Lista alfabética de métodos ObjectPAL

compact
constantNameToValue
contains
convertPointWithRespectTo
copy
copyFromArray
copyRecord
copyToArray
cos
cosh
count
countOf
CPUClockTime
create
cSamStd
cSamVar
cStd
cSum
currRecord
currency
currentPage
cVar
data
dataType
date
DateTime
dateVal
day
daysInMonth
debug
delayScreenUpdates
delete
deleteDir
deleteRecord
design
didFlyAway
disableBreakMessage
distance
dlgAdd
dlgCopy
dlgCreate
dlgDelete
dlgEmpty
dlgNetDrivers
dlgNetLocks
dlgNetRefresh
dlgNetRetry
dlgNetSetLocks
dlgNetSystem
dlgNetUserName
dlgNetWho
dlgRename
dlgRestructure
dlgSort
dlgSubtract

dlgTableInfo
DMAddTable
DMGet
DMHasTable
DMPut
DMRemoveTable
dow
dowOrd
doy
drives
dropIndex
edit
empty
end
endEdit
enumAliasNames
enumDataBaseTables
enumDesktopWindowNames
enumDriverCapabilities
enumDriverInfo
enumDriverNames
enumDriverTopics
enumEngineInfo
enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumFileList
enumFolder
enumFonts
enumFormNames
enumIndexStruct
enumLocks
enumObjectNames
enumOpenDatabases
enumRefIntStruct
enumReportNames
enumRTLClassNames
enumRTLConstants
enumRTLMethods
enumSecStruct
enumSource
enumSourceToFile
enumTableLinks
enumTableProperties
enumUIClasses
enumUIObjectNames
enumUIObjectProperties
enumUsers
enumWindowNames
eof
eot
errorClear
errorCode
errorLog
errorMessage

errorPop
errorTrapOnWarnings
exchange
execMethod
execute
executeQBE
executeQBEFile
executeQBEString
existDrive
exit
exp
fail
familyRights
fieldName
fieldNo
fieldRights
fieldSize
fieldType
fieldUnits2
fieldValue
fileBrowser
fill
findFirst
findNext
floor
formCaller
formReturn
format
fraction
freeDiskSpace
fullName
fv
getAliasPath
getBoundingBox
getDestination
getDir
getDrive
getLanguageDriver
getLanguageDriverDesc
getMenuChoiceAttribute
getMenuChoiceAttributeById
getMousePosition
getMouseScreenPosition
getNetUserName
getObjectHit
getPosition
getProperty
getPropertyAsString
getTarget
getTitle
getValidFileExtensions
grow
hasMenuChoiceAttribute
hasMouse
helpOnHelp

helpQuit
helpSetIndex
helpShowContext
helpShowIndex
helpShowTopic
helpShowTopicInKeywordTable
hide
hideSpeedBar
home
hour
id
ignoreCaseInLocate
ignoreCaseInStringCompares
ln
indexOf
initRecord
insert
insertAfter
insertAfterRecord
insertBefore
insertBeforeRecord
insertFirst
insertRecord
int
isAbove
isAdvancedWildcardsInLocate
isAltKeyDown
isAssigned
isBelow
isBlank
isBlankZero
isContainerValid
isControlKeyDown
isDir
isEdit
isEmpty
isEncrypted
isFile
isFirstTime
isFromUI
isFixed
isFixedType
isIgnoreCaseInLocate
isIgnoreCaseInStringCompares
isInside
isLastMouseClickedValid
isLastMouseRightClickedValid
isLeapYear
isLeft
isLeftDown
isMaximized
isMiddleDown
isMinimized
isPreFilter
isRecordDeleted

isRemote
isRemovable
isResizeable
isRight
isRightDown
isShared
isShiftKeyDown
isShowDeletedOn
isSpace
isSpeedBarShowing
isTable
isTargetSelf
isValid
isVisible
keyChar
keyNameToChar
keyNameToVKCode
keyPhysical
killTimer
load
locate
locateNext
locateNextPattern
locatePattern
locatePrior
locatePriorPattern
lock
lockRecord
lockStatus
log
logical
longInt
lower
makeDir
match
max
maximize
menuAction
menuChoice
message
methodDelete
methodGet
methodSet
milliSec
min
minimize
minute
mod
month
mouseDouble
mouseDown
mouseEnter
mouseExit
mouseMove
mouseRightDouble

mouseRightDown
mouseRightUp
mouseUp
moveTo
moveToPage
movetoRecNo
moveToRecord
moy
msgAbortRetryIgnore
msgInfo
msgQuestion
msgRetryCancel
msgStop
msgYesNoCancel
nFields
nKeyFields
nRecords
name
nextRecord
numVal
number
oemCode
open
openAsDialog
pixelsToTwips
play
pmt
point
position
postAction
postRecord
pow
pow10
pushButton
priorRecord
privDir
protect
pv
qLocate
rand
reIndex
reIndexAll
readChars
readEnvironmentString
readFromClipboard
readFromFile
readLine
readProfileString
reason
recNo
recordStatus
remove
removeAlias
removeAllItems
removeAllPasswords

removeItem
removeMenu
removePassword
rename
replaceItem
resync
retryPeriod
rgb
run
save
saveCFG
search
second
seqNo
setAliasPath
setAltKeyDown
setChar
setControlKeyDown
setData
setDir
setDrive
setErrorCode
setFieldValue
setFilter
setFlyAwayControl
setId
setIndex
setInside
setItem
setLeftDown
setMenuChoiceAttribute
setMenuChoiceAttributeById
setMiddleDown
setMousePosition
setMouseScreenPosition
setMouseShape
setNewValue
setPosition
setProperty
setReason
setRetryPeriod
setRightDown
setShiftKeyDown
setSize
setStatusValue
setTimer
setTitle
setVChar
setVCharCode
setX
setXY
setY
show
showDeleted
showSpeedBar

sin
sinh
size
skip
sleep
smallInt
sort
sortTo
sound
space
splitFullFileName
sqrt
startUpDir
statusValue
strVal
string
substr
subtract
switchIndex
sysInfo
tableName
tableRights
tan
tanh
time
toANSI
toOEM
today
totalDiskSpace
twipsToPixels
type
unAssign
unAttach
unDeleteRecord
unLock
unLockRecord
unprotect
updateRecord
upper
usesIndexes
vChar
vCharCode
version
view
vkCodeToKeyName
wait
wasLastClicked
wasLastRightClicked
windowClientHandle
windowHandle
windowsDir
windowsSystemDir
workingDir
writeEnvironmentString
writeLine

writeProfileString
writeQBE
writeString
writeToClipboard
writeToFile
x
y
year

bitAND

Selecciona el TIPO
- LongInt
- SmallInt

bitOR

Selecciona el TIPO

- LongInt
- SmallInt

bitXOR

Selecciona el TIPO

- LongInt
- SmallInt

CPUClockTime

Selecciona el TIPO
System

formReturn

Selecciona el TIPO

Form

pixelsToTwips

Selecciona el TIPO
- System
- UIObject

statusValue

Selecciona el TIPO
StatusEvent

twipsToPixels

Selecciona el TIPO
- System
- UIObject

abs

Selecciona el TIPO
    Number

accessRights

Selecciona el TIPO
FileSystem

acos

Selecciona el TIPO

Number

action

Selecciona el TIPO

- Form
- UIObject
- TableView

actionClass

Selecciona el TIPO
ActionEvent

add

Selecciona el TIPO

- TCursor
- Table

addAlias

Selecciona el TIPO
    Session

addArray

Selecciona el TIPO
- Menu
- PopupMenu

addBar

Selecciona el TIPO

PopupMenu

addBreak

Selecciona el TIPO
- Menu
- PopupMenu

addLast

Selecciona el TIPO

Array

addPassword

Selecciona el TIPO
Session

addPopup

Selecciona el TIPO
- Menu
- PopupMenu

addSeparator

Selecciona el TIPO
PopupMenu

addStaticText

Selecciona el TIPO

- Menu
- PopupMenu

addText

Selecciona el TIPO
- Menu
- PopupMenu

advMatch

Selecciona el TIPO
- String
- TextStream

advancedWildcardsInLocate

Selecciona el TIPO
    Session

ansiCode

Selecciona el TIPO
    String

append

Selecciona el TIPO

Array

asin

Selecciona el TIPO
Number

atFirst

Selecciona el TIPO
- TCursor
- UIObject

atLast

Selecciona el TIPO
- TCursor
- UIObject

atan2

Selecciona el TIPO

Number

attach

Selecciona el TIPO

- Form
- TCursor
- Table
- UIObject

attachToKeyViol

Select theTYPE
TCursor

beep

Selecciona el TIPO

System

bitIsSet

Selecciona el TIPO
- longint
- SmallInt

blank

Selecciona el TIPO
    AnyType

blankAsZero

Selecciona el TIPO
    Session

bot

Selecciona el TIPO
    TCursor

breakApart

Selecciona el TIPO
    String

bringToTop

Selecciona el TIPO

Form

broadcastAction

Selecciona el TIPO
UIObject

canReadFromClipboard

Selecciona el TIPO
OLE

cAverage

Selecciona el TIPO

- Table
- TCursor

cCount

Selecciona el TIPO

- Table
- TCursor

cMax

Selecciona el TIPO

- Table
- TCursor

cMin

Selecciona el TIPO
- Table
- TCursor

cNpv

Selecciona el TIPO

- Table
- TCursor

cSamStd

Selecciona el TIPO

- Table
- TCursor

cSamVar

Selecciona el TIPO

- Table
- TCursor

cStd

Selecciona el TIPO
- Table
- TCursor

cSum

Selecciona el TIPO

- Table
- TCursor

currentPage

Selecciona el TIPO
Report

cVar

Selecciona el TIPO

- Table
- TCursor

cancel

Selecciona el TIPO

cancelEdit

Selecciona el TIPO
- UIObject
- TCursor

ceil

Selecciona el TIPO
    Number

char

Selecciona el TIPO
KeyEvent

charAnsiCode

Selecciona el TIPO
KeyEvent

chr

Selecciona el TIPO

String

chrOEM

Selecciona el TIPO

String

chrToKeyName

Selecciona el TIPO
    String

close

Selecciona el TIPO

- DDE
- DataBase
- Form
- Report
- Session
- TCursor
- TableView
- TextStream
- Library
- System

commit

Selecciona el TIPO
    TextStream

compact

Selecciona el TIPO
- TCursor
- Table

constantNameToValue

Selecciona el TIPO
    System

constantValueToName

Selecciona el TIPO
    System

contains

Selecciona el TIPO
- Array
- DynArray
- Menu

convertPointWithRespectTo

Selecciona el TIPO
UIObject

copy

Selecciona el TIPO

- FileSystem
- TCursor
- Table

copyFromArray

Selecciona el TIPO
- TCursor
- UIObject

copyRecord

Selecciona el TIPO
    TCursor

copyToArray

Selecciona el TIPO
- TCursor
- UIObject

cos

Selecciona el TIPO
Number

cosh

Selecciona el TIPO
    Number

count

Selecciona el TIPO

Menu

countOf

Selecciona el TIPO
Array

create

Selecciona el TIPO

- Form
- TextStream
- UIObject

currRecord

Selecciona el TIPO

- TCursor
- UIObject

currency

Selecciona el TIPO
    Currency

data

Selecciona el TIPO

MenuEvent

dataType

Selecciona el TIPO
AnyType

date

Selecciona el TIPO
Date

DateTime

Selecciona el TIPO
datetime

dateVal

Selecciona el TIPO
Date

day

Selecciona el TIPO
DateTime

daysInMonth

Selecciona el TIPO
    DateTime

debug

Selecciona el TIPO

System

delayScreenUpdates

Selecciona el TIPO
Form

delete

Selecciona el TIPO
- DataBase
- FileSystem
- UIObject

deleteDir

Selecciona el TIPO
FileSystem

deleteRecord

Selecciona el TIPO
- TCursor
- UIObject

design

Selecciona el TIPO

- Form
- Report

didFlyAway

Selecciona el TIPO
TCursorx

disableBreakMessage

Selecciona el TIPO

Form

distance

Selecciona el TIPO
    Point

dlgAdd

Selecciona el TIPO
System

dlgCopy

Selecciona el TIPO
System

dlgCreate

Selecciona el TIPO

System

dlgDelete

Selecciona el TIPO
System

dlgEmpty

Selecciona el TIPO
System

dlgNetDrivers

Selecciona el TIPO
    System

dlgNetLocks

Selecciona el TIPO
System

dlgNetRefresh

Selecciona el TIPO
    System

dlgNetRetry

Selecciona el TIPO
    System

dlgNetSetLocks

Selecciona el TIPO
    System

dlgNetSystem

Selecciona el TIPO
System

dlgNetUserName

Selecciona el TIPO
System

dlgNetWho

Selecciona el TIPO
    System

dlgRename

Selecciona el TIPO
System

dlgRestructure

Selecciona el TIPO
    System

dlgSort

Selecciona el TIPO
System

dlgSubtract

Selecciona el TIPO
System

dlgTableInfo

Selecciona el TIPO
    System

DMAddTable

Selecciona el TIPO
Form

DMGet

Selecciona el TIPO
Form

DMHasTable

Selecciona el TIPO
- Form

DMPut

Selecciona el TIPO

Form

DMRemoveTable

Selecciona el TIPO
Form

dow

Selecciona el TIPO
    DateTime

dowOrd

Selecciona el TIPO
DateTime

doy

Selecciona el TIPO
DateTime

drives

Selecciona el TIPO
    FileSystem

dropIndex

Selecciona el TIPO
- TCursor
- Table

edit

Selecciona el TIPO

- TCursor
- UIObject
- OLE

empty

Selecciona el TIPO

- Array
- DynArray
- Menu
- TCursor
- Table
- UIObject

end

Selecciona el TIPO

- TCursor
- UIObject
- TextStream

endEdit

Selecciona el TIPO
- TCursor
- UIObject

enumAliasNames

Selecciona el TIPO
    Session

enumDataBaseTables

Selecciona el TIPO
    Session

enumDesktopWindowNames

Selecciona el TIPO
System

enumDriverCapabilities

Selecciona el TIPO
    Session

enumDriverInfo

Selecciona el TIPO
Session

enumDriverNames

Selecciona el TIPO
Session

enumDriverTopics

Selecciona el TIPO
    Session

enumEngineInfo

Selecciona el TIPO
Session

enumFieldNames

Selecciona el TIPO
- Table
- TCursor
- UIObject

enumFieldNamesInIndex

Selecciona el TIPO
- Table
- TCursor

enumFieldStruct

Selecciona el TIPO
- Table

enumFileList

Selecciona el TIPO
FileSystem

enumFolder

Selecciona el TIPO

Session

enumFonts

Selecciona el TIPO
System

enumFormNames

Selecciona el TIPO
    System

enumIndexStruct

Selecciona el TIPO
- Table
- TCursor

enumLocks

Selecciona el TIPO
- TCursor
- UIObject

enumObjectNames

Selecciona el TIPO
    UIObject

enumOpenDatabases

Selecciona el TIPO
    Session

enumRefIntStruct

Selecciona el TIPO

- Table
- TCursor

enumReportNames

Selecciona el TIPO
    System

enumRTLClassNames

Selecciona el TIPO
    System

enumRTLConstants

Selecciona el TIPO
    System

enumRTLMethods

Selecciona el TIPO

System

enumSecStruct

Selecciona el TIPO

- Table
- TCursor

enumSource

Selecciona el TIPO

- Library
- UIObject
- Form

enumSourceToFile

Selecciona el TIPO

- Library
- UIObject
- Form

enumTableLinks

Selecciona el TIPO
Form

enumTableProperties

Selecciona el TIPO
    TCursor

enumUIClasses

Selecciona el TIPO
    UIObject

enumUIObjectNames

Selecciona el TIPO

- Form
- Report
- UIObject

enumUIObjectProperties

Selecciona el TIPO

- Form
- Report
- UIObject

enumUsers

Selecciona el TIPO
    Session

enumVerbs

Selecciona el TIPO
OLE

enumWindowNames

Selecciona el TIPO
    System

eof

Selecciona el TIPO
TextStream

eot

Selecciona el TIPO
    TCursor

errorClear

Selecciona el TIPO
    System

errorCode

Selecciona el TIPO
- System
- Event

errorLog

Selecciona el TIPO

System

errorMessage

Selecciona el TIPO
    System

errorPop

Selecciona el TIPO
System

errorShow

Selecciona el TIPO
System

errorTrapOnWarnings

Selecciona el TIPO
    System

exchange

Selecciona el TIPO
Array

execMethod

Selecciona el TIPO
- UIObject
- Library

execute

Selecciona el TIPO
- DDE
- System

executeQBE

Selecciona el TIPO
- DataBase
- Query

executeQBEFile

Selecciona el TIPO
DataBase

executeQBEString

Selecciona el TIPO
DataBase

existDrive

Selecciona el TIPO
FileSystem

exit

Selecciona el TIPO

System

exp

Selecciona el TIPO
Number

fail

Selecciona el TIPO
System

familyRights

Selecciona el TIPO
- Table
- TCursor

fieldName

Selecciona el TIPO
Table

fieldNo

Selecciona el TIPO
- Table
- TCursor

fieldRights

Selecciona el TIPO
    TCursor

fieldSize

Selecciona el TIPO
TCursor

fieldType

Selecciona el TIPO

- Table
- TCursor

fieldUnits2

Selecciona el TIPO
    TCursor

fieldValue

Selecciona el TIPO
TCursor

fileBrowser

Selecciona el TIPO
    System

fill

Selecciona el TIPO

- Array
- String

findFirst

Selecciona el TIPO
    FileSystem

findNext

Selecciona el TIPO
    FileSystem

formCaller

Selecciona el TIPO
Form

floor

Selecciona el TIPO
Number

format

Selecciona el TIPO
String

formatAdd

Selecciona el TIPO
System

formatDelete

Selecciona el TIPO
    System

formatExist

Selecciona el TIPO
System

formatSetCurrencyDefault

Selecciona el TIPO
    System

formatSetDateDefault

Selecciona el TIPO
    System

formatSetDateTimeDefault

Selecciona el TIPO

System

formatSetLogicalDefault

Selecciona el TIPO
System

formatSetLongIntDefault

Selecciona el TIPO
    System

formatSetNumberDefault

Selecciona el TIPO
    System

formatSetSmallIntDefault

Selecciona el TIPO
    System

formatSetTimeDefault

Selecciona el TIPO
System

fraction

Selecciona el TIPO
    Number

freeDiskSpace

Selecciona el TIPO
FileSystem

fullName

Selecciona el TIPO
FileSystem

fv

Selecciona el TIPO
Number

getAliasPath

Selecciona el TIPO
Session

getBoundingBox

Selecciona el TIPO
    UIObject

getDestination

Selecciona el TIPO
MoveEvent

getDir

Selecciona el TIPO
    FileSystem

getDrive

Selecciona el TIPO
FileSystem

getFileAccessRights

Selecciona el TIPO
FileSystem

getKeys

Selecciona el TIPO

DynArray

getLanguageDriver

Selecciona el TIPO
    TCursor

getLanguageDriverDesc

Selecciona el TIPO
    TCursor

getMenuChoiceAttribute

Selecciona el TIPO
    Menu

getMenuChoiceAttributeById

Selecciona el TIPO

Menu

getMousePosition

Selecciona el TIPO
MouseEvent

getMouseScreenPosition

Selecciona el TIPO

System

getNetUserName

Selecciona el TIPO
Session

getObjectHit

Selecciona el TIPO
MouseEvent

getPosition

Selecciona el TIPO
- UIObject
- Form

getProperty

Selecciona el TIPO
    UIObject

getPropertyAsString

Selecciona el TIPO
    UIObject

getRGB

Selecciona el TIPO
    UIObject

getServerName

Selecciona el TIPO
OLE

getTarget

Selecciona el TIPO
Event

getTitle

Selecciona el TIPO

Form

getValidFileExtensions

Selecciona el TIPO
FileSystem

grow

Selecciona el TIPO
Array

hasMenuChoiceAttribute

Selecciona el TIPO

Menu

hasMouse

Selecciona el TIPO
    UIObject

helpOnHelp

Selecciona el TIPO

System

helpQuit

Selecciona el TIPO

System

helpSetIndex

Selecciona el TIPO
    System

helpShowContext

Selecciona el TIPO
    System

helpShowIndex

Selecciona el TIPO
    System

helpShowTopic

Selecciona el TIPO
    System

helpShowTopicInKeywordTable

Selecciona el TIPO
    System

hide

Selecciona el TIPO

Form

hideSpeedBar

Selecciona el TIPO
Form

home

Selecciona el TIPO

- TCursor
- UIObject
- TextStream

hour

Selecciona el TIPO
DateTime

id

Selecciona el TIPO
- ActionEvent
- MenuEvent

ignoreCaseInLocate

Selecciona el TIPO
Session

ignoreCaseInStringCompares

Selecciona el TIPO
    String

indexOf

Selecciona el TIPO

Array

initRecord

Selecciona el TIPO
    TCursor

insert

Selecciona el TIPO

Array

insertAfter

Selecciona el TIPO
Array

insertAfterRecord

Selecciona el TIPO
- TCursor
- UIObject

insertBefore

Selecciona el TIPO

Array

insertBeforeRecord

Selecciona el TIPO
- TCursor
- UIObject

insertFirst

Selecciona el TIPO

Array

insertRecord

Selecciona el TIPO
- TCursor
- UIObject

int

Selecciona el TIPO
SmallInt

isAbove

Selecciona el TIPO

Point

isAdvancedWildcardsInLocate

Selecciona el TIPO
Session

isAltKeyDown

Selecciona el TIPO
KeyEvent

isAssigned

Selecciona el TIPO
- AnyType
- Session
- DataBase
- Table

isBelow

Selecciona el TIPO

Point

isBlank

Selecciona el TIPO
AnyType

isBlankZero

Selecciona el TIPO
    Session

isContainerValid

Selecciona el TIPO
UIObject

isControlKeyDown

Selecciona el TIPO
- KeyEvent
- MouseEvent

isDir

Selecciona el TIPO
    FileSystem

isEdit

Selecciona el TIPO
- UIObject
- TCursor

isEmpty

Selecciona el TIPO

- Table
- TCursor
- UIObject

isEncrypted

Selecciona el TIPO
- Table
- TCursor

isFile

Selecciona el TIPO
FileSystem

isFirstTime

Selecciona el TIPO
    Event

isFixed

Selecciona el TIPO
FileSystem

isFixedType

Selecciona el TIPO
AnyType

isFromUI

Selecciona el TIPO
- KeyEvent
- MenuEvent
- MouseEvent

isIgnoreCaseInLocate

Selecciona el TIPO
Session

isIgnoreCaseInStringCompares

Selecciona el TIPO
    String

isInside

Selecciona el TIPO
MouseEvent

isLastMouseClickedValid

Selecciona el TIPO
    UIObject

isLastMouseRightClickedValid

Selecciona el TIPO
UIObject

isLeapYear

Selecciona el TIPO
DateTime

isLeft

Selecciona el TIPO

Point

isLeftDown

Selecciona el TIPO
MouseEvent

isMaximized

Selecciona el TIPO

Form

isMiddleDown

Selecciona el TIPO
MouseEvent

isMinimized

Selecciona el TIPO

Form

isPreFilter

Selecciona el TIPO
Event

isRecordDeleted

Selecciona el TIPO
- TCursor
- UIObject

isRemote

Selecciona el TIPO
FileSystem

isRemovable

Selecciona el TIPO
FileSystem

isResizeable

Selecciona el TIPO
Array

isRight

Selecciona el TIPO
    Point

isRightDown

Selecciona el TIPO
MouseEvent

isShared

Selecciona el TIPO

- Table
- TCursor

isShiftKeyDown

Selecciona el TIPO
- KeyEvent
- MouseEvent

isShowDeletedOn

Selecciona el TIPO
TCursor

isSpace

Selecciona el TIPO
    String

isSpeedBarShowing

Selecciona el TIPO
Form

isTable

Selecciona el TIPO
- DataBase
- Table

isTargetSelf

Selecciona el TIPO
Event

isValid

Selecciona el TIPO
    TCursor

isVisible

Selecciona el TIPO

Form

keyChar

Selecciona el TIPO
- Form
- UIObject

keyNameToChar

Selecciona el TIPO
    String

keyNameToVKCode

Selecciona el TIPO
    String

keyPhysical

Selecciona el TIPO

- Form
- UIObject

killTimer

Selecciona el TIPO
    UIObject

In

Selecciona el TIPO
Number

load

Selecciona el TIPO

- Form
- Report

locate

Selecciona el TIPO
- TCursor
- UIObject

locateNext

Selecciona el TIPO
- TCursor
- UIObject

locateNextPattern

Selecciona el TIPO
- TCursor
- UIObject

locatePattern

Selecciona el TIPO
- TCursor
- UIObject

locatePrior

Selecciona el TIPO
- TCursor
- UIObject

locatePriorPattern

Selecciona el TIPO
- TCursor
- UIObject

lock

Selecciona el TIPO

- Session
- Table
- TCursor

lockRecord

Selecciona el TIPO
- TCursor
- UIObject

lockStatus

Selecciona el TIPO
- TCursor
- UIObject

log

Selecciona el TIPO
Number

logical

Selecciona el TIPO
    Logical

longInt

Selecciona el TIPO
    LongInt

lower

Selecciona el TIPO

String

lTrim

Selecciona el TIPO

String

makeDir

Selecciona el TIPO
    FileSystem

match

Selecciona el TIPO
    String

max

Selecciona el TIPO
Number

maximize

Selecciona el TIPO

Form

memo

Selecciona el TIPO

Memo

menuAction

Selecciona el TIPO
- Form
- UIObject

menuChoice

Selecciona el TIPO
MenuEvent

message

Selecciona el TIPO
System

methodDelete

Selecciona el TIPO

- Form
- UIObject

methodGet

Selecciona el TIPO

- Form
- UIObject

methodSet

Selecciona el TIPO
- UIObject
- Form

milliSec

Selecciona el TIPO
DateTime

min

Selecciona el TIPO
Number

minimize

Selecciona el TIPO

Form

minute

Selecciona el TIPO
DateTime

mod

Selecciona el TIPO

Number

month

Selecciona el TIPO
DateTime

mouseClick

Selecciona el TIPO
    UIObject

mouseDouble

Selecciona el TIPO

- Form
- UIObject

mouseDown

Selecciona el TIPO

- Form
- UIObject

mouseEnter

Selecciona el TIPO
- Form
- UIObject

mouseExit

Selecciona el TIPO

- Form
- UIObject

mouseMove

Selecciona el TIPO

- Form
- UIObject

mouseRightDouble

Selecciona el TIPO

- Form
- UIObject

mouseRightDown

Selecciona el TIPO

- Form
- UIObject

mouseRightUp

Selecciona el TIPO

- Form
- UIObject

mouseUp

Selecciona el TIPO

- Form
- UIObject

moveTo

Selecciona el TIPO

- Form
- UIObject

moveToPage

Selecciona el TIPO

- Form
- Report

moveToRecNo

Selecciona el TIPO
- UIObject
- TCursor

moveToRecord

Selecciona el TIPO
- TCursor
- UIObject

moy

Selecciona el TIPO
DateTime

msgAbortRetryIgnore

Selecciona el TIPO
    System

msgInfo

Selecciona el TIPO
System

msgQuestion

Selecciona el TIPO
    System

msgRetryCancel

Selecciona el TIPO
System

msgStop

Selecciona el TIPO
    System

msgYesNoCancel

Selecciona el TIPO
    System

nFields

Selecciona el TIPO

- Table
- TCursor
- UIObject

nKeyFields

Selecciona el TIPO
- Table
- TCursor
- UIObject

nRecords

Selecciona el TIPO
- Table
- TCursor
- UIObject

name

Selecciona el TIPO
    FileSystem

nextRecord

Selecciona el TIPO
- TCursor
- UIObject

numVal

Selecciona el TIPO
    Number

number

Selecciona el TIPO
Number

oemCode

Selecciona el TIPO
String

open

Selecciona el TIPO

- DataBase
- DDE
- Form
- Report
- Session
- TableView
- TextStream
- TCursor
- Library

openAsDialog

Selecciona el TIPO

Form

play

Selecciona el TIPO
    System

pmt

Selecciona el TIPO
    Number

point

Selecciona el TIPO

Point

position

Selecciona el TIPO
    TextStream

postAction

Selecciona el TIPO

- Form
- UIObject

postRecord

Selecciona el TIPO

- TCursor
- UIObject

pow

Selecciona el TIPO
    Number

pow10

Selecciona el TIPO

Number

priorRecord

Selecciona el TIPO
- TCursor
- UIObject

privDir

Selecciona el TIPO
FileSystem

pushButton

Selecciona el TIPO
    UIObject

protect

Selecciona el TIPO

Table

pv

Selecciona el TIPO
Number

qLocate

Selecciona el TIPO
    TCursor

rTrim

Selecciona el TIPO
    String

rand

Selecciona el TIPO

Number

reIndex

Selecciona el TIPO
- Table
- TCursor

reIndexAll

Selecciona el TIPO
- Table
- TCursor

readChars

Selecciona el TIPO
    TextStream

readEnvironmentString

Selecciona el TIPO
    System

readFromClipboard

Selecciona el TIPO
- Graphic
- OLE

readFromFile

Selecciona el TIPO
- Graphic
- Memo
- Binary

readLine

Selecciona el TIPO
TextStream

readProfileString

Selecciona el TIPO
System

reason

Selecciona el TIPO
- Event
- MenuEvent
- StatusEvent
- ErrorEvent
- MoveEvent

recNo

Selecciona el TIPO
    TCursor

recordStatus

Selecciona el TIPO
- TCursor
- UIObject

remove

Selecciona el TIPO

- Array
- Menu

removeAlias

Selecciona el TIPO
    Session

removeAllItems

Selecciona el TIPO

Array

removeAllPasswords

Selecciona el TIPO
    Session

removeItem

Selecciona el TIPO
- Array
- DynArray

removeMenu

Selecciona el TIPO
    Menu

removePassword

Selecciona el TIPO
    Session

rename

Selecciona el TIPO
- FileSystem
- Table

replaceItem

Selecciona el TIPO

Array

resync

Selecciona el TIPO
    UIObject

retryPeriod

Selecciona el TIPO
Session

rgb

Selecciona el TIPO
    UIObject

run

Selecciona el TIPO

Form
Report

save

Selecciona el TIPO
Form

saveCFG

Selecciona el TIPO

Session

search

Selecciona el TIPO
    String

second

Selecciona el TIPO
DateTime

seqNo

Selecciona el TIPO
    TCursor

setAliasPath

Selecciona el TIPO
Session

setAltKeyDown

Selecciona el TIPO
KeyEvent

setChar

Selecciona el TIPO
KeyEvent

setControlKeyDown

Selecciona el TIPO
- MouseEvent
- KeyEvent

setData

Selecciona el TIPO
MenuEvent

setDir

Selecciona el TIPO
    FileSystem

setDrive

Selecciona el TIPO
FileSystem

setErrorCode

Selecciona el TIPO
Event

setFieldValue

Selecciona el TIPO
TCursor

setFilter

Selecciona el TIPO
- Table
- TCursor
- UIObject

setFlyAwayControl

Selecciona el TIPO
TCursor

setId

Selecciona el TIPO
- ActionEvent
- MenuEvent

setIndex

Selecciona el TIPO
Table

setInside

Selecciona el TIPO
MouseEvent

setItem

Selecciona el TIPO
DDE

setLeftDown

Selecciona el TIPO
MouseEvent

setMenuChoiceAttribute

Selecciona el TIPO
Menu

setMenuChoiceAttributeById

Selecciona el TIPO
    Menu

setMiddleDown

Selecciona el TIPO
MouseEvent

setMousePosition

Selecciona el TIPO
MouseEvent

setMouseScreenPosition

Selecciona el TIPO

System

setMouseShape

Selecciona el TIPO
    System

setNewValue

Selecciona el TIPO
ValueEvent

setPosition

Selecciona el TIPO

- Form
- TextStream
- UIObject

setProperty

Selecciona el TIPO
UIObject

setReason

Selecciona el TIPO
- Event
- MenuEvent
- StatusEvent
- ErrorEvent
- MoveEvent

setRetryPeriod

Selecciona el TIPO
Session

setRightDown

Selecciona el TIPO
MouseEvent

setShiftKeyDown

Selecciona el TIPO
MouseEvent

setSize

Selecciona el TIPO

Array

setstatusValue

Selecciona el TIPO
StatusEvent

setTimer

Selecciona el TIPO
    UIObject

setTitle

Selecciona el TIPO

Form

setVChar

Selecciona el TIPO
KeyEvent

setVCharCode

Selecciona el TIPO
KeyEvent

setX

Selecciona el TIPO
- MouseEvent
- Point

setXY

Selecciona el TIPO
Point

setY

Selecciona el TIPO

- MouseEvent
- Point

show

Selecciona el TIPO
- Form
- Menu
- PopupMenu

showDeleted

Selecciona el TIPO
- Table
- TCursor

showSpeedBar

Selecciona el TIPO
Form

sin

Selecciona el TIPO
    Number

sinh

Selecciona el TIPO
Number

size

Selecciona el TIPO
- Array
- DynArray
- FileSystem
- String
- TextStream
- Binary

skip

Selecciona el TIPO
- TCursor
- UIObject

sleep

Selecciona el TIPO

System

smallInt

Selecciona el TIPO
SmallInt

sort

Selecciona el TIPO

Table

sortTo

Selecciona el TIPO
    TCursor

sound

Selecciona el TIPO

System

space

Selecciona el TIPO

String

splitFullFileName

Selecciona el TIPO
FileSystem

sqrt

Selecciona el TIPO
    Number

startUpDir

Selecciona el TIPO
FileSystem

strVal

Selecciona el TIPO
    String

string

Selecciona el TIPO
String

substr

Selecciona el TIPO

String

subtract

Selecciona el TIPO

- Table
- TCursor

switchIndex

Selecciona el TIPO
- TCursor
- UIObject

sysInfo

Selecciona el TIPO

System

tableName

Selecciona el TIPO
TCursor

tableRights

Selecciona el TIPO
- Table
- TCursor

tan

Selecciona el TIPO
Number

tanh

Selecciona el TIPO
Number

time

Selecciona el TIPO
- Time
- FileSystem

toANSI

Selecciona el TIPO
    String

toOEM

Selecciona el TIPO

String

today

Selecciona el TIPO
Date

totalDiskSpace

Selecciona el TIPO
FileSystem

tracerClear

Selecciona el TIPO
    System

tracerHide

Selecciona el TIPO
    System

tracerOff

Selecciona el TIPO
    System

tracerOn

Selecciona el TIPO
    System

tracerSave

Selecciona el TIPO
System

tracerShow

Selecciona el TIPO
System

tracerToTop

Selecciona el TIPO
    System

tracerWrite

Selecciona el TIPO
System

type

Selecciona el TIPO
- Table
- TCursor

unAssign

Selecciona el TIPO
    AnyType

unAttach

Selecciona el TIPO
    Table

unDeleteRecord

Selecciona el TIPO
- UIObject
- TCursor

unLock

Selecciona el TIPO

- Session
- Table
- TCursor

unlockRecord

Selecciona el TIPO
- TCursor
- UIObject

unprotect

Selecciona el TIPO

Table

updateRecord

Selecciona el TIPO
    TCursor

upper

Selecciona el TIPO

String

usesIndexes

Selecciona el TIPO
Table

vChar

Selecciona el TIPO
    KeyEvent

vCharCode

Selecciona el TIPO
KeyEvent

version

Selecciona el TIPO

System

view

Selecciona el TIPO

- Array
- UIObject
- Record
- DynArray

vkCodeToKeyName

Selecciona el TIPO
    String

wait

Selecciona el TIPO

- Form
- TableView

wasLastClicked

Selecciona el TIPO
UIObject

wasLastRightClicked

Selecciona el TIPO
UIObject

winGetMessageID

Selecciona el TIPO
    System

winPostMessage

Selecciona el TIPO
    System

winSendMessage

Selecciona el TIPO
    System

windowClientHandle

Selecciona el TIPO
Form

windowsDir

Selecciona el TIPO
FileSystem

windowHandle

Selecciona el TIPO

Form

windowsSystemDir

Selecciona el TIPO
    FileSystem

workingDir

Selecciona el TIPO
FileSystem

writeEnvironmentString

Selecciona el TIPO
    System

writeLine

Selecciona el TIPO
    TextStream

writeProfileString

Selecciona el TIPO
System

writeQBE

Selecciona el TIPO
- DataBase
- Query

writeString

Selecciona el TIPO
TextStream

writeToClipboard

Selecciona el TIPO

- Graphic
- OLE

writeToFile

Selecciona el TIPO
- Graphic
- Memo
- Binary

x

Selecciona el TIPO
- MouseEvent
- Point

y

Selecciona el TIPO
- MouseEvent
- Point

year

Selecciona el TIPO
DateTime

Introducción a ObjectPAL

ObjectPAL es el lenguaje de aplicación de Paradox basado en objetos y controlado por sucesos, diferente de un lenguaje de procedimientos tradicional en diversos aspectos:

- Los lenguajes tradicionales permiten crear archivos de comandos que se ejecutan uno detrás de otro.
- ObjectPAL permite situar objetos (por ejemplo, botones y campos) en una ficha o un informe y anexarles módulos de código, denominados métodos, que se ejecutan cuando le sucede algo al objeto.

ObjectPAL cuenta con dos aspectos:

- el propio lenguaje (sus tipos de objetos, métodos, procedimientos y construcciones)
- el Entorno integrado de desarrollo (IDE), que incluye:
  - El Editor de ObjectPAL.
  - El Depurador
  - Un mecanismo que permite crear y ejecutar macros
  - Las funciones de envío de la aplicación.

Vea también

Referencia a tipos de ObjectPAL
Tareas de programación
Componentes principales
Objetos
Sucesos
Componentes del lenguaje de ObjectPAL
Estructura del lenguaje de ObjectPAL
Propiedades
ObjectPAL IDE
Uso de tablas

Tareas de programación

A continuación se incluye un esquema del lenguaje de ObjectPAL, que sirve para acceder a sus aspectos concretos. Antes de consultar cualquiera de estos temas, sin embargo, asegúrese de que consulta la información sobre los objetos y sucesos.

Mensajes y cuadros de diálogo	Los mensajes y cuadros de diálogo estándar proporcionan una forma de interacción con el usuario.
Manejo de sucesos de teclado	Es posible controlar cualquier pulsación de tecla en ObjectPAL, lo que significa que pueden desarrollarse con toda facilidad teclas rápidas para las aplicaciones.
Uso de menús	ObjectPAL permite definir menús y menús emergentes para mostrar opciones a los usuarios.
Uso de listas	Puede utilizar cuadros de lista y cuadros de edición desplegables para permitir que un usuario seleccione un elemento entre un grupo de ellos.
Aplicaciones multificha	Para diseñar aplicaciones que utilizan más de una ficha, es necesario conocer la forma de apertura y control de una ficha desde otra. Las fichas también pueden abrirse como cuadros de diálogo.
Uso de archivos de texto	Puede utilizar ObjectPAL para trabajar con archivos de texto, que se denominan TextStreams en ObjectPAL.
Uso de DLL	Mediante la cláusula Uses pueden declararse y utilizarse funciones activadas desde DLL (Bibliotecas de vínculo dinámico).
Uso del sistema de archivos	Mediante los métodos en el tipo FileSystem, es posible acceder a y obtener información sobre archivos de disco, unidades y directorios. ObjectPAL también incluye un cuadro de diálogo estándar para localizar archivos.

Vea también

Objetos
Sucesos

Componentes principales

Paradox y ObjectPAL permiten crear aplicaciones compiladas a partir de los siguientes componentes principales:

Categoría	Descripción	Tipos de objeto
Objetos de modelo de datos	Permiten utilizar datos guardados en tablas	Database, Query, Table, TCursor
Objetos de datos del sistema	Permiten guardar datos, excepto en tablas	DDE, FileSystem, Library, Session, System, TextStream
Tipos de datos	Los tipos de objetos básicos de ObjectPAL	AnyType, Array, Binary,Currency, Date, DateTime,DynArray, Graphic, Logical, LongInt, Memo, Number, OLE,Point, Record,SmallInt, String, Time
Objetos de diseño	Permiten crear el interfaz de usuario para la aplicación	Menu, PopupMenu, UIObject
Gestores de visualización	Permiten controlar la presentación de los datos ante el usuario	Application, Form, Report, TableView
Sucesos	Contienen información sobre las acciones en Paradox	ActionEvent, ErrorEvent,Event, KeyEvent, MenuEvent, MouseEvent, MoveEvent,StatusEvent, ValueEvent

Mediante Paradox y ObjectPAL, pueden contruirse aplicaciones que se incrementen, es decir, pueden construirse algunas tablas, crear algunos objetos y escribir algunos métodos, posteriormente comprobarlos y mejorarlos, después añadir más tablas, objetos y métodos y así sucesivamente hasta finalizar la aplicación.

Objetos

En Paradox, todo es un objeto, desde los botones y campos que se crean mediante las herramientas de la barra rápida, hasta las tablas y archivos de texto guardados en disco o los menús y menús emergentes que se crean con código. Paradox reconoce dos tipos de objetos:

- objetos de diseño — objetos que se incluyen en una ficha, como un botón
- objetos de datos — archivos, tipos de datos y estructuras de programación

Todos los objetos de Paradox contienen

Propiedades como color, fuente o grosor de línea

Métodos código que define la reacción de un objeto ante un suceso

Es posible modificar ambas características. Cualquier objeto que se crea o modifica a través de un uso interactivo de Paradox, puede crearse o modificarse mediante ObjectPAL.

Objetos y métodos

La creación de aplicaciones de Paradox es principalmente un proceso de integración de objetos en fichas y de escritura de métodos de ObjectPAL para definir la respuesta de estos objetos ante sucesos. Este tipo de aplicaciones se denomina "Eh tú, haz esto". La parte "Eh tu" (llamada suceso) se produce cuando el usuario hace algo con un objeto (por ejemplo, señala un botón en una ficha y hace clic). La parte "haz esto" se define mediante métodos, es decir, código que se ejecuta cuando el objeto gestiona un suceso. Algunos objetos pueden provocar la visualización de otros (como una lista de referencia) o estar encadenados a otro paso en la aplicación (por ejemplo, otra ficha, consulta o informe).

Referencia a tipos de objetos

Los objetos de ObjectPAL están agrupados por tipo. La ventana de ayuda Referencia a tipos de ObjectPAL agrupa estos tipos y proporciona acceso a la ayuda sobre los métodos de cada uno de los tipos.

En cada método encontrará la sintaxis, la descripción y el código de ejemplo, que puede copiar y pegar a través del Portapapeles en el código que está creando.

Vea también

Referencia a tipos de ObjectPAL

Sucesos

A continuación se incluyen algunos ejemplos de sucesos:

- pulsación del botón del ratón
- liberación del botón del ratón
- desplazamiento del puntero del ratón sobre un objeto
- pulsación de una tecla
- desplazamiento del cursor a un campo
- desplazamiento del cursor fuera del campo
- selección de un elemento en un menú

Los sucesos pueden producirse además por otras razones. Por ejemplo, el suceso timer se produce cuando transcurre un tiempo determinado. También pueden generarse sucesos desde los métodos que se crean.

Mediante ObjectPAL pueden crearse métodos que definen la reacción de los objetos ante los sucesos. Todos los objetos tienen métodos por defecto para los sucesos de ObjectPAL. No es, por tanto, necesario escribir métodos para todos los sucesos que puede manejar un objeto; además, un suceso nunca deja de reconocerse.

Propiedades

Los objetos tienen propiedades como el color, la trama, la fuente y el grosor de línea.

Para definir y modificar estas propiedades puede utilizar Paradox u ObjectPAL. Todo lo que puede realizarse en Paradox, puede también realizarse en ObjectPAL.

Por ejemplo, las sentencias siguientes definen el color del rectángulo cuadro1 como rojo (red), la fuente del campo campo1 como Times y el objeto micírc como invisible (asegúrese de que ve los códigos de ejemplo a pantalla completa).

```
cuadro1.color = "Red"    ; define el color de cuadro1 como rojo
campo1.font = "Times" ; define la fuente de campo1 como times
micírc.visible = No
```

Componentes del lenguaje de ObjectPAL

El lenguaje de ObjectPAL cuenta con tres componentes:

Métodos	Código anexado a un objeto que define su comportamiento
Procedimientos	Métodos encerrados entre los comandos PROC y ENDPROC
Elementos básicos del lenguaje	Los elementos de estructura básicos de ObjectPAL

Procedimientos

En ObjectPAL existen dos tipos de procedimientos, los procedimientos de la biblioteca run-time (RTL) de ObjectPAL y los procedimientos personalizados creados por el usuario. Los primeros son similares a los métodos, excepto en que nunca especifican de forma explícita un objeto. Los segundos son semejantes a otros lenguajes de programación, rutinas que escribe el usuario y que se utilizan como una subrutina.

Vea también

Procedimientos RTL
Procedimientos personalizados

Procedimientos RTL

Los procedimientos de la biblioteca run-time de ObjectPAL son similares a los métodos de ObjectPAL, con una diferencia: los procedimientos nunca especifican un objeto. Cualquier método en cualquier objeto puede activar cualquier procedimiento de ObjectPAL y éste sabrá su función. Por ejemplo, la sentencia

```
close()
```

activa el procedimiento tipo Form **quit**, que cierra la ficha actual. El tipo System incluye un número de procedimientos para interactuar con los usuarios, como message y msgInfo, msgStop.

```
msgStop("¡Alerta!", "El archivo ya existe.")
```

El tipo System incluye asimismo los procedimientos **beep** y **sleep** además de varios procedimientos de eumeración para obtener y definir la forma y posición del ratón.

```
method pushButton(var eventInfo Event)
beep()              ; ejecuta la señal sonora del sistema
sleep(2000)         ; espera 2 segundos
beep()
message("¿Ha oido la señal sonora dos veces?")
                    ; muestra un mensaje en la línea de estado
sleep(2000)
enumAllObjectSource("miFuente.db")
                    ; crea una tabla con todos los métodos de esta ficha
endMethod
```

Al igual que los métodos de ObjectPAL, sus procedimientos están relacionados con tipos de objetos y se ejecutan como respuesta a sucesos. Puede resultar de ayuda considerar los procedimientos de ObjectPAL como métodos con el objeto implícito.

Vea también

Procedimientos personalizados

Procedimientos personalizados

Los procedimientos personalizados de ObjectPAL son similares a los que pueden crearse con cualquier otro tipo de lenguaje de programación; son rutinas que escribe el usuario y que utiliza como subrutinas.

Este tipo de procedimiento puede estar anexado al propio objeto, a cualquier objeto presente en la jerarquía de contenedores o a la propia ficha.

Los procedimientos personalizados pueden incluirse en bibliotecas, pero sólo pueden ejecutarse desde la biblioteca.

Nota: ObjectPAL puede activar un procedimiento personalizado con más rapidez que un método personalizado.

La estructura es:

```
PROC nombre (DescripciónParámetros) [TipoDevuelto]
   [CONST sección]
   [TYPE sección]
   [VAR sección]
   [ObjectPAL sentencias]
ENDPROC
```

El bloque PROC...ENDPROC es un elemento básico del lenguaje, que se trata con más profundidad en la *Guía de referencia ObjectPAL.*

Los procedimientos pueden declararse en dos posiciones:

- desde un método
- en una ventana Proc de un objeto

Vea también

Procedimientos declarados en métodos
Procedimientos declarados en una ventana Proc de un objeto
Procedimientos RTL
Elementos básicos del lenguaje
Control del ámbito de una biblioteca
Contenedores
Bibliotecas

Procedimientos declarados en métodos

Los procedimientos que se declaran en métodos son privados: su ámbito se limita al método en el que se definen.

A continuación se incluye un ejemplo de procedimiento personalizado:

```
proc inc (x SmallInt) SmallInt
   return x+1 ; suma uno al número
endProc
```

El ejemplo siguiente muestra cómo activar este procedimiento (además de otro) desde el método. En este ejemplo se trata del método **pushButton**, pero podría tratarse de cualquier otro.

```
proc inc (x SmallInt) SmallInt
   return x+1
endProc

proc showMe (x SmallInt)
   msgInfo("miNúm <M>=<D> ", x)
endProc

method pushButton (var eventInfo Event)
var
   miNúm SmallInt
endVar
   miNúm <M>=<D> 3
   showMe(miNúm)
   miNúm <M>=<D> inc(miNúm)
   showMe(miNúm)
endMethod
```

Vea también

Procedimientos declarados en una ventana Proc de un objeto
Procedimientos personalizados
Ambito

Procedimientos declarados en una ventana Proc de un objeto

Los procedimientos de este tipo utilizan la misma sintaxis que los declarados en un método, pero actúan sobre un ámbito diferente.

Un procedimiento declarado en una ventana Proc de un objeto resulta visible para todos los métodos anexados al objeto y para todos los métodos de los objetos contenidos por el primero. Así pues, para que un procedimiento esté disponible para todos los objetos de una ficha, se declara en la ventana Proc de la ficha.

Para más información, consulte la *Guía de referencia ObjectPAL.*

Vea también

Procedimientos declarados en métodos
Procedimientos personalizados
Lenguaje | Arbol de objetos
Anexación de métodos a una ficha
Contenedores

Métodos

El método es código que define el comportamiento de un objeto, es decir, la respuesta de un objeto a sucesos. Los métodos de ObjectPAL se incluyen en tres categorías:

- Métodos estándar incluidos con todos los objetos de Paradox
- Métodos de la biblioteca run-time de ObjectPAL
- Métodos personalizados creados por el usuario

Vea también

Lista alfabética de métodos
Edición de métodos
El cuadro de diálogo Métodos
Métodos estándar
Métodos de la biblioteca run-time
Métodos personalizados
Métodos en otros objetos
Ambito

Edición de métodos

Para editar el método de un objeto, haga clic con el botón derecho sobre el objeto y seleccione Métodos en el menú de propiedades. Haga clic dos veces sobre un método en el cuadro de diálogo Métodos para seleccionarlo; el código del método aparecerá en la ventana del Editor.

Para editar el método de una ficha, seleccione la ficha y elija Propiedad | Ficha | Métodos. Aparece el cuadro de diálogo Métodos.

El texto de un método puede teclearse directamente en el Editor; pero también puede utilizarse el Portapapeles para copiar, cortar y pegar métodos o partes de métodos de otros objetos. Sin embargo, no se creará ninguna relación entre el método original y la copia; los cambios que se realicen en uno no se reproducirán en el otro.

También puede copiarse un método mediante la copia de un objeto desde un documento de diseño. Cuando se copia un objeto, se copian todos los métodos anexados a él. Sin embargo, tampoco así se crea ninguna relación después de la copia.

Vea también

El cuadro de diálogo Métodos

El cuadro de diálogo Métodos

Este cuadro de diálogo permite seleccionar un método en las listas de métodos estándar o personalizados y definir un nuevo método personalizado.

Métodos estándar Seleccione el método estándar que desea editar y haga clic sobre Aceptar; aparece una ventana del Editor.

Métodos personalizados Elija el método personalizado que desea editar.

Método personalizado nuevo Para crear un nuevo método personalizado introduzca su nombre en el cuadro de texto y haga clic sobre Aceptar. Aparece una ventana del Editor donde puede introducir el método.

El cuadro de diálogo Métodos también incluye las opciones siguientes:

Utilice	Para declarar
Var	variables
Const	constantes
Type	tipos
Procs	procedimientos
Uses	procedimientos que utilizan los métodos del objeto

Cada uno de estos elementos presenta su propia ventana del Editor, que se abre cuando se elige la opción. Cuando se declara una variable, constante o procedimiento en una de estas ventanas, resulta visible para todos los métodos anexados al objeto.

Para obtener más información sobre un método o procedimiento determinados, haga clic sobre el botón Buscar e introduzca el nombre del método o procedimiento en el cuadro de diálogo que aparece. También puede utilizar la Lista alfabética de métodos.

Para abrir el cuadro de diálogo Métodos, elija Lenguaje | Métodos.

Vea también

El Editor de ObjectPAL
El Depurador de ObjectPAL
Lista alfabética de métodos
Introducción a ObjectPAL
Macros
Bibliotecas

Métodos estándar

Todos los objetos de Paradox incluyen métodos estándar (como **open**, **close** y **mouseUp**) para los sucesos a los que pueden responder. Estos métodos determinan el comportamiento por defecto del objeto ante un suceso determinado. El Editor de ObjectPAL permite añadir código a los métodos estándar. Para editarlos, inspeccione el objeto y seleccione Métodos en el menú de propiedades. Seleccione uno o más métodos en el cuadro de diálogo que aparece y elija Aceptar para abrir una o más ventanas del Editor de ObjectPAL. Puede teclear el texto del método directamente en el Editor o utilizar el Portapapeles para copiar, cortar y pegar métodos o partes de métodos de otros objetos.

Los métodos estándar se describen en la *Guía de referencia ObjectPAL*.

Vea también

El cuadro de diálogo Métodos

Métodos en la biblioteca run-time

La biblioteca run-time de ObjectPAL (RTL) es una colección de rutinas predefinidas: incluye métodos que pueden utilizarse para realizar una gran variedad de tareas, desde la lectura y edición de datos en las tablas hasta la creación y visualización de menús. Cada uno de estos métodos está relacionado con un tipo de objeto; por ejemplo, todos los métodos que se utilizan con una ficha están en el tipo Form y los que se utilizan para trabajar con archivos de texto están en el tipo TextStream. Estos métodos, ordenados alfabéticamente por tipo, se incluyen en la *Guía de referencia ObjectPAL* así como en el apartado Referencia a tipos de ObjectPAL de ? (Ayuda).

Los métodos de ObjectPAL son simétricos y coherentes. Dentro de los tipos, los métodos normalmente aparecen en grupos de dos; por ejemplo, si un tipo incluye un método **open** puede esperarse que incluya también un método **close**. Si puede leer información de un objeto, también puede escribir en él; si puede obtener un valor, también puede definirlo.

Los métodos de ObjectPAL son coherentes entre los diferentes tipos, ya que métodos con nombres similares realizan acciones similares. Por ejemplo, **open** consigue que un objeto esté disponible para su manipulación, independientemente de que el objeto sea una tabla o un archivo de texto y **close** lo cierra. El código puede variar pero los resultados son los mismos.

Los métodos de la biblioteca run-time precisan que se utilice notación de puntos para especificar el objeto sobre el que van a actuar.

Vea también

Referencia a tipos de ObjectPAL
El cuadro de diálogo Métodos
Contenedores

Métodos personalizados

Los métodos personalizados son métodos complementarios creados por el usuario. Resultan convenientes para conseguir que rutinas de uso frecuente estén disponibles para varios objetos.

Los métodos personalizados anexados a una ficha están disponibles para todos los objetos de la ficha. De esta forma, sólo es necesario conservar el código en una posición.

Para crear un método personalizado, inspeccione un objeto, elija Métodos y, a continuación, el cuadro de texto Método personalizado nuevo. En el cuadro de edición introduzca un nombre para el método y elija Aceptar para acceder a una ventana del Editor de ObjectPAL. En ella puede teclear o pegar texto al igual que puede hacerlo en los métodos estándar.

Después de guardar un método personalizado, su nombre aparece en la lista del cuadro de diálogo Métodos. Para realizar modificaciones en él, elija su nombre y abra una ventana del Editor de ObjectPAL, de la misma forma que en el caso de un método estándar.

Es posible copiar, cortar y pegar un objeto completo. Si se realiza de esta forma, todos los métodos anexados a dicho objeto se copian con él. Sin embargo, no existe relación alguna entre el método original y la copia; los cambios que se realicen en uno de ellos, no afectarán al otro.

Vea también

El cuadro de diálogo Métodos
Contenedores

Métodos en otros objetos

Los métodos son públicos, es decir, los métodos anexados a un objeto puede ser activados por otro objeto. Por ejemplo, suponga que una ficha contiene dos cuadros, *cuadro1* y *cuadro2*. Si *cuadro1* tiene un método **pedrín**, *cuadro2* podría utilizar la notación de puntos para activarlo:

```
cuadro1.pedrín()
```

Si se anexa un método personalizado a una ficha, que constituye el nivel máximo en la jerarquía de contenedores, todos los objetos contenidos en la ficha tienen acceso directo a dicho método. Por ejemplo, si se anexa el método personalizado **goNextPage** a una ficha, un botón de la ficha podría llamar a **goNextPage** de la forma siguiente:

```
method pushButton (var eventInfo Event)
goNextPage() ; éste es un método personalizado anexo a la ficha
endMethod
```

En este ejemplo, no ha sido necesario utilizar notación de puntos ya que el método **pushButton** está anexado al botón y el botón, a su vez, está contenido por la ficha, por lo que tiene acceso directo a los métodos de la ficha.

Si se compila este método, ObjectPAL buscará otros objetos que utilicen **goNextPage**, de forma que no se produzcan retrasos en el momento de la ejecución.

Vea también

Ambito

Elementos básicos del lenguaje

Los elementos básicos del lenguaje son los elementos de estructura fundamentales de ObjectPAL. La mayor parte de ellos no está relacionada con ningún tipo de objeto en particular, ya que se utilizan con todos los tipos de objetos. Es posible utilizar estos elementos para asignar valores, activar funciones desde DLL o construir estructuras de control como bucles **if...then...else...endIf**, bucles **while...endWhile** y estructuras **switch...case...endSwitch**. También es posible declarar métodos, procedimientos, constantes, variables y tipos de datos.

Los elementos básicos del lenguaje son:

= (igual)	iif	switch
const	loop	try
disableDefault	method	type
doDefault	passEvent	uses
enableDefault	proc	var
for	quitLoop	while
forEach	return	
if	scan	

Estructura y sintaxis del lenguaje de métodos de ObjectPAL

Los métodos de ObjectPAL, en lo que concierne a su estructura y sintaxis, son similares a los de programas tradicionales. Algunos aspectos de su estructura son:

- Los métodos pueden tener parámetros (también llamados argumentos).
- Los métodos están perfilados por las palabras clave METHOD...ENDMETHOD. Es posible definir una estructura ordenada de ejecución, ya que ObjectPAL acepta estructuras y bucles de control como WHILE...ENDWHILE, IF...THEN...ELSE y SWITCH...CASE...ENDSWITCH.
- Al igual que en Pascal y C, es posible definir procedimientos para realizar una o más tareas. Los procedimientos pueden, además, recibir argumentos de y devolver resultados al método que los activa.
- También al igual que sucede en C, es posible utilizar espacios en blanco (tabuladores, espacios y líneas vacías). Pueden sangrarse las líneas de método subordinadas, incluir una o más sentencias en una línea y añadir un comentario en cualquier línea del método; el espacio en blanco no afecta a la ejecución de las sentencias.

Contenedores

Los objetos de Paradox coexisten en una jerarquía de contenedores. Cuando se incluye un objeto de tabla, por ejemplo, en la página de una ficha, la página **contiene** la tabla. Las fichas contienen tablas, las tablas contienen registros, los registros contienen campos, los campos contienen botones y así sucesivamente.

Los objetos sólo están contenidos si están incluidos por completo dentro de los márgenes del contenedor.

La posición en una jerarquía es importante, ya que define qué puede ver un objeto de otros: sus propiedades y variables.

Un objeto no puede ver variables de los objetos que contiene.

Un objeto puede ver sus propias variables, así como las de los objetos que lo contienen.

Para decirlo de otra forma, si se consideran los objetos como cajas que a su vez contienen cajas más pequeñas, la caja más pequeña es la que tiene la mejor vista.

Variables

Las variables son como ranuras en las que pueden guardarse de forma temporal elementos de información.

El valor de una variable puede pertenecer a cualquier tipo de ObjectPAL (también llamado tipo de datos). No es necesario indicar de forma explícita un tipo de dato para las variables.

El proceso de especificación de un tipo de dato en una variable antes de utilizarla se denomina declarar una variable.

La forma más sencilla de asignar un valor a una variable es utilizar el operador de asignación (=).

Vea también

Ambito

El ámbito de una variable

El término "ámbito" significa "accesiblidad". El ámbito de una variable, es decir, el rango de objetos que tienen acceso a ella, viene determinado por el objeto en que se declara y por la jerarquía de contenedores. Los objetos sólo pueden acceder a sus propias variables y a las variables definidas en los objetos que los contienen. Además, el ámbito de una variable depende de la posición donde se declara:

Dentro de un método

Las variables así declaradas sólo son visibles en este método y sólo se puede acceder a ellas mientras el método se ejecuta. Se inicializan (restauran) cada vez que se ejecuta el método.

Fuera de un método

Las variables de este tipo declaradas antes de la palabra **method** sólo están visibles en este método, pero no se inicializan cada vez que se ejecuta.

En la ventana Var

Las variables declaradas en una ventana Var del objeto son visibles a todos los métodos anexados a dicho objeto y a todos los objetos *que* contiene el objeto. Una variable de este tipo está anexada al objeto y permanece accesible en tanto el objeto existe en la ficha y la ficha está abierta.

Desde la jerarquía de contenedores (asociación en el momento de la compilación)

En términos de programación, la "asociación" de una variable es el proceso de conexión de una variable a un tipo de datos. El compilador de ObjectPAL asocia las variables cuando compila el código fuente, ya que no se produce ninguna asociación en el momento de la ejecución. Si el compilador encuentra una variable en una sentencia, busca en el resto del código fuente el lugar donde se ha declarado la variable para así poder asociarla al tipo de datos declarado.

Constantes

Las constantes son similares a las variables, excepto en que están protegidas de forma que no se pueden modificar cuando se ejecuta el programa; así el compilador puede generar un código más eficaz.

Es posible definir constantes para un sólo método o abrir una ventana Const para definir constantes para todos los métodos del objeto.

Las constantes se incluyen de forma automática en los recursos, donde pueden modificarse sin afectar al código fuente.

El lenguaje de ObjectPAL incluye varias constantes predefinidas.

Para obtener una lista de constantes predefinidas, consulte la lista Tipos de constantes.

ObjectPAL IDE

Cuando se trabaja en el Entorno integrado de desarrollo (IDE) de ObjectPAL, se utiliza el Editor o el Depurador.

En el Editor pueden modificarse:

- Métodos (estándar o personalizados)
- Procedimientos
- Usos
- Tipos
- Constantes
- Variables

En el Depurador pueden depurarse métodos o procedimientos.

El código que se edita o depura puede estar anexado a una macro, a una biblioteca, a una ficha o a un objeto en una ficha.

Vea también

El Editor
El Depurador
Macros
Bibliotecas

El Editor de ObjectPAL

El Editor de ObjectPAL proporciona las funciones de un editor de texto de Windows, además de funciones especiales para editar los métodos de ObjectPAL. Además, el Editor utiliza el Depurador para proporcionar un entorno integrado en el que se pueden crear, comprobar y modificar métodos.

El Editor de ObjectPAL funciona de la misma forma si se utiliza con un objeto, una ficha, una biblioteca o una macro.

Vea también

Inicio del Editor
Uso del Editor de ObjectPAL
Barra rápida del Editor de ObjectPAL
Salida del Editor
Menú del Editor de ObjectPAL
El Depurador de ObjectPAL

Inicio del Editor

Para iniciar el Editor

en una ficha o en un objeto de una ficha:

1. Haga clic con el botón derecho sobre un objeto para acceder a su menú Propiedades.
2. Elija Métodos en la lista. El cuadro de diálogo Métodos presenta una lista con los métodos que pueden editarse. Desplácese por la lista para ver todas las entradas.
3. Cuando elige un método, se abre una ventana del Editor con texto por defecto.

en una biblioteca:

1. Abra la ventana Biblioteca y haga clic con el botón derecho sobre ella.
2. Cuando se abra el cuadro de diálogo Métodos, elija el método que desea editar.

en una macro:

1. Cuando abre una macro, ésta se presenta de forma automática en una ventana del Editor.
2. La primera línea asigna un nombre al método y la última lo finaliza. El cursor se sitúa en la tercera línea, donde puede comenzar a escribir el método.

El Editor siempre se encuentra en modo Insertar y dispone de los métodos de edición habituales.

Es posible tener varias ventanas del Editor abiertas a la vez.

Nota: Las variables, las constantes y los procedimientos que se declaran en una ventana de método sólo están visibles en dicho método. Para que estos elementos estén visibles para todos los métodos de un objeto, active Uses, Type, Const , Var y Procs; tantos como desee en el cuadro de diálogo Métodos. Paradox abre una ventana diferente para cada cuadro que se activa.

Vea también

El cuadro de diálogo Métodos
Uso del Editor de ObjectPAL
Menú del Editor de ObjectPAL
Salida del Editor
El Depurador de ObjectPAL

Uso del Editor de ObjectPAL

La ventana del Editor dispone de los métodos habituales de edición, con dos excepciones:

- El Editor siempre se encuentra en modo insertar.
- El Editor no realiza automáticamente saltos de línea. Las líneas continúan hacia la derecha mientras se teclea hasta que se pulsa Intro para comenzar una línea nueva.

Uso del teclado:

Ctrl+flecha izquierda	desplaza el cursor una palabra hacia la izquierda.
Ctrl+flecha derecha	desplaza el cursor una palabra hacia la derecha.
Inicio	desplaza el cursor al comienzo de una línea.
Fin	desplaza el cursor al final de una línea.
Ctrl+Inicio	desplaza el cursor al comienzo del texto.
Ctrl+Fin	desplaza el cursor al final del texto.
Re Pág	se desplaza una ventana hacia atrás.
Av Pág	se desplaza una ventana hacia delante.
Retroceso	borra el carácter situado a la izquierda del cursor.
Supr	borra el carácter situado a la derecha del cursor.
Insert	no tiene ningún efecto ya que el Editor se encuentra siempre en modo Insertar. A medida que se teclea, los caracteres se desplazan a la derecha. No es posible sobrescribir caracteres.
Ctrl+Insert	copia el texto seleccionado en el portapapeles.
Mayús+Insert	pega el texto del portapapeles en el método.
Tab	inserta un carácter de tabulador y desplaza el texto hacia la derecha.

Selección de texto:

Para seleccionar una palabra, haga doble clic sobre ella.

Para seleccionar una línea completa, haga clic hacia la izquierda de la línea (el ratón adopta la forma de flecha).

Para seleccionar un bloque de texto:

- haga clic y arrastre con el ratón
- pulse **Mayús** y utilice las teclas de flecha
- haga clic para indicar la posición donde comienza la selección y pulse **Mayús** para ampliarla

Vea también

Menú del Editor de ObjectPAL
Barra rápida del Editor de ObjectPAL
Salida del Editor

Barra rápida del Editor de ObjectPAL

Haga clic sobre los botones de la barra rápida para realizar las acciones de forma más rápida en la ventana del Editor de ObjectPAL. Estos son los botones disponibles:

Cortar

Copiar

Pegar

Ejecutar

Imprimir

Verificar sintaxis

Establecer puntos de ruptura

Métodos

Guardar y salir

Arbol de objetos

Abrir carpeta

Para más información sobre cada botón, haga clic sobre su imagen en esta pantalla.

Vea también

El Editor de ObjectPAL

Botón Cortar

El botón Cortar de la barra rápida equivale a Edición | Cortar en el menú. El botón Cortar elimina el texto u objetos seleccionados y los sitúa en el Portapapeles.

A continuación puede utilizarse el botón Pegar o elegir Edición | Pegar para pegar el contenido del Portapapeles en otro archivo o en otra posición del mismo archivo.

El contenido del Portapapeles no se borra cuando se pega, por lo que puede pegarse tantas veces como sea necesario.

Para eliminar una selección sin que afecte al contenido del Portapapeles, pulse Supr o elija Edición | Eliminar.

Método abreviado Mayús+Supr

Botón Copiar

El botón Copiar de la barra rápida equivale a Edición | Copiar en el menú.

Elija el botón Copiar para copiar texto u objetos seleccionados en el Portapapeles sin eliminarlos del documento o la consulta.

Para pegar el contenido del Portapapeles en el documento puede utilizar:

- Edición | Pegar
- Mayús+Ins
- Botón Pegar

El contenido del Portapapeles no se elimina cuando se pega, por lo que puede realizar tantas operaciones de pegado como desee.

Método abreviado Ctrl+Ins

Botón Pegar

El botón Pegar de la barra rápida equivale a Edición | Pegar en el menú.

El efecto del botón Pegar depende de qué ventana está activa y de si se diseña o se visualizan los datos.

El contenido del Portapapeles no se elimina cuando se pega, por lo que pueden realizarse tantas operaciones de pegado como sea necesario.

Método abreviado Mayús+Ins

Botón Ejecutar

Haga clic sobre este botón para ejecutar la ficha o macro que está editando. Paradox guarda los métodos anexados, compila el código y pasa a una ventana de visualización.

Nota: Si ha definido puntos de ruptura, Paradox finaliza la ejecución en el primero que encuentra y abre una ventana del Depurador.

Nota: El botón Ejecutar no está disponible en una ventana Biblioteca. Sus métodos y procedimientos se activan a través de una macro o una ficha.

Botón Imprimir

Haga clic sobre este botón para imprimir el código que está editando.

Si está en	Paradox imprime
una macro	todo el código de la macro
una biblioteca sin ninguna ventana del Editor con método abierta	todo el código de la biblioteca
una biblioteca con una ventana del Editor con método abierta	el código del método abierto
una ficha con una ventana del Editor con método abierta	el código del método abierto

Botón Verificar sintaxis

Haga clic sobre este botón para compilar todos los métodos de la ficha, macro, o biblioteca (no sólo la ventana del Editor actual). Si se encuentra un error de sintaxis, aparece una ventana con el método correspondiente y el cursor situado cerca del error; además la línea de estado muestra un mensaje de error.

Nota: Si modifica el código en más de una ventana del Editor, guarde los cambios antes de comprobar la sintaxis. En caso contrario, el verificador puede informar sobre errores inesperados ya que no comprueba el último código.

Botón Establecer puntos de ruptura

La ejecución finalizará en el primer punto de ruptura que se encuentre. Es posible definir tantos puntos de ruptura como permita la memoria del sistema.

Cuando se hace clic sobre este botón, aparece el cuadro de diálogo Establecer puntos de ruptura.

El botón equivale a Depurar | Establecer puntos de ruptura.

Nota: Es posible establecer puntos de ruptura en una biblioteca, pero no ejecutar ésta de forma independiente; es necesario activar el método desde una ficha o una macro.

Botón Métodos

Haga clic sobre este botón para acceder al cuadro de diálogo Métodos, donde puede seleccionar un método en las listas de métodos estándar o personalizados. También es posible definir un método personalizado nuevo o abrir una ventana del Editor para declarar usos, tipos, constantes, variables o procedimientos.

Equivale a Lenguaje | Métodos.

Botón Guardar y salir

Haga clic sobre este botón para guardar el código y cerrar la ventana del Editor. Permanecerá en la ventana de diseño.

Botón Arbol de objetos de ObjectPAL

Haga clic sobre este botón de la barra rápida para ver el Arbol de objetos. Paradox muestra este diagrama en una ventana independiente.

El Arbol de objetos de Paradox muestra un diagrama esquemático de los objetos en la ficha y las relaciones entre ellos. En el diagrama se observa la jerarquía del objeto, con el objeto actual en el extremo izquierdo y el árbol, que muestra la jerarquía de contenedores, extendiéndose hacia la derecha.

Nota: El botón Arbol de objetos no está disponible en una ventana Biblioteca.

Nota: Las macros no pueden contener otros objetos, por lo que el árbol de objetos de una macro sólo muestra la macro.

Botón Abrir carpeta

Haga clic sobre este botón para abrir una carpeta para el directorio de trabajo. Equivale a Archivo | Abrir | Carpeta.

La carpeta es una ventana que muestra los objetos seleccionados en el directorio de trabajo. Los iconos representan los objetos del directorio.

En la ventana Carpeta puede hacer clic con el botón derecho sobre los iconos para inspeccionar objetos, o hacer doble clic para realizar la acción por defecto (la primera opción del menú).

Salida del Editor de ObjectPAL

La salida del Editor depende de la acción que se vaya a realizar a continuación.

Para continuar con el diseño de fichas, elija Cerrar en el menú de control de la ventana o haga doble clic sobre el cuadro del menú de control. A continuación, puede utilizar Paradox de forma interactiva para seguir con el diseño de la ficha. Para ver el resultado de la ficha, elija Ficha | Ver datos.

Para ejecutar los métodos al salir del Editor, señale la barra de menús, haga clic sobre Conservar y después sobre Ejecutar para que Paradox pase a modo ver datos.

El Depurador de ObjectPAL

El Depurador de ObjectPAL permite comprobar y seguir de forma interactiva la ejecución de los comandos en los métodos.

Mediante el Depurador se puede:

- establecer puntos de ruptura, que permiten ejecutar instrucciones hasta un determinado punto, parar y comprobar el resultado.
- inspeccionar variables para asegurarse de que los valores se tratan de la forma esperada.
- ejecutar un método línea a línea (llamado paso único).
- excluir llamadas a procedimientos y funciones que sabe que carecen de errores.

Para iniciar el Depurador

1. Muestre un método en una ventana del Editor.

2. Elija Depurar para visualizar un menú desplegable con las funciones del Depurador.

Para salir del Depurador

Para abandonar el Depurador de ObjectPAL puede:

- Elegir Depurar | Ejecutar. Si ya no hay más puntos de ruptura establecidos, se cerrará la ventana de depuración y se pasará al modo ver datos.
- Hacer clic sobre la ficha de la que se procede para volver a ella.
- Elegir Cerrar en el menú de control de la ventana Depurar o hacer doble clic sobre el cuadro del menú de control para cerrar la ventana.

Vea también

El menú Depurar
Barra rápida del Depurador de ObjectPAL

Barra rápida del Depurador de ObjectPAL

Haga clic sobre los botones de la barra rápida para ejecutar comandos rápidamente en la ventana del Depurador de ObjectPAL. A continuación se incluyen los botones disponibles:

Ejecutar

Imprimir

Establecer puntos de ruptura

Métodos

Incluir llamadas

Excluir llamadas

Inspeccionar

Arbol de objetos

Abrir carpeta

Para más información sobre cada uno de los botones, haga clic sobre su imagen en esta pantalla.

Vea también

El Depurador de ObjectPAL

Botón Ejecutar

Haga clic sobre este botón para ejecutar el Depurador de ObjectPAL. Paradox guarda los métodos anexados, compila el código y pasa a una ventana de visualización. Si encuentra un punto de ruptura, finaliza la ejecución y abre una ventana del Depurador.

Si ya está en una ventana del Depurador, la ejecución continúa a partir del punto de ruptura.

Nota: Para ejecutar el Depurador es necesario establecer puntos de ruptura en el código.

Nota: El botón Ejecutar no está disponible en una ventana Biblioteca.

Botón Imprimir

Haga clic sobre este botón para imprimir el código que está depurando.

Si está en	Paradox imprime
una macro	todo el código de la macro
una biblioteca sin ninguna ventana del Editor con método abierta	todo el código de la biblioteca
una biblioteca con una ventana del Editor con método abierta	el código del método abierto
una ficha con una ventana del Editor con método abierta	el código del método abierto

Botón Establecer puntos de ruptura

Haga clic sobre este botón para establecer puntos de ruptura en un método y detener así la ejecución en líneas determinadas. Pueden definirse tantos puntos de ruptura como permita la memoria del sistema.

Cuando se hace clic sobre este botón aparece el cuadro de diálogo Establecer puntos de ruptura.

Este botón equivale a Depurar | Establecer puntos de ruptura.

Botón Métodos

Haga clic sobre este botón para abrir el cuadro de diálogo Métodos, donde puede seleccionar un método en listas de métodos estándar o personalizados; además puede definir un nuevo método personalizado o abrir una ventana del Editor para declarar usos, tipos, constantes, variables o procedimientos.

Este botón equivale a Lenguaje | Métodos.

Botón Excluir llamadas

Haga clic sobre este botón para realizar un paso único en un método y considerar los procedimientos y métodos personalizados como pasos únicos. Esta función sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Este botón equivale a Depurar | Excluir llamadas.

Botón Incluir llamadas

Haga clic sobre este botón para realizar un paso único en cada línea de un método y en cada línea de los procedimientos y métodos personalizados que activa el método. Esta función sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Este botón equivale a Depurar | Incluir llamadas.

Botón Inspector

Haga clic sobre este botón para visualizar y modificar (optativo) el valor de una variable. Esta función sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Este botón equivale a Depurar | Inspector.

Botón Arbol de objetos de ObjectPAL

Haga clic sobre este botón de la barra rápida para ver el Arbol de objetos. Paradox muestra este diagrama en una ventana independiente.

El Arbol de objetos de Paradox muestra un diagrama esquemático de los objetos en la ficha y las relaciones entre ellos. En el diagrama se observa la jerarquía del objeto, con el objeto actual en el extremo izquierdo y el árbol, que muestra la jerarquía de contenedores, extendiéndose hacia la derecha.

Nota: El botón Arbol de objetos no está disponible en una ventana Biblioteca.

Nota: Las macros no pueden contener otros objetos, por lo que el árbol de objetos de una macro sólo muestra la macro.

Botón Abrir carpeta

Haga clic sobre este botón para abrir una carpeta para el directorio de trabajo. Equivale a Archivo | Abrir | Carpeta.

La carpeta es una ventana que muestra los objetos seleccionados en el directorio de trabajo. Los iconos representan los objetos del directorio.

En la ventana Carpeta puede hacer clic con el botón derecho sobre los iconos para inspeccionar objetos, o hacer doble clic para realizar la acción por defecto (la primera opción del menú).

Envío de aplicaciones

Cuando Paradox envía una ficha, una macro o una biblioteca, elimina el código fuente de ObjectPAL, pero conserva el código compilado con el archivo. Esto permite que otros usuarios utilicen la aplicación sin que puedan ver o modificar el código.

La extensión en los nombres de archivo de aplicaciones enviadas se modifica como se muestra a continuación:

aplicación	extensión sin enviar	extensión enviada
ficha	.FSL	.FDL
macro	.SSL	.SDL
biblioteca	.LSL	.LDL

Nota: Una ficha enviada también está protegida para evitar modificaciones en el diseño. No puede abrirse en una ventana de diseño. Cuando envíe una ficha, no olvide enviar copias de todas las tablas en su modelo de datos, así como cualquier archivo de índices e integridad referencial.

Elija Lenguaje | Enviar para enviar una ficha, macro o biblioteca.

Uso de las tablas

Si desea utilizar ObjectPAL para manipular tablas, debería conocer los tipos Table, TCursor, TableFrame y TableView. Esta tabla resume las diferencias:

Tipo	Descripción
Table	Datos tabulares guardados en un archivo
TCursor	Un puntero a una tabla, guardada y manipulada en memoria
TableView	Una tabla que aparece en su propia ventana.

Biblioteca: recopilación de código personalizado

Una biblioteca es un archivo que almacena métodos personalizados, y procedimiéntos personalizados, variables constantes y tipos de datos definidos por el usuario. Puede usar las bibliotecas para almacenar y mantener rutinas de uso frecuente y compartir código entre varias fichas.

Importante: Las fichas no tienen acceso directo a las variables declaradas en las bibliotecas. Es necesario escribir métodos o procedimientos personalizados para definir y recuperar los valores.

El trabajo con bibliotecas es semejante en muchos aspectos al trabajo con fichas. Por ejemplo, para crear una ficha se selecciona Archivo|Nuevo|Ficha y para crear una biblioteca, Archivo|Nuevo|Biblioteca. Una biblioteca, al igual que una ficha, tiene incorporados métodos estándar. La anexión de código a una biblioteca se realiza igual que la anexión a una ficha, mediante el cuadro de diálogo de métodos y el Editor de ObjectPAL. Como ocurre con una ficha, pueden abrirse ventanas del Editor para declarar tipos de datos, constantes, variables, métodos y procedimientos personalizados, así como rutinas externas.

Pero también existen diferencias importantes:

- En tiempo de ejecución las bibliotecas no aparecen en ventanas.
- Las bibliotecas no pueden contener objetos de diseño; sólo contienen código.
- En los métodos de una biblioteca, las sentencias que utilizan *Self* no se refieren a la biblioteca, sino al objeto que llamó al método. De igual manera, las sentencias que usan *Container* se refieren al contenedor del objeto llamado por el método de la biblioteca. Este principio también afecta a las demás variables estándar de objeto: *active*, *lastMouseClicked*, *lastMouseRightClicked* y *subject*.
- Las normas de ámbito son diferentes para las bibliotecas.

Vea también

Taréas de la ventana de la biblioteca
Comandos del Editor ObjectPAL
Control del ámbito de una biblioteca

Taréas de la ventana de la biblioteca

Creación de una biblioteca

Para crear una nueva biblioteca, seleccione Archivo|Nuevo|Biblioteca. Inspeccione la ventana de diseño de biblioteca para abrir el cuadro de diálogo Métodos.

Cuando haya terminado de escribir código:

- seleccione Archivo|Guardar para almacenar la biblioteca (tanto el código fuente como el código ejecutable) en el disco.
- seleccione Lenguaje|Enviar desde una ventana del Editor si desea guardar sólo el código ejecutable.

Vea también

Adición de código a la biblioteca
Llamada a los métodos de una biblioteca
Almacenamiento de una biblioteca

Adición de código a la biblioteca

Para añadir código a una biblioteca, inspeccione la ventana de diseño de bilbioteca para abrir el cuadro de diálogo Métodos. Después añada el código como se se tratase de cualquier otro objeto. Puede realizar esta operación con una biblioteca nueva o con una ya existente.

Cuando haya terminado de escribir código:

- seleccione Archivo|Guardar para almacenar la biblioteca (tanto el código fuente como el código ejecutable) en el disco.
- seleccione Lenguaje|Enviar desde una ventana del Editor si desea guardar sólo el código ejecutable.

Puede añadir código a una biblioteca a través del cuadro de diálogo de métodos y de las ventanas del Editor de ObjectPAL, de las siguientes maneras:

- Anexar código a los methods.estándar.
- Añadir métodos personalizados.
- Añadir procedimientos personalizados.
- Declarar variables, constantes, tipos de datos y rutinas externas.

Vea también

Anexión de código a los métodos estándar
Anexión de código a los métodos personalizados
Adición de procedimientos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas

Anexión de código a los métodos estándar

Todas las bibliotecas tienen los métodos estándar **open**, **close** y **error**. En la ventana del Editor, puede anexar código a estos métodos estándar de la misma forma que lo haría con qualqier otro objeto.

Cuando se abre por primera vez una biblioteca, se produce una llamada al método estándar **open** de la misma; cuando la biblioteca se cierra, se llama al método estándar **close**, y cuando el código de la biblioteca genera un error, se llama al método **error**. Generalmente se usa **open** para inicializar las variables, globales de una biblioteca y **close** para "dejar limpia" la biblioteca después de usarla. Por defecto, el método **error** de una biblioteca llama al método **error** de la ficha llamada por la rutina de biblioteca.

Para información sobre la anexión de código a los métodos estándar y el uso de ObjectPAL, consulte la *Guía del Programador de ObjectPAL.*

Adición de código a la biblioteca
Anexión de código a los métodos personalizados
Adición de procedimientos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas

Adición de métodos personalizados

Los métodos personalizados de una biblioteca pueden ser llamados por otros métodos de la misma biblioteca, por métodos de otras fichas y por métodos de objetos de otras fichas. Esta facilidad de acceso hace de las bibliotecas un instrumento muy útil.

Para visualizar el cuadro de díalogo Métodos, inspeccione la ventana de diseño de biblioteca o pulse Ctrl+Espaciador. Introduzca un nombre para el nuevo método personalizado como se si tratase de cualquier otro objeto y seleccione ACEPTAR para abrir otra ventana del Editor.

Consulte en la *Guía del Programador de ObjectPAL para* los detalles sobre la anexión de métodos personalizados y el uso del ObjectPAL.

Vea también

Adición de código a la biblioteca
Anexión de código a los métodos estándar
Anexión de código a los métodos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas

Adición de procedimientos personalizados

Desde el cuadro de diálogo de métodos de una biblioteca puede seleccionar Procs para abrir una ventana del Editor donde declarar procedimientos personalizados para la biblioteca.

Nota: A diferencia de los métodos personalizados, que pueden llamarse desde otras fichas y objetos, los procedimientos personalizados pueden llamarse sólo desde la biblioteca donde están declarados.

Consulte la *Guía del Programador de ObjectPAL para* los detalles sobre la adición de procedimientos personalizados y el uso de ObjectPAL.

Vea también

Adición de código a la biblioteca
Anexión de código a los métodos estándar
Anexión de código a los métodos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas

Declaración de variables, constantes, tipos de datos y rutinas externas

Desde el cuadro de diálogo de métodos de una biblioteca puede declarar variables, constantes, tipos de datos y rutinas externas. Seleccione Var, Const, Tipo o Usos, respectivamente, para abrir la ventana del Editor apropiada. Los elementos que se declaran en esas ventanas son globales para la biblioteca pero no puede accederse a ellos desde otras fichas u objetos. No obstante, las rutinas de biblioteca con acceso a esas variables pueden llamar a otros objetos o fichas.

Para más información sobre la declaración de variables, constantes, tipos de datos y rutinas externas, consulte la *Guía del Programador de ObjectPAL*.

Vea también

Adición de código a la biblioteca
Anexión de código a los métodos estándar
Anexión de código a los métodos personalizados
Adición de procedimientos personalizados

▪

Edición de una biblioteca

Puede editar una bibilioteca desde la ventana del Editor de ObjectPAL.

Para editar una biblioteca:

1. Abra la biblioteca deseada. Aparecerá una en vacía.
2. Inspeccione la ventana de biblioteca para abrir el cuadro de diálogo Métodos.
3. Seleccione el método a editar y pulse ACEPTAR.
Aparecerá una ventana del editor con el método seleccionado. Utilice el Editor ObjectPAL para modificar cualquier método.

Vea también
Editor ObjectPAL

Almacenamiento de una biblioteca

Utilice Lenguaje | Enviar para almacenar una biblioteca. Cuando realiza esta operación, Parados elimina todo el código fuente. No a perdido su libería, simplemente está protegida.

Si almacena la bilbioteca con Archivo | Guardar cualquier usuario puede modificar el código de la biblioteca y modificar así el programa. Con la opción Enviar se asegura de que ningún otro usuario pueda modificar su aplicación.

Cuando selecciona Lenguaje | Enviar, Paradox guarda una copia de la biblioteca con la extensión.LDL . Puede modificar el código en este archivo .LSL, pero si quiere estar seguro de la integridad de su aplicación entrege al resto de usuarios la biblioteca almacenada con Enviar.

Para más información consulte la *Guía del Programador ObjectPAL .*

Vea también

Métodos

Llamada a los métodos de una biblioteca

Para llamar a un método de una biblioteca, primero debe declarar el método en la ventana Usos del objeto que realiza la llamada. Por ejemplo, suponga que desea que el método **pushButton** del botón llame a un método personalizado de la biblioteca. Declare el método en la ventana Usos del botón (o en la ventana Usos del objeto que contenga el botón), para que Paradox sepa dónde debe buscar el método y qué argumentos tomar. Puede declarar varios métodos de biblioteca en la misma ventana Usos.

Nota: Puede escribir código que llame a un método de biblioteca, y el código se compilará en una ventana de diseño, incluso aunque no exista la biblioteca. Pero la biblioteca debe existir para que se ejecute la ficha.

Vea también

Métodos de Biblioteca

Métodos de Biblioteca

Puede usar los métodos runtime (tiempo de ejecución) del tipo Library en su código (anexados a cualquier objeto o a una biblioteca) para manipular una biblioteca. Los métodos runtime son:

open	Abre una biblioteca y la carga en la memoria del sistema,, con lo que queda disponible para una o más fichas y Escritorios Escritorios..
close	Borra la biblioteca de la memoria. No es necesario cerrar de forma explícita una biblioteca,, pues se suprime automáticamente al cerrar todas las fichas o bibliotecas de referencia,, pero el uso de este método facilita la lectura de código y el conocimiento de lo que supuestamente debe suceder.
enumSource	Escribe código de biblioteca en una tabla de Paradox.
enumSourceToFile	Escribe código de biblioteca en un archivo de texto.
execMethod	Ejecuta un método de biblioteca específico.

Vea también

Llamada a los métodos de una biblioteca

Control del ámbito de una biblioteca

El *ámbito* de una biblioteca expresa su accesibilidad; es decir, indica los objetos que pueden acceder al código de la biblioteca. Una variable Library sigue las mismas normas de ámbito que el resto de las variables de ObjectPAL.

El ámbito de una biblioteca depende de dónde se ha declarado la variable Libraryy cómo se ha abierto la biblioteca.

Vea también

Declaración de una variable Library
Apertura de una biblioteca

Declaración de una variable Library

Las variablesLibrary siguen las mismas normas de ámbito que el resto de las variables de ObjectPAL. Una variable declarada en un método está disponible mientras dura la ejecución de dicho método. Una variable declarada en la ventana Var de un objeto está disponible para todos los métodos anexos a ese objeto y para todos los objetos contenidos en él.

Por tanto, si desea que una biblioteca esté disponible para todos los objetos de una ficha mientras está ejecutándose, declare la variable Library en la ventana Var de la ficha y declare las rutinas de biblioteca en la ventana Usos de la ficha.

Vea también

Control del ámbito de una biblioteca
Apertura de una biblioteca

Apertura de una biblioteca

El método **open** del tipo Library toma argumentos que especifican el ámbito. Una biblioteca puede abrirse de una de estas maneras:

Global para el Escritorio	cada ficha del Escritorio (en una sesión de Paradox) puede acceder a la biblioteca. Esto permita que varias formas tengan acceso a los mismos metódos estándar y compartan las mismas variables. globales. Este es el ámbito por defecto. La apertura por defecto de las bibliotecas es global para el Escritorio. **Nota:** Para que dos o más fichas compartan la misma biblioteca, todas deben abrir la biblioteca global para el Escritorio y todas deben tener una ventana Usos que declare qué rutinas de biblioteca usar.
Privada de una ficha	sólo la ficha que abrió la biblioteca puede acceder a su código.

Vea también

Control del ámbito de una biblioteca
Declaración de una variable Library

Uso de las variables Library como argumentos

Es posible usar una variable as Library como argumento de un método o procedimiento personalizado anexo a un objeto de una ficha (o a la propia ficha).

Si se pasa una Library como argumento, puede modificarse el comportamiento de una rutina (método o procedimiento) sin perder la independencia de la misma. Una rutina puede usar una biblioteca y rutinas de una biblioteca, pero el que llama puede determinar la función de las rutinas simplemente cambiando la biblioteca.

Macros

Una macro es un objeto formado por código incluido en un archivo específico (no asociado a una ficha), que aparece en el Escritorio en forma de icono. Es posible:

- anexar uno o más métodos.
- declarar variables, constantes, tipos de datos y procedimientos.
- ejecutar DLLs personalizados.

Las macros no tienen tipo, no se visualizan dentro de ventanas ni contienen objetos de diseño, pero sí tienen los métodos estándar **run**, **error** y **status** (este último sólo disponible en el nivel avanzado de ObjectPAL). Puede ejecutar estos métodos con Paradox de forma interactiva o activarlos desde un método o procedimiento de ObjectPAL. Como todos los objetos, las macros también tienen ventanas donde se declaran variables, constantes, procedimientos, tipos de datos y rutinas externas, además de poder declarar métodos personalizados. Utilice una macro cuando desee ejecutar código sin abrir una ventana de ficha.

Desde una macro tiene acceso total a la biblioteca run-time de ObjectPAL, lo que permite controlar otros objetos. Por ejemplo, puede ejecutar otras macros, abrir y utilizar tablas, fichas e informes, así como ejecutar consultas. Asimismo, puede activar métodos anexos a otros objetos, y obtener y definir sus propiedades.

Para cambiar el nivel de ObjectPAL actual, elija Propiedad | Escritorio. Aparecerá el cuadro de diálogo Propiedades del Escritorio, donde puede seleccionar Principiante o Avanzado.

Vea también

Tareas de la ventana Macro
Menú del Editor de ObjectPAL

Tareas de la ventana Macro

Creación de macros

Para crear una macro, elija Archivo | Nuevo | Macro. Se abrirá una ventana del Editor de ObjectPAL con el método **run** de macros. Se trata de una ventana normal del Editor de ObjectPAL, donde puede escribir el código de la macro, además de editar, verificar la sintaxis y depurar el método **run,** como haría con cualquier objeto.

También como en cualquier objeto, desde una macro es posible abrir y cerrar fichas, crear objetos, ver y definir propiedades y valores, mostrar mensajes y activar métodos.

Para acceder al cuadro de diálogo Métodos, elija Lenguaje | Métodos. Desde este cuadro puede declarar las variables, constantes, tipos de datos, procedimientos, métodos personalizados y DLLs que quiera usar. No obstante, recuerde que todo lo que declare sólo será visible en el método **run** de la macro.

Cuando termine de editar, cierre la ventana. Aparecerá un cuadro de diálogo para que introduzca el nombre de la macro; hágalo y elija Aceptar para guardarla en disco. Al igual que las fichas, las macros pueden guardarse eligiendo Archivo | Guardar o Archivo | Guardar como, y pueden enviarse eligiendo Lenguaje | Enviar. Es posible cambiar las macros guardadas, pero no las enviadas.

Las macros se explican detalladamente en la *Guía de Referencia ObjectPAL*.

Vea también
Anexión de código a macros
Envío de macros

Anexión de código a macros

Para anexar código a una macro, elija Archivo | Abrir | Macro o Archivo | Nuevo | Macro. A continuación, escriba el código en la ventana del Editor que aparece; cuando termine, elija Archivo | Guardar para guardar en disco tanto el código fuente como el ejecutable. Si sólo quiere guardar el código ejecutable de la macro, elija Lenguaje | Enviar.

Con el cuadro de diálogo Métodos y la ventana del Editor de ObjectPAL puede anexar código a una macro de las siguientes formas:

- Anexando código a los métodos estándar.
- Anexando métodos personalizados.
- Anexando procedimientos personalizados
- Declarando variables, constantes, tipos de datos y rutinas externas.

Vea también

Anexión de código a métodos estándar
Anexión de métodos personalizados
Anexión de procedimientos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas
Envío de macros
Cuadro de diálogo Métodos

Anexión de código a métodos estándar

Cada macro tiene los siguientes métodos estándar: **run**, **error** y **status** (este último sólo disponible en el nivel avanzado de ObjectPAL). Es posible anexar código a estos métodos estándar lo mismo que en el caso de los demás objetos. Para ello se emplea la ventana Editor.

Puede ejecutar estos métodos con Paradox de forma interactiva o activarlos desde un método o procedimiento de ObjectPAL.

Vea también
Anexión de código a macros
Anexión de métodos personalizados
Anexión de procedimientos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas
Editor de ObjectPAL

Anexión de métodos personalizados

Para anexar métodos personalizados a una macro, use el cuadro de diálogo Métodos.

Para acceder a dicho cuadro de diálogo, inspeccione la ventana Macro o pulse Ctrl+Espaciador, escriba el nombre del nuevo método personalizado y elija Aceptar para abrir otra ventana del Editor.

Encontrará información sobre la anexión de métodos personalizados y la programación con ObjectPAL en la *Guía del Programador ObjectPAL.*

Vea también
Anexión de código a macros
Anexión de código a métodos estándar
Anexión de métodos personalizados
Anexión de procedimientos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas

Anexión de procedimientos personalizados

Desde el cuadro de diálogo Métodos de una macro, puede elegir Procs para abrir una ventana del Editor donde puede declarar procedimientos personalizados para la macro.

Encontrará información sobre la anexión de procedimientos personalizados y la programación con ObjectPAL en la *Guía del Programador ObjectPAL.*

Vea también
Anexión de código a macros
Anexión de código a métodos estándar
Anexión de métodos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas
El cuadro de diálogo Métodos
El Editor de ObjectPAL

Declaración de variables, constantes, tipos de datos y rutinas externas

Desde el cuadro de diálogo Métodos de una macro, puede declarar variables, constantes, tipos de datos y rutinas externas eligiendo Var, Const, Type o Uses, respectivamente, para abrir la correspondiente ventana del Editor. Los elementos que se declaran en estas ventanas son globales para la biblioteca, pero no son accesibles por otras fichas u objetos. En cambio, otras fichas y objetos pueden activar rutinas de biblioteca que accedan a estas variables.

Anexión de código a macros
Anexión de código a métodos estándar
Anexión de métodos personalizados
Anexión de procedimientos personalizados
Declaración de variables, constantes, tipos de datos y rutinas externas

Edición de macros

Las macros pueden editarse en ventanas del Editor de ObjectPAL.

Para editar una macro

1. Elija Archivo | Abrir | Macro y seleccione la macro correspondiente.
2. Haga clic en Diseñar y luego en Aceptar. ObjectPAL abrirá la macro en una ventana del Editor con el método estándar **run** visible.
3. Use el Editor de ObjectPAL para editar la macro de igual forma que un método.

Vea también

El Editor de ObjectPAL

Depuración de macros

Puede depurar macros con el Depurador de ObjectPAL.

Para depurar una macro

1. Elija Archivo | Abrir | Macro y seleccione la macro correspondiente.
2. Haga clic en Diseñar y luego en Aceptar. La macro aparecerá en una ventana del Editor.
3. Elija Depurar | Establecer puntos de ruptura, o haga clic sobre el botón Establecer puntos de ruptura, para abrir el cuadro de diálogo del mismo nombre.
4. Introduzca el número de la línea donde desee establecer el punto de ruptura y haga clic en Aceptar.
5. Elija Depurar | Ejecutar para ejecutar la macro.

Cuando ObjectPAL encuentre el punto de ruptura, dejará de ejecutar la macro y la mostrará en una ventana del Depurador de ObjectPAL, donde puede depurar la macro como cualquier otro método.

Vea también

El Depurador de ObjectPAL

Ejecución de macros

Es posible ejecutar una macro con Paradox de forma interactiva o desde un método. En ambos casos ejecutará el método **run** de la macro.

Uso interactivo de Paradox

1. Elija Archivo | Abrir | Macro. Aparece un cuadro de diálogo con las macros disponibles.

2. Seleccione una macro, elija Ejecutar y luego Aceptar; se ejecutará la macro.

Desde un método

Use el método **play** de tipo System para ejecutar una macro desde dentro de un método o procedimiento. Por ejemplo:

```
switch
   case theValue = "esto" : play("hazEsto")   ; ejecuta la macro "hazEsto"
   case theValue = "eso" : play("hazEso")   ; ejecuta la macro "hazEso"
   otherwise              : play("loOtro") ; ejecuta la macro "loOtro"
endSwitch
```

Envío de macros

Use Lenguaje | Enviar para enviar las macros que haya creado. Cuando se envía una macro, Paradox elimina todo el código fuente; el código no se pierde, sino que se protege.

Si guarda una macro con Archivo | Guardar, cualquier usuario que la emplee podrá cambiar el código de ObjectPAL y, por tanto, la aplicación. Para evitarlo, Enviar permite a otros usuarios acceder a dicho código, pero no modificarlo.

Cuando se elige Lenguaje | Enviar, Paradox guarda una copia de la macro con extensión .LDL. Así podrá cambiar el código de ObjectPAL en la macro que tiene extensión .LSL, pero si quiere que los demás usuarios la utilicen sin ningún riesgo, proporcióneles la macro enviada.

Menú del Entorno integrado de desarrollo (IDE) de ObjectPAL

El Entorno integrado de desarrollo (IDE) de ObjectPAL incluye varias herramientas especiales para la edición de métodos de ObjectPAL. Estas herramientas aparecen en los siguientes menús cuando se abre una ventana del Editor:

Edición
Lenguaje
Depurar
Propiedad

Los menús Archivo, Ventana y ? (Ayuda) no varían. Consulte Comandos de menú habituales para más información sobre los comandos comunes a todas las ventanas de Paradox.

Vea también

Conceptos básicos de ObjectPAL

Edición

Deshacer anula la última edición. Esta opción aparece inactiva si no se ha realizado ningún cambio.

Cortar desplaza el texto seleccionado al portapapeles y lo borra de la ventana. Esta opción aparece inactiva si no se ha realizado ninguna selección.

Copiar copia el texto seleccionado en el portapapeles. Esta opción aparece inactiva si no se ha realizado ninguna selección.

Pegar copia texto del portapapeles en la posición actual del cursor. Si se ha seleccionado texto, éste se sustituye por el contenido del portapapeles. Esta opción aparece inactiva si no hay texto en el portapapeles.

Eliminar borra texto seleccionado. La opción aparece inactiva si no se ha realizado ninguna selección.

Seleccionar todo selecciona todo el texto de la ventana.

Vea también

Edición | Pegar desde
Edición | Copiar en
Edición | Buscar
Edición | Buscar siguiente
Edición | Sustituir
Edición | Sustituir siguiente
Edición | Ir a

Edición | Pegar desde

Utilice esta opción para copiar un archivo de texto en el método actual, en la posición del cursor.

Cuando elige Pegar desde aparece el cuadro de diálogo Copy From File. Introduzca el nombre del archivo de texto y elija Aceptar o seleccione Examinar para buscar el archivo que desea copiar.

Edición | Copiar a

Utilice esta opción para copiar texto seleccionado en el archivo de texto que especifique.

Cuando elige Copiar a, aparece el cuadro de diálogo Copiar en archivo. Introduzca el nombre del archivo en el que desea copiar el texto seleccionado.

Nota: El texto que se copia en el archivo especificado es el texto seleccionado, *no* el contenido del Portapapeles de Windows.

Edición | Buscar

Utilice esta opción para localizar cadenas de texto en el código.

Cuando elige Buscar, aparece el cuadro de diálogo del mismo nombre, en el que puede especificar el texto que va a buscar, la dirección de búsqueda y la concordancia de mayúsculas y minúsculas.

Vea también

Cuadro de diálogo Buscar
Edición | Buscar siguiente
Edición | Sustituir

Cuadro de diálogo Buscar

Utilice este cuadro de diálogo para localizar cadenas de texto en el código. Para definir la búsqueda especifique:

Buscar	el texto que desea localizar.
Dirección	hacia delante o atrás en el código.
Atrás	desde la posición actual del cursor hasta el comienzo del método.
Adelante	desde la posición actual del cursor hasta el final del método.
Distinguir mayús/minús	si se activa, la búsqueda distingue mayúsculas y minúsculas.

Para abrir el cuadro de diálogo Buscar elija Edición | Buscar.

Vea también

Edición | Buscar
Edición | Buscar siguiente
Edición | Sustituir

Edición | Buscar siguiente

Utilice esta opción para buscar la siguiente ocurrencia del texto especificado con Buscar. Buscar siguiente aparece inactivo si no se ha realizado ninguna búsqueda durante la sesión actual.

Vea también

Edición | Buscar
Edición | Sustituir

Edición | Sustituir

Utilice esta opción para buscar texto y sustituirlo por el valor que especifique. En el cuadro de diálogo Buscar y sustituir especifique el texto que desea localizar y el valor que desea que lo sustituya.

Vea también

Cuadro de diálogo Buscar y sustituir
Edición | Buscar

Cuadro de diálogo Buscar y sustituir

Utilice este cuadro de diálogo para modificar cadenas de texto en el código.

Buscar	Introduzca el texto que desea localizar.
Sustituir por	Introduzca el texto de sustitución.
Distinguir mayús/minús	Active esta opción para buscar el texto tal como está tecleado, incluida la diferencia entre mayúsculas y minúsculas.
Sustituir todo	Active esta opción para sustituir el texto de la cadena siempre que aparezca en el método.
Dirección	Elija Atrás o Adelante para buscar en dichas direcciones en el archivo.

Para abrir el cuadro de diálogo Buscar y sustituir elija Edición | Sustituir.

Vea también

Edición | Sustituir

Edición | Sustituir siguiente

Elija esta opción para sustituir la siguiente ocurrencia del texto especificado en el cuadro de diálogo Buscar y sustituir. Sustituir siguiente aparece inactivo hasta que se sustituye texto.

Vea también

Cuadro de diálogo Buscar y sustituir
Edición | Buscar
Edición | Sustituir

Edición | Ir a

Utilice esta opción para desplazarse rápidamente a una línea del código.

Cuando elige Ir a, aparece el cuadro de diálogo Ir a línea. Para desplazarse a una línea, teclee su número y haga clic sobre Aceptar; si el número no existe, el cursor no se desplaza.

Lenguaje

Los comandos de este menú se utilizan cuando se escribe o edita código de ObjectPAL.

Verificar sintaxis compila todos los métodos de la ficha (no sólo de la ventana actual). Si se encuentran errores de sintaxis, se abre la ventana del método correspondiente con el cursor situado cerca del error y aparece un mensaje de error en la línea de estado.

Advertencia siguiente activa y desactiva la aparición de mensajes de advertencia procedentes del compilador. Si esta opción está activada, aparecen mensajes en la línea de estado que advierten sobre la introducción de variables sin declarar y otras condiciones que pueden producir un error en el momento de la ejecución.

Métodos proporciona un acceso rápido al menú Métodos.

Arbol de objetos muestra un diagrama de árbol con los objetos relacionados en la ficha actual.

Palabras clave éste es un menú en cascada con palabras clave de uso frecuente; se utiliza para incluir una palabra clave en un método sin necesidad de teclearla.

Tipos muestra un cuadro de diálogo que contiene una lista de todos los tipos de objetos y sus métodos. Puede elegir un nombre de tipo para visualizar sus métodos y procedimientos o insertar un prototipo del método o procedimiento en el método que está creando, en la posición actual del cursor.

Propiedad muestra un cuadro de diálogo con una lista de los objetos y sus propiedades. Puede elegir un nombre de objeto para visualizar sus propiedades e insertarlas en el método.

Constantes muestra una lista de constantes que proporciona ObjectPAL para especificar colores, formas del ratón, atributos de menú y estilos de ventana. Puede elegir una constante e insertarla en el método.

Examinar recursos crea y muestra un informe con el código fuente de la ficha o el informe actuales.

Enviar crea una ficha o un informe compilados a partir del código fuente de ObjectPAL. Los usuarios no pueden editar el diseño o el código fuente.

Para más información sobre los comandos, seleccione el apropiado y pulse F1.

Lenguaje | Verificar sintaxis

Utilice esta opción para compilar todos los métodos de la ficha (no sólo la ventana actual). Si se encuentra un error de sintaxis, aparece una ventana con el cursor situado cerca del error y un mensaje de error en la línea de estado.

Nota: Si modifica el código en más de una ventana del Editor, guarde los cambios antes de verificar la sintaxis. Si no lo hace así, pueden producirse errores inesperados ya que no se verifica el último código.

Lenguaje | Advertencia siguiente

Utilice esta opción para desplazarse al siguiente mensaje de advertencia del compilador. Estas advertencias sólo aparecen si Propiedad | Advertencias compilador está activado; en caso contrario, se suprimen.

Lenguaje | Métodos

Elija esta opción para acceder al cuadro de diálogo Métodos, donde puede elegir en listas de métodos estándar o personalizados, definir un nuevo método personalizado o abrir una ventana del Editor para declarar rutinas, tipos, constantes, variables o procedimientos externos.

Vea también

El cuadro de diálogo Métodos

El cuadro de diálogo Métodos

Utilice este cuadro de diálogo para seleccionar un método en las listas de métodos estándar o personalizados o para definir un nuevo método personalizado.

Métodos estándar	Seleccione el método estándar que va a editar y haga clic sobre Aceptar. Se abre una ventana del Editor.
Métodos personalizados	Elija el método personalizado que va a editar.
Método personalizado nuevo	Para crear un nuevo método personalizado, introduzca su nombre en el cuadro de texto y haga clic sobre Aceptar. Aparece una ventana del Editor donde puede introducir el método.

El cuadro de diálogo Métodos también incluye las opciones siguientes:

Utilice	Para declarar
Var	variables
Const	constantes
Type	tipos
Procs	procedimientos
Uses	procedimientos que utilizan los métodos del objeto

Cada una de estas opciones tiene su propia ventana del Editor, que se abre cuando se selecciona. Si se declara una variable, una constante o un procedimiento en una de estas ventanas, será visible para todos los métodos anexados.

Para obtener ayuda sobre un método o procedimiento determinados, haga clic sobre el botón Buscar e introduzca su nombre en el cuadro de diálogo Buscar. También puede utilizar la Lista alfabética de métodos.

Para abrir el cuadro de diálogo Métodos elija Lenguaje | Métodos.

Vea también

El Editor de ObjectPAL
El Depurador de ObjectPAL
Lista alfabética de métodos
Introducción a ObjectPAL
Macros
Bibliotecas

Lenguaje | Arbol de objetos

Utilice esta opción para visualizar un diagrama de árbol con los objetos relacionados en la ficha actual. Este diagrama muestra la jerarquía del objeto, con el objeto actual en el extremo izquierdo y el árbol, que presenta la jerarquía del contenedor, extendiéndose hacia la derecha.

Cuando se incluye un objeto en una ficha, Paradox le asigna un nombre por defecto que comienza con el signo de almohadilla (#). El Arbol de objetos muestra los objetos incluidos con y sin nombre. Si se han escrito métodos para un objeto, su nombre aparece subrayado y señalado con un asterisco. Es posible inspeccionar el objeto y elegir Métodos para visualizar el cuadro de diálogo Métodos.

Mediante el Arbol de objetos pueden anexarse y editarse métodos para una ficha, al igual que se realiza para un botón.

Vea también

Anexación de métodos a una ficha
Métodos

Anexación de métodos a una ficha

1. En una ventana de diseño, pulse Esc para anular la selección de todos los demás objetos (incluida la página). Es posible que deba pulsar Esc repetidas veces.
2. Elija Ficha | Arbol de objetos para visualizar un diagrama de los objetos de la ficha.
3. El objeto denominado #form design1, que aparece en el extremo izquierdo del diagrama (el elemento "superior" del árbol de objetos) representa la ficha. Inspecciónelo y elija Métodos para visualizar el cuadro de diálogo Métodos correspondiente.
4. Elija y edite un método o cree un método personalizado como haría con cualquier otro objeto.

Consejo: Anexe a la ficha métodos personalizados de uso frecuente; esta solución es más eficaz que copiar el método en cada objeto que lo activa.

Vea también

Lenguaje | Arbol de objetos
Métodos
Edición de métodos
Métodos personalizados

Lenguaje | Palabras clave

Utilice esta opción para visualizar un menú en cascada de palabras clave de uso frecuente. Elija una palabra clave en este menú para introducirla en un método sin necesidad de teclearla.

Las palabras clave son elementos básicos del lenguaje. Vea Elementos básicos del lenguaje para obtener información más concreta sobre las palabras clave.

Lenguaje | Tipos

Esta opción presenta un cuadro de diálogo que incluye una lista de todos los tipos de objeto y sus métodos. Si elige un nombre de tipo puede visualizar sus métodos y procedimientos. Por ejemplo, para visualizar una lista de los métodos y procedimientos del tipo Array, elija Array.

En primer lugar se incluyen los métodos y, a continuación, los procedimientos, ambos en orden alfabético.

Vea también

Cuadro de diálogo Tipos y métodos

Cuadro de diálogo Tipos y métodos

En este cuadro de diálogo se incluye una lista con todos los tipos de objeto y sus métodos y procedimientos.

Tipos presenta una lista de todos los tipos de ObjectPAL en orden alfabético. Puede seleccionar un nombre de tipo y visualizar sus métodos y procedimientos.

Métodos presenta una lista con todos los métodos y procedimientos del tipo seleccionado. Los métodos aparecen en primer lugar y, a continuación, los procedimientos, ambos en orden alfabético.

Insertar tipo inserta el nombre de tipo en el método, en la posición actual del cursor. Después de seleccionar el tipo, haga clic sobre el botón Insertar tipo y, a continuación, sobre Aceptar.

Insertar método inserta el nombre del método o procedimiento en el método principal, en la posición actual del cursor. Después de seleccionar el método o el procedimiento, haga clic sobre el botón Insertar método y, a continuación, sobre Aceptar.

Para abrir el cuadro de diálogo Tipos y métodos, elija Lenguaje | Tipos.

Lenguaje | Propiedad

Esta opción presenta un cuadro de diálogo que incluye una lista de objetos y sus propiedades.

Puede elegir un nombre de objeto para visualizar sus propiedades e insertarlas en el método. Por ejemplo, para visualizar una lista con las propiedades de Button, elija Button. En la columna Propiedad aparecen las disponibles para este elemento. Elija la que desee incluir y haga clic sobre el botón Insertar propiedad; de esta forma, la propiedad se incluye en el código.

Vea también

Cuadro de diálogo Objetos y propiedades

Cuadro de diálogo Objetos y propiedades

En este cuadro de diálogo se visualizan los objetos y sus propiedades.

Objetos	presenta una lista con todos los objetos en pantalla. Elija un nombre de objeto para visualizar sus propiedades.
Propiedad	presenta una lista con todas las propiedades del objeto seleccionado.
Valores	presenta una lista con los valores posibles de una propiedad. Seleccione el valor que desea de la lista; si sólo hay un valor posible, este cuadro aparece vacío.
Insertar objeto	inserta el nombre del objeto en el método. Después de seleccionar el objeto, haga clic sobre este botón y sobre Aceptar.
Insertar propiedad	inserta el nombre de la propiedad en el método. Después de seleccionar la propiedad, haga clic sobre este botón y sobre Aceptar.

Para abrir el cuadro de diálogo Objetos y propiedades, elija Lenguaje | Propiedad.

Lenguaje | Constantes

Esta opción permite visualizar listas de constantes que proporciona ObjectPAL y que permiten especificar colores, formas del ratón, atributos de menú y estilos de ventana. Puede elegir una constante y hacer clic sobre el botón Insertar para incluirla en el método.

Vea también
Cuadro de diálogo Constantes

Cuadro de diálogo Constantes

Este cuadro de diálogo permite visualizar una lista de constantes para cada tipo de constante.

Tipos de constantes	presenta una lista con todas las categorías de constantes de ObjectPAL. Elija un tipo para visualizar todas sus constantes.
Constantes	presenta una lista con todas las constantes del tipo seleccionado.
Insertar constante	inserta el nombre de la constante en el código, en la posición actual del cursor. Después de seleccionar la constante, haga clic sobre el botón Insertar constante y sobre Aceptar.

Para abrir el cuadro de diálogo Constantes, elija Lenguaje | Constantes.

Lenguaje | Examinar recursos

Esta opción permite crear y visualizar un informe con el código fuente de la ficha o el informe actuales.

Lenguaje | Enviar

Esta opción permite crear una ficha, macro o biblioteca compiladas a partir del código fuente de ObjectPAL. Los usuarios no pueden editar el diseño o el código fuente.

Para más información sobre el envío de aplicaciones, consulte la *Guía del Programador ObjectPAL.*

Vea también

Envío de aplicaciones

Depurar

Utilice los comandos de este menú cuando depure el código de ObjectPAL.

Inspector Muestra y modifica (optativo) el valor de una variable. Sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Seguimiento retroactivo Presenta una lista con los métodos y procedimientos activados desde que la ficha comenzó a ejecutarse. La rutina llamada en último lugar aparece en la primera posición, seguida por el objeto que la activó; a continuación se incluye la penúltima rutina activada y así sucesivamente hasta llegar al primer método o procedimiento. Sólo está disponible cuando se interrumpe la ejecución en un punto de ruptura.

Establecer puntos de ruptura Define los puntos de ruptura que detienen la ejecución de un método en líneas determinadas. Pueden establecerse tantos puntos de ruptura como permita la memoria del sistema, pero no en procedimientos escritos por el usuario, donde se puede pasar por ellos mediante Depurar | Incluir llamadas.

Listar puntos de ruptura Presenta una lista con los números de línea de los puntos de ruptura y, de forma optativa, permite borrarlos.

Seguimiento de la ejecución Abre una ventana y presenta una lista con cada una de las líneas de código a medida que lo ejecuta. El parámetro de este elemento se guarda con la ficha, por lo que no es necesario activarlo cada vez que se desea realizar un seguimiento.

Métodos de seguimiento Muestra un cuadro de diálogo que incluye una lista con los métodos de seguimiento del objeto seleccionado. Active uno de estos métodos para visualizar información sobre él en la ventana Controlador de seguimiento a medida que se ejecuta. Es necesario disponer de código de ObjectPAL anexado por lo menos a un objeto.

Activar modo DEPURADOR Detiene la ejecución siempre que encuentra la sentencia DEBUG. La introducción de una sentencia DEBUG en un método tiene el mismo efecto que la definición de un punto de ruptura en la misma línea, con la ventaja de que se guarda con el código fuente, por lo que no es necesario definirlo cada vez, como sucede con los puntos de ruptura. El parámetro de este elemento se guarda con la ficha.

Activar Ctrl+Inter para DEPURADOR Permite utilizar Ctrl+Inter para detener la ejecución y abrir una ventana del Depurador que contiene el método o procedimiento activos, como si se hubiese encontrado un punto de ruptura.

Si este elemento no está activado, la pulsación de Ctrl+Inter finaliza la ejecución.

Ver fuente Muestra otro código del método en una ventana del Editor.

Origen Muestra el método que contiene el punto de ruptura actual y sitúa el cursor en la línea donde está incluido. Resulta útil si se trabaja con varios métodos en diferentes ventanas y se desea volver rápidamente al punto de ruptura. Sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Excluir llamadas Permite realizar un paso único en un método y omitir las llamadas a procedimientos y funciones. Sólo está disponible cuando se detiene la ejecución

en un punto de ruptura.

Incluir llamadas	Permite realizar un paso único en un método e incluir las llamadas a procedimientos y funciones. Sólo está disponible cuando se detiene la ejecución en un punto de ruptura.
Salir del método	Finaliza la ejecución y cierra todas las ventanas del Depurador. Sólo está disponible cuando se ha detenido la ejecución en un punto de ruptura.
Ejecutar	Abandona el Depurador, guarda los cambios y vuelve al modo ver datos.

Para más información sobre cada comando, selecciónelo y pulse F1.

Depurar | Inspector

Esta opción muestra y modifica (de forma optativa) el valor de una variable. Sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Vea también

Depurar | Establecer puntos de ruptura

Depurar | Seguimiento retroactivo

Esta opción presenta una lista con los métodos y procedimientos activados desde que comenzó la ejecución de la ficha. En primer lugar aparece la última rutina activada seguida del objeto que la activó y, a continuación la penúltima rutina activada, hasta llegar al primer método o procedimiento. Esta opción sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Vea también

Depurar | Establecer puntos de ruptura

Depurar | Establecer puntos de ruptura

Esta opción permite definir puntos de ruptura en un método para detener la ejecución en líneas determinadas. Pueden definirse tantos puntos de ruptura como permita la memoria del sistema.

No es posible definir puntos de ruptura en procedimientos escritos por el usuario, pero sí pasar por ellos mediante Depurar | Incluir llamadas.

Cuando elige Depurar | Establecer puntos de ruptura aparece el cuadro de diálogo del mismo nombre.

Vea también

Cuadro de diálogo Establecer puntos de ruptura
Depurar | Incluir llamadas

Cuadro de diálogo Establecer puntos de ruptura

Este cuadro de diálogo permite detener la ejecución de métodos en líneas determinadas.

La lista de objetos informa sobre el método en el que se incluye el punto de ruptura.

En el cuadro de texto Número línea introduzca el número de la línea donde desea incluir el punto de ruptura y haga clic sobre Aceptar. El número de línea por defecto viene determinado por la posición actual del cursor.

Los puntos de ruptura deben incluirse en líneas ejecutables. Si intenta incluirlos en otro tipo de línea, Paradox los introducirá en la siguiente línea ejecutable; si no hay ninguna línea ejecutable, el programa muestra un mensaje de error de número de línea incorrecto.

Cuando se define un punto de ruptura, ObjectPAL ejecuta una verificación de sintaxis en el código. El punto de ruptura no se definirá si existe un error de sintaxis.

Cuando se guarda el código, los puntos de ruptura se eliminan de forma automática; también se suprimen si se modifica el código fuente.

Para abrir el cuadro de diálogo Establecer puntos de ruptura, elija Depurar | Establecer puntos de ruptura o haga clic sobre el botón correspondiente de la barra rápida.

Depurar | Listar puntos de ruptura

Esta opción presenta un cuadro de diálogo con los número de línea donde están situados los puntos de ruptura y permite borrar cualquiera de ellos.

Nota: Si se modifica el código fuente, los puntos de ruptura se eliminan automáticamente.

Depurar | Seguimiento de la ejecución

Si está activada esta opción, el Controlador de seguimiento de ObjectPAL abre una ventana en la que incluye cada línea de código a medida que la ejecuta. El parámetro de este elemento se guarda con la ficha, por lo que no es necesario activarlo cada vez que se desea realizar un seguimiento.

Si esta opción no está activada, la ejecución se realiza de forma normal. ObjectPAL, además, incluye procedimientos para modificar parámetros del Controlador de seguimiento.

Depurar | Métodos de seguimiento

Esta opción presenta un cuadro de diálogo con todos los métodos de seguimiento. Marque uno de ellos para visualizar información relacionada con el método en la ventana del Controlador de seguimiento a medida que se ejecuta. Para que esta opción pueda utilizarse es necesario disponer de código anexado por lo menos a un objeto.

Nota: La activación de componentes de este cuadro de diálogo no activa de forma automática el controlador. Es necesario elegir Depurar | Seguimiento de la ejecucion o Depurar | Activar modo DEPURADOR y utilizar el procedimiento **tracerOn** en el código de ObjectPAL anexado a un objeto.

La activación de un método de seguimiento indica que se desea controlar dicho método; los métodos que no se activan, no se controlan. No es relevante el que un método incluya código anexado, ya que por el simple hecho de estar activado se controlará.

Si se activa el cuadro "form prefilter", los métodos se controlan a medida que se ejecutan para la ficha y el objeto que se pretende destino; en caso contrario, sólo se controlan para el objeto destino.

Vea también

Depurar | Seguimiento de la ejecución
Depurar | Activar modo DEPURADOR

Cuadro de diálogo Métodos estándar de seguimiento

Este cuadro de diálogo se utiliza para visualizar todos los métodos estándar. Si se activa uno de estos métodos, se indica que debe controlarse, independientemente de que incluya o no código de ObjectPAL anexado.

Todos activa todos los métodos

Ninguno desactiva todos los métodos

Para abrir el cuadro de diálogo Métodos estándar de seguimiento, elija Depurar | Métodos de seguimiento.

Depurar | Activar modo DEPURADOR

Si esta opción está activada, la ejecución se detiene siempre que se encuentra la sentencia DEBUG. La inclusión de esta sentencia en un método tiene el mismo efecto que la definición de un punto de ruptura en la misma línea, con la ventaja de que se guarda con el código fuente, por lo que no es necesario volver a definirlo como sucede con los puntos de ruptura. El parámetro de este elemento se guarda con la ficha.

Si no se activa esta opción, las sentencias DEBUG se ignoran. Asegúrese de que desactiva esta opción antes de enviar una ficha o un informe; en caso contrario, las sentencias DEBUG actuarán cuando el usuario ejecute la ficha o el informe.

Nota: Las sentencias DEBUG sólo tienen efecto en métodos escritos por el usuario y no en los procedimientos.

Nota: Si esta opción está activada, Paradox proporciona información de error más detallada. En realidad, esta opción es útil incluso si no se utiliza nunca una sentencia DEBUG.

Depurar | Activar Ctrl+Inter para DEPURADOR

Si esta opción está activada y se pulsa Ctrl+Inter, se detiene la ejecución y se abre una ventana del Depurador con el método o procedimiento activos, de igual forma que sucede cuando se encuentra un punto de ruptura.

Si no está activada, Ctrl+Inter finaliza la ejecución.

Nota: Ctrl+Inter finaliza la ejecución de métodos y procedimientos de ObjectPAL. Otras operaciones, como las consultas, no se ven afectadas.

Depurar | Ver fuente

Depurar | Ver fuente muestra otro código del método en una ventana del Editor. Esta es una forma rápida de desplazarse a un método específico para localizar un objeto determinado. Aunque este cuadro de diálogo no incluye barras de desplazamiento, puede utilizar el ratón o las teclas de flecha para desplazarse por los elementos.

Depurar | Origen

Esta opción muestra el método que contiene el punto de ruptura actual, con el cursor situado en la línea donde se incluye.

Cuando la ejecución se detiene en un punto de ruptura, una ventana del Editor muestra el método que lo contiene; en este momento es posible abrir otras ventanas del Editor con otros métodos, por lo que la pantalla puede aparecer desordenada. Esta función resulta útil cuando existen varias ventanas abiertas y se desea volver rápidamente al punto de ruptura. La opción Origen sólo está disponible si la ejecución se ha detenido en un punto de ruptura.

Depurar | Excluir llamadas

Esta opción permite realizar pasos únicos en un método, tratando los procedimientos y métodos personalizados como pasos únicos. Esta función sólo está disponible cuando se detiene la ejecución en un punto de ruptura.

Vea también

Depurar | Incluir llamadas
Depurar | Establecer puntos de ruptura

Depurar | Incluir llamadas

Esta opción permite realizar un paso único por cada línea de un método y cada línea de los procedimientos y métodos personalizados que activa el método prinicpal. Esta función sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Vea también

Depurar | Excluir llamadas
Depurar | Establecer puntos de ruptura

Depurar | Salir del método

Esta opción finaliza la ejecución y cierra las ventanas del Depurador; sólo está disponible cuando la ejecución se detiene en un punto de ruptura.

Vea también

Depurar | Establecer puntos de ruptura

Depurar | Ejecutar

Elija esta opción para ejecutar el Depurador de ObjectPAL. Paradox guarda todos los métodos anexados, compila el código y pasa a una ventana de visualización. Cuando encuentra un punto de ruptura detiene la ejecución y abre una ventana del Depurador.

Desde una ventana del Depurador, la ejecución continúa a partir del punto de ruptura.

Nota: Para abrir una ventana del Depurador debe establecer puntos de ruptura en el código.

Vea también
Depurar | Establecer puntos de ruptura

Propiedad

Utilice los comandos de este menú para adaptar la ventana de ObjectPAL a sus necesidades.

Escritorio	muestra un cuadro de diálogo en el que se definen las propiedades de la ventana del Escritorio.
Tamaño ventana	define el tamaño por defecto de las ventanas del Editor y el Depurador.
Editor alternativo	muestra un cuadro de diálogo en el que se puede especificar otro editor para escribir o modificar los métodos.
Advertencias compilador	especifica si el compilador debe mostrar mensajes de advertencia cuando se encuentra con variables sin declarar.

Vea también

Propiedad | Escritorio
Propiedad | Tamaño ventana
Propiedad | Editor alternativo
Propiedad | Advertencias compilador

Propiedad | Escritorio

Esta opción presenta un cuadro de diálogo donde puede modificarse el aspecto del Escritorio.

Título	Introduzca otro título para la barra de título del Escritorio.
Mapa de bits de fondo	Introduzca el nombre de un archivo de mapa de bits o elija Buscar para consultar una lista en el cuadro de diálogo Seleccionar archivo.
Barra rápida	Convierta la barra rápida en una paleta flotante ordenada en una o dos columnas o filas. Para devolver la barra rápida a su posición original bajo el menú, elija Fijo en su menú de Control.
Nivel de ObjectPAL	Modifique el nivel de ObjectPAL en el que trabaja.

Cuadro de diálogo Propiedades del Escritorio

Este cuadro de diálogo permite modificar el aspecto del Escritorio.

Título	Introduzca otro título para la barra de título del Escritorio.
Mapa de bits de fondo	Introduzca el nombre de un archivo de mapa de bits o elija Buscar para consultar una lista en el cuadro de diálogo Seleccionar archivo.
Barra rápida	Convierta la barra rápida en una paleta flotante ordenada en una o dos columnas o filas. Para devolver la barra rápida a su posición original bajo el menú, elija Fijo en su menú de Control.
Nivel de ObjectPAL	Modifique el nivel de ObjectPAL en el que trabaja. El nivel por defecto es Principiante.

Para abrir el cuadro de diálogo Propiedades del Escritorio, elija Propiedad | Escritorio.

Propiedad | Tamaño ventana

Elija esta opción para modificar el tamaño por defecto de las ventanas del Editor y el Depurador. Seleccione el tamaño que desea y elija Tamaño ventana para acceder al cuadro de diálogo Tamaño de la ventana del Editor, en el que puede convertir el tamaño actual en el valor por defecto o volver al tamaño por defecto de Windows.

Vea también

Cuadro de diálogo Tamaño de la ventana del Editor

Cuadro de diálogo Tamaño de la ventana del Editor

En este cuadro de diálogo puede modificar el tamaño por defecto de las ventanas del Editor y el Depurador.

Usar tamaño actual desde ahora	modifica el tamaño por defecto de la ventana del Editor y el Depurador y le asigna el tamaño de la ventana actual.
Restablecer tamaño por omisión	modifica el tamaño por defecto de la ventana del Editor y el Depurador y le asigna el tamaño por defecto de Windows.

Para acceder a este cuadro de diálogo, elija Propiedad | Tamaño ventana.

Propiedad | Editor alternativo

Esta opción presenta un cuadro de diálogo en el que puede especificar otro editor para su uso en la escritura y modificación de métodos. Introduzca la vía de acceso completa al editor en el cuadro de texto Editor de ObjectPAL y haga clic sobre Aceptar.

Preferencias de ObjectPAL

Este cuadro de diálogo permite especificar otro editor para su uso en la escritura y edición de código de ObjectPAL.

Editor de ObjectPAL alternativo	presenta la vía de acceso al editor alternativo que especifique. Introduzca la vía de acceso completa al editor en el cuadro de texto y haga clic sobre Aceptar.
Usar estándar	seleccione esta opción para utilizar el Editor estándar de ObjectPAL.
Usar alternativo	seleccione esta opción si ha especificado un editor alternativo; esta casilla sólo está disponible si ha introducido un editor alternativo.

Propiedad | Advertencias compilador

Utilice esta opción para activar y desactivar la visualización de mensajes de advertencia procedentes del compilador. Si la opción está activada, los mensajes de la línea de estado advierten sobre variables sin declarar y otras condiciones que pueden producir un error en el momento de la ejecución. Si la opción está desactivada, estos mensajes no aparecen.

Tipos de Constantes

ActionClasses
ActionDataCommands
ActionEditCommands
ActionFieldCommands
ActionMoveCommands
ActionSelectCommands
ButtonStyles
ButtonTypes
Colors
CompleteDisplay
ErrorReasons
EventErrorCodes
ExecuteOptions
FieldDisplayTypes
FileBrowserFileTypes
FontAttributes
FrameStyles
General
GraphBindTypes
GraphicMagnification
GraphLabelFormats
GraphLegendPosition
GraphMarkers
GraphTypeOverRide
GraphTypes
IdRanges
Keyboard
KeyBoardStates
LibraryScope
LineEnds
LineStyles
LineThickness
LineTypes
MenuChoiceAttributes
MenuCommands
MenuReasons
MouseShapes
MoveReasons
PatternStyles
RasterOperations
ReportOrientation
ReportPrintPanel
Status Reasons
TableFrameStyles
TextAlignment
TextDesignSizing
TextSpacing
UIObjectTypes
ValueReasons
WindowStyles

ActionClasses

Constante	Tipo de datos	Descripción
DataAction	SmallInt	Estas acciones son para el desplazamiento en una tabla y para tareas tales como el bloqueo y almacenamiento de registros.
EditAction	SmallInt	La mayoría de estas acciones alteran los datos dentro de un campo.
FieldAction	SmallInt	Son una categoría especial de acciones Move que permiten el movimiento entre objetos de campo.
MoveAction	SmallInt	Tienen relación con el desplazamiento dentro de un objeto de campo.
SelectAction	SmallInt	Estas acciones equivalen a las acciones Move.

ActionDataCommands

Constante	Tipo de datos	Descripción
DataArriveRecord	SmallInt	Indica un cambio en el registro actual, independientemente de la razón. Algunas razones posibles: movimiento, edición, actualización de red y desplazamiento.
DataBegin	SmallInt	Se desplaza al primer registro de la tabla asociada con el UIObject dado. Forzará una acción (DataUnlockRecord) recursiva si el registro actual se ha modificado. Si se produce un error, ejecutará el método **error**. Se activa mediante el botón primer registro VCR, acción de menú Registro|Primero.
DataBeginEdit	SmallInt	Se utiliza para acceder al modo editar en la ficha. Normalmente siempre se permite.
DataBeginFirstField	SmallInt	Se desplaza al primer campo del primer registro de la tabla asociada con el UIObject dado.
DataCancelRecord	SmallInt	Se utiliza para desechar cambios en un registro. Por defecto tiene éxito, pero el usuario puede impedirla. Se activa mediante Edición|Deshacer, *Alt+Retroceso* o la opción de menú Registro|Cancelar cambios. También se utiliza internamente cuando se sale de un registro bloqueado pero no modificado.
DataDeleteRecord	SmallInt	Borra el registro actual. Los errores que se produzcan activarán el método **error**. Esta acción es irreversible excepto en las tablas de dBASE. Se activa mediante Registro|Eliminar o *Ctrl+Supr*.
DataDesign	SmallInt	Cambia la ficha al modo diseñar. Se activa mediante *F8*.
DataDitto	SmallInt	Copia en el registro actual todos los campos desde el registro anterior. Se activa mediante *Ctrl+D*.
DataEnd	SmallInt	Se desplaza al registro final de la tabla asociada con el UIObject dado. Forzará un action(DataUnlockRecord) recursivo si el registro actual se ha modificado. La presencia de un error activará el método **error**. Se activa mediante el botón último registro de la barra rápida.
DataEndEdit	SmallInt	Se utiliza para salir del modo editar en la ficha. Se activa mediante un segundo *F9*, Editar datos en la barra rápida o la acción de menú Ficha|Terminar edición.
DataEndLastField	SmallInt	Se desplaza al último campo del último registro de la tabla asociada con un UIObject.
DataFastBackward	SmallInt	Retrocede un grupo de registros (donde grupo se define como el número de filas de un marco de tabla o MRO).
DataFastForward	SmallInt	Avanza un grupo de registros (donde grupo se define como el número de filas de un marco de tabla o MRO).
DataHideDeleted	SmallInt	Altera el modo de la ficha de forma que se ocultarán los registros borrados (disponible sólo en tablas de dBASE). Se activa mediante Ficha|Ocultar borrados.
DataInsertRecord	SmallInt	Insertará un nuevo registro (vacío) antes del registro actual. El estado del registro aparecerá como bloqueado , y el registro vacío no existirá en la tabla subyacente hasta que se modifique o desbloquee. Se activa mediante Registro|

		Insertar, o *Ins*. Obsérvese que los registros creados de esta forma pueden desecharse mediante DataDeleteRecord o DataCancelRecord antes de que se hayan desbloqueado. Si se sale de uno de estos registros sin realizar cambios, se utilizará internamente DataCancelRecord para desecharlo.
DataLockRecord	SmallInt	Se utiliza para bloquear el registro actual. Los errores que se produzcan activarán el método **error**. Se activa mediante *F5*.
DataLookup	SmallInt	Se utiliza para activar la tabla de referencia para el campo actual, para aceptar la elección de un nuevo valor por parte del usuario y, si es pertinente
DataLookupMove	SmallInt	Forma especial de referencia que permite al usuario elegir un nuevo registro principal para este detalle.
DataNextRecord	SmallInt	Se desplaza (si es posible) al siguiente registro secuencial de la tabla asociada con el UIObject. Forzará un action(DataUnlockRecord) recursivo si se ha modificado el registro actual. Los errores que se produzcan ejecutarán el método **error**. Se activa mediante la opción de menú Registro|Siguiente, el botón registro siguiente de la barra rápida, *F12*, etc.
DataNextSet	SmallInt	Avanza un grupo de registros (donde grupo se define como el número de filas de un marco de tabla o MRO).
DataPostRecord	SmallInt	Igual que DataUnlockRecord, pero el bloqueo de registro no se liberará. Como consecuencia, si se producen cambios en campos clave que implican que el registro se desplazará a otra posición en la tabla, la posición de la tabla cambia con ese registro (seguirá siendo el registro actual). Se activa mediante *Ctrl+F5*.
DataPrint	SmallInt	Imprime una ventana Ficha o Tabla.
DataPriorRecord	SmallInt	Se desplaza (si es posible) al registro anterior de la tabla asociada con el UIObject. Forzará un action(DataUnlockRecord) recursivo si se ha modificado el registro actual. Los errores que se produzcan ejecutarán el método **error**. Se activa mediante la opción de menú Registro|Anterior, el botón registro anterior de la barra rápida, *F11*, etc.
DataPriorSet	SmallInt	Retrocede un grupo de registros (donde grupo se define como el número de filas de un marco de tabla o MRO, o 1 en el caso de una ficha con un solo registro). Forzará un action(DataUnlockRecord) recursivo si se ha modificado el registro actual. Los errores que se produzcan ejecutarán el método **error**. Se activa mediante *RePág*, Registro|Grupo anterior, *Mayús+F11*, etc.
DataRecalc	SmallInt	Fuerza que un campo calculado recalcule su valor.
DataRefresh	SmallInt	Especifica la actualización de un valor de un registro mostrado en la pantalla.
DataRefreshOutside	SmallInt	Especifica la actualización de un valor de un registro no mostrado en la pantalla.
DataSaveCrosstab	SmallInt	Escribe la tabla cartesiana dada en CARTESIA.DB.
DataSearch	SmallInt	Muestra un cuadro de diálogo para permitir que el usuario busque un valor específico dentro de un campo especificado. Se activa mediante Registro|Buscar|Valor o *Ctrl+Z*.

DataSearchNext	SmallInt	Buscará el siguiente registro que contenga el último valor especificado en respuesta a la última acción DataSearch. Se activa mediante Registro|Buscar siguiente o *Ctrl+A*.
DataSearchReplace	SmallInt	Muestra un cuadro de diálogo para permitir que el usuario busque un valor específico dentro de un campo especificado y lo sustituya por otro valor. Se activa mediante Registro| Buscar|Y sustituir, o *Ctl+Mayús+Z*.
DataShowDeleted	SmallInt	Altera el modo de la ficha para que los registros borrados se muestren (disponible sólo en tablas de dBASE). Su aspecto no será diferente de los registros normales, pero la línea de estado reflejará su estado. Se activa mediante Ficha|Mostrar borrados.
DataTableView	SmallInt	Se utiliza para mostrar la tabla principal de una ficha en modo tabla. Si la ficha se activó originalmente como ficha preferente del modo tabla existente, esta constante sólo devuelve el foco a ese modo tabla. Se activa mediante *F7*, Modo tabla en la barra rápida o Ficha|Modo tabla.
DataToggleDeleted	SmallInt	Se utiliza para invertir el estado de mostrar registros borrados en las tablas de dBASE.
DataToggleDeleteRecord	SmallInt	Se utiliza para invertir el estado de borrado de los registros de las tablas de dBASE.
DataToggleEdit	SmallInt	Se utiliza para invertir el estado Editar de la ficha. Ejecuta recursivamente action(DataBeginEdit) o action(DataEndEdit), según sea conveniente. Se activa mediante *F9*, Editar datos en la barra rápida o Ficha|Editar datos.
DataToggleLockRecord	SmallInt	Se utiliza para invertir el estado de bloqueo del registro actual. En realidad, sólo utiliza recursivamente action(DataLockRecord) o action(DataUnlockRecord), según sea conveniente. Los errores que se produzcan ejecutarán el método **error**.
DataUnDeleteRecord	SmallInt	En las tablas de dBASE, considerará el registro borrado anteriormente como no borrado .
DataUnlock Record	SmallInt	Se utiliza para consignar las modificaciones del registro en la tabla y (si es satisfactorio) desbloquear el registro. Los errores que se produzcan activarán el método **error**.

ActionEditCommands

Constante	Tipo de datos	Descripción
EditCommitField	SmallInt	Escribe las modificaciones del registro actual en la memoria intermedia de registro (sin salir del campo).
EditCopySelection	SmallInt	Copia un área seleccionada de texto en el Portapapeles.
EditCopyToFile	SmallInt	Activa un cuadro de diálogo para copiar la selección en un archivo.
EditCutSelection	SmallInt	Copia un área seleccionada de texto en el Portapapeles y la borra.
EditDeleteBeginLine	SmallInt	Borra desde la posición actual hasta el comienzo de la línea.
EditDeleteEndLine	SmallInt	Borra desde la posición actual hasta el final de la línea.
EditDeleteLeft	SmallInt	Borra un carácter a la izquierda. Se activa mediante *Retroceso* en modo campo.
EditDeleteLeftWord	SmallInt	Borra hasta el principio de la palabra situada a la izquierda de la posición de carácter actual. Se activa mediante *Ctrl+Retroceso*.
EditDeleteLine	SmallInt	Borra la línea en que se encuentra la posición actual.
EditDeleteRight	SmallInt	Borra un carácter a la derecha. Se activa mediante *Supr* en modo campo.
EditDeleteRightWord	SmallInt	Borra hasta el final de la palabra situada a la derecha de la posición de carácter actual.
EditDeleteSelection	SmallInt	Borra el área de texto seleccionada actualmente.
EditDeleteWord	SmallInt	Borra la palabra situada alrededor de la posición actual.
EditDropDownList	SmallInt	Se utiliza por campos de edición desplegables. Desplegará la lista de selección asociada.
EditEnterFieldView	SmallInt	Entra en el modo campo para el campo actual (permitiendo el desplazamiento dentro del campo mediante las teclas del cursor). Comienza desplazando la posición actual al final del campo y cancelando su resaltado.
EditEnterMemoView	SmallInt	Entra en el modo memo en campos memo u OLE.
EditEnterPersistFieldView	SmallInt	Entra en el modo campo persistente , lo cual implica que las teclas del cursor siempre realizan desplazamientos dentro de las posiciones de carácter dentro de un campo, incluso cuando se realiza un desplazamiento a nuevos campos.
EditExitFieldView	SmallInt	Sale del modo campo (las teclas del cursor se desplazarán de nuevo entre campos) y resalta todo el campo.
EditExitMemoView	SmallInt	Sale del modo memo en campos memo u OLE, por lo que *Intro* y *Tab* volverán a desplazarse entre campos.
EditExitPersistField View	SmallInt	Sale del modo campo persistente , por lo que las teclas del cursor se moverán entre campos.

EditHelp	SmallInt	Muestra el subsistema de ayuda. Se activa mediante *F1*.
EditInsertBlank	SmallInt	Inserta un carácter vacío en la posición actual.
EditInsertLine	SmallInt	Inserta una línea vacía en la posición actual.
EditLaunchServer	SmallInt	Utilizada sólo por campos OLE, ejecutará la aplicación servidora adecuada para el campo actual.
EditPaste	SmallInt	Pega el contenido del Portapapeles en la posición actual (sustituyendo la selección actual, i es apropriado).
EditPasteFromFile	SmallInt	Muestra un cuadro de diálogo, ermitiendo que el usuario seleccione un archivo para su inserción en la posición actual.
EditProperties	SmallInt	Muestra el menú de inspección de propiedades para el objeto dado. Actualmente, ólo los campos de imagen y los de memo formateados permiten esta acción.
EditReplace	SmallInt	Activa y desactiva el modo sobrescritura en un objeto de campo.
EditTextSearch	SmallInt	Muestra un cuadro de diálogo que permite al usuario buscar y sustituir texto dentro del campo actual.
EditToggleFieldView	SmallInt	Invierte el estado actual de modo campo. Ejecuta recursivamente action(EditEnterFieldView) o action(EditExitFieldView). Se activa mediante *F2*, todo campo en la barra rápida o Tabla|Modo campo.
EditUndoField	SmallInt	Desecha las modificaciones en el campo actual y vuelve al valor de la memoria intermedia de registro actual. Se activa mediante *Esc*.

ActionFieldCommands

Constante	Tipo de datos	Descripción
FieldBackward	SmallInt	Se utiliza para retroceder un campo en el orden de tabulación. Buscará el UIObject anterior marcado como Posición de tabulación por orden de izquierda a derecha y de arriba a abajo. Se activa mediante *Mayús+Tab*.
FieldDown	SmallInt	Se utiliza para ir al campo situado bajo el actual,tanto si se está en modo campo como si no. Se activa mediante *Alt*+ ↓.
FieldEnter	SmallInt	Se utiliza para consignar las modificaciones en un campo (si las hay) y para avanzar un campo en el orden de tabulación. Se activa mediante *Intro*.
FieldFirst	SmallInt	Se utiliza para ir al primer campo dentro de un registro. Se activa mediante *Alt+Inicio* o mediante *Inicio* (cuando no se está en modo campo).
FieldForward	SmallInt	Se utiliza para avanzar un campo en el orden de tabulación. Buscará el UIObject siguiente marcado como Posición de tabulación por orden de izquierda a derecha y de arriba a abajo. Se activa mediante *Tab*.
FieldGroupBackward	SmallInt	Se utiliza para retroceder un supergrupo de tabulación (por ejemplo, entre diferentes marcos de tabla de la misma ficha). Se activa mediante *F3*.
FieldGroupForward	SmallInt	Se utiliza para retroceder un supergrupo de tabulación (por ejemplo, entre diferentes marcos de tabla de la misma ficha). Se activa mediante *F4*.
FieldLast	SmallInt	Se utiliza para ir al último campo dentro de un registro. Se activa mediante *Alt+Fin* o mediante *Fin* (cuando no se está en modo campo).
FieldLeft	SmallInt	Se utiliza para ir al campo situado a la izquierda del campo actual.
FieldNextPage	SmallInt	Se utiliza para ir a la siguiente página secuencial de una ficha multipágina. Se activa mediante Ficha|Página|Siguiente o *Mayús+F4*.
FieldPriorPage	SmallInt	Se utiliza para ir a la página anterior de una ficha multipágina. Se activa mediante Ficha|Página|Siguiente o *Mayús+F3*.
FieldRight	SmallInt	Se utiliza para ir al campo situado a la derecha del campo actual, tanto si se está en modo campo como si no. Se activa mediante *Alt*+ →.
FieldRotate	SmallInt	Se utiliza para rotar columnas dentro de un marco de tabla. Se activa mediante *Ctrl+R*.
FieldUp	SmallInt	Se utiliza para ir al campo situado por encima del campo actual, tanto si se está en modo campo como si no. Se activa mediante *Alt*+ ↑.

ActionMoveCommands

Constante	Tipo de datos	Descripción
MoveBegin	SmallInt	En modo memo, se desplaza al principio del documento. En caso contrario, se desplaza al primer campo del primer registro de la tabla. Se activa mediante *Ctrl+Inicio*.
MoveBeginLine	SmallInt	En modo memo, se desplaza al comienzo de la línea; en caso contrario, se desplaza al primer campo del registro. Se activa mediante *Inicio*.
MoveBottom	SmallInt	En modo memo, se desplaza a la línea inferior del área de texto. En caso contrario, se desplaza al último registro de la tabla.
MoveBottomLeft	SmallInt	En modo memo, se desplaza al principio de la última línea mostrada en pantalla. Se activa mediante *Ctrl+RePág*.
MoveBottomRight	SmallInt	En modo memo, se desplaza al final de la última línea mostrada en pantalla. Se activa mediante *Ctrl+AvPág*.
MoveDown	SmallInt	Se desplaza hacia abajo según corresponda. En modo memo, baja una línea en campos multilínea. En caso contrario, va al siguiente objeto Posición de tabulación situado bajo el objeto actual. En los objetos de marco de tabla, se desplaza al registro siguiente. Se activa mediante ↓.
MoveEnd	SmallInt	En modo memo, se desplaza al final del documento; en caso contrario,se desplaza al último campo del último registro de la tabla. Se activa mediante *Ctrl+Fin*.
MoveEndLine	SmallInt	En modo memo, se desplaza al final de la línea; de lo contrario, se desplaza al último campo del registro. Se activa mediante *Fin*.
MoveLeft	SmallInt	Se desplaza hacia la izquierda según corresponda. En modo memo, se desplaza una posición de carácter a la izquierda; de lo contrario, se desplaza al siguiente objeto Posición de tabulación situado a la izquierda del objeto actual. Se activa mediante ← .
MoveLeftWord	SmallInt	En modo memo, desplaza el punto de inserción al principio de la palabra situada a la izquierda del punto de inserción actual. Se activa mediante *Ctrl+* ←.

MoveRight SmallInt Se desplaza a la derecha según corresponda. En modo memo, se desplaza una posición de carácter a la derecha; de lo contrario, se desplaza al siguiente objeto Posición de tabulación situado a la derecha del objeto actual. Se activa mediante ▪.
MoveRightWordSmallInt En modo memo,desplaza el punto de inserción al principio de la palabra situada a la derecha del punto de inserción actual. Se activa mediante *Ctrl+Ctrl+* ▪.

MoveScrollDown	SmallInt	Desplaza la imagen hacia abajo (desplazando en realidad el área de visualización hacia arriba) en la cantidad adecuada. Los campos activos se desplazan en líneas regulares de texto. En las tablas, se desplaza a un nuevo registro. En modo memo, se desplaza hacia el final del texto. El punto

		de inserción permanece en la misma línea del área de visualización a menos que la última línea del texto sea visible, en cuyo caso el punto de inserción baja una línea hasta que llegue a la última línea. Se activa mediante *Ctrl*+ ▪.
MoveScrollLeft	SmallInt	Desplaza la imagen hacia la izquierda (desplazando en realidad el área de visualización hacia la derecha) en la cantidad adecuada. Los campos activos se desplazan, aproximadamente, una posición de carácter. En las tablas, se desplaza a una nueva columna.
MoveScrollPageDown	SmallInt	Desplaza la imagen hacia abajo (desplazando en realidad el área de visualización hacia arriba) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
MoveScrollPageLeft	SmallInt	Desplaza la imagen hacia la izquierda (desplazando en realidad el área de visualización hacia la derecha) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
MoveScrollPageRight	SmallInt	Desplaza la imagen hacia la derecha (desplazando en realidad el área de visualización hacia la izquierda) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
MoveScrollPageUp	SmallInt	Desplaza la imagen hacia arriba (desplazando en realidad el área de visualización hacia abajo) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
MoveScrollRight	SmallInt	Desplaza la imagen hacia la derecha (desplazando en realidad el área de visualización hacia la izquierda) en la cantidad adecuada. Los campos activos se desplazan, aproximadamente, una posición de carácter. En las tablas, se desplaza a una nueva columna.
MoveScrollScreenDown	SmallInt	Desplaza la imagen hacia abajo (desplazando en realidad el área de visualización hacia arriba) el tamaño del área de visualización (por ejemplo, el tamaño de un campo). En modo memo, se desplaza en el documento hacia abajo un espacio igual a la altura del área de visualización. Se activa mediante *AvPág*.
MoveScrollScreenLeft	SmallInt	Desplaza la imagen hacia la izquierda (desplazando en realidad el área de visualización hacia la derecha) el tamaño del área de visualización (por ejemplo, el tamaño de un campo).
MoveScrollScreenRight	SmallInt	Desplaza la imagen hacia la derecha (desplazando en realidad el área de visualización hacia la izquierda) el tamaño del área de visualización (por ejemplo,el tamaño de un campo).
MoveScrollScreenUp	SmallInt	Desplaza la imagen hacia arriba (desplazando en realidad el área de visualización hacia abajo) el tamaño del área de visualización (por ejemplo, el tamaño de un campo). En modo memo, se desplaza en el documento hacia arriba un espacio igual a la

		altura del área de visualización. Se activa mediante *RePág*.
MoveScrollUp	SmallInt	Desplaza la imagen hacia arriba (desplazando en realidad el área de visualización hacia abajo) en la cantidad adecuada. Los campos activos se desplazan en líneas regulares de texto. En modo memo, se desplaza hacia el principio del texto. El punto de inserción permanece en la misma línea del área de visualización a menos que la primera línea del documento sea visible, en cuyo caso el punto de inserción sube una línea si puede. Se activa mediante *Ctrl*+ ▪.
MoveTop	SmallInt	En modo memo, desplaza el punto de inserción a la primera línea de texto visible en el área de visualización; en caso contrario, lo desplaza al primer registro de la tabla.
MoveTopLeft	SmallInt	En modo memo, va a la parte superior izquierda del área de visualización; en caso contrario, se desplaza al campo superior izquierdo. Se activa mediante *Ctrl+RePág*.
MoveTopRight	SmallInt	En modo memo, va a la parte superior derecha del área de visualización; en caso contrario, se desplaza al campo superior derecho. Se activa mediante *Ctrl+AvPág*.
MoveUp	SmallInt	Se desplaza hacia arriba según corresponda. En modo memo, sube una línea en campos multilínea; en caso contrario, va al siguiente objeto Posición de tabulación situado por encima del objeto actual. En los objetos de marco de tabla,va al registro anterior. Se activa mediante ▪.

ActionSelectCommands

Constante	Tipo de datos	Descripción
SelectBegin	SmallInt	En modo memo, selecciona desde la posición actual hasta el principio del documento; en caso contrario, selecciona desde la posición actual hasta el primer campo del primer registro de la tabla. Se activa mediante *Mayús+Ctrl+Inicio*.
SelectBeginLine	SmallInt	En modo memo, selecciona desde la posición actual hasta el principio de la línea; en caso contrario, selecciona desde la posición actual hasta el primer campo del registro. Se activa mediante *Mayús+Inicio*.
SelectBottom	SmallInt	En modo campo y modo memo, selecciona desde la posición actual hasta el final del área de visualización; en caso contrario, selecciona desde la posición actual hasta el último registro de la tabla.
SelectBottomLeft	SmallInt	En modo memo, selecciona desde la posición actual hasta el principio de la última línea mostrada en el área de visualización. Se activa mediante *Mayús+Ctrl+RePág*.
SelectBottomRight	SmallInt	En modo memo,selecciona desde la posición actual hasta el final de la última línea mostrada en el área de visualización. Se activa mediante *Mayús+Ctrl+AvPág*.
SelectDown	SmallInt	Selecciona hacia abajo según corresponda. En modo campo o modo memo, selecciona hacia abajo una línea en campos multilínea. En las fichas, no puede ampliarse la selección a otros campos. En los objetos de marco de tabla, selecciona hasta el registro siguiente. Se activa mediante *Mayús+* ▪.
SelectEnd	SmallInt	En modo campo o modo memo, selecciona desde la posición actual hasta el final del documento; en caso contrario, selecciona desde la posición actual hasta el último campo del último registro de la tabla. Se activa mediante *Mayús+Ctrl+Fin*.
SelectEndLine	SmallInt	En modo campo o modo memo, selecciona desde la posición actual hasta el final de la línea; en caso contrario, selecciona desde la posición actual hasta el último campo del registro. Se activa mediante *Mayús+Fin*.
SelectLeft	SmallInt	Selecciona hacia la izquierda según corresponda. En modo campo o modo memo, selecciona una posición de carácter hacia la izquierda; en caso contrario, selecciona el siguiente objeto Posición de tabulación situado a la izquierda del objeto actual. Se activa mediante *Mayús+* ←.
SelectLeftWord	SmallInt	En modo campo o modo memo, si el punto de inserción se halla entre dos palabras, selecciona la palabra situada a la izquierda del punto de inserción. Si el punto de inserción se halla dentro de una palabra, selecciona hasta el principio de esa palabra. Se activa mediante *Mayús+Ctrl+* ←.
SelectRight	SmallInt	Selecciona hacia la derecha según corresponda. En modo campo o

modo memo, selecciona una posición de carácter hacia la derecha. Se activa mediante *Mayús+* ▪.

SelectRightWord	SmallInt	En modo campo o modo memo, selecciona hasta el principio de la palabra siguiente. Si el punto de inserción precede a uno o más espacios o tabuladores, la selección sólo incluye esos espacios o tabuladores. Se activa mediante *Mayús*+Ctrl+ ▪.

SelectScrollDown SmallInt Selecciona la imagen hacia abajo (desplazando en realidad el área de visualización hacia arriba) en la cantidad adecuada. En los campos activos, se selecciona en líneas regulares de texto. En las tablas, se selecciona un nuevo registro. Se activa mediante *Mayús+Ctrl+* ▪.

SelectScrollLeft	SmallInt	Selecciona la imagen hacia la izquierda (desplazando en realidad el área de visualización hacia la derecha) en la cantidad adecuada. En los campos activos, se selecciona una posición de carácter. En las tablas, se selecciona hasta una nueva columna.
SelectScrollPageDown	SmallInt	Selecciona la imagen hacia abajo (desplazando en realidad el área de visualización hacia arriba) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
SelectScrollPageLeft	SmallInt	Selecciona la imagen hacia la izquierda (desplazando en realidad el área de visualización hacia la derecha) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
SelectScrollPageRight	SmallInt	Selecciona la imagen hacia la derecha (desplazando en realidad el área de visualización hacia la izquierda) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
SelectScrollPageUp	SmallInt	Selecciona la imagen hacia arriba (desplazando en realidad el área de visualización hacia abajo) el tamaño lógico del objeto (por ejemplo, la página completa de un documento).
SelectScrollRight	SmallInt	Selecciona la imagen hacia la derecha (desplazando en realidad el área de visualización hacia la izquierda) en la cantidad adecuada. En los campos activos, se selecciona una posición de carácter. En las tablas, se selecciona una nueva columna.
SelectScrollScreenDown	SmallInt	Selecciona la imagen hacia abajo (desplazando en realidad el área de visualización hacia arriba) el tamaño del área de visualización (por ejemplo, el tamaño de un campo). Se activa mediante *Mayús+Crtl+AvPág*.
SelectScrollScreenLeft	SmallInt	Selecciona la imagen hacia la izquierda (desplazando en realidad el área de visualización hacia la derecha) el tamaño del área de visualización (por ejemplo, el tamaño de un campo).
SelectScrollScreenRight	SmallInt	Selecciona la imagen hacia la derecha (desplazando en realidad el área de visualización hacia la izquierda) el tamaño del área de visualización (por ejemplo, el tamaño de un campo).
SelectScrollScreenUp	SmallInt	Selecciona la imagen hacia arriba (desplazando en realidad el área de visualización hacia abajo) el tamaño del área de visualización (por ejemplo, el

		tamaño de un campo). Se activa mediante *Mayús+Ctrl+RePág*.
SelectScrollUp	SmallInt	Selecciona la imagen hacia arriba (desplazando en realidad el área de visualización hacia abajo) en la cantidad adecuada. En los campos activos, se selecciona en líneas regulares de texto.
SelectSelectAll	SmallInt	Selecciona todo el campo.
SelectTop	SmallInt	En modo campo o modo memo, selecciona desde la posición actual hasta la parte superior del área de visualización; en caso contrario, selecciona desde la posición actual hasta el primer registro de la tabla.
SelectTopLeft	SmallInt	En modo campo o modo memo, selecciona desde la posición actual hasta el principio de la pantalla; en caso contrario, selecciona desde la posición actual hasta el campo superior izquierdo. Se activa mediante *Mayús+Crtl+RePág*.
SelectTopRight	SmallInt	En modo campo o modo memo, selecciona desde la posición actual hasta el final de la línea superior de la pantalla; en caso contrario, selecciona desde la posición actual hasta el campo superior derecho. Se activa mediante *Mayús+Crtl+AvPág*.
SelectUp	SmallInt	Selecciona hacia arriba según corresponda. En modo campo o modo memo, selecciona hacia arriba una línea en campos multilínea; en caso contrario, selecciona el siguiente objeto Posición de tabulación situado por encima del objeto actual. En los objetos de marco de tabla, se selecciona hasta el registro anterior. Se activa mediante *Mayús+* ▪..

ButtonStyles

Constante	Tipo de datos	Descripción
BorlandButton	SmallInt	Da a un botón el aspecto 3-D de los botones de los productos de Borland.
WindowsButton	SmallInt	Da a un botón el aspecto de los botones de otros productos Windows.

ButtonTypes

Constante	Tipo de datos	Descripción
CheckboxType	SmallInt	Muestra un botón como casilla de verificación.
PushButtonType	SmallInt	Muestra un botón como botón de comando.
RadioButtonType	SmallInt	Muestra un botón como botón de radio.

Colors

Constante	Tipo de datos	Descripción
Black	LongInt	Negro
Blue	LongInt	Azul
Brown	LongInt	Marrón
DarkBlue	LongInt	Azul oscuro
DarkCyan	LongInt	Cian oscuro
DarkGray	LongInt	Gris oscuro
DarkGreen	LongInt	Verde oscuro
DarkMagenta	LongInt	Magenta oscuro
DarkRed	LongInt	Rojo oscuro
Gray	LongInt	Gris
Green	LongInt	Verde
LightBlue	LongInt	Azul claro
Magenta	LongInt	Magenta
Red	LongInt	Rojo
Translucent	LongInt	Traslúcido
Transparent	LongInt	Transparente
White	LongInt	Blanco
Yellow	LongInt	Amarillo

CompleteDisplay

Constante	Tipo de datos	Descripción
DisplayAll	SmallInt	Especifica CompleteDisplay para todos los objetos de campo de la ficha.
DisplayCurrent	SmallInt	Especifica CompleteDisplay para el objeto de campo actual.

ErrorReasons

Constante	Tipo de datos	Descripción
ErrorCritical	SmallInt	Muestra un mensaje en un cuadro de diálogo modal.
ErrorWarning	SmallInt	Muestra un mensaje en el área de estado.

EventErrorCodes

Constante	Tipo de datos	Descripción
Can_Arrive	SmallInt	Concede permiso para acceder a un objeto.
Can_Depart	SmallInt	Concede permiso para salir de un objeto.
CanNotArrive	SmallInt	Concede permiso para acceder a un objeto (impide el desplazamiento).
CanNotDepart	SmallInt	Concede permiso para salir de un objeto (impide el desplazamiento).

ExecuteOptions

Constante	Tipo de datos	Descripción
ExeHidden	SmallInt	Oculta la ventana de aplicación y pasa la activación a otra ventana.
ExeMinimized	SmallInt	Minimiza la ventana de aplicación y activa la ventana de nivel superior en la lista del administrador de ventanas.
ExeShowMaximized	SmallInt	Activa la ventana de aplicación y la muestra como una ventana maximizada.
ExeShowMinimized	SmallInt	Activa la ventana de aplicación y la muestra minimizada (como icono).
ExeShowMinimized NoActivate	SmallInt	Muestra la aplicación como un icono. La ventana que está activa actualmente permanece activa.
ExeShowNoActivate	SmallInt	Muestra la ventana de aplicación con su tamaño y posición más recientes. La ventana activa actualmente permanece activa.
ExeShowNormal	SmallInt	Activa y muestra una ventana.

FieldDisplayTypes

Constante	Tipo de datos	Descripción
BitmapField	SmallInt	Permite que un objeto de campo muestre un mapa de bits.
CheckboxField	SmallInt	Muestra un campo como casilla de verificación.
ComboField	SmallInt	Muestra un campo como lista de edición desplegable (también llamado cuadro combinado).
EditField	SmallInt	Muestra un campo sin etiqueta.
LabeledField	SmallInt	Muestra un campo con etiqueta.
ListField	SmallInt	Muestra un cuadro de lista.
OleField	SmallInt	Permite que un campo contenga datos OLE.
RadioButtonField	SmallInt	Muestra un campo como uno o más botones de radio.

FileBrowserFileTypes

Constante	Tipo de datos	Descripción
fbASCII	LongInt	Archivos de texto.
fbAllTables	LongInt	Todos los tipos de tabla admitidos por Paradox.
fbBitmap	LongInt	Imagen de mapa de bits.
fbDBase	LongInt	Tablas de dBASE.
fbExcel	LongInt	Hojas de cálculo de Excel.
fbFiles	LongInt	Todos los archivos.
fbForm	LongInt	Fichas de Paradox.
fbGraphic	LongInt	Archivos de imagen.
fbIni	LongInt	Archivos cuya extensión sea .INI.
fbLibrary	LongInt	Bibliotecas de ObjectPAL.
fbLotus1	LongInt	Hojas de cálculo de Lotus 1-2-3 versión 1.
fbLotus2	LongInt	Hojas de cálculo de Lotus 1-2-3 versión 2.
fbParadox	LongInt	Tablas de Paradox.
fbQuattro	LongInt	Hojas de cálculo de Quattro.
fbQuattroPro	LongInt	Hojas de cálculo de Quattro Pro.
fbQuattroProWindows	LongInt	Cuadernos de Quattro Pro para Windows.
fbQuery	LongInt	Archivos de consulta.
fbReport	LongInt	Informes de Paradox.
fbScript	LongInt	Macros de ObjectPAL.
fbTable	LongInt	Tablas de Paradox.
fbTableView	LongInt	Archivos de modo tabla de Paradox.
fbText	LongInt	Todos los archivos de texto.

FontAttributes

Constante	Tipo de datos	Descripción
FontAttribBold	SmallInt	Ejemplo: negrita
FontAttribItalic	SmallInt	Ejemplo: *cursiva*
FontAttribNormal	SmallInt	Ejemplo: normal
FontAttribStrikeOut	SmallInt	Ejemplo:
FontAttribUnderline	SmallInt	Ejemplo: subrayado

FrameStyles

Constante	Tipo de datos	Descripción
DashDotDotFrame	SmallInt	Secuencia repetida de un guión seguido por dos puntos.
DashDotFrame	SmallInt	Secuencia repetida de un guión seguido por un punto.
DashedFrame	SmallInt	Secuencia repetida de guiones.
DottedFrame	SmallInt	Secuencia repetida de puntos.
DoubleFrame	SmallInt	Dos cuadros concéntricos.
Inside3DFrame	SmallInt	El marco aparece insertado en la ficha.
NoFrame	SmallInt	Sin marco.
Outside3DFrame	SmallInt	El marco aparece sacado de la ficha.
ShadowFrame	SmallInt	Sombra.
SolidFrame	SmallInt	Un solo cuadro sólido (sin guiones ni puntos).
WideInsideDoubleFrame	SmallInt	Dos cuadros concéntricos; el interior es ancho.
WideOutsideDoubleFrame	SmallInt	Dos cuadros concéntricos; el exterior es ancho.

General

Constante	Tipo de datos	Descripción
No	Logical	False
Off	Logical	False
On	Logical	True
PI	Number	3,14159265358979323846
Yes	Logical	True

GraphBindTypes

Constante	Tipo de datos	Descripción
Graph1DSummary	SmallInt	Especifica un gráfico de resumen unidimensional. Activa los operadores de resumen.
Graph2DSummary	SmallInt	Especifica un gráfico de resumen bidimensional. Activa los operadores de resumen y la especificación por grupos.
GraphTabular	SmallInt	Especifica un gráfico tabular (por defecto).

GraphLabelFormats

Constante	Tipo de datos	Descripción
GraphHideY	SmallInt	Oculta el valor Y (sólo en gráficos circulares y de columna 2-D y 3-D).
GraphPercent	SmallInt	Muestra el valor Y como un porcentaje (sólo en gráficos circulares y de columna 2-D y 3-D).
GraphShowY	SmallInt	Muestra el valor Y en las unidades empleadas en la tabla (sólo en gráficos circulares y de columna 2-D y 3-D).

GraphLegendPosition

Constante	Tipo de datos	Descripción
LegendCenter	SmallInt	Muestra la leyenda centrada bajo el gráfico.
LegendLeft	SmallInt	Muestra la leyenda a la izquierda del gráfico.

GraphMarkers

Constante	Tipo de datos	Descripción
MarkerBoxedCross	SmallInt	El símbolo es un cuadro con una cruz en su interior.
MarkerBoxed_Plus	SmallInt	El símbolo es un cuadro con un signo más en su interior.
MarkerCross	SmallInt	El símbolo es una cruz.
MarkerFilledBox	SmallInt	El símbolo es un cuadro relleno.
MarkerFilledCircle	SmallInt	El símbolo es un círculo relleno.
MarkerFilledDownTriangle	SmallInt	El símbolo es un triángulo relleno que señala hacia abajo.
MarkerFilledTriangle	SmallInt	El símbolo es un triángulo relleno que señala hacia arriba.
MarkerFilledTriangles	SmallInt	El símbolo son dos triángulos rellenos señalando cada uno al otro.
MarkerHollowBox	SmallInt	El símbolo es un cuadro hueco (sin rellenar).
MarkerHollowCircle	SmallInt	El símbolo es un círculo hueco.
MarkerHollowDown Triangle	SmallInt	El símbolo es un triángulo hueco que señala hacia abajo.
MarkerHollowTriangle	SmallInt	El símbolo es un triángulo hueco que señala hacia arriba.
MarkerHollowTriangles	SmallInt	El símbolo son dos triángulos huecos señalando cada uno al otro.
MarkerHorizontalLine	SmallInt	El símbolo es una línea horizontal.
MarkerPlus	SmallInt	El símbolo es un signo más.
MarkerVerticalLine	SmallInt	El símbolo es una línea vertical.

GraphTypeOverRide

Constante	Tipo de datos	Descripción
GraphArea	SmallInt	Muestra una serie especificada como un gráfico de áreas.
GraphBar	SmallInt	Muestra una serie especificada como un gráfico de barras.
GraphDefault	SmallInt	Muestra una serie especificada en el tipo de gráfico por defecto.
GraphDefault	SmallInt	Muestra una serie especificada como un gráfico de líneas.

GraphTypes

Constante	Tipo de datos	Descripción
Graph2DArea	SmallInt	Gráfico de áreas bidimensional.
Graph2DBar	SmallInt	Gráfico de barras bidimensional.
Graph2DColumns	SmallInt	Gráfico de columnas bidimensional.
Graph2DLine	SmallInt	Gráfico de líneas bidimensional.
Graph2DPie	SmallInt	Gráfico circular bidimensional.
Graph2DRotatedBar	SmallInt	Gráfico rotado de barras bidimensional.
Graph2DStackedBar	SmallInt	Gráfico de barras apiladas bidimensional.
Graph3DArea	SmallInt	Gráfico de áreas tridimensional.
Graph3DBar	SmallInt	Gráfico de barras tridimensional.
Graph3DColumns	SmallInt	Gráfico de columnas tridimensional.
Graph3DPie	SmallInt	Gráfico circular tridimensional.
Graph3DRibbon	SmallInt	Gráfico de cintas tridimensional.
Graph3DRotatedBar	SmallInt	Gráfico rotado de barras tridimensional.
Graph3DStackedBar	SmallInt	Gráfico de barras apiladas tridimensional.
Graph3DStep	SmallInt	Gráfico de escalera tridimensional.
Graph3DSurface	SmallInt	Gráfico de superficie tridimensional.
GraphXY	SmallInt	Gráfico XY.

GraphicMagnification

Constante	Tipo de datos	Descripción
Magnify100	SmallInt	Muestra el gráfico con su tamaño real.
Magnify200	SmallInt	Muestra el gráfico al doble de su tamaño real.
Magnify25	SmallInt	Muestra el gráfico a un cuarto de su tamaño real.
Magnify400	SmallInt	Muestra el gráfico a cuatro veces su tamaño real.
Magnify50	SmallInt	Muestra el gráfico a la mitad de su tamaño real.
MagnifyBestFit	SmallInt	Redimensiona el gráfico lo necesario para que se ajuste en el marco.

IdRanges

Constante	Tipo de datos	Descripción
UserAction	SmallInt	Valor mínimo para una constante de acción definida por el usuario.
UserActionMax	SmallInt	Valor máximo para una constante de acción definida por el usuario.
UserError	SmallInt	Valor mínimo para una constante de error definida por el usuario.
UserErrorMax	SmallInt	Valor máximo para una constante de error definida por el usuario.
UserMenu	SmallInt	Valor mínimo para una constante de ID de menú definida por el usuario.
UserMenuMax	SmallInt	Valor máximo para una constante de ID de menú definida por el usuario.

KeyBoardStates

Constante	Tipo de datos	Descripción
Alt	SmallInt	Pulsación de la tecla *Alt*.
Control	SmallInt	Pulsación de la tecla *Ctrl*.
LeftButton	SmallInt	Clic del botón izquierdo del ratón.
RightButton	SmallInt	Clic del botón derecho del ratón.
Shift	SmallInt	Pulsación de la tecla *Mayús*.

Keyboard

Constante	Tipo de datos	Descripción
VK_ADD	SmallInt	Tecla de suma
VK_BACK	SmallInt	Tecla *Retroceso*
VK_CANCEL	SmallInt	Se utiliza para el proceso de Control-Pausa
VK_CAPITAL	SmallInt	Tecla *Bloq Mayús*
VK_CLEAR	SmallInt	Tecla Borrar
VK_CONTROL	SmallInt	Tecla *Ctrl*
VK_DECIMAL	SmallInt	Tecla de decimal
VK_DELETE	SmallInt	Tecla *Supr*
VK_DIVIDE	SmallInt	Tecla de división
VK_DOWN	SmallInt	Tecla ▪
VK_END	SmallInt	Tecla *Fin*
VK_ESCAPE	SmallInt	Tecla *Escape*
VK_EXECUTE	SmallInt	Tecla de ejecución
VK_F1	SmallInt	Tecla *F1*
VK_F10	SmallInt	Tecla *F10*
VK_F11	SmallInt	Tecla *F11*
VK_F12	SmallInt	Tecla *F12*
VK_F13	SmallInt	Tecla *F13*
VK_F14	SmallInt	Tecla *F14*
VK_F15	SmallInt	Tecla *F15*
VK_F16	SmallInt	Tecla *F16*
VK_F2	SmallInt	Tecla *F2*
VK_F3	SmallInt	Tecla *F3*
VK_F4	SmallInt	Tecla *F4*
VK_F5	SmallInt	Tecla *F5*
VK_F6	SmallInt	Tecla *F6*
VK_F7	SmallInt	Tecla *F7*
VK_F8	SmallInt	Tecla *F8*
VK_F9	SmallInt	Tecla *F9*
VK_HELP	SmallInt	Tecla *Ayuda*
VK_HOME	SmallInt	Tecla *Inicio*
VK_INSERT	SmallInt	Tecla *Insert*
VK_LBUTTON	SmallInt	Botón izquierdo del ratón
VK_LEFT	SmallInt	Tecla ▪
VK_MBUTTON	SmallInt	Botón central del ratón (ratón de 3 botones)
VK_MENU	SmallInt	Tecla *Menú*
VK_MULTIPLY	SmallInt	Tecla de multiplicación
VK_NEXT	SmallInt	Tecla *Av Pág*
VK_NUMLOCK	SmallInt	Tecla *Bloq Num*
VK_NUMPAD0	SmallInt	Tecla *0* del teclado numérico
VK_NUMPAD1	SmallInt	Tecla *1* del teclado numérico
VK_NUMPAD2	SmallInt	Tecla *2* del teclado numérico
VK_NUMPAD3	SmallInt	Tecla *3* del teclado numérico
VK_NUMPAD4	SmallInt	Tecla *4* del teclado numérico
VK_NUMPAD5	SmallInt	Tecla *5* del teclado numérico

VK_NUMPAD6	SmallInt	Tecla *6* del teclado numérico
VK_NUMPAD7	SmallInt	Tecla *7* del teclado numérico
VK_NUMPAD8	SmallInt	Tecla *8* del teclado numérico
VK_NUMPAD9	SmallInt	Tecla *9* del teclado numérico
VK_PAUSE	SmallInt	Tecla *Pausa*
VK_PRINT	SmallInt	Específica de OEM
VK_PRIOR	SmallInt	Tecla *Re Pág*
VK_RBUTTON	SmallInt	Botón derecho del ratón
VK_RETURN	SmallInt	Tecla *Retorno*
VK_RIGHT	SmallInt	Tecla ▪
VK_SELECT	SmallInt	Tecla de selección
VK_SEPARATOR	SmallInt	Tecla de separador
VK_SHIFT	SmallInt	Tecla *Mayús*
VK_SNAPSHOT	SmallInt	Tecla Impr Pant para Windows 3.0 y posterior
VK_SPACE	SmallInt	Espacio
VK_SUBTRACT	SmallInt	Tecla de resta
VK_TAB	SmallInt	Tecla *Tab*
VK_UP	SmallInt	Tecla ▪.

LibraryScope

Constante	Tipo de datos	Descripción
GlobalToDesktop	SmallInt	Hace que las variables de una biblioteca de ObjectPAL estén disponibles para una o más fichas.
PrivateToForm	SmallInt	Hace que las variables de una biblioteca de ObjectPAL estén disponibles para una ficha sólo.

LineEnds

Constante	Tipo de datos	Descripción
ArrowBothEnds	SmallInt	Añade flechas en ambos extremos de una línea (sólo si LineType = StraightLine)
ArrowOneEnd	SmallInt	Añade una flecha en el extremo final de una línea (sólo si LineType = StraightLine)
NoArrowEnd	SmallInt	Muestra una línea sin flechas en ninguno de sus extremos.

LineStyles

Constante	Tipo de datos	Descripción
DashDotDotLine	SmallInt	Secuencia repetida de un guión seguido por dos puntos.
DashDotLine	SmallInt	Secuencia repetida de un guión seguido por un punto.
DashedLine	SmallInt	Secuencia repetida de guiones.
DottedLine	SmallInt	Secuencia repetida de puntos.
NoLine	SmallInt	Sin línea.
SolidLine	SmallInt	Línea continua.

LineThickness

Constante	Tipo de datos	Descripción
LWidth10Points	SmallInt	Especifica un grosor de 10 puntos de impresora.
LWidth1Point	SmallInt	Especifica un grosor de 1 punto de impresora.
LWidth2Points	SmallInt	Especifica un grosor de 2 puntos de impresora.
LWidth3Points	SmallInt	Especifica un grosor de 3 puntos de impresora.
LWidth6Points	SmallInt	Especifica un grosor de 6 puntos de impresora.
LWidthHairline	SmallInt	Especifica una línea muy fina.
LWidthHalfPoint	SmallInt	Especifica un grosor de medio punto de impresora.

LineTypes

Constante	Tipo de datos	Descripción
CurvedLine	SmallInt	Especifica una línea curva (elíptica).
StraightLine	SmallInt	Especifica una línea recta.

MenuChoiceAttributes

Constante	Tipo de datos	Descripción
MenuChecked	SmallInt	Inserta una marca de verificación antes de la opción de menú.
MenuDisabled	SmallInt	La opción de menú no puede seleccionarse. El menú permanece abierto.
MenuEnabled	SmallInt	La opción de menú puede seleccionarse. El menú se cierra.
MenuGrayed	SmallInt	La opción de menú se muestra en caracteres grises (atenuada).
MenuHilited	SmallInt	La opción de menú está resaltada.
MenuNotChecked	SmallInt	Muestra la opción de menú sin marca de verificación.
MenuNotGrayed	SmallInt	Muestra la opción de menú normalmente (sin atenuar).
MenuNotHilited	SmallInt	Muestra la opción de menú sin barra de selección.

MenuCommands

Constante	Tipo de datos	Descripción
MenuCanClose	SmallInt	Solicita permiso para continuar después de elegir Cerrar en el menú Control.
MenuControlClose	SmallInt	Equivale a elegir Cerrar en el menú Control.
MenuControlKeyMenu	SmallInt	El menú Control se activó por la pulsación de una tecla.
MenuControlMaximize	SmallInt	Equivale a elegir Maximizar en el menú Control.
MenuControlMinimize	SmallInt	Equivale a elegir Minimizar en el menú Control.
MenuControlMouseMenu	SmallInt	El menú Control se activó por un clic del ratón.
MenuControlMove	SmallInt	Equivale a elegir Mover en el menú Control.
MenuControlNextWindow	SmallInt	Equivale a elegir Ventana siguiente en el menú Control.
MenuControlPrevWindow	SmallInt	Equivale a elegir Ventana anterior en el menú Control.
MenuControlRestore	SmallInt	Equivale a elegir Restaurar en el menú Control.
MenuControlSize	SmallInt	Equivale a elegir Tamaño en el menú Control.
MenuEditCopy	SmallInt	Edición|Copiar
MenuEditCopyTo	SmallInt	Edición|Copiar en
MenuEditCut	SmallInt	Edición|Cortar
MenuEditDelete	SmallInt	Edición|Eliminar
MenuEditPaste	SmallInt	Edición|Pegar
MenuEditPasteFrom	SmallInt	Edición|Pegar desde
MenuEditSearchText	SmallInt	Edición|Buscar texto
MenuEditUndo	SmallInt	Edición|Deshacer
MenuFileAliases	SmallInt	Archivo|Alias
MenuFileExit	SmallInt	Archivo|Salir
MenuFileExport	SmallInt	Archivo|Utilidades|Exportar
MenuFileImport	SmallInt	Archivo|Utilidades|Importar
MenuFileMultiBlankZero	SmallInt	Archivo|Sistema|Blancos como ceros
MenuFileMultiUser AutoRefresh	SmallInt	Archivo|Sistema|Autoactualizar
MenuFileMultiUserDrivers	SmallInt	Archivo|Sistema|Tablas configuración
MenuFileMultiUserLock	SmallInt	Archivo|Multiusuario|Establecer cloqueos
MenuFileMultiUser LockInfo	SmallInt	Archivo|Multiusuario|Ver bloqueos
MenuFileMultiUserRetry	SmallInt	Archivo|Multiusuario|Establecer reintentos
MenuFileMultiUser UserName	SmallInt	Archivo|Multiusuario|Nombre de usuario
MenuFileMultiUserWho	SmallInt	Archivo|Multiusuario|Usuarios
MenuFilePrint	SmallInt	Archivo|Imprimir
MenuFilePrinterSetup	SmallInt	Archivo|Impresora
MenuFilePrivateDir	SmallInt	Archivo|Directorio personal
MenuFileTableAdd	SmallInt	Archivo|Utilidades|Añadir
MenuFileTableCopy	SmallInt	Archivo|Utilidades|Copiar
MenuFileTableDelete	SmallInt	Archivo|Utilidades|Eliminar
MenuFileTableEmpty	SmallInt	Archivo|Utilidades|Vaciar

MenuFileTableInfoStructure	SmallInt	Archivo|Utilidades|Estructura
MenuFileTablePasswords	SmallInt	Archivo|Utilidades|Contraseñas
MenuFileTableRename	SmallInt	Archivo|Utilidades|Renombrar
MenuFileTableRestructure	SmallInt	Archivo|Utilidades|Restructurar
MenuFileTableSort	SmallInt	Archivo|Utilidades|Ordenar
MenuFileTableSubtract	SmallInt	Archivo|Utilidades|Extraer
MenuFileWorkingDir	SmallInt	Archivo|Directorio trabajo
MenuFolderOpen	SmallInt	Archivo|Abrir|Carpeta
MenuFormDesign	SmallInt	Ficha|Diseño
MenuFormEditData	SmallInt	Ficha|Editar datos
MenuFormFieldView	SmallInt	Ficha|Modo campo
MenuFormNew	SmallInt	Archivo|Nuevo|Ficha
MenuFormOpen	SmallInt	Archivo|Abrir|Ficha
MenuFormOrderRange	SmallInt	Ficha|Orden/Rango
MenuFormPageFirst	SmallInt	Ficha|Página|Primera
MenuFormPageGoto	SmallInt	Ficha|Página|Ir a
MenuFormPageLast	SmallInt	Ficha|Página|Ultima
MenuFormPageNext	SmallInt	Ficha|Página|Siguiente
MenuFormPagePrevious	SmallInt	Ficha|Página|Anterior
MenuFormShowDeleted	SmallInt	Ficha|Mostrar borrados
MenuFormTableView	SmallInt	Ficha|Modo tabla
MenuHelpAbout	SmallInt	Ayuda|Acerca de
MenuHelpContents	SmallInt	Ayuda|Contenido
MenuHelpKeyboard	SmallInt	Ayuda|Teclado
MenuHelpSpeedBar	SmallInt	Ayuda|Barra rápida
MenuHelpSupport	SmallInt	Ayuda|Información de soporte
MenuHelpUsingHelp	SmallInt	Ayuda|Uso de la Ayuda
MenuInit	SmallInt	Se genera por el clic sobre una opción de menú
MenuLibraryNew	SmallInt	Archivo|Nuevo|Biblioteca
MenuLibraryOpen	SmallInt	Archivo|Abrir|Biblioteca
MenuPasteLink	SmallInt	Edición|Pegar vínculo
MenuPropertiesCurrent	SmallInt	Propiedad|Objeto actual
MenuPropertiesDesigner	SmallInt	Propiedad|Diseño
MenuPropertiesDesktop	SmallInt	Propiedad|Escritorio
MenuPropertiesExpandedRuler	SmallInt	Propiedad|Regla expandida
MenuPropertiesFormRestoreDefaults	SmallInt	Propiedad|Parámetros defecto|Restablecer
MenuPropertiesFormSaveDefaults	SmallInt	Propiedad|Parámetros defecto|Guardar
MenuPropertiesHorizontalRuler	SmallInt	Propiedad|Regla horizontal
MenuPropertiesVerticalRuler	SmallInt	Propiedad|Regla vertical
MenuPropertiesZoom100	SmallInt	Propiedad|Zoom|100%
MenuPropertiesZoom200	SmallInt	Propiedad|Zoom|200%
MenuPropertiesZoom25	SmallInt	Propiedad|Zoom|25%

MenuPropertiesZoom400	SmallInt	Propiedad|Zoom|400%
MenuPropertiesZoom50	SmallInt	Propiedad|Zoom|50%
MenuPropertiesZoom BestFit	SmallInt	Propiedad|Zoom|Ajuste automático
MenuPropertiesZoom FitHeight	SmallInt	Propiedad|Zoom|Ajustar altura
MenuPropertiesZoom FitWidth	SmallInt	Propiedad|Zoom|Ajustar anchura
MenuQueryNew	SmallInt	Archivo|Nuevo|Consulta
MenuQueryOpen	SmallInt	Archivo|Abrir|Consulta
MenuRecordCancel	SmallInt	Registro|Cancelar cambios
MenuRecordDelete	SmallInt	Registro|Eliminar
MenuRecordFast Backward	SmallInt	Registro|Grupo anterior
MenuRecordFastForward	SmallInt	Registro|Grupo siguiente
MenuRecordFirst	SmallInt	Registro|Primero
MenuRecordInsert	SmallInt	Registro|Insertar
MenuRecordLast	SmallInt	Registro|Ultimo
MenuRecordLocateNext	SmallInt	Registro|Buscar siguiente
MenuRecordLocate RecordNumber	SmallInt	Registro|Buscar|Nº registro
MenuRecordLocate SearchAndReplace	SmallInt	Registro|Buscar|Y sustituir
MenuRecordLocateValue	SmallInt	Registro|Buscar|Valor
MenuRecordLock	SmallInt	Registro|Bloquear
MenuRecordLookup	SmallInt	Registro|Tabla de referencia
MenuRecordMove	SmallInt	Registro|Mover grupo
MenuRecordNext	SmallInt	Registro|Siguiente
MenuRecordPost	SmallInt	Registro|Mantener bloqueo
MenuRecordPrevious	SmallInt	Registro|Anterior
MenuReportNew	SmallInt	Archivo|Nuevo|Informe
MenuReportOpen	SmallInt	Archivo|Abrir|Informe
MenuSave	SmallInt	Archivo|Guardar
MenuScriptNew	SmallInt	Archivo|Nuevo|Macro
MenuScriptOpen	SmallInt	Archivo|Abrir|Macro
MenuSelectAll	SmallInt	Edición|Seleccionar todo
MenuTableNew	SmallInt	Archivo|Nuevo|Tabla
MenuTableOpen	SmallInt	Archivo|Abrir|Tabla
MenuWindow ArrangeIcons	SmallInt	Ventana|Redistribuir iconos
MenuWindowCascade	SmallInt	Ventana|Cascada
MenuWindowCloseAll	SmallInt	Ventana|Cerrar todas
MenuWindowTile	SmallInt	Ventana|Mosaico

MenuReasons

Constante	Tipo de datos	Descripción
MenuControl	SmallInt	Se activa mediante la elección de una opción en el menú de control.
MenuDesktop	SmallInt	Se activa mediante la elección de una opción en un menú estándar de Paradox.
enuNormal	SmallInt	Se activa mediante la elección de una opción en un menú personalizado de ObjectPAL o mediante el clic sobre un botón de la barra rápida.

MouseShapes

Constante	Tipo de datos	Descripción
MouseArrow	LongInt	Puntero de flecha estándar.
MouseCross	LongInt	El puntero es una cruz.
MouseIBeam	LongInt	El puntero es vertical (cursor de inserción de texto).
MouseUpArrow	LongInt	El puntero es una flecha que apunta hacia arriba.
MouseWait	LongInt	El puntero es un reloj de arena.

MoveReasons

Constante	Tipo de datos	Descripción
PalMove	SmallInt	Provocado por una sentencia de ObjectPAL.
RefreshMove	SmallInt	Provocado cuando los datos se actualizan,por ejemplo,por el desplazamiento por una tabla.
ShutDownMove	SmallInt	Provocado cuando se cierra la ficha.
StartupMove	SmallInt	Provocado cuando se abre la ficha.
UserMove	SmallInt	Provocado por el usuario.

PatternStyles

Constante	Tipo de datos
BricksPattern	SmallInt
CrosshatchPattern	SmallInt
DiagonalCrosshatchPattern	SmallInt
DottedLinePattern	SmallInt
EmptyPattern	SmallInt
FuzzyStripesDownPattern	SmallInt
HeavyDotPattern	SmallInt
HorizontalLinesPattern	SmallInt
LatticePattern	SmallInt
LeftDiagonalLinesPattern	SmallInt
LightDotPattern	SmallInt
MaximumDotPattern	SmallInt
MediumDotPattern	SmallInt
RightDiagonalLinePattern	SmallInt
ScalesPattern	SmallInt
StaggeredDashPattern	SmallInt
ThickHorizontalLinePattern	SmallInt
ThickStripesDownPattern	SmallInt
ThickStripesUpPattern	SmallInt
ThickVerticalLinesPattern	SmallInt
VerticalLinesPattern	SmallInt
VeryHeavyDotPattern	SmallInt
WeavePattern	SmallInt
ZigZagPattern	SmallInt

RasterOperations

Constante	Tipo de datos	Descripción
MergePaint	LongInt	Invierte la imagen fuente y la combina con el destino mediante el operador booleano OR.
NotSourceCopy	LongInt	Invierte la imagen fuente y la copia al destino.
NotSourceErase	LongInt	Combina la imagen fuente y el destino e invierte el resultado mediante el operador booleano OR.
SourceAnd	LongInt	Combina la imagen fuente y el destino mediante el operador booleano AND.
SourceCopy	LongInt	Copia una imagen fuente no modificada al destino.
SourceErase	LongInt	Invierte el destino y lo combina con la imagen fuente mediante el operador booleano AND.
SourceInvert	LongInt	Combina la imagen fuente y el destino mediante el operador booleano XOR.
SourcePaint	LongInt	Combina la imagen fuente y el destino mediante el operador booleano OR.

ReportOrientation

ReportPrintPanel

Constante	Tipo de datos	Descripción
PrintFromCopy	SmallInt	Imprime el informe a partir de copias de las tablas del modelo de datos del informe.
PrintLock	SmallInt	Bloquea tablas en el modelo de datos del informe antes de su impresión.
PrintNoLock	SmallInt	Imprime sin bloquear las tablas del modelo de tabla del informe.
PrintRestart	SmallInt	Reinicia el trabajo de impresión cuando los datos cambian en las tablas del modelo de datos del informe.
PrintReturn	SmallInt	Cancela el trabajo de impresión cuando los datos cambian en las tablas del modelo de datos del informe.

Status Reasons

Constante	Tipo de datos	Descripción
ModeWindow1	SmallInt	El área de la barra de estado situada la segunda por la izquierda.
ModeWindow2	SmallInt	El área de la barra de estado situada la tercera por la izquierda.
ModeWindow3	SmallInt	El área de la barra de estado situada más a la derecha.
StatusWindow	SmallInt	El área de la barra de estado situada más a la izquierda (y la más grande).

TableFrameStyles

Constante	Tipo de datos	Descripción
tf3D	SmallInt	El marco de tabla tiene un marco 3D.
tfDoubleLine	SmallInt	El marco de tabla tiene un marco de cuadro doble.
tfNoGrid	SmallInt	EL marco de tabla no tiene cuadrícula.
tfSingleLine	SmallInt	El marco de tabla tiene un marco de cuadro.
tfTripleLine	SmallInt	El marco de tabla tiene un marco de cuadro triple.

TextAlignment

Constante	Tipo de datos	Descripción
TextAlignBotton	SamllInt	Se alinea la parte inferior del texto (sólo ventana de tabla)
TextAlignCenter	SmallInt	El texto se centra en horizontal.
TextAlignJustify	SmallInt	El texto se justifica a la izquierda y a la derecha (no se aplica a la ventana de tabla).
TextAlignLeft	SmallInt	El texto se justifica a la izquierda.
TextAlignRight	SmallInt	El texto se justifica a la derecha.
TextAlignTop	SmallInt	Se alinea la parte superior del texto (sólo ventana de tabla)
TextAlignVCenter	SmallInt	El texto se centra en vertical (sólo ventana de tabla).

TextDesignSizing

Constante	Tipo de datos	Descripción
TextFixedSized	SmallInt	El cuadro de texto no cambia de tamaño.
TextGrowOnly	SmallInt	El cuadro de texto aumenta para albergar texto.
TextSizeToFit	SmallInt	El cuadro de texto aumenta o disminuye según sea necesario para albergar texto.

TextSpacing

Constante	Tipo de datos	Descripción
TextDoubleSpacing	SmallInt	2 líneas.
TextDoubleSpacing2	SmallInt	2,5 líneas.
TextSingleSpacing	SmallInt	1 línea.
TextSingleSpacing2	SmallInt	1,5 líneas.
TextTripleSpacing	SmallInt	3 líneas.

UIObjectTypes

Constante	Tipo de datos	Descripción
BoxTool	SmallInt	Crea un cuadro.
ButtonTool	SmallInt	Crea un botón.
ChartTool	SmallInt	Crea un gráfico.
EllipseTool	SmallInt	Crea una elipse.
FieldTool	SmallInt	Crea un campo.
GraphicTool	SmallInt	Crea un objeto de imagen.
LineTool	SmallInt	Crea una línea.
OleTool	SmallInt	Crea un objeto OLE.
RecordTool	SmallInt	Crea un registro.
TableFrameTool	SmallInt	Crea un marco de tabla.
TextTool	SmallInt	Crea un cuadro de texto.
XtabTool	SmallInt	Crea un objeto cartesiano.

ValueReasons

Constante	Tipo de datos	Descripción
EditValue	SmallInt	Se ha activado el método estándar **newValue** de un campo de botón de radio, lista o lista de edición desplegable (por ejemplo,accediendo a un botón de radio o eligiendo un elemento de una lista), pero no se ha consignado el valor del campo (por ejemplo,saliendo del campo).
FieldValue	SmallInt	Se ha activado el método estándar **newValue**, y el valor se ha consignado.
StartupValue	SmallInt	Se ha activado el método estándar **newValue** porque la ficha se ha abierto.

WindowStyles

Constante	Tipo de datos	Descripción
WinDefaultCoordinate	LongInt	Muestra una ventana con su tamaño y posición por defecto.
WinStyleBorder	LongInt	Especifica un borde con tamaño.
WinStyleControlMenu	LongInt	Especifica un menú de control del sistema.
WinStyleDefault	LongInt	Especifica los atributos de visualización por defecto.
WinStyleDialog	LongInt	Especifica los atributos de los cuadros de diálogo.
WinStyleDialogFrame	LongInt	Especifica un marco de cuadro de diálogo.
WinStyleHScroll	LongInt	Especifica una barra de desplazamiento horizontal.
WinStyleHidden	LongInt	Convierte una ventana en invisible.
WinStyleMaximize	LongInt	Muestra una ventana con su tamaño máximo.
WinStyleMaximizeButton	LongInt	Especifica un botón Maximizar.
WinStyleMinimize	LongInt	Muestra una ventana como un icono (minimizada).
WinStyleMinimizeButton	LongInt	Especifica un botón Minimizar.
WinStyleModal	LongInt	Convierte una ventana en modal.
WinStyleThickFrame	LongInt	Especifica una franja.
WinStyleTitleBar	LongInt	Especifica una barra de título.
WinStyleVScroll	LongInt	Especifica una barra de desplazamiento vertical.

Elementos básicos del lenguaje

Este capítulo presenta los elementos estructurales fundamentales de ObjectPAL. La mayoría de estos elementos no están asociados con tipos de objetos específicos funcionan para todos los tipos de objetos . Estos elementos pueden utilizarse para asignar valores, activar funciones desde los DLL, construir estructuras de control como bucles **if**...**then**...**else**...**endIf**, bucles **while**...**endWhile** y estructuras **switch**...**case**...**endSwitch**. También es posible declarar métodos, procedimientos, constantes, variables y tipos de datos.

Los elementos básicos del lenguaje son:

= (equals)	iif	switch
const	loop	try
disableDefault	method	type
doDefault	passEvent	uses
enableDefault	proc	var
for	quitLoop	while
forEach	return	
if	scan	

=

Palabra clave

Sintaxis **especificaciónElemento** = *expresión*

Descripción La sentencia **=** asigna el valor de *expresión* a *especificaciónElemento*. Cualquier valor anterior contenido en *especificaciónElemento* se pierde. Cuando se asigna un valor a *especificaciónElemento*, la información de *especificaciónElemento* puede incluir el recorrido de contenedores.

En una sentencia que contiene más de una sentencia **=**, el primer **=** realiza la asignación, mientras que los demás comparan dos valores o expresiones.

Cuando se utiliza **=** con los UIObject, se asigna el valor de un UIObject a otro UIObject. Por ejemplo, supóngase que una ficha contiene dos campos, *campoUno* y *campoDos*. La sentencia siguiente copia el valor de *campoDos* a *campoUno*.

```
campoUno = campoDos ; campoUno obtiene el valor de campoDos
```

También es posible utilizar **=** con variables de UIObject. ObjectPAL utiliza **attach** de la misma forma que C y Pascal emplean los punteros. Por ejemplo,

```
var iu UIObject endVar
iu.attach(campoUno) ; hace que iu "apunte a" campoUno
iu.view() ; muestra el valor de iu (el mismo que campoUno) en
un cuadro
    ; de diálogo.
iu = campoDos ; iu toma el valor de campoDos (también cambia el
valor
    ; de campoUno)
iu.view() ; muestra el valor de iu (el mismo que campoDos) en
un cuadro
    ; de diálogo
iu.color = Red ; cambia a rojo el color de iu y por lo tanto el
color de campoUno
```

La sentencia siguiente asigna a *iu* todo lo que ObjectPAL sabe sobre *campoUno*:

```
iu.attach(campoUno)
```

En contraste, la sentencia siguiente sólo asigna el valor de *campoDos* a *iu* (y a *campoUno*):

```
iu = campoDos
```

Ejemplo

```
var
   x AnyType
   matriz Array[5] AnyType
   w Logical
   y, z SmallInt
   iuUno, iuDos UIObject
endVar
```

```
x = 5,14    ; x toma el valor 5,14 (es de tipo Number)
matriz[1] = "Hola"    ; el elemento 1 de matriz toma el valor
"Hola" (cadena)
y = 5   ; y toma el valor 5
z = 12    ; z toma el valor 12
x = "zoo"    ; x toma un nuevo valor : la cadena "zoo"
miFicha.micampo = y + z    ; el campo micampo toma el resultado
de y + z campototal = campototaltemporal
grancaja.grancirculo.minicaja.minicirculo.color = Blue
; asignamos el color Blue como característico de minicirculo

; el primer signo = asigna un valor, los demás permiten
comparar
w = (y = z)    ; w toma el valor verdadero cuando y es igual a
z,
<|>              ; en cualquier otro caso w toma el valor falso
iuUno.attach(campoUno)      ; iuUno "apunta" a campoUno
iuDos = iuUno ; asigna a iuDos el valor de iuUno
```

Vea también Containers
Properties
Variables

const

Palabra clave Declara constantes.

Tipo

Sintaxis **const**

nombreConstante = *tipoDatos***(***valor*)|*valor*

endConst

Descripción **const** declara uno o más valores de constante, donde *tipoDatos*, si se incluye, especifica el tipo de datos de la constante. Si se omite *tipoDatos*, el tipo de datos se deduce del valor como un LongInt, un Number, un SmallInt o un String.

Ejemplo

```
const
   a = -1000                ; tipo smallInt, inferido
   x = 123,45               ; tipo Number, inferido
   AñoNuevo = Date("01/01/99")                ; tipo Date,
asignado
   NombreEmpresa = String ("Borland")   ; tipo string, asignado
endconst
```

Vea también var

disableDefault

Palabra clave Desactiva el código por defecto de un método estándar.

Sintaxis **disableDefault**

Descripción **disableDefault** impide la ejecución del código estándar de un suceso. Normalmente, el código estándar se ejecuta implícitamente al final de un método, justo antes de la sentencia **endmethod**. El empleo de **disableDefault** en un método desactiva la llamada implícita al código estándar.

Ejemplo El ejemplo siguiente define el valor de un campo como hola , cuando el usuario teclea un carácter. La llamada a **disableDefault** impide la ejecución del código estándar, por lo que el carácter no se muestra en el campo. La sentencia **message** muestra el carácter en la ventana de estado.

```
method KeyChar(var eventInfo KeyEvent)
   self.value = "hola"      ; aparece el literal hola en el
campo
   disableDefault     ; se desactiva el código estándar
   message(eventInfo.char())      ; muestra el carácter en la
ventana de estado endMethod
```

Vea también doDefault
enableDefault
passEvent

doDefault

Palabra clave Ejecuta el código por defecto de un método estándar.

Sintaxis **doDefault**

Descripción **doDefault** ejecuta el código estándar de un suceso inmediatamente, y no al final del método. La utilización de **doDefault** en un método desactiva la llamada implícita al código estándar. Si un método contiene más de una sentencia **doDefault**, sólo se ejecuta la primera; las demás no se tienen en cuenta.

Ejemplo En el método siguiente, se pulsa el botón, el sistema espera dos segundos y emite una señal sonora, y se suelta el botón. El código estándar se llama implícitamente, justo antes de la sentencia **endMethod**.

```
method pushButton(var eventInfo Event)
   sleep(2000)
   beep()
endMethod
```

En el método siguiente, la llamada a **doDefault** implica que se suelta el botón antes de realizar la pausa y emitir la señal, y desactiva el código implícito al final del método.

```
method pushButton(var eventInfo Event)
   doDefault
   sleep(2000)
   beep()
endMethod
```

Vea también disableDefault
enableDefault
passEvent

enableDefault

Palabra clave Activa el código por defecto de un método estándar.

Sintaxis **enableDefault**

Descripción **enableDefault** permite que el código estándar se ejecute normalmente al final de un método, justo antes de la sentencia **endmethod**. Compare **enableDefault** con **doDefault**, que ejecuta el código estándar de inmediato.

Ejemplo

```
method AcciónMenú(var eventInfo MenuEvent)
var elección string endVar
disableDefault
elección = eventInfo.OpciónMenú()
switch
   case elección = "Abrir"  : Abrelo()
   case elección = "Terminar"     : Terminalo()
   otherwise    : enableDefault
endSwitch
endMethod
```

Vea también disableDefault
doDefault
passEvent

for

Palabra clave Ejecuta una secuencia de sentencias un número especificado de veces.

Tipo

Sintaxis **for** contador [**from** *valorInicial*] [**to** *valorFinal*] [**step** *valorPaso*]

Sentencias

endFor

valorInicial, valorFinal y *valorPaso* son valores o expresiones que representan los valores inicial y final del contador y el número en que se incrementa el contador cada vez que se procesa el bucle. Estos contadores pueden ser de cualquier tipo de datos representado por AnyTipo, *excepto* Point, Memo, Graphic, String, OLE y Binary.

Descripción **for** ejecuta una secuencia de *Sentencias* tantas veces como especifique un contador, que se almacena en *contador* y se controla mediante las palabras clave optativas **from**, **to** y **step**. Cualquier combinación de éstas puede utilizarse para especificar el número de veces que se ejecutan las sentencias del bucle. No es necesario declarar explícitamente *contador*, pero un bucle **for** se ejecuta con mayor rapidez si se declara.

for puede utilizarse sin las palabras clave **from**, **to** y **step**:

- Si se omite *valorInicial*, el contador comienza en el valor actual de *contador*.
- Si se omite *valorFinal*, el bucle **for** se ejecuta indefinidamente.
- Si se omite *valorPaso*, el contador se incrementa en 1 cada vez que se procesa el bucle.
- *valorInicial*, *valorFinal* y *valorPaso* se almacenan en una memoria intermedia temporal; no se envalúan cada vez que se procesa el bucle.

Si se utiliza **quitLoop** dentro del cuerpo de sentencias del bucle **for**, se sale del bucle **for endFor**. Si se utiliza **loop** dentro del cuerpo de sentencias, las que siguen a **loop** no se procesan, el contador se incrementa y la repetición continúa desde el principio del bucle **for**.

Si **step** es positivo y una cláusula **to** está presente, la repetición continúa mientras que el valor de *contador* sea menor o igual que el valor de *valorFinal*. Si **step** es negativo, la repetición continuará mientras el valor de *contador* sea mayor o igual que el de *valorFinal*. En cualquier caso, una vez que el valor de *contador* alcance o supere el límite definido por **step**, el bucle **for** deja de ejecutarse, pero *contador* mantiene su valor, como se muestra en el ejemplo.

Si no se ha asignado previamente un valor a *contador*, **from** crea la variable y le asigna el valor de *valorInicial*.

Ejemplo A continuación se muestra un sencillo bucle **for**. Obsérvese el valor de la variable de contador *i* después de que haya concluido el bucle **for**.

```
var i smallInt endVar
for i from 1 to 3
   i.view("dentro del bucle") ; i = 1, i = 2, i = 3
endFor
```

```
i.view("fuera del bucle") ; i = 4
```

Vea también loop
quitLoop
while

forEach

Palabra clave Repite la secuencia de sentencias especificada sobre elementos includidos en un DynArray.

Tipo

Sintaxis **forEach** *nombreVar* **in** *nombreMatrizDinámica*

Sentencias

endForEach

Descripción **forEach** pasa por los elementos de un DynArray. En general, no es posible usar la sentencia for para pasar por un DynArray porque los índices de un DynArray no son necesariamente números enteros.

Puesto que los índices de DynArray no son enteros, los elementos de un DynArray no están ordenados secuencialmente. La sentencia **forEach** opera sobre los elementos del DynArray en un orden arbitrario. El programador no debería depender de una ordenación específica de los índices.

Si *nombreMatrizDinámica* no existe, la sentencia **forEach** provoca un error cuando se compila el método.

Si se utiliza la sentencia **quitLoop** dentro del cuerpo de sentencias del bucle **forEach**, se sale del bucle **forEach endForEach**. Si se utiliza la sentencia **loop** dentro del cuerpo de *Sentencias*, las sentencias que siguen a **loop** no se procesan y la repetición continúa desde el principio del bucle **forEach**.

Ejemplo El ejemplo siguiente utiliza la sentencia **forEach** para mostrar los elementos del DynArray creado por la sentencia **sysInfo**:

```
var
   matrizSistema DynArray[] AnyType
   elemento string
endVar
sysInfo(matrizSistema)
forEach elemento IN matrizSistema
   message(elemento," : ", matrizSistema[elemento])
   sleep(1500)
endForEach
```

Vea también for
loop
quitLoop
while

if

Palabra clave Ejecuta una de dos secuencias de sentencias dependiendo del valor de una condición lógica.

Sintaxis **if** *Condición* **then**

Sentencias1

[**else**

Sentencias2]

endIf

Descripción Cuando ObjectPAL llega a una sentencia **if**, evalúa si la *Condición* es True. Si lo es, ejecuta secuencialmente las sentencias enumeradas en *Sentencias1*. Si no lo es, no tiene en cuenta *Sentencias1* y, si la palabra clave optativa **else** está presente, ejecuta las sentencias de *Sentencias2*. En cualquier caso, la ejecución continúa después de la palabra clave **endIf**.

Una construcción **if** puede abarcar varias líneas, especialmente si hay muchas sentencias en *Sentencias1* o *Sentencias2*. Es recomendable sangrar las cláusulas **then** y **else** y así evidenciar el flujo de control.

```
if Condición then
   Sentencias1
else
   Sentencias2
endIf
```

El código siguiente es un ejemplo de una sentencia **if**:

```
if existencias < 100 then
   pedirMás()   ; ejecuta un procedimiento personalizado
    ; llamado pedirMás
   existencias = existencias + 10 ; y suma 10 al valor de
existencias
endIf
```

Las sentencias **if** pueden anidarse; es decir, cualquiera de las sentencias de *Sentencias1* o *Sentencias2* también pueden ser sentencias **if**. Las sentencias **if** anidadas deben estar contenidas completamente dentro de la estructura **if** superior en otras palabras, cada sentencia **if** anidada debe tener un **endIf** dentro de la anidación . Cada conjunto **if endIf** debe incluir código o código y otro conjunto completo **if endIf**:

```
if Condición then
   if Condición then
     Condición
   endIf
endIf
```

Ejemplo El ejemplo siguiente proporciona el código de una sentencia **if** anidada:

```
if nivelManejo = "Iniciado" then
   if cajaManejo.color = "Red" or cajaManejo.color = "Yellow"
then
```

```
      cajaManejo.color = "Green"
   endIf
endIf
```

Vea también for
iif
switch
while

iif

Palabra clave Devuelve uno de dos valores dependiendo del valor de una condición lógica.

Sintaxis **iif** (*Condición, ValorSiTrue***,** *ValorSiFalse*)

Descripción **iif** permite la bifurcación dentro de una sola sentencia. **iif** puede utilizarse en cualquier posición en que se puede emplear cualquier otra expresión. **iif** es útil especialmente en campos calculados de fichas o informes porque las sentencias **if endIf** no son válidas en ellos.

Ejemplo

```
a = iif(x  1, b, c) ; if x > 1, a = b; else a = c
```

Vea también if

loop

Palabra clave Pasa el control al principio del bucle **for**, **forEach**, **scan** o **while** más cercano en que se encuentra.

Sintaxis **loop**

Descripción Cuando **loop** se ejecuta dentro de una estructura **for**, **forEach**, **scan**, or **while**, no se procesan las sentencias situadas entre **loop** y **endFor**, **endForEach**, **endScan** o **endWhile** y vuelve al comienzo de la estructura. En caso contrario, **loop** produce un error.

Ejemplo

```
var x smallInt endVar
for x from 1
   if x <> 5 then
      loop ; retorna a la sentencia for y toma un nuevo valor
para x
      message("Esto no aparece") ; esta sentencia no se
ejecutará nunca
   else
      quitLoop
endif
endFor
message(x) ; muestra el número 5
```

Vea también for
forEach
quitLoop
scan
while

method

Palabra clave Define un método de ObjectPAL.

Sintaxis ***method*** *Nombre* (*descripciónParámetros* [**,***descripciónParámetros*]*]) *[tipoDevuelto*]

[***type*** *sección*]

[***const*** *sección*]

[***var*** *sección*]

Sentencias

endMethod

Descripción **method** marca el comienzo de un método. Como mínimo, es necesario proporcionar lo siguiente:

- El nombre del método, en *Nombre*
- Paréntesis, aunque el método no tenga argumentos
- Las *Sentencias* de que consta el método

La definición termina con la palabra clave obligatoria **endMethod**.

Adicionalmente, es posible declarar constantes, tipos de datos, variables y procedimientos antes de la palabra clave **method**, y declarar variables y constantes después de **method**.

También son optativas una o más *descripciónParámetros*, donde cada *descripciónParámetros* tiene la forma

[var | const] ***tipo parámetro***

El argumento optativo *tipoDevuelto* declara el tipo de datos del valor devuelto por el método. *tipoDevuelto* es optativo porque un método puede no devolver un valor. Sin embargo, si el método devuelve un valor, es necesario especificar el tipo de datos del valor.

Los métodos y los procedimientos son similares. Las principales diferencias son:

- Los métodos son visibles y exportables a otros objetos, mientras que los procedimientos son privados dentro de una jerarquía de contenedores.
- Un método puede contener una definición de procedimiento, pero no es posible definir un método dentro de un procedimiento.

Nota: El ámbito de un método depende de dónde se declara.

Ejemplo

```
method pushButton(var eventInfo Event)
   var
      texto   string
      número  number
   endVar
   número = 123.321
   texto  = string(número)
   msgInfo("número = ",texto)
endmethod
```

Vea también proc

passEvent

Palabra clave Pasa el suceso al contenedor del objeto.

Sintaxis **passEvent**

Descripción **passEvent** pasa el paquete de sucesos al contenedor del objeto. El empleo de **passEvent** en un método no afecta a la llamada implícita al código estándar.

Ejemplo El código del ejemplo siguiente se anexa a un objeto de campo. Se ejecuta cuando el puntero está en el objto de campo. Si *Mayús* se mantiene pulsada cuando se pulsa el ratón, el código ejecuta **disableDefault** para impedir la ejecución del código estándar y activa **passEvent** para enviar el suceso al contenedor del objeto de campo. Esta técnica es útil cuando se desea que varios objetos respondan de la misma forma a un suceso dado.

```
method mouseDown(var eventInfo MouseEvent)
   if eventInfo.isShiftKeyDown() then
      disableDefault
      passEvent ; deja que lo maneje el contenedor
   endIf
endmethod
```

Vea también disableDefault
doDefault
enableDefault

proc

Palabra clave Define un procedimiento de ObjectPAL.

Sintaxis **proc** *Nombre* (*descripciónParámetros* [,descripciónParámetros]*]) [tipoDevuelto]

[**const** *sección*]

[**type** *sección*]

[**var** *sección*]

Sentencias

endProc

Descripción proc comienza la definición de un procedimiento. El programador debe proporcionar lo siguiente:

El nombre del procedimiento, en *Nombre*

Paréntesis, aunque el procedimiento no tenga argumentos

Una o más descripciones de parámetros, representada por el prototipo de *descripciónParámetros*, donde cada descripción tiene la forma

[**var|const**] ***tipo parámetro***

Use *tipoRetorno* para declarar el tipo de datos del valor devuelto por el procedimiento (si devuelve un valor)

Secciones que declaran variables, constantes y tipos

Las *Sentencias* de que consta el procedimiento

La definición termina con la palabra clave obligatoria **endProc**.

Es posible utilizar **return** en el cuerpo de un procedimiento para devolver un valor al método o procedimiento que lo ejecutó.

Un procedimiento utilizado en una expresión debe devolver un valor, como

```
x = NúmeroRegistros("Pedidos")   ; NúmeroRegistros es un
procedimiento
```

Los métodos y los procedimientos son similares. Las principales diferencias son:

Los métodos son visibles y exportables a otros objetos, mientras que los procedimientos son privados dentro de una jerarquía de contenedores.

Un método puede contener una definición de procedimiento, pero no es posible definir un método dentro de un procedimiento.

Nota: El ámbito de un método depende de dónde se declara.

Ejemplo

```
proc inc (x SmallInt) SmallInt
   return x + 1
endProc
method pushButton(var eventInfo Event)
   var  número smallInt endVar
```

```
   número = 5
   número = incrementa(número) ; ejecuta el procedimiento
   message(número) ; muestra 6
endmethod
```

Vea también method

quitLoop

Palabra clave Termina el bucle **for**, **forEach**, **scan** o **while** en el que aparece.

Sintaxis **quitLoop**

Descripción **quitLoop** sale inmediatamente del bucle **for**, **forEach**, **scan** o **while** más inmediato en que se encuentra. El método continúa con la sentencia que sigue al **endFor**, **endForEach**, **endScan** o **endWhile**.

quitLoop provoca un error si se ejecuta fuera de alguna estructura **for**, **scan** o **while**.

Ejemplo En este ejemplo, se utiliza **quitLoop** en un bucle **for** que determina si una matriz tiene algún elemento sin asignar:

```
var
   Matriz  Array[12]
   sinAsignar  Logical
endVar
sinAsignar = False
for i from 1 to Matriz.size()
   if not isAssigned(Matriz[i]) then
      sinAsignar = True
      quitLoop
   endIf
endFor
```

Vea también loop
return

return

Palabra clave Devuelve el control desde un método o procedimiento, pasando optativamente un valor.

Sintaxis **return** [Expresión]

Descripción **return** sirve para devolver el control del procedimiento o método actual al procedimiento o método que lo ejecutó. Lo siguiente se aplica a **return**:

- Debe haberse declarado que el procedimiento va a devolver un valor antes de que se pueda utilizar **return**.
- Si se ejecuta **return** dentro del cuerpo de un procedimiento, se sale del procedimiento.
- Si se ejecuta **return** dentro de un método (pero fuera del cuerpo de un procedimiento), se sale del método.

Optativamente es posible devolver el valor de *Expresión* cuando se vuelve de un procedimiento o de un método. Si se ejecuta un procedimiento dentro de una expresión, el procedimiento debe devolver un valor que se convierte en el valor de la llamada al procedimiento.

```
y = miProc(x) + 3     ; miProc es un procedimiento
```

Si un procedimiento se llama en un contexto autónomo, no se tiene en cuenta el valor que pueda devolver.

```
miProc(x)
```

Si no se suministra ninguna *Expresión*, **return** no debe ir seguido de nada más en la línea, aparte de un comentario.

Los tipos de datos siguientes *no pueden* devolverse: DDE, Database, Query, Session, Table o TCursor.

No es necesario emplear **return** para devolver el control a un método o procedimiento de nivel superior, puesto que eso se produce automáticamente cuando el método o procedimiento de nivel inferior termina. Sin embargo, si se ha declarado que el método o procedimiento devuelvan un valor, es necesario emplear **return** para devolver el valor; el valor no se devolverá automáticamente.

Ejemplo A continuación, se presenta un simple ejemplo que suma 1 al valor de una variable y devuelve el nuevo valor al método que lo ejecutó:

```
proc añadirUno (x SmallInt) SmallInt
   return x + 1
endProc
```

En un método estándar, una sentencia **return** ejecuta el código estándar a menos que se desactive explícitamente. Por ejemplo, el código siguiente ejecuta **return** cuando el usuario teclea "**?**" en un objeto de campo. La ejecución de **disableDefault** impide que el código estándar muestre ? en el objeto de campo.

```
method KeyChar(var eventInfo KeyEvent)
   if eventInfo.char()="?" then
      disableDefault
```

```
            return
        endIf
    endMethod
```

Vea también quitLoop

scan

Palabra clave Explora el TCursor y ejecuta las instrucciones de ObjectPAL.

Sintaxis **scan** *tcVar* [**for** *expresiónBooleana*] *:*

Sentencias

endScan

Los dos puntos son necesarios para separar **scan** de una sentencia **for** siguiente.

Descripción **scan** analiza *tcVar* (un TCursor) y ejecuta *Sentencias* (instrucciones de ObjectPAL) para cada registro. **scan** siempre comienza en el primer registro de la tabla y analiza cada registro secuencialmente. Cuando las sentencias del bucle **scan** cambian un campo indexado, ese registro va a su posición ordenada en la tabla, por lo que es posible encontrar el mismo registro más de una vez en el mismo bucle.

Si se suministra la cláusula **for**, las *Sentencias* sólo se ejecutan para los registros que cumplan la condición; los demás no se procesan. Si la tabla está vacía o si ningún registro cumple la condición, **scan** no tiene ningún efecto.

Nota: for es una palabra clave para **scan**, por lo que debe ir seguida de un signo de dos puntos para diferenciarla de un bucle **for**.

scan es muy potente ya que es posible crear un prototipo con una secuencia de sentencias para un solo registro de una tabla, y situar dicha secuencia dentro de un bucle **scan** para que funcione en toda una tabla.

Es posible utilizar **loop**, **return** y **quitLoop** en el cuerpo de la estructura **scan**. **loop** omite las sentencias restantes entre él y **endScan**, va al registro siguiente y vuelve al principio del bucle **scan**. **quitLoop** termina el **scan** del todo, dejando el registro que se está analizando como registro actual.

Puesto que **scan** repite toda una secuencia de sentencias para cada registro, no incluya acciones que sólo deban realizarse una vez para la tabla. Incluya dichas sentencias fuera del bucle **scan**. **scan** va automáticamente de registro en registro en toda la tabla, por lo que no es necesario ejecutar **nextRecord**.

Ejemplo Este ejemplo utiliza un bucle **scan** para actualizar la tabla *Empleado*. Analiza el campo Dept de cada registro, y si el valor es Personal , lo cambia a Recursos Humanos .

```
var
   emple TCursor
endVar

emple.open("empleado.db")   ; está sentencia debe ejecutarse una
sola vez     emple.edit()

scan emple for emple.departa = "Personal" : ; los dos puntos
son necesarios
   emple.departa ="Recursos Humanos"
```

```
endScan

emple.endEdit()
emple.close()
```

Vea también if
for
while

switch

Palabra clave Ejecuta una secuencia de entre un conjunto de sentencias alternativas, dependiendo de que se cumplan varias condiciones.

Sintaxis **switch**

listaCase

[**otherWise**: *Sentencias*]

endSwitch

listaCase es cualquier número de sentencias con la forma siguiente:

case Condición : Sentencias

Descripción **switch** utiliza los valores de las sentencias Condición de listaCase para determinar qué secuencia de Sentencias debería ejecutarse, si es necesario ejecutar alguna. **switch** funciona como muchas sentencias **if**, y cada listaCase funciona como una sola sentencia **if**.

Las Condiciones de **case** se evalúan en el orden en que aparecen:

- Si una tiene un valor True, se ejecuta la secuencia de Sentencias correspondiente, y el resto no se procesan.
- Si ninguna tiene el valor True, y la cláusula optativa **otherWise** está presente, se ejecutan las Sentencias de **otherWise**.
- Si ninguna tiene el valor True y no hay ninguna cláusula **otherWise**, **switch** no tiene ningún efecto.

Por tanto, como máximo, se ejecuta un conjunto de Sentencias. El método continúa en la sentencia que sigue a **endSwitch**.

Ejemplo Este ejemplo crea una matriz de 100 números aleatorios y, a continuación, utiliza el algoritmo de orden de lanzamiento para ordenarlas numéricamente:

```
method PushButton(var eventInfo Event)
var
   cien, indice, segundo, primero smallInt
   matriz Array[100] smallInt
   temporal number
endVar

   cien = 100
   matriz.fill(0)

for indice from 1 to cien step 1
   temporal = Rand()
   switch
      case temporal < .1 : matriz[indice] = 1
      case temporal < .2 : matriz[indice] = 2
      case temporal < .3 : matriz[indice] = 3
      case temporal < .4 : matriz[indice] = 4
      case temporal < .5 : matriz[indice] = 5
      case temporal < .6 : matriz[indice] = 6
      case temporal < .7 : matriz[indice] = 7
      case temporal < .8 : matriz[indice] = 8
```

```
      case temporal < .9 : matriz[indice] = 9
       otherwise        : matriz[indice] = 10
   endSwitch
endFor

for primero from 1 to cien-1 step 1
   for segundo from 1 to cien-primero step 1
      if matriz[segundo] <>  matriz[segundo+1] then
          matriz.exchange(segundo,segundo+1)
      endIf
   endFor
endFor

endMethod
```

Vea también if
while

try

Palabra clave Marca un bloque de sentencias que se van a intentar y especifica una respuesta en caso de que se produzca un error.

Sintaxis

try

[*Sentencias*] ; bloque de transacción

onFail

[*Sentencias*] ; bloque de recuperación

[**reTry**] ; optativo

EndTry

Descripción El mecanismo para incorporar la recuperación de fallos en una aplicación es el bloque **try onFail**.

El bloque de transacción es un conjunto de *Sentencias*, todas las cuales se desea que sean satisfactorias. Si la transacción es satisfactoria, el programa va directamente a **endTry**. Si la transacción falla, se ejecuta el bloque de recuperación. Es posible llamar a **reTry** para ejecutar el bloque de transacción de nuevo.

Un fallo en el intento se produce cuando el programa llama al procedimiento **fail** del tipo System en algún punto dentro del bloque de transacción o dentro de procedimientos llamados por el bloque de transacción. Esto impide que las funciones del sistema devuelvan errores de estado o valores nulos a quien las llamó.

Una llamada a **fail** puede anidarse varios niveles de llamadas a procedimientos desde donde comenzó el bloque. Sus variables locales se eliminan de la pila y cualquier objeto (como bloques de texto grandes) se desasignan. Si los objetos de referencia (como tablas) están en uso, se cierran, y cualquier actualización pendiente se cancela. Es casi como si la transacción nunca se hubiera iniciado. Lo único que permanece son los cambios realizados en las variables externas al bloque o los datos añadidos satisfactoriamente a las tablas y consignadas antes de que se produjera el fallo.

Si durante un bloque de recuperación, se decide que el código de error no es el esperado o es más grave que los que pueden gestionarse a este nivel, ejecute **fail** de nuevo para pasar ese código de error. Si no existe ningún bloque **try onFail** de nivel superior, toda la aplicación falla, cancela las acciones existentes, cierra los recursos y sale.

Ejemplo Este ejemplo intenta definir la propiedad Color de algunos objetos de diseño y utiliza un bloque **try onFail** para gestionar la situación si no puede definir la propiedad.

```
method PushButton(var eventInfo Event)
var literal string endVar
cajaUno.cajaDos.color = Blue      ; esto funciona
literal = "cajaCinco"       ; cajaCinco no existe

try
```

```
  cajaUno.(literal).color = Red ; intenta asignar color a
cajaCinco
onFail    ; gestiona el error
   msgStop("Error", "No puedo encontrar " +literal)
   literal = "cajaDos"      ; cajaDos existe
   reTry  ; lo intenta otra vez
endTry

literal = "cajaSeis"  ; cajaSeis no existe
try
   cajaUno.(literal).color = Green
onFail
   fail(ObjetoInexistente,"El objeto " + literal + " no
existe.")          endTry
endMethod
```

Vea también System::fail

type

Palabra clave Declara tipos de datos.

Sintaxis

type

[*nombreNuevoTipo* **=** *tipoExistente*]*

endType

Descripción Mediante **type**, pueden definirse nuevos tipos de datos. Una vez definidos, es posible utilizar estos tipos para declarar variables en los métodos.

Por ejemplo, una aplicación que controla el número de piezas en un almacén podría declarar un tipo *unidadesPedido* y declarar, a continuación, una variable que sea del tipo *unidadesPedido*, de esta forma:

```
type
   unidadesPedido = SmallInt      ; declara un tipo nuevo
endType

var ; utiliza el tipo nuevo para declarar una variable
undPed unidadesPedido       ; undPed es un SmallInt
endVar    ; porque unidadesPedido es SmallInt
```

Más tarde, si el número de piezas se aproxima a 32.767 (el valor máximo de un SmallInt), sólo es necesario cambiar la definición del tipo, por ejemplo,

```
type
   unidadesPedido = LongInt       ; cambia la declaración
endType

var ; utiliza el tipo nuevo para declarar una variable
   undPed unidadesPedido    ; undPed es un LongInt
endVar    ; porque unidadesPedido es LongInt
```

Ejemplo Un **type** útil es el **record**. Los registros definidos en la ventana Type de un objeto no tienen ninguna relación con las tablas. En su lugar, son similares a los registros de Pascal y a las STRUCT de C, porque permiten unir varios elementos de datos relacionados bajo un solo nombre. Por ejemplo, el código siguiente declara un **record** *Empleado* que puede usarse para declarar variables en métodos y procedimientos.

```
type
   empleado = record
              Apellidos     String
              Nombre  String
              Título  String
              Salario       Currency
              FechadeAlta   Date
              endRecord
endType
```

Vea también var

uses

uses

Palabra clave Declara rutinas de bibliotecas externas para su uso en un método o procedimiento.

Sintaxis **uses** [biblioteca|ObjectPAL]

nombreRutina (listaParámetros)

endUses

Descripción El bloque **uses**, declarado en la ventana Uses de un objeto, hace que rutinas almacenadas en bibliotecas externas estén disponibles en los métodos de ObjectPAL. Las rutinas deben ajustarse a una de las descripciones siguientes:

Rutinas escritas en ObjectPAL y almacenadas en una biblioteca de ObjectPAL library.

- Rutinas escritas en ObjectPAL y anexadas a una ficha. La sintaxis para ejecutar métodos anexados a una ficha es la misma que para ejecutarlos desde una biblioteca.

- Rutinas escritas en lenguaje ensamblador, C, C++ o Pascal y almacenadas en una biblioteca de ObjectPAL o en una biblioteca de vínculo dinámico de Windows (DLL). Un DLL es una biblioteca de código ejecutable o datos que pueden vincularse con la aplicación en tiempo de ejecución. El uso de DDL permite añadir características y funciones sin modificar la aplicación ObjectPAL compilada.

Un bloque **uses** de una biblioteca de ObjectPAL varía ligeramente de un bloque **uses** de un DDL, por lo que se tratan en secciones distintas.

Bloque uses de una biblioteca de ObjectPAL

Para utilizar métodos contenidos en una biblioteca de ObjectPAL o anexados a una ficha, escriba un bloque **uses** en la ventana Uses de un objeto. Esta es la sintaxis:

uses ObjectPAL

[*nombreMétodo* ([**var | const**] listaArgumentos) [tipoDevuelto]]*

endUses

La palabra clave **ObjectPAL** es necesaria para indicar que se llama a métodos de una biblioteca de ObjectPAL o a una ficha, y no a un DLL.

Es necesario abrir una biblioteca antes de ejecutar un método desde ella; es necesario abrir y ejecutar una ficha antes de ejecutar un método desde ella.

Entonces, *nombreMétodo* representa el nombre del método que se llama mientras que *listaArgumentos* representa una lista de pares argumento/tipo de datos separados por comas y va precedida optativamente por las palabras clave **var** y **const**, según sea pertinente.

El argumento optativo *tipoDevuelto* especifica el tipo de datos del valor (si lo hay) devuelto por el método.

Dentro de una sola ventana Uses, es posible declarar más de un método de biblioteca, y métodos de más de una biblioteca.

Los argumentos y tipos de datos declarados en la ventana Uses deben declararse exactamente como se halla en la biblioteca.

El ejemplo siguiente declara dos métodos de biblioteca, **buildMenu** y **calcInterest**. **buildMenu** toma un argumento, una constante String llamada *Nombrepágina*. **calcInterest** toma dos argumentos: una variable Number llamada *porcent* (pasada mediante referencia) y una variable SmallInt llamada *períodos* (pasada mediante valor).

```
uses ObjectPAL
   hacermenu(const Nombrepágina string)
   calculaInteres(var porcent number, períodos smallInt) Number
endUses
```

El código siguiente, anexado a la ventana Uses de un botón, declara el método **calcInterest** para que el botón pueda utilizarlo. Obsérvese que no es necesario declarar cada método de una biblioteca, sólo los que se desee emplear.

```
uses ObjectPal
      calculaInteres(var porcent number, períodos smallInt)
Number
endUses
```

El código siguiente, anexado al método estándar **pushButton** de un botón, abre la biblioteca y ejecuta estos métodos.

```
method PushButton(var eventInfo Event)
   var
       matLib Library
       porcent Number
       períodos SmallInt
       interés  Number
   endVar

   if matLib.open("matemat.lsl") then
      porcent = hipoteca.porcent.value
      períodos = hipoteca.años.value * 12
      interés = matemat.calcuInterés(porcent,períodos)
     interés.view("Intereses")
   endIf
edMethod
```

En este ejemplo, la notación de punto especifica dónde encontrar el método **calcInterest**. La sentencia siguiente mira en la biblioteca representada por la variable Library *thamLib*.

```
interés = matemat.calcuInterés(porcent,períodos)
```

Ejecución de métodos personalizados anexados a otras fichas

El concepto de ejecución de un método personalizado anexado a otra ficha es el mismo: utilizar notación de punto para especificar la ficha en que buscar el método. El ejemplo siguiente supone que se ha declarado previamente la variable Form *códigoFicha*, que se ha abierto la ficha y que

se ha declarado el método **getObjHelp** en una ventana Uses adecuada.

```
interés = codigoFicha.ObtenerAyuda(self.name)
```

Para utilizar rutinas contenidas en un DLL, escriba un bloque uses en uno de los lugares siguientes:

- Ventana Uses de un objeto de diseño
- Ventana de un método estándar
- Ventana de un método personalizado
- Ventana de un procedimiento personalizado

La ventana en que escriba el bloque depende del ámbito (disponibilidad) que se desee para la rutina. Independientemente de donde se escriba, la estructura básica (mostrada en el pseudocódigo siguiente) es la misma:

uses *nombreBiblioteca*

nombreRutina* (*listaParámetros*) *tipoDevuelto

endUses

El argumento nombreBiblioteca especifica el nombre del archivo DLL, donde nombreBiblioteca es un nombre de archivo válido del DOS con un máximo de ocho caracteres. Paradox supone una extensión .DLL o .EXE. Windows busca el archivo por este orden:

1. Directorio actual.

2. Directorio de Windows (directorio que contiene WIN.COM). Es posible emplear el procedimiento windowsDir de FileSystem para obtener esta información (normalmente, es C:\WINDOWS).

3. Directorio SYSTEM de Windows (directorio que contiene archivos de sistema como KERNEL.EXE). Es posible utilizar el procedimiento **windowsSystemDir** de FileSystem para obtener esta información (normalmente, es C:\WINDOWS\SYSTEM).

4. Directorios incluidos en la variable PATH del entorno. Para más información, consulte la documentación del DOS.

5.Lista de directorios asignados en una red.

Nota para programadores avanzados de Windows: Si se ejecuta una rutina desde un DLL cargado previamente (por ejemplo, un DLL cargado automáticamente por Windows), es posible utilizar nombreBiblioteca para especificar el nombre del módulo DLL, en lugar del nombre de archivo. Consulte la documentación de su lenguaje de programación para más información sobre los nombres de los módulos DLL.

Un bloque **uses** puede contener uno o más *nombreRutina*, y cada *nombreRutina* puede tener su propia *listaParámetros*. Una *listaParámetros* especifica uno o más nombres de argumentos y tipos de datos y, si la rutina devuelve un valor, el *tipoDevuelto* especifica el tipo de datos del valor devuelto. ObjectPAL comprueba en las especificaciones de estos argumentos las correspondencias *exactas* con los declarados en la rutina, pero ésa es toda la comprobación que realiza.

Declare un bloque **uses** en la ventana Uses de un objeto y, dentro de esa ventana, declare un bloque **uses** para cada biblioteca o DLL que desea utilizar. No es necesario que declare cada rutina que la biblioteca o DLL

contiene, sólo las que desea utilizar. Una vez declaradas, las rutinas están disponibles para todos los métodos anexados a ese objeto y para todos los objetos que el objeto contiene.

En un bloque **uses**, declare tipos de datos mediante las palabras clave siguientes:

Tipo de datos	ObjectPAL	C	Pascal	
Entero de 16 bits	CWORD	int	Integer	
Entero de 32 bits	CLONG	long	Longint	
Número de 64 bits de coma flotante		CDOUBLE	double	Double
Número de 80 bits de coma flotante	Extended	CLONGDOUBLE	long double	
puntero	CPTR	char far *	String	
datos binarios	CHANDLE o gráficos	Handle	THandle (Windows)	(Windows)

Estas palabras clave sólo son válidas dentro de un bloque **uses**. No las utilice en ningún otro lugar.

Para utilizar una rutina en un método, declare variables para su uso como argumentos y, a continuación, ejecute la rutina. Por ejemplo,

```
; este código va en la ventana Uses de un objeto
uses Tontada ; lee la rutina de TONTADA.DLL
   hazAlgo (esteNumero CLONG, aquelNumero CLONG) CDOUBLE ;
declara la rutina     endUses

; este código modifica el método mouseUp de un objeto
method mouseUp(var eventInfo MouseEvent)
var
   esteNumero,aquelNumero LongInt ; declara las variables que
se pasan
     ; a la rutina
   miResultado Number
endVar

esteNumero  = 3155111
aquelNumero = 5535345
miResultado = hazAlgo(esteNumero, aquelNumero) ; ejecuta la
rutina,
    ; devuelve un resultado
```

En el ejemplo anterior, los argumentos del bloque **uses endUses** se declaran utilizando CLONG y CDOUBLE, y las variables del método se declaran mediante LongInt y Number. Ello se debe a que los tipos de datos de ObjectPAL son más complejos (y potentes) que los tipos correspondientes de C y Pascal:

- CWORD corresponde a SmallInt.
- CLONG corresponde a LongInt.
- CDOUBLE y CLONGDOUBLE corresponden a Number.

- CPTR corresponde a String.
- CHANDLE corresponde a Binary y Graphic.

Nota: No modifique el contenido de un CPTR pasado.

Ejemplo Este ejemplo utiliza rutinas de MINMAX.DLL, escritas utilizando Turbo Pascal para Windows de Borland. El código Pascal para el DLL es:

Código Pascal para definir un DLL

```
{ este es el código en Pascal que define la DLL }
library MinMax;

function Min(x,y:integer) : integer ; export;
begin
   if x < y then Min:=x else  Min:=y;
end;

function Max(x,y:integer) : integer ; export;
begin
   if x > y then Max:=x else  Max:=y;
end;

exports
   Min index 1,
   Max index 2;

begin
end
```

A continuación se presenta el código de ObjectPAL para utilizar las rutinas del DDL. Primero se muestra el código para la ventana Uses y, a continuación, el código que modifica el método **pushButton** de un botón:

Código de ObjectPAL en la ventana Uses\

```
; el siguiente código va en la ventana Uses de un botón
uses MinMax ; carga las rutinas desde MINMAX.DLL
   Min (x CWORD, y CWORD) ; declara las funciones que van a
emplearse
   Max (x CWORD, y CWORD)
endUses
```

Código de ObjectPAL en la ventana del método

El código siguiente modifica el método estándar **pushButton** de un botón:

```
method pushButtton(var eventInfo Event)

   var
      x, y, z SmallInt
   endVar
   x = 2
   y = 6
   z = Min(x,y)       ; ejecuta la función Min de la DLL
   msgInfo("Mínimo .: ",z)
   z = Max(x,y)       ; ejecuta la función Max de la DLL
   msgInfo("Máximo .: ",z)
endMethod
```

Aspectos adicionales sobre las llamadas a rutinas C

Windows utiliza la misma convención de llamada que Pascal. La convención de llamada de Pascal implica lo siguiente:

- Los parámetros se introducen en la pila en el orden en que aparecen en la llamada a la función.
- El código que restaura la pila es parte de la función llamada (no de la función que realiza la llamada).

La convención de llamada de Pascal es distinta de la utilizada en C. En C, los parámetros se introducen en la pila en orden inverso, y la función que realiza la llamada es responsable de restaurar la pila. Cuando escriba un DLL en un lenguaje que no utilice normalmente la convención de llamada de Pascal, como C, asegúrese de que se emplea dicha convención en cualquier función que Windows llame. En C, esto precisa el uso de la palabra clave **PASCAL** cuando se declara la función.

Los ejemplos de las tablas siguientes dan por supuesto que se han declarado estas variables de ObjectPAL:

si	SmallInt
li	LongInt
nu	Number
gr	Graphic
st	String

Cuando se pasa un valor a un procedimiento C, la variable de ObjectPAL debe declararse y teclearse explícitamente. Sin embargo, AnyType no está permitido.

Vea también Pase mediante valor

Pase mediante puntero

Valores de retorno

Nota sobre datos gráficos y binarios

Usando C++

Archivo .DEF

Pase mediante valor

La tabla presenta las sintaxis para pasar varios tipos de datos mediante valor a un procedimiento C. ObjectPAL pasa y devuelve los valores de coma flotante mediante valor, tal y como precisa el compilador C++ de Borland. Otros compiladores C pueden tener requisitos diferentes. Para garantizar la compatibilidad, pase los valores mediante puntero, como se describe en la sección siguiente.

Tipo de datos de C	Sintaxis de C	En bloque USES	Llamada de ObjectPAL
Long double CLONGDOUBLE)	void pascal far _loadds cproc(long double value)	cproc(si)	cproc(value cproc(li) cproc(nu)
Double	void pascal far _loadds cproc(double value)	cproc(value CDOUBLE)	cproc(si), cproc(li) cproc(nu)
Long int	void pascal far _loadds cproc(long int value)	cproc(value CLONGINT)	cproc(si) cproc(li)
Int	void pascal far _loadd cproc(int value)	cproc(value CWORD)	cproc(si)
String	void pascal far _loadds cproc(char * value)	cproc(value CPTR)	cproc(st)
Graphic	void pascal far _loadds cproc(HANDLE value)	cproc(value CHANDLE)	cproc(gr)
Binary	void pascal far _loadds cproc(HANDLE value)	cproc(value CHANDLE)	cproc(gr)

Pase mediante puntero

Esta sección explica cómo pasar información si el procedimiento C utiliza punteros para la información. El puntero señala directamente al valor correspondiente de la variable de ObjectPAL. Por ejemplo, si desea un int* y pasa un SmallInt, obtendrá un puntero que señala directamente al int dentro de la variable SmallInt. El programador puede modificar el valor del SmallInt, pero esto podría ser muy peligroso para programadores C principiantes, puesto que pueden corromper ObjectPAL sobrescribiendo la memoria (escribiendo más allá de los límites del puntero de memoria). Utilice punteros para:

- Cambiar la información (esto debería hacerse mediante valores devueltos por la función, si es possible).posible).
- Pasar valores de coma flotante a procedimientos C que no se compilaron mediante el compilador C de Borland. Los diferentes compiladores C utilizan convenciones distintas para pasar y devolver valores de coma flotante (double y long double). La única forma de pasar información independiente del compilador es mediante punteros.

La tabla presenta las sintaxis para pasar varios tipos de datos mediante puntero a un procedimiento C.

Tipo de datos de C	**Sintaxis de C**	**En bloque USES**	**Llamada de ObjectPAL**
C data type	C syntax	In USES block	ObjectPAL call
LONG DOUBLE *	void pascal far _loadds cproc(long double * value)		cproc(value CPTR) cproc(nu)
LONG INT *	void pascal far _loadds cproc(long int * value)	cproc(value CPTR)	cproc(li)
INT *	void pascal far _loadds cproc(int * value)	cproc(value CPTR)	cproc(si)
STRING *	void pascal far _loadds cproc(char * value)	cproc(value CPTR)	cproc(st)

Valores de retorno

La siguiente tabla muestra la sintaxis de los valores de retorno de procedimientos C

C data type	C syntax	In USES block	ObjectPAL call
Long double	long double pascal far _loadds cproc(void)	cproc() CLONGDOUBLE	nu = cproc()
Double	double pascal far _loadds cproc(void)	cproc() CDOUBLE	nu = cproc()
Long int	long int pascal far _loadds cproc(void)	cproc() CLONGINT	nu = cproc() li = cproc()
Int	long int pascal far _loadds cproc(void)	cproc() CWORD	nu = cproc() li = cproc() sm = cproc()
Char	char * pascal far _loadds cproc(void)	cproc() CPTR	st = cproc()

Nota sobre datos gráficos y binarios (CHANDLE)

Nota sobre datos gráficos y binarios (CHANDLE)

Los datos Graphic y Binary se pasan mediante CHANDLE. En términos de C, es una definición de tipo HANDLE. Un CHANDLE es un gestor para memoria de Windows. Para utilizarlo, incluya:

```
void pascal far_loadds cproc (HANDLE valor)
{
   huge *ptr = (huge *) GlobalLock(valor)
   // ....... utilice ptr.
   // ....... no utilice GlobalFree(valor)
   GlobalUnlock(value)
}
```

Para una variable Binary, HANDLE es un gestor para memoria que contiene directamente la información contenida en el BLOB Binario. No hay ninguna información de cabecera . Es posible leer o modificar los datos, pero no cambiar su tamaño.

Para una variable Graphic, HANDLE es un mapa de bits de Windows.

Utilide éste como utilizaría cualquier otro HANDLE de mapa de bits.

Usando C++

Si está utilizando C++, rodee su código utilizando la siguiente sintaxis:

```
extern "C"
{
// Your procs
}
```

Archivo DEF

Todas las funciones C que desee que ObjectPAL llame deben exportarse en el archivo .DEF.

Vea también proc

var

Palabra clave Declara variables.

Tipo

Sintaxis **var**

[nombreVar [**,** *nombreVar*] ***** *tipoVar*]*

endVar

Descripción El bloque **var endVar** declara variables mediante la asociación de un nombre de variable nombreVar con un tipo de datos tipoVar. Cuando declare más de una variable del mismo tipo en la misma línea, utilice comas para separar los nombres.

Nota: El ámbito de una variable depende del lugar en que se declara.

Ejemplo

```
var
   misLetras, xx String
   miNúmero Number
   pedidos, ventas, entregas TCursor
   proteo AnyType
   miCaja UIObject
   a,b Array[5] SamllInt
   miOtroNúmero Number
endVar
```

Vea también method

while

Palabra clave Repite una secuencia de sentencias mientras que una condición especificada sea True.

Tipo

Sintaxis **while** *Condición*

[Sentencias]

endWhile

Descripción **while** comienza evaluando la expresión lógica *Condición*. Si *Condición* es False, las *Sentencias* no se ejecutan. Si el valor es True, se ejecutan secuencialmente las *Sentencias* incluidas entre la *Condición* y **endWhile**. Entonces, el control vuelve al principio del bucle y se evalúa de nuevo la *Condición*. Estos pasos se repiten hasta que la *Condición* se evalúe como False; en este momento, se sale del bucle y el control avanza a la sentencia que sigue a **endWhile**.

Es posible utilizar **loop** dentro del cuerpo del **while** para forzar la devolución del control al principio del bucle y omitir así las sentencias situadas entre **loop** y **endWhile**. También puede emplearse **quitLoop** para salir del bucle por completo. Las sentencias **while** pueden anidarse unas dentro de otras a cualquier nivel.

while y **for** son similares pero, en general, se utilizan por razones diferentes. **for** es útil para ejecutar una secuencia de sentencias un número conocido de veces, mientras que **while** la ejecuta un número de veces arbitario.

Ejemplo

```
; este ejemplo genera una matriz con los apellidos
var
   misNombres TCursor
   MatrizNombres Array[] String
   n SmallInt
endVar

misNombres.open("nombres.db")
MatrizNombres.grow(1)
MatrizNombres[1] = misNombres."Apellido"
n = 1

while misNombres.nextRecord()
   n = n + 1
   MatrizNombres.grow(1)
   MatrizNombres[n] = misNombres."Apellido"
endWhile
```

Vea también

for
forEach
if
loop
quitLoop
scan

Tipos de ObjectPAL

Los métodos y procedimientos ObjectPAL están divididos en seis categorías. Cada una de ellas está compuesta de diferentes tipos. Los elementos básicos del lenguaje son comunes a todos los métodos y procedimientos. Seleccione cualquiera de las siguientes categorías para visualizar los tipos que contiene.

Modelo de datos
Datos del sistema
Tipos de datos
Elementos básicos
Diseño de objetoss
Visualización managers
Sucesos

El nivel ObjectPAL con el que está trabajando biene determinado por el número de métodos disponibles para cada tipo. Puede cambiar el nivel de ObjectPAL en el Cuadro de diálogo Propiedades del escritorio

Puede copiar y pegar los ejemplos en su propio código a través del Portapapeles.

- Desde la ventana de Ayuda, seleccione Edición | Copiar, y a continuación el bloque de código que desea copiar.
- Situe el cursor en el lugar donde desea pegar el contenido de Portapapeles y seleccione Edición | Pegar del menú de Paradox.

Vea también

El editor de ObjectPAL
El depurardor de ObjectPAL
Macros
Bibliotecas
Lista alfabética de métodos

Copiando ejemplos en métodos

Puede copiar y pegar los ejemplos en su propio código a través del Portapapeles.

- Desde la ventana de Ayuda, seleccione Edición | Copiar, y a continuación el bloque de código que desea copiar.
- Situe el cursor en el lugar donde desea pegar el contenido de Portapapeles y seleccione Edición | Pegar del menú de Paradox.

Modelo de datos

Seleccione un tipo para visualizar la lista de métodos y procedimientos. Para cada método y procedimiento aparecerán los campos sintaxis, descripción y un ejemplo.

DataBase
Query
Table
TCursor

Datos del sistema

Seleccione un tipo para visualizar la lista de métodos y procedimientos. Para cada método y procedimiento aparecerán los campos sintaxis, descripción y un ejemplo.

DDE
FileSystem
Library
Session
System
TextStream

Tipos de datos

Seleccione un tipo para visualizar la lista de métodos y procedimientos. Para cada método y procedimiento aparecerán los campos sintaxis, descripción y un ejemplo.

AnyType
Array
Binary
Currency
Date
DateTime
DynArray
Graphic
Logical
LongInt
Memo
Number
OLE
Point
Record
SmallInt
String
Time

Diseño de objetos

Seleccione un tipo para visualizar la lista de métodos y procedimientos. Para cada método y procedimiento aparecerán los campos sintaxis, descripción y un ejemplo.

Menu
PopupMenu
UIObject

Visualización

Seleccione un tipo para visualizar la lista de métodos y procedimientos. Para cada método y procedimiento aparecerán los campos sintaxis, descripción y un ejemplo.

Application
Form
Report
TableView

Sucesos

Seleccione un tipo para visualizar la lista de métodos y procedimientos. Para cada método y procedimiento aparecerán los campos sintaxis, descripción y un ejemplo.

ActionEvent	MouseEvent
ErrorEvent	MoveEvent
Event	StatusEvent
KeyEvent	TimerEvent
MenuEvent	ValueEvent

Tipo DataBase

Database proporciona un gestor para una base de datos (un directorio). Cuando se ejecuta una aplicación de Paradox, Paradox abre la base de datos por defecto (directorio de trabajo). La base de datos por defecto contiene la vía de acceso al directorio de trabajo actual. Si lo que se desea es trabajar con dichas tablas, no es necesario abrir ninguna otra base de datos. Para trabajar con tablas almacenadas en otro lugar, declare una variable Database y utilice una sentencia open para crear un gestor para otra base de datos (sería posible especificar la vía de acceso completa a cada tabla cada vez que se va a usar, pero es más fácil mantener el código que utiliza variables Database).

Utilizando **open** y un alias, puede especificarse qué base de datos se abre, como se muestra en el ejemplo siguiente:

```
var
   InfoCliente Database
endVar
; addAlias está definido en el tipo Session
addAlias("InfoClientes", "Standard", "D:\\pdoxwin\\tables\\datclien")
InfoCliente.open("InfoClientes") ; abre la base de datos InfoClientes
                            ; InfoClientes debe ser un alias válido
```

Paradox conoce ahora dos bases de datos: la base de datos por defecto y InfoClientes. La variable *InfoCliente* es un *gestor* para la base de datos InfoClientes es decir, es posible emplear *InfoCliente* en sentencias para referenciar la base de datos InfoClientes . Por ejemplo, supóngase que hay dos archivos llamados PEDIDOS.DB (uno en el directorio de trabajo y otro en InfoClientes) y que se desea averiguar si estos archivos son tablas. El ejemplo siguiente comprueba, en primer lugar, PEDIDOS.DB en el directorio de trabajo y, a continuación, emplea *InfoCliente* como gestor para la base de datos InfoClientes y comprueba PEDIDOS.DB ahí:

```
var
   InfoCliente Database
endVar
addAlias("InfoClientes", "Standard", "D:\\pdoxwin\\tables\\datclien")
InfoCliente.open("InfoClientes")

if isTable("Pedidos.db") then ; comprobamos PEDIDOS.DB en la base
             ; de datos por defecto
msgInfo("Directorio de Trabajo", "PEDIDOS.DB es una tabla.")
endIf

if InfoCliente.isTable("Pedidos.db") then   ; utilizamos MiBD como apuntador
a
                        ; la base de datos InfoClientes
   msgInfo("InfoClientes", "PEDIDOS.DB es una tabla.")
endIf
```

Si se utiliza open pero no se especifica una base de datos, Paradox supone que se desea un gestor para la base de datos por defecto. Por ejemplo, esta sintaxis proporciona un gestor para la base de datos por defecto, que podría pasarse a un método personalizado que precisa un gestor de base de datos.

```
var BDPorDefecto Database endVar
BDPorDefecto.open() ; abre la base de datos por defecto
```

El uso de un gestor para la base de datos por defecto también puede hacer más legible el

código, especialmente cuando se trabaja con varias bases de datos al mismo tiempo.

close
delete
executeQBE
executeQBEFile
executeQBEString
isAssigned
isTable
open
writeQBE

Tipo Query

Una variable Query de ObjectPAL representa una consulta QBE. Es posible utilizar ObjectPAL para crear y ejecutar consultas desde métodos de la misma forma que si se estuviese empleando Paradox interactivamente. Una consulta puede ejecutarse desde un archivo de consulta, una sentencia de consulta o una cadena.

La mayoría de los métodos para trabajar con consultas se definen para el tipo Database, puesto que en ciertas aplicaciones será necesario especificar la base de datos que contiene las tablas que se desean consultar.

executeQBE
isAssigned
query
writeQBE

Tipo Table

Una variable Table representa una descripción de una tabla. Es diferente de un TCursor, que es un puntero a los datos, y de un marco de tabla y un TableView, que son objetos que muestran datos.

Utilizando objetos Table, es posible añadir, copiar, crear e indexar tablas, realizar cálculos en columnas, obtener información sobre la estructura de una tabla y otras operaciones, pero no es posible editar registros. Emplee un TCursor o un marco de tabla (UIObject) para ello.

add
attach
cAverage
cCount
cMax
cMin
cNpv
compact
copy
create
cSamStd
cSamVar
cStd
cSum
cVar
delete
dropIndex
empty
enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct
enumRefIntStruct
enumSecStruct
familyRights
fieldName
fieldNo
fieldType
index
isAssigned
isEmpty
isEncrypted
isShared
isTable
lock
nFields
nKeyFields
nRecords
protect
reIndex
reIndexAll
rename
setExclusive
setFilter
setIndex

setReadOnly
showDeleted
sort
subtract
tableRights
type
unAttach
unlock
unProtect
usesIndexes

Tipo TCursor

Un TCursor es un puntero a los datos de una tabla y permite manipular los datos sin tener que mostrar la tabla. No es una copia de la tabla la edición de registros en un TCursor cambia la tabla subyacente, y los bloqueos aplicados a la tabla afectan al TCursor .

Para información sobre objetos relacionados, consulte los tipos Table y TableViewTable.

add
atFirst
atLast
attach
attachToKeyViol
bot
cancelEdit
cAverage
cCount
close
cMax
cMin
cNpv
compact
copy
copyFromArray
copyRecord
copyToArray
cSamStd
cSamVar
cStd
cSum
currRecord
cVar
deleteRecord
didFlyAway
dropIndex
edit
empty
end
endEdit
enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct
enumLocks
enumRefIntStruct
enumTableProperties
eot
familyRights
fieldNo
fieldRights
fieldSize
fieldType
fieldUnits2
fieldValue
getLanguageDriver
getLanguageDriverDesc

Tipo DDE

El intercambio dinámico de datos (DDE) es un protocolo Windows que permite que Paradox comparta datos con otras aplicaciones que se comportan según el protocolo DDE. Mediante los métodos de DDE, es posible acceder a datos creados y almacenados en otra aplicación, así como enviar comandos y datos a la otra aplicación.

Nota: Paradox y ObjectPAL también permiten el uso de OLE, otro protocolo para compartir datos entre aplicaciones.

execute
open
setItem

Tipo FileSystem

Las variables FileSystem proporcionan el acceso a archivos en disco, unidades y directorios, e información sobre los mismos. Una variable FileSystem proporciona un gestor, variable que puede utilizarse en sentencias de ObjectPAL para trabajar con un directorio o archivo. En muchos casos, el primer paso al trabajar con variables FileSystem es utilizar **findFirst** para comprobar si hay alguna información. Puede ser útil considerar este paso como la inicialización de la variable FileSystem.

accessRights
copy
delete
deleteDir
drives
enumFileList
existDrive
findFirst
findNext
size
freeDiskSpace
fullName
getDir
getDrive
getFileAccessRights
getValidFileExtensions
isDir
isFile
isFixed
isRemote
isRemovable
makeDir
name
privDir
rename
setDir
setDrive
setFileAccessRights
splitFullFileName
startUpDir
time
totalDiskSpace
windowsDir
windowsSystemDir
workingDir

Tipo Library

Una biblioteca es un objeto de Paradox que contiene métodos personalizados, procedimientos personalizados, variables, tipos de datos definidos por el usuario y constantes. Las bibliotecas son útiles para almacenar y mantener rutinas empleadas frecuentemente y para compartir variables y métodos personalizados entre varias fichas.

En muchos sentidos, trabajar con una biblioteca equivale a trabajar con una ficha. Por ejemplo, para crear una ficha, elija Archivo|Nuevo|Ficha; para crear una biblioteca, elija Archivo|Nuevo|Biblioteca. Al igual que una ficha, una biblioteca tiene métodos estándar. El código se añade en una biblioteca igual que en una ficha, utilizando el cuadro de diálogo Métodos y el Editor de ObjectPAL. Al igual que en una ficha, es posible abrir ventanas del Editor para declarar métodos personalizados, procedimientos, variables, constantes, tipos de datos y rutinas externas.

close
enumSource
enumSourceToFile
execMethod
open

Tipo Session

Un objeto Session representa un canal con el Engine de base de datos. La apertura de una aplicación de Paradox abre una sesión por defecto, y es posible utilizar ObjectPAL para abrir otras sesiones desde dentro de una aplicación; no es necesario abrir otras sesiones para utilizar procedimientos del tipo Session. El número de sesiones que pueden abrirse depende del entorno del sistema. Cada sesión utiliza un contador de usuario.

Sólo la sesión por defecto puede gestionarse interactivamente mediante Paradox.. Las demás sesiones deben gestionare con ObjectPAL.

Los bloqueos definidos por ObjectPAL interactúan como iguales con los definidos interactivamente en la misma sesión.

addAlias
addPassword
advancedWildcardsInLocate
blankAsZero
close
enumDataBaseTables
enumDriverCapabilities
enumDriverInfo
enumDriverNames
enumDriverTopics
enumEngineInfo
enumFolder
enumOpenDatabases
enumUsers
getAliasPath
getNetUserName
ignoreCaseInLocate
isAdvancedWildcardsInLocate
isAssigned
isBlankZero
isIgnoreCaseInLocate
lock
open
removeAlias
removeAllPasswords
removePassword
retryPeriod
saveCFG
setAliasPath
setRetryPeriod
unLock

Tipo System

El tipo System contiene procedimientos para mostrar mensajes, averiguar datos acerca del ordenador del usuario, manipular el cuadro de diálogo Examinar, trabajar con el sistema de ayuda, y más.

beep
close
constantNameToValue
constantValueToName
cpuClockTime
debug
dlgAdd
dlgCopy
dlgCreate
dlgDelete
dlgEmpty
dlgNetDrivers
dlgNetLocks
dlgNetRefresh
dlgNetRetry
dlgNetSetLocks
dlgNetSystem
dlgNetUserName
dlgNetWho
dlgRename
dlgRestructure
dlgSort
dlgSubtract
dlgTableInfo
enumDesktopWindowNames
enumFonts
enumFormNames
enumReportNames
enumRTLClassNames
enumRTLConstants
enumRTLMethods
enumWindowNames
errorClear
errorCode
errorLog
errorMessage
errorPop
errorShow
errorTrapOnWarnings
execute
exit
fail
fileBrowser
formatAdd
formatDelete
formatExist
formatSetCurrencyDefault
formatSetDateDefault
formatSetDateTimeDefault

formatSetLogicalDefault
formatSetLongIntDefault
formatSetNumberDefault
formatSetSmallIntDefault
formatSetTimeDefault
getMouseScreenPosition
helpOnHelp
helpQuit
helpSetIndex
helpShowContext
helpShowIndex
helpShowTopic
helpShowTopicInKeywordTable
message
msgAbortRetryIgnore
msgInfo
msgQuestion
msgRetryCancel
msgStop
msgYesNoCancel
pixelsToTwips
play
readEnvironmentString
readProfileString
setMouseScreenPosition
setMouseShape
sleep
sound
sysInfo
tracerClear
tracerHide
tracerOff
tracerOn
tracerSave
tracerShow
tracerToTop
tracerWrite
version
winGetMessageID
winPostMessage
winSendMessage
writeEnvironmentString
writeProfileString

Tipo TextStream

Un TextStream es una secuencia de caracteres leídos de un archivo de texto (o escritos en él). Un TextStream sólo contiene caracteres ANSI; no se incluye información de formato, como fuente, alineación y márgenes. Sin embargo, sí se incluyen los caracteres no imprimibles, como retorno de carro y avance de línea (CR/LF).

Paradox mantiene un puntero de posición en el archivo que se comporta como un cursor de punto de inserción en un procesador de textos. El puntero indica la distancia (el número de caracteres) a que se está desde el principio del archivo. La numeración comienza en 1 (no en cero, como en otros lenguajes).

advMatch
close
commit
create
end
eof
home
open
position
readChars
readLine
setPosition
size
writeLine
writeString

Tipo AnyType

Un valor AnyType puede ser cualquiera de los tipos de datos enumerados en la tabla siguiente.

Tipo	Descripción
AnyType	Un tipo de datos que engloba los tipos de datos básicos
Array	Conjunto indexado de datos
Binary	Datos leíbles por el ordenador
Currency	Utilizado para manipular valores monetarios
Date	Datos de fecha
DateTime	Datos de fecha y hora combinados
DynArray	Matriz dinámica
Graphic	Imagen de mapa de bits
Logical	True o False
LongInt	Utilizado para representar valores enteros relativamente grandes
Memo	Alberga mucho texto
Number	Valores de coma flotante
OLE	Un vínculo con otra aplicación
Point	Información sobre una posición en la pantalla
Record	Record, Estructura definida por el usuario
SmallInt	Utilizado para representar valores enteros relativamente pequeÀ9_ÀosUsed to represent relatively small integer values
String	LetrasLetters
Time	Datos horariosClock data

Un AnyType no puede ser un tipo complejo como TCursor o TextStream. Una variable AnyType hereda las características del valor que se le ha asignado. Es decir, se comporta como un String cuando se le asigna un valor String, como un Number cuando se le asigna un valor Number, etc.

Los objetos de datos AnyType se incluyen en ObjectPAL para que puedan usarse variables para los tipos de datos básicos sin necesidad de declararlos antes (recuerde, no obstante, que es mejor declarar las variables siempre que sea posible).

blank
dataType
isAssigned
isBlank
isFixedType
view

Tipo Array

Un Array (matriz) contiene valores (llamados *elementos*) en *celdas* similares a los casilleros de correos que contienen cartas. Una matriz de ObjectPAL es unidimensional, como una sola fila de casilleros donde cada uno contiene un solo elemento. El tipo Array también incluye métodos definidos para el tipo AnyType. Consulte el tipo AnyType para más información.

Para utilizar matrices en los métodos, es necesario declararlas especificando un nombre, tamaño (número de elementos) y tipo de datos de los elementos.

Nota: En ObjectPAL, los elementos de una matriz se cuentan comenzando en 1, no en 0, como en otros lenguajes.

Nota: ObjectPAL también admite matrices dinámicas. Consulte también los métodos y procedimientos del tipo DynArray.

addLast
append
contains
countOf
empty
exchange
fill
grow
indexOf
insert
insertAfter
insertBefore
insertFirst
isResizeable
remove
removeAllItems
removeItem
replaceItem
setSize
size
view

Tipo Binary

Un objeto binario (en ocasiones denominado objeto binario grande o BLOB) contiene datos que sólo los ordenadores pueden leer e interpretar. Un ejemplo de un objeto binario es un archivo de sonido: una persona no puede leer ni interpretar el archivo tal cual, pero un ordenador sí.

Cuando se declara una variable Binary, se crea un gestor para un objeto binario, que puede referenciarse en el código para desplazar datos binarios entre un archivo en disco y una tabla, o desde un archivo en disco o una tabla a un método o procedimiento.

El tipo Binary también incluye métodos definidos para el tipo AnyType.Consulte el tipo AnyType para más información.

readFromFile
size
writeToFile

Tipo Currency

Los valores de Currency pueden hallarse entre 3,4 * ± 10-4930 to 1,1 * ± 104930 con una precisión de seis cifras decimales. El número de cifras decimales que se muestra depende de los valores definidos por el usuario en el Panel de control, pero no así el valor contenido en una tabla, ya que ésta almacena las seis cifras decimales. El tipo Currency también incluye métodos definidos para el tipo AnyType y para el tipo Number. Consulte los tipos AnyType y Number para más información.

Currency

Tipo Date

En ObjectPAL, los valores de fecha pueden representarse en los formatos mes/día/año, día-Mes-año o día.mes.año. Las fechas deben declararse explícitamente. Por ejemplo,

```
Var
   Fecha Date
endVar
Fecha = date ("12/21/1997")
```

asigna a *Fecha* la fecha 21 diciembre 1997. No omita las comillas alrededor del valor de fecha si lo hace, ObjectPAL realiza la división de los valores .

El tipo Date incluye métodos definidos para el tipo AnyType y para el tipo DateTime.Consulte los tipos AnyType y DateTime para más información.

El formato de los valores de Date se especifica mediante el método **formatSetDateDefault** (tipo System) o mediante las sentencias de formateo de ObjectPAL.

Aunque es posible utilizar ObjectPAL para realizar cálculos sobre cualquier fecha válida, los valores de fecha contenidos en una tabla de Paradox deben hallarse entre el 1 de enero de 100 y 31 de diciembre de 9999.

Las fechas del siglo 20 pueden especificarse utilizando dos dígitos para el año, como en

```
Día = date ("11/09/59")
```

Las fechas de los siglos segundo a décimo deben incluir tres dígitos para el año (como en 12/17/243); las fechas de los siglos undécimo a decimonoveno deben tener cuatro dígitos (12/17/1043). El año no puede omitirse completamente.

El tipo Date incluye varios métodos definidos para el tipo DateTime. Para más información, consulte el tipo DateTime.

date
dateVal
today

Tipo DateTime

Una variable DateTime contiene datos en la forma hora-minuto-segundo-milisegundo año-mes-día. Los valores de DateTime sólo se utilizan en los cálculos de ObjectPAL; no es posible almacenar un valor de DateTime en una tabla de Paradox. Los valores de DateTime deben declararse explícitamente. Por ejemplo, las sentencias siguientes asignan a la variable DateTime *FH* una hora de 11 horas, 10 minutos y 40 segundos y una fecha de 21 de diciembre de 1997.

```
var FH DateTime endVar
FH = DateTime ("11:10:40 am 12/21/97")
```

Las comillas que delimitan el valor son obligatorias.

Es posible utilizar los caracteres siguientes como separadores: blanco, tabulador, espacio, coma (,), guión (-), barra inclinada (/), punto (.), dos puntos (:) y punto y coma (;). El formato de los valores de DateTime depende de la especificación definida por el método **formatSetDateTimeDefault** (tipo System) o por las sentencias de formateo de ObjectPAL.

Un valor de DateTime debe especificarse completamente; no es posible omitir ninguno de los campos, pero sí especificar un valor de cero para cualquier campo.

Consulte también los métodos y procedimientos definidos para el tipo Date y para el tipo Time.

El tipo DateTime incluye métodos definidos para el tipo AnyType. Consulte el tipo AnyTypepara más información. Además, tanto el tipo Date como el tipo Time incluyen métodos definidos para el tipo DateTime.

dateTime
day
daysInMonth
dow
dowOrd
doy
hour
isLeapYear
milliSec
minute
month
moy
second
year

Tipo DynArray

Un DynArray es una matriz dinámica estructurada flexiblemente. Una matriz dinámica es una estructura de almacenamiento compacta para cualquier combinación de tipos de datos. Mediante un DynArray, es posible consultar valores rápidamente, incluso cuando la matriz dinámica contiene un gran número de elementos.

Estas matrices son dinámicas porque no se especifica su tamaño; las dimensiones de un DynArray cambian automáticamente conforme se añaden o se eliminan elementos. El tamaño de un DynArray sólo está limitado por la memoria del ordenador.

Nota: ObjectPAL también permite el uso de matrices de tamaño fijo y redimensionables. Para más información, consulte la descripción del tipo Array.

A diferencia de las matrices de tamaño fijo, los índices de las matrices dinámicas no son números enteros; pueden ser cualquier expresión de ObjectPAL válida que se evalúe en un String. Cada índice de una matriz dinámica está asociado con un valor.

El tipo DynArray también incluye métodos definidos para el tipo AnyType.Consulte el tipo AnyType para más información.

contains
getKeys
removeItem
size
view

Tipo Graphic

Una variable Graphic proporciona un gestor para manipular un objeto de imagen. Es decir, las variables Graphic pueden utilizarse en código de ObjectPAL para manipular objetos gráficos. Los objetos gráficos contienen y muestran imágenes en el formato de mapa de bits (BMP). Sin embargo, Paradox puede importar los siguientes formatos gráficos: mapa de bits (BMP), Postscript encapsulado (EPS), formato de intercambio gráfico (GIF), Paintbrush (PCX) y formato de archivo con información etiquetada (TIF).

Mediante los métodos **readFromClipboard**, **writeToClipboard**, **readFromFile** y **writeToFile** del tipo Graphic, es posible utilizar variables Graphic para transferir mapas de bits entre fichas (e informes), tablas, el Portapapeles y archivos en disco.

El tipo Graphic también incluye métodos definidos para el tipo AnyType.Consulte AnyType para más información.

readFromClipboard
readFromFile
writeToClipboard
writeToFile

Tipo Logical

Las variables Logical tienen dos valores posibles: True o False. En lugar de True, pueden utilizarse las constantes de ObjectPAL Yes u On, y en lugar de False, las constantes No u Off.

Una variable Logical ocupa 1 byte de espacio. Por orden de prioridad, los operadores lógicos son NOT, AND y OR.

Las variables Logical suelen contestar preguntas sobre otros objetos y operaciones, por ejemplo:

¿Está vacía esa tabla?

¿Esa ficha se muestra como un icono?

¿Pudo esa operación crear un archivo de texto?

El tipo Logical también incluye métodos definidos para el tipo AnyType.

logical

Tipo LongInt

Los valores de LongInt son números enteros grandes; es decir, pueden representarse mediante una serie larga de dígitos. Una variable LongInt ocupa 4 bytes.

ObjectPAL convierte los valores de LongInt al rango de -2.147.483.648 a 2.147.483.647. Un intento de asignar un valor fuera de este rango a una variable LongInt provoca un error. Por ejemplo,

```
var
  x, y, z LongInt
endVar

x = 2147483647   ; el número máximo que se puede utilizar con
             ; una variable de tipo Longint
y = 1
z = X + y        ; causa un error
```

Para trabajar con valores fuera de estos límites, almacene el resultado en una variable Number.

Nota: Los métodos de biblioteca run-time definidos para el tipo Number también funcionan con variables LongInt. La sintaxis es la misma, y el valor devuelto es un número. El tipo LongInt también incluye métodos definidos para el tipo AnyType.

Consulte el tipo AnyType para más información.

bitAND
bitIsSet
bitOR
bitXOR
LongInt

Tipo Memo

Los memos contienen texto y datos de formato hasta 512MB en las tablas de Paradox. Mediante los métodos **readFromFile** y **writeToFile** del tipo Memo, es posible transferir memos entre fichas (o informes), tablas y archivos.

También puede utilizarse el operador = para asignar el valor de un campo memo a una variable Memo o una variable String.

Nota: No existen operadores aritméticos ni de comparación para las variables Memo.

Si se asigna un campo memo a una variable String, sólo se obtiene el texto memo sin formateo. Si se asigna un campo memo a una variable Memo, se obtiene el texto y el formato.

El tipo Memo también incluye métodos definidos para el tipo AnyType.

Consulte el tipo AnyType para más información.

readFromFile
memo
writeToFile

Tipo Number

Las variables Number representan valores de coma flotante que constan de un significando (parte fraccionaria, por ejemplo, 3,224) multiplicado por una potencia de 10. El significando puede contener hasta 18 dígitos significativos, y la potencia de 10 puede encontrarse entre ± 3,4 * 10 -4930 to ± 1,1 * 10 4930. Un intento de asignar un valor fuera de este rango a una variable Number provoca un error.

Nota: El tipo Number incluye métodos definidos para el tipo AnyType.Consulte el tipo AnyType para más información. Además, los métodos y procedimientos de biblioteca run-time definidos para el tipo Number también funcionan con las variables LongInt y SmallInt. La sintaxis es la misma y el valor devuelto es un Number. Por ejemplo, el código siguiente funciona, incluso aunque **sin** no aparezca en la lista de métodos del tipo LongInt:

```
var
  abc LongInt
  xyz Number
endVar
abc = 43
xyz = abc.sin()
```

Nota: ObjectPAL permite una sintaxis alternativa:

nombreMétodo (***varObj***, *argumento* [,*argumento*])

nombreMétodo representa el nombre del método, ***varObj*** es la variable que representa a un objeto y *argumento* representa uno o más argumentos. Por ejemplo, la sentencia siguiente utiliza la sintaxis estándar de ObjectPAL para devolver el seno de un número:

```
ElNúmero.sin()
```

La sentencia siguiente emplea la sintaxis alternativa:

```
sin (ElNúmero)
```

Es recomendable utilizar la sintaxis estándar por razones de claridad y coherencia, pero es posible usar la alternativa cuando sea conveniente.

Nota: Los formatos de visualización de los métodos de Number pueden variar según el formato numérico de Windows definido en el ordenador del usuario, pero la representación interna de ObjectPAL es siempre la misma.

numVal
pmt
pow
pow10
pv
rand
round
sinh
sin
sqrt
tanh
tan
truncate

Tipo OLE

OLE es un acrónimo del inglés *Object Linking and Embedding* (vínculo e incrustación de objetos), protocolo que proporciona acceso a la funcionalidad de otra aplicación sin tener que salir de Paradox y abrir la aplicación cada vez que se desea realizar un cambio.

Por ejemplo, suponga que tiene tablas que contienen imágenes de mapa de bits y desea crear una aplicación Paradox que permita a los usuarios editar dichas imágenes. Un planteamiento es crear la imagen utilizando un programa de dibujo que sea servidor de OLE (concepto definido más abajo) y utilizar los métodos del tipo OLE de ObjectPAL para permitir que los usuarios dispongan de la funcionalidad del programa de dibujo (suponiendo, por supuesto, que los usuarios tengan dicho programa instalado en sus ordenadores).

El tipo OLE también incluye métodos definidos para el tipo AnyType. Consulte el tipo AnyType para más información.

Nota: ObjectPAL y Paradox también permiten el uso de DDE (intercambio dinámico de datos), otro protocolo para compartir datos.

Los términos siguientes se utilizan al explicar operaciones OLE:

Un *servidor* de OLE es una aplicación que puede proporcionar acceso a sus documentos a través del mecanismo OLE. Paradox no es un servidor de OLE.

Un *cliente* de OLE es una aplicación que puede utilizar el mecanismo OLE para acceder a documentos creados por un servidor de OLE. Paradox es un cliente de OLE.

Un *objeto* OLE es un documento creado utilizando un servidor de OLE. Contiene los datos que se desean utilizar en la aplicación Paradox.

Una *variable* OLE es una variable de ObjectPAL declarada como perteneciente al tipo OLE. Una variable OLE proporciona un gestor para manipular un objeto OLE. En otras palabras, es posible utilizar variables OLE en código ObjectPAL para manipular objetos OLE.

canReadFromClipboard
edit
enumVerbs
getServerName
readFromClipboard
writeToClipboard

Tipo Point

Una variable Point contiene información sobre un punto en la pantalla. Para ObjectPAL, la pantalla es una cuadrícula bidimensional, con el origen situado en el ángulo superior izquierdo del contenedor del objeto de diseño, extendiéndose los valores *x* positivos hacia la derecha y los valores *y* positivos hacia abajo. Un Point tiene un valor *x* y un valor *y*, donde *x* e *y* se miden en twips (un twip es 1/1440 de pulgada, 1/20 de un punto de la impresora).

Los métodos definidos para el tipo Point obtienen y definen información sobre las coordenadas de pantalla y las posiciones relativas de los puntos. Por ejemplo, las propiedades de tamaño y posición de un objeto de diseño se especifican en puntos.

Nota: ObjectPAL calcula los valores de Point respecto al contenedor del objeto de diseño en cuestión. Por ejemplo, si un cuadro contiene un botón, ObjectPAL calcula la posición del botón respecto al cuadro. Si el botón se halla en una página vacía, ObjectPAL calcula su posición respecto a la página. Los métodos que toman o devuelven valores Point como argumentos utilizan este marco relativo.

El tipo Point también incluye métodos definidos para el tipo AnyType. Para más información, consulte la sección AnyType.

distance
isAbove
isBelow
isLeft
isRight
point
setX
setXY
setY
x
y

Tipo Record

ObjectPAL proporciona el tipo Record como un conjunto programático y definido por el usuario de información, similar a un **record** de Pascal o a un **struct** de C. Los registros definidos en código de ObjectPAL son distintos y están aparte de los registros asociados con una tabla.

Esta es la sintaxis para declarar un tipo de datos Record:

```
type
NombreRegistro = record
                     NombreCampo TipoCampo
                   [  NombreCampo    TipoCampo ]*
                 endRecord
endType
```

Uno o más *nombreCampo* identifican a los campos (columnas) del registro y *tipoCampo* es uno de los tipos de datos. Declare los registros en la ventana Type de un objeto de diseño.

Una vez que se declara un tipo de datos Record, es posible utilizar los operadores de comparación = y <> para comparar un registro con otro. También puede utilizarse el operador de asignación (=) para copiar el contenido de un registro en otro.

El tipo Record también incluye métodos definidos para el tipo AnyType. Consulte el tipo AnyType para más información.

view

Tipo SmallInt

Los valores SmallInt son números enteros pequeños; es decir, pueden representarse mediante una serie pequeña de dígitos. Una variable SmallInt ocupa 2 bytes de espacio.

ObjectPAL convierte los valores SmallInt al rango de -32.768 a 32.767. Un intento de asignar un valor fuera de este rango a una variable SmallInt provoca un error. Por ejemplo,

```
var
   x, y, z SmallInt
endVar
x = 32767 ; el valor límite superior de una variable SmallInt
y = 1
z = x + y ; causa un error
```

Para trabajar con valores fuera de estos límites, almacene el resultado en una variable de un tipo que pueda contenerlos. Por ejemplo,

```
var
   x, y SmallInt
   z    LongInt   ; declaramos z de tipo LongInt, de modo que pueda
almacenar
                  ; el resultado
endVar

x = 32767 ; el valor límite superior de una variable SmallInt
y = 1
z = x + y

z.view() ; muestra 32768 porque z es de tipo LongInt
         ; y puede manejar valores grandes
```

El valor de SmallInt -32.768 no puede almacenarse en una tabla de Paradox porque, para Paradox, -32.768 = Vacío. Sin embargo, es posible utilizar este valor en los cálculos y almacenarlo en una tabla de dBASE.

Nota: El tipo SmallInt incluye métodos definidos para el tipo AnyType. Consulte la sección AnyType para más información. Además, los métodos y procedimientos de biblioteca run-time definidos para el tipo Number también funcionan con variables LongInt y SmallInt. La sintaxis es la misma, y el valor devuelto es un Number. Por ejemplo, el código siguiente funcionará, aunque **sin** no aparezca en la lista de métodos del tipo SmallInt:

```
var
   abc LongInt
   xyz Number
endVar
abc = 43
xyz = abc.sin()
```

Nota: ObjectPAL permite una sintaxis alternativa:

nombreMétodo (***varObj***, *argumento* ,*argumento*])

nombreMétodo representa el nombre del método, ***varObj*** es la variable que representa a un objeto y argumento representa uno o más argumentos. Por ejemplo, la sentencia siguiente utiliza la sintaxis estándar de ObjectPAL para devolver el seno de un número:

```
ElNúmero.sin()
```

La sentencia siguiente emplea la sintaxis alternativa:

```
sin(ElNúmero)
```

Es recomendable que se utilice la sintaxis estándar por razones de claridad y coherencia, pero es posible emplear la alternativa donde sea conveniente.

bitAND
bitIsSet
bitOR
bitXOR
int
smallInt

Tipo String

Una variable String para cadenas vacías puede contener hasta 32.000 caracteres (utilice objetos Memo para texto más amplio). Una cadena entrecomillada puede contener hasta 255 caracteres. Utilice dos comillas ( ) para representar una cadena vacía. Las variables String ocupan 1 byte de espacio por carácter.

El tipo String también incluye métodos definidos para el tipo AnyType. Para más información, consulte la sección AnyType.

Nota: ObjectPAL admite una sintaxis alternativa:

nombreMétodo (V*arObj, argumento* [,*argumento*])

nombreMétodo representa el nombre del método, *varObj* es la variable que representa a un objeto y *argumento* representa a uno o más argumentos. Por ejemplo, la sentencia siguiente emplea la sintaxis estándar de ObjectPAL para devolver una versión en minúsculas de una cadena:

```
Cadena.lower()
```

La sentencia siguiente emplea la sintaxis alternativa:

```
lower(Cadena)
```

Es recomendable que se utilice la sintaxis estándar por razones de claridad y coherencia, pero es posible emplear la alternativa donde sea conveniente.

advMatch
ansiCode
breakApart
chr
chrOEM
chrToKeyName
fill
format
ignoreCaseInStringCompares
isIgnoreCaseInStringCompares
isSpace
keyNameToChar
keyNameToVKCode
lower
lTrim
match
oemCode
rTrim
search
size
space
string
strVal
substr
toANSI
toOEM
upper
vkCodeToKeyName

Tipo Time

Las variables Time almacenan valores horarios en la forma horas-minutos-segundos-milisegundos. Es posible emplear los caracteres siguientes como separadores: blanco, tabulador, espacio, coma (,), guión (-), barra inclinada (/), punto (.), dos puntos (:) y punto y coma (;).

Los valores de Time deben declararse explícitamente. Por ejemplo, las sentencias siguientes asignan el valor de 11 horas, 10 minutos y 40 segundos de la mañana a la variable Time Hora.

```
var
  Hora Time
endVar
Hora = Time ( 11:10:40 am )
```

Las comillas que rodean el valor son necesarias. La validez de una hora depende del formato de hora actual de Paradox. Por ejemplo, si el formato de hora está definido para 12 horas (como hh:mm:ss), los métodos del tipo Time consideran hh:mm:ss un formato de hora válido. Para definir los formatos de hora de Paradox con ObjectPAL, utilice **formatSetTimeDefault** y **formatSetDateTimeDefault** del tipo System.

El tipo Time incluye varios métodos definidos para los tipos AnyType y DateTime. Para más información y ejemplos, consulte las secciones DateTime y AnyType.

time

Tipo Menu

Un objeto Menu es una lista de elementos que aparece en la barra de menú de la aplicación. Cuando el usuario elige una opción en un menú, se devuelve el texto de esa opción. Los menús creados en ObjectPAL sustituyen completamente a los menús estándar de Paradox (pero pueden recuperarse utilizando **removeMenu**).

Por defecto, los menús no persisten de una ficha a otra; cada ficha tiene su propio sistema de menús asociado con ella. Si se crea un menú para una ficha, el menú sólo aparece cuando esa ficha está activa. Si posteriormente abre una segunda ficha, ésta utiliza los menús estándar, no los creados para la primera ficha. Si se crea un menú personalizado para cada ficha, es posible simular menús relativos al contexto en una aplicación.

Si desea que dos (o más) fichas utilicen el mismo menú personalizado, defina la propiedad StandardMenu de cada ficha como Off. Esto instruye a Paradox para que mantenga el menú actual cuando el usuario se desplaza de una ficha a otra. Es posible emplear la propiedad StandardMenu para construir un solo sistema de menús para toda una aplicación.

addArray
addBreak
addPopUp
addStaticText
addText
contains
count
empty
getMenuChoiceAttribute
getMenuChoiceAttributeById
hasMenuChoiceAttribute
remove
removeMenu
setMenuChoiceAttribute
setMenuChoiceAttributeById
show

Tipo PopupMenu

Un PopUpMenu es una lista de opciones que se muestran en vertical en respuesta a un Event (normalmente, un clic del ratón). Cuando el usuario elige una opción en un menú emergente, el texto de esa opción se devuelve al método.

Un PopUpMenu es distinto de un Menu, que es una lista de opciones que se muestra en horizontal en la barra de menús de la aplicación, si bien el tipo PopUpMenu incluye métodos definidos para el tipo Menu.

Nota: La elección de una opción en un menú emergente no activa el método estándar **menuAction**, a menos que el menú emergente se anexe a un método personalizado.

Mediante los métodos de PopUpMenu, es posible

Crear un menú emergente

Mostrar el menú emergente y devolver la opción seleccionada

Inspeccionar las opciones de un menú emergente

Proporcionar acceso al teclado

Nota: Una aplicación típica utiliza tanto objetos Menu como objetos PopUpMenu. Consulte también la sección Menu en este capítulo.

addArray
addBar
addBreak
addPopUp
addSeparator
addStaticText
addText
show
switchMenu

Tipo UIObject

Los UIObject crean el interfaz de usuario para una aplicación: todo lo que puede situarse en una ficha es UIObject. Sólo los UIObject tienen métodos estándar. Los diferentes UIObject son: mapa de bits, cuadro, botón, cartesiano, elipse, objeto de campo, ficha, gráfico, línea, objeto multirregistro, objeto OLE, página, objeto de registro, marco de tabla y cuadro de texto.

Nota: La ficha se comporta como un UIObject: una ficha tiene métodos estándar, es posible anexar código a esos métodos estándar y una ficha responde a los sucesos. También hay otro tipo, Form, para los métodos y procedimientos que sólo funciona con fichas.

Muchos métodos de UIObject duplican a los métodos de TCursor. Los métodos de UIObject que funcionan con tablas trabajan sobre la tabla subyacente a través del objeto visible. Las acciones dirigidas al UIObject que afectan a la tabla son visibles inmediatamente en el objeto con que está asociada la tabla. Los métodos de TCursor, por contraste, trabajan con una tabla detrás del escenario; las acciones que afectan a la tabla no son visibles necesariamente en ningún objeto, incluso aunque el TCursor actúe sobre la misma tabla con que está asociado un objeto visible.

Algunas operaciones de tabla son bastante más rápidas con los TCursor que con los UIObject. Por ejemplo, si se precisa realizar una operación orientada a tabla que provocará una gran cantidad de actualizaciones de pantalla que podrían ser lentas , es posible emplear esta técnica: declarar un TCursor, anexar al objeto la tabla con que ya está asociado (por ejemplo, un marco de tabla), realizar la operación con el TCursor y resincronizar el objeto de visualización con el TCursor. Cuando se anexa un TCursor con un objeto asociado con una tabla, el puntero de registro del TCursor se sitúa en el registro actual del objeto. Después de realizar una operación de TCursor, como **locate**, el TCursor podría señalar a un registro diferente que el objeto. Para que el objeto señale al mismo registro que el TCursor, utilice **resync** o **moveToRecord**; para que el TCursor señale al mismo registro que el objeto, utilice el método **attach**. Consulte el ejemplo de **insertRecord**, en esta sección.

enumUIClasses
enumUIObjectNames
enumUIObjectProperties
execMethod
getBoundingBox
getPosition
getProperty
getPropertyAsString
getRGB
hasMouse
home
insertAfterRecord
insertBeforeRecord
insertRecord
isContainerValid
isEdit
isEmpty
isLastMouseClickedValid
isLastMouseRightClickedValid
isRecordDeleted
keyChar
keyPhysical
killTimer
locate
locateNext
locateNextPattern
locatePattern
locatePrior
locatePriorPattern
lockRecord
lockStatus
menuAction
mouseClick
mouseDouble
mouseDown
mouseEnter
mouseExit
mouseMove
mouseRightDouble
mouseRightDown
mouseRightUp
mouseUp
moveTo
moveToRecNo
moveToRecord
nextRecord
nFields
nKeyFields
nRecords
pixelsToTwips
postAction
postRecord
priorRecord
pushButton
recordStatus

resync
rgb
setFilter
setPosition
setProperty
setTimer
skip
switchIndex
twipsToPixels
unDeleteRecord
unlockRecord
view
wasLastClicked
wasLastRightClicked

Tipo Application

Una variable Application proporciona un gestor para trabajar con la ventana Escritorio de la aplicación Paradox actual. Es posible utilizar una variable Application en el código para controlar el tamaño, posición y aspecto del Escritorio. El tipo Application incluye métodos definidos para el tipo Form. Consulte el tipo Form para más información.

Aunque puede haber más de una aplicación ejecutándose al mismo tiempo, los objetos de Application no pueden comunicarse ni operar uno sobre otro. Una variable del tipo Application referencia sólo al escritorio actual de Paradox; sin embargo, es posible utilizar variables Session para abrir múltiples canales al Engine de base de datos (consulte el tipo Session type).

Puesto que sólo puede haber una aplicación actual, para obtener un gestor de aplicación, simplemente se declara una variable del tipo Application. Mientras que una variable Application esté dentro del ámbito, sirve como gestor: se utiliza esa variable para acceder a los métodos del tipo Application. Por ejemplo, en el código siguiente, se declara la variable Application llamada *estaAplic* y, a continuación, se utiliza en el código del método.

```
;reduceTamaño::pushButton
method pushButton(var eventInfo Event)
var
   estaAplic Application
   x, y, l, a LongInt
endVar
estaAplic.getPosition(x,y,l,a)                ; recoge la posición actual
estaAplic.setPosition(x,y,l*0.9,a*0.9)  ; reduce la ventana principal un 10%
endmethod
```

Tipo Form

Una variable Form proporciona un gestor para trabajar con una ficha de Paradox. Los métodos del tipo Form permiten:

- Cargar una ficha en una ventana de diseño y guardar un diseño
- Abrir y cerrar una ficha
- Realizar una anexión con una ficha abierta
- Añadir y eliminar tablas en el modelo de datos de una ficha
- Enumerar nombres de objeto, propiedades y código fuente para los métodos
- Determinar y cambiar la posición de una ficha, así como maximizarla o minimizarla
- minimize the forEnviar sucesos a una ficha
- Obtener y definir métodos para una ficha

El tipo Form es el tipo básico del que se derivan los demás tipos de gestor de visualización. Muchos de los métodos enumerados en esta sección también son utilizados por los tipos Application, Report y TableView.

action
attach
bringToTop
close
create
delayScreenUpdates
design
disableBreakMessage
dmAddTable
dmGet
dmHasTable
dmPut
dmRemoveTable
enumSource
enumSourceToFile
enumTableLinks
enumUIObjectNames
enumUIObjectProperties
formCaller
formReturn
getPosition
getTitle
hide
hideSpeedBar
isMaximized
isMinimized
isSpeedBarShowing
isVisible
keyChar
keyPhysical
load
maximize
menuAction
methodDelete
methodGet
methodSet
minimize
mouseDouble
mouseDown

mouseEnter
mouseExit
mouseMove
mouseRightDouble
mouseRightDown
mouseRightUp
mouseUp
moveToPage
open
openAsDialog
postAction
run
save
setPosition
setTitle
show
showSpeedBar
wait
windowClientHandle
windowHandle

Tipo Report

Una variable Report es un gestor de un informe. En otras palabras, es posible utilizar variables Report en el código para manipular el informe en pantalla y para mostrarlo e imprimirlo. Los métodos del tipo Report controlan el tamaño, posición y aspecto de la ventana.

Utilice **load** para cargar un archivo de informe en la ventana Diseñar informe; utilice **open** para abrir el informe en una ventana de diseño, y **print** para abrir un informe e imprimirlo. No es posible anexar métodos a los objetos de un informe.

El tipo Report incluye varios métodos definidos para el tipo Form. Para más información, consulte el tipo Form.

attach
close
currentPage
design
enumUIObjectNames
enumUIObjectProperties
load
moveToPage
open
print
run

Tipo TableView

Un objeto TableView muestra los datos de una tabla en su propia ventana. Un objeto TableView es distinto de un marco de tabla, que es un UIObject situado en una ficha, y de un TCursor, que señala a los datos de una tabla.

Cuando se declara una variable TableView y se abre una tabla usando esa variable, se crea un gestor a la ventana de tabla, algo que puede referenciarse en el código para manipular la ventana de tabla.

Los métodos de TableView son un subconjunto de los métodos del tipo Form. Es posible utilizarlos para controlar el tamaño, posición y aspecto de la ventana de tabla. Aunque es posible iniciar y terminar el modo editar para una ventana de tabla, no es posible utilizar ObjectPAL para editar directamente los datos de una ventana de tabla.

ObjectPAL puede utilizarse para manipular las propiedades de TableView en tres áreas principales:

Ventana de tabla en su conjunto (por ejemplo, color de fondo, estilo de cuadrícula y número de registros)

Datos a nivel de campo de la tabla (por ejemplo, fuente, color, formato de visualización y el valor del registro actual)

Cabecera de la ventana de tabla (por ejemplo, fuente, color y alineación)

Para ver una lista de propiedades de ventana de tabla, consulte las categorías TableView, TVData y TVHeading en el cuadro de diálogo Propiedades.

El tipo TableView incluye varios métodos definidos para el tipo Form.

Consulte el tipo Form para más información.

action
close
open
wait

Tipo ActionEvent

Los ActionEvent se generan principalmente mediante la edición y desplazamiento en una tabla. El tipo ActionEvent incluye varios métodos definidos para el tipo Event.

El único método estándar que un ActionEvent activa es **action**. Normalmente, cuando trabaje con los ActionEvent, también utilizará constantes de acción de ObjectPAL. Por ejemplo, para impedir que los usuarios editen una tabla, podría hacer algo como lo siguiente:

```
;recuadroTabla::action
method action (var eventInfo ActionEvent)
; si el usuario intenta cambiar a modo Edición, visualiza una caja de
diálogo          if eventInfo.id() = DataBeginEdit then ; DataBeginEdit es
una constante
   msgStop("Alto","Usted no puede editar esta tabla.")
   eventInfo.setErrorCode(1)
endif
endmethod
```

Las constantes de acción se enumeran en el cuadro de diálogo Constantes. Para mostrar la lista, abra una ventana del Editor de ObjectPAL y elija Lenguaje|Constantes. A continuación, en la columna Tipos de constantes, elija una que comience con Action, por ejemplo, ActionDataCommands. Las constantes se muestran en la columna Constantes.

Consulte el tipo Event para más información.

actionClass
id
setId

Tipo ErrorEvent

El tipo ErrorEvent proporciona métodos que el programador puede utilizar para obtener y definir información sobre errores que se producen al ejecutar código de ObjectPAL. El tipo ErrorEvent incluye varios métodos definidos para el tipo Event.

El único método estándar activado por un ErrorEvent es **error**.

reason
setReason

Tipo Event

El tipo Event es el tipo básico del que se derivan los demás tipos de sucesos (por ejemplo, ActionEvent). Muchos de los métodos enumerados en esta sección pueden ser utilizados por otros tipos de sucesos.

Los Event activan los métodos estándar siguientes: **open**, **close**, **setFocus**, **removeFocus**, **newValue** y **pushButton**.

errorCode
getTarget
isFirstTime
isPreFilter
isTargetSelf
reason
setErrorCode
setReason

Tipo KeyEvent

Un objeto KeyEvent obtiene y define información sobre sucesos de pulsación de teclas.

Los KeyEvent activan los métodos estándar siguientes: **keyChar** y **keyPhysical**.

El tipo KeyEvent incluye métodos definidos para el tipo Event. Consulte el tipo Event para más información.

char
charAnsiCode
isAltKeyDown
isControlKeyDown
isFromUI
isShiftKeyDown
setAltKeyDown
setChar
setControlKeyDown
setShiftKeyDown
setVChar
setVCharCode
vChar
vCharCode

Tipo MenuEvent

Las variables MenuEvent contienen datos relacionados con selecciones de menú en la barra de menús de una aplicación. Cuando el usuario elige una opción en un menú, activa el método **menuAction**. Modificando el método estándar **menuAction** de un objeto, es posible definir cómo responde el objeto.

El tipo MenuEvent incluye varios métodos definidos para el tipo Event. Consulte el tipo Event para más información.

data
id
isFromUI
menuChoice
reason
setData
setId
setReason

Tipo MouseEvent

Un objeto MouseEvent responde preguntas sobre el ratón, como

¿Dónde está el ratón?

¿Se ha hecho clic con un botón del ratón?

¿Con qué botón del ratón se hizo clic o se mantuvo pulsado durante una operación?

El tipo MouseEvent incluye varios métodos definidos para el tipo Event.

Los MouseEvents activan los métodos estándar siguientes: **mouseClick**, **mouseDown**, **mouseUp**, **mouseDouble**, **mouseRightUp**, **mouseRightDown**, **mouseRightDouble**, **mouseMove**, **mouseEnter** y **mouseExit**.

getMousePosition
getObjectHit
isControlKeyDown
isFromUI
isLeftDown
isMiddleDown
isRightDown
isShiftKeyDown
setControlKeyDown
setInside
setLeftDown
setMiddleDown
setMousePosition
setRightDown
setShiftKeyDown
setX
setY
x
y

Tipo MoveEvent

Los métodos del tipo MoveEvent permiten obtener y definir información sobre los sucesos que se producen al desplazarse el usuario de un objeto de una ficha a otro. El tipo MoveEvent incluye varios métodos definidos para el tipo Event. Consulte la sección Event para más información.

Los MoveEvent activan los siguientes métodos estándar: **arrive**, **canArrive**, **canDepart** y **depart**.

getDestination
reason
setReason

Tipo StatusEvent

Los métodos del tipo StatusEvent controlan los mensajes que se muestran en la barra de estado del Escritorio. El tipo StatusEvent incluye varios métodos definidos para el tipo Event.

Todos los objetos de diseño tienen un método estándar **status**, que es activado por un StatusEvent.

Mediante los métodos del tipo StatusEvent, es posible anexar código al método estándar para averiguar dónde y por qué se mostrarán los mensajes, y es posible impedirlos o mostrarlos en algún otro lugar: en un área de estado diferente o en otro objeto (por ejemplo, un objeto de campo o un archivo de texto). También puede especificarse el texto que se muestra en el mensaje.

Es posible utilizar las constantes de StatusWindows (ModeWindow1, ModeWindow2, ModeWindow3 y StatusWindow) para referenciar las áreas de la barra de estado.

reason
setReason
setStatusValue
statusValue

Tipo TimerEvent

Methods in the TimerEvent type process information used by the timer method built into every design object. The TimerEvent type includes several methods defined for the Event type. Refer to Eventidh_pobj_even for more information.

Los métodos del tipo TimerEvent proporcionan información utilizada por los métodos estándar **timer**. El tipo TimerEvent incluye varios métodos definidos para el tipo Event. Consulte la sección Eventidh_pobj_even para más información

Utilice **setTimer** del tipo UIObject para especificar cuándo se envían los sucesos de temporizador (TimerEvent) a un objeto, y a continuación modifique el método estándar **timer** del objeto para controlar cómo responde el objeto cuando se agota un temporizador. Utilice **killTimer**, definido para el tipo UIObject, para desactivar el temporizador de un objeto. El ejemplo siguiente muestra cómo utilizar estos métodos con los TimerEvent.

En este ejemplo, suponga que una ficha contiene un objeto multirregistro asociado con la tabla *Clientes*. El contenedor de registros del objeto multirregistro se llama *registroCliente*.

Suponga que dispone de un programa de introducción de datos y que desea dar al usuario 60 segundos para editar un registro. Después de ese tiempo, desea advertir al usuario. Para ello, el método **action** de *registroCliente* comprueba cada acción. Si la acción es DataArriveRecord, el método detiene cualquier temporizador antiguo con **killTimer** y define un nuevo temporizador para el objeto de registro con **setTimer**. Cuando el temporizador se agota, aparece un mensaje alertando al usuario. Para facilitar la modificación del intervalo del temporizador, se define una constante en la ventana Const de *registroCliente*, como sigue:

```
; registroCliente::Const
const
  HoraDeAlerta = 60000     ; Alerta de entrada de datos en 60 segundos
endConst
```

El código siguiente es para el método **action** de *registroCliente*.

```
; registroCliente::action
method action(var eventInfo ActionEvent)

if eventInfo.id() = DataArriveRecord then      ; cuando abrimos un nuevo
registro
  self.killTimer()                             ; sólo en caso de que no haya
expirado
                                                     ; aún, destruímos el
antiguo temporizador
  self.setTimer(HoraDeAlerta)                  ; iniciamos el temporizador
para este
                             ; registro
endif
endmethod
```

Este código se anexa al método **timer** de *registroCliente*.

```
; registroCliente::timer
method timer(var eventInfo TimerEvent)
beep()
msgInfo( ¡Atención! ,  Ha estado procesando este registro durante un minuto.
)
self.killTimer()
```

```
endmethod
```

Consulte los métodos **killTimer**, **setTimer** y **action** del tipo UIObject para más información.

Tipo ValueEvent

Los métodos de ValueEvent controlan los cambios en los valores de los campos. El método estándar **changeValue** es el único método activado por un ValueEvent. El método estándar **newValue** no se activa con un ValueEvent; por el contrario, **newValue** toma un Event.

El tipo ValueEvent incluye varios métodos definidos para el tipo Event.Consulte el tipo Event para más información.

No confunda **changeValue** con **newValue**. El método estándar **changeValue** se ejecuta cuando el valor de un campo está a punto de cambiar. **changeValue** da la oportunidad de comprobar el valor y decidir si se desea almacenar. El método estándar **newValue** informa de cuándo un campo ha recibido un nuevo valor; **newValue** se suele ejecutar después del hecho (los campos definidos como botones y las listas se comportan de forma distinta). Observe también que el método estándar **newValue** *no* es igual que el método **newValue** descrito en esta sección.

newValue
setNewValue

close

Método Cierra una base de datos.

Tipo Database

Sintaxis **close** ( ) Logical

Descripción Termina la asociación entre una variable Database y una base de datos, quedando la variable sin asignación. **close** devuelve True si es satisfactorio; de lo contrario, devuelve False.

Ejemplo El código siguiente abre la base de datos con el alias AlgunasTablas. Si la tabla Pedidos no existe en AlgunasTablas, este código cierra AlgunasTablas y abre otra base de datos con el alias MásTablas. Este código supone que ambos alias se han definido en algún otro lugar y que son válidos.

```
;BotónDeSumar::pushButton
method pushButton(var eventInfo Event)
var
   bd Database
   tc TCursor
endVar
bd.open("AlgunasTablas")    ; abre el alias de la base de datos
AlgunasTablas     if bd.isTable("Pedidos.db") then   ; si
Pedidos.db está en la base de datos,
   tc.open("Pedidos.db", bd)      ; abrimos un TCursor para ella

    ; calculamos el Total del campo Pendiente
   msgInfo("Pendiente", tc.cSum("Pendiente"))
else
   bd.close()   ; cerramos la base de datos AlgunasTablas
   bd.open("MásTablas")     ; y abrimos otra
   if bd.isTable("Pedidos.db") then
      tc.open("Pedidos.db", bd)
      msgInfo("Pendiente", tc.cSum("Pendiente"))
   endIf
endIf
endmethod
```

Vea también open

delete

Método/ procedimiento Borra una tabla de una base de datos.

Tipo Database

Sintaxis

1. delete (const ***nombreTabla*** String [, const ***tipoTabla*** String]) Logical

2. delete (const ***varTabla*** Table) Logical

Descripción Borra la base de datos de una tabla y los archivos de índice asociados o archivos de vista de tabla sin pedir confirmación. Si utiliza la sintaxis 1, y si la extensión de archivo no es estándar o no se suministra, puede emplear el argumento optativo tipoTabla para especificar el tipo de la tabla que desea borrar (Paradox o dBASE). Si no especifica tipoTabla o no es estándar, se supone Paradox . Si utiliza la sintaxis 2, puede emplear el argumento varTabla para especificar una variable Table. Sin embargo, este método sólo usa el nombre y tipo de la tabla descrita por la variable Table, no su asociación con la base de datos.

La operación no puede anularse. Este método devuelve True si se lograr borrar la tabla; en caso contrario, devuelve False. la tabla está abierta, delete falla.

Ejemplo En este ejemplo, el método **pushButton** de *EliminarTabla* borra una tabla de la base de datos con el alias *megaDato*.

```
;EliminarTabla::pushButton
method pushButton(var eventInfo Event)
var
   Bd Database
   NombreTabla String
endVar
NombreTabla = "TablaAnt"
Bd.open("megadato")
if isTable(NombreTabla) then
   Bd.delete(NombreTabla, "dBASE") ; elimina TablaAnt.dbf de
megadato
endif
endmethod
```

executeQBE

Método/ procedimiento Ejecuta una consulta QBE.

Tipo Database

Sintaxis

1. **executeQBE** (const ***varQBE*** Query) Logical
2. **executeQBE** (const ***varQBE*** Query, const ***tablaRespuesta*** String) Logical
3. **executeQBE** (const ***varQBE*** Query, const ***tablaRespuesta*** Table) Logical
4. **executeQBE** (const ***varQBE*** Query, var ***tablaRespuesta*** TCursor) Logical

Descripción Ejecuta una consulta creada en un método de ObjectPAL y escribe el resultado en tablaRespuesta. En la sintaxis 1, donde no se especifica tablaRespuesta, **executeQBE** lo escribe en SOLUCION.DB en el directorio personal del usuario. En la sintaxis 2, la tabla de respuesta se especifica como una cadena; si no se incluye una extensión de archivo, *tablaRespuesta* es una tabla de Paradox por defecto. En la sintaxis 3, donde tablaRespuesta es una variable Table, la variable Table debe estar asignada y ser válida. Si se utiliza la sintaxis 4 para escribir el resultado de la consulta en un TCursor, el resultado sólo se almacena en la memoria del sistema; no se crea una tabla en disco.

Si **executeQBE** es satisfactorio si se crea *tablaRespuesta* o SOLUCION.DB este método devuelve True (incluso si la tabla resultante está vacía); en caso contrario, devuelve False.

Una consulta en un método comienza con una variable Query , el signo = y la palabra clave **query** seguida de una línea en blanco. A continuación, viene el cuerpo de la consulta y otra línea en blanco. La consulta termina con la palabra clave **endquery**. Para especificar una vía de acceso, no son necesarias las barras invertidas dobles.

Puesto que este tipo de consulta no es una cadena entrecomillada, puede contener variables de tilde (compare este método con **executeQBEString**). Es posible utilizar vías de acceso absolutas o alias en la definición de la consulta.

Ejemplo El código de este ejemplo modifica el método estándar **pushButton** de *encontrarNombre*. Cuando se pulsa el botón, el método define una variable Query y utiliza **executeQBE** para ejecutar la consulta y almacenar la consulta en la tabla NOMCLIEN.DB.

```
;encontrarNombre::pushButton
method pushButton(var eventInfo Event)
var
   qq Query
   Nombre String
   tv TableView
endVar

Nombre = "Unisco"
qq = Query
    c:\pdoxwin\ejemplos\clientes | N° de Cliente | Nombre
                                 | Check         | Check
~Nombre |
    EndQuery
executeQBE(qq, "NomClien.db") ; lleva los resultados a
NomClien.db     tv.open("NomClien")          ; vemos la tabla
endmethod
```

El ejemplo siguiente añade alias, utiliza el alias para abrir una base de datos y ejecuta una consulta sobre una tabla de esa base de datos:

```
; esteBotón
method pushButton(var eventInfo Event)
var
    bd Database
    qq Query
    tc TCursor
    tv TableView
endVar

qq = Query
     contacto.db | Ultimo Nombre | Primer Nombre | Compañía |
Teléfono
                 | Check         | Check         | Check    |
Check 808.. |
     EndQuery

; creamos el alias DatosEjemplo y luego lo abrimos
addAlias("DatosEjemplo", "Standard", "C:\\pdoxwin\\ejemplos")
db.open("DatosEjemplo")

; ahora consultamos CONTACTO.DB en la base de datos
DatosEjemplo y escribimos
; los resultados en TELEFONO.DBF de la base de datos por
defecto     db.executeQBE(qq, ":trabajo:telefono.dbf")
tv.open("telefono.dbf")     ; abrimos la tabla en la base de
datos por defecto endmethod
```

Vea también executeQBEFile
executeQBEString

executeQBEFile

Método/ procedimiento Abre y ejecuta un archivo QBE.

Tipo Database

Sintaxis

1. **executeQBEFile** (const ***nombreArchivoQBE*** String) Logical
2. **executeQBEFile** (const ***nombreArchivoQBE*** String,
   const tablaRespuesta String) Logical
3. **executeQBEFile** (const nombreArchivoQBE String,
   const ***tablaRespuesta*** Table) Logical
4. **executeQBEFile** (const ***nombreArchivoQBE*** String,
   var ***tablaRespuesta*** TCursor) Logical

Descripción Abre nombreArchivoQBE (creado con el método **writeQBE** o interactivamente mediante el Editor de consultas), lo ejecuta y escribe el resultado en la tabla tablaRespuesta. En la sintaxis 1, donde no se especifica tablaRespuesta, **executeQBEFile** lo escribe en SOLUCION.DB en el directorio personal del usuario. En la sintaxis 2, la tabla de respuesta se especifica como una cadena; si no se incluye una extensión de archivo, *tablaRespuesta* es una tabla de Paradox por defecto (utilice .DBF para escribir en una tabla de dBASE). En la sintaxis 3, donde tablaRespuesta es una variable Table, la variable Table debe estar asignada y ser válida. Si se utiliza la sintaxis 4 para escribir el resultado de la consulta en un TCursor, el resultado sólo se almacena en la memoria del sistema; no se crea una tabla en disco.

Si **executeQBEFile** es satisfactorio si se crea *tablaRespuesta* o SOLUCION.DB este método devuelve True (incluso si la tabla resultante está vacía); en caso contrario, devuelve False.

Ejemplo Este código demuestra cómo funciona cada forma de la sintaxis:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   bd    Database
  Tabla  Table
  tc    TCursor
endVar
addAlias("Datos",  "Standard", "c:\\PdoxWin\\Tablas")
db.open("Datos")
Tabla.attach("Cliente1.db", bd)

; esto escribe los resultados en :PERSONAL:RESPUEST.DB
executeQBEFile("ObtClien.qbe")

; esto escribe los resultados en MiClient.db en la base de
datos Datos     executeQBEFile("ObtClien.qbe", "MiClient.db")

; esto escribe el resultado en Cliente1.db en la base de datos
Datos     executeQBEFile("ObtClien.qbe", Tabla)

; esto escribe el resultado en el TCursor tc
executeQBEFile("ObtClien.qbe", tc)

endmethod
```

Vea también executeQBE
executeQBEString

executeQBEString

Método/ procedimiento Ejecuta una cadena QBE.

Tipo Database

Sintaxis

1. executeQBEString (const ***cadenaQBE*** String) Logical

2. executeQBEString (const ***cadenaQBE*** String,
const tablaRespuesta String) Logical

3. executeQBEString (const ***cadenaQBE*** String,
const tablaRespuesta Table) Logical

4. executeQBEString (const ***cadenaQBE*** String,
var tablaRespuesta TCursor) Logical

Descripción Ejecuta una cadena QBE y escribe el resultado en tablaRespuesta. En la sintaxis 1, donde no se especifica tablaRespuesta, **executeQBEString** lo escribe en SOLUCION.DB en el directorio personal del usuario. En la sintaxis 2, la tabla de respuesta se especifica como una cadena; si no se incluye una extensión de archivo, *tablaRespuesta* es una tabla de Paradox por defecto. En la sintaxis 3, donde tablaRespuesta es una variable Table, la variable Table debe estar asignada y ser válida. Si se utiliza la sintaxis 4 para escribir el resultado de la consulta en un TCursor, el resultado sólo se almacena en la memoria del sistema; no se crea una tabla en disco.

Si **executeQBEString** es satisfactorio si se crea *tablaRespuesta* o SOLUCION.DB este método devuelve True (incluso si la tabla resultante está vacía); en caso contrario, devuelve False.

Una cadena QBE puede ser una combinación de cadenas entrecomilladas y variables de cadena.

executeQBEString es útil cuando se crea una cadena QBE a partir de cadenas más pequeñas. Al especificar una vía de acceso, son necesarias barras invertidas dobles.

Puesto que una cadena QBE es una cadena entrecomillada, no puede contener variables de tilde. Si desea emplear variables de tilde en una consulta, utilice **executeQBE**.

Ejemplo En este ejemplo, el método **pushButton** de *encontrarNombre* define una consulta como un valor de cadena y utiliza **executeQBEString** para ejecutar la consulta.

```
;encontrarNombre::pushButton
method pushButton(var eventInfo Event)
var
   bd Database
   qs String
   tv TableView
   tc TCursor
   NumExistenc String
endVar

; añadimos el alias DatosEjemplo y luego abrimos la base de
datos     addAlias("DatosEjemplo", "Standard", "c:\\pdoxwin\\
ejemplos")     bd.open("DatosEjemplo")

; abrimos un TCursor para la tabla Existenc
tc.open("Existenc.db", bd)

; si locate encuentra Luminosos de Kriptón en el campo
Descripción
if tc.locate("Descripción", "Luminosos de Kriptón") then

  ; ahora utilizamos el valor del campo N° de Stock en
Existenc.db dentro
     ; de una cadena QBE
  qs = "Query\n\n" +
          ":DatosEjemplo:Articulo | N° de Pedido | N° de Stock
|\n" +
                                "| _NumPed |" + tc."N° de Stock"
+ "|\n\n" +
          ":DatosEjemplo:Pedidos | N° de Pedido | N° de Cliente
|\n" +
                                "| _NumPed | _Clien |\n\n" +
          ":DatosEjemplo:Clientes | N° de Cliente | Nombre |
Teléfono |\n" +
                                "| _Clien | Check | Check |\n\n"
+
            "EndQuery"
   ; nótese que las líneas verticales (|) no tienen que estar
alineadas

   if executeQBEString(qs) then        ; escribir
en :PERSONAL:SOLUCION
     tv.open(":personal:Solución.db")  ; mostrar la tabla
Solución
else
  msgStop("Error", "Fallo en la consulta") ; de otro modo,
falló la consulta       endif

else
   msgStop("Error", "No encuentro Luminosos de Kriptón")
endif

endmethod
```

Vea también executeQBE
executeQBEFile

isAssigned

Método Informa de si se ha asignado un valor a una variable Database.

Tipo Database

Sintaxis **isAssigned** ( ) Logical

Descripción Devuelve True si se ha asignado un valor a una variable Database; en caso contrario, devuelve False.

Ejemplo En este ejemplo, una ficha tiene un campo sin asignar llamado *EstimaComp* y un botón llamado *mostrarEstimación*. El código anexado al método **pushButton** de *mostrarEstimación* utiliza **isAssigned** para determinar si se ha asignado algún valor a la variable Database *bd*. Si no se ha asignado un valor, se establece un alias y se asigna a la variable Database. Una vez definida la variable, el código abre un TCursor para la tabla *NueClien* contenida en la base de datos. El TCursor busca un valor en el campo Company y muestra la valoración crediticia de esa compañía en el campo *EstimaComp* de la ficha. A continuación, se presenta el código anexado al método **pushButton** de *mostrarEstimación*:

```
;mostrarEstimación::pushButton
method pushButton(var eventInfo Event)
var
   bd Database
   tc TCursor
endVar

if not isAssigned(bd) then
   addAlias("Tablas", "Standard", "c:\\pdoxwin\\Tablas")
   bd.open("Tablas")
endif

tc.open("NueClien.dbf", bd)
if tc.locatePattern("Compañía", "Informática..") then
   EstimaComp.value = tc.Estimación
else
   message("Error", "No encuentro Informática..")
endif

endmethod
```

Vea también isTable

isTable

Método/ procedimiento Informa de si una tabla existe en una base de datos.

Tipo Database

Sintaxis

1. isTable (const ***nombreTabla*** String [, const ***tipoTabla*** String]) Logical

2. isTable (const ***varTabla*** Table) Logical

Descripción Devuelve True si una tabla especificada se halla en la base de datos; en caso contrario, devuelve False.

Si utiliza la sintaxis 1, puede especificar un nombre y un tipo de tabla en los argumentos nombreTabla y tipoTabla. Si utiliza la sintaxis 2, puede especificar una variable Table en varTabla. Sin embargo, este método sólo emplea el nombre y tipo de la tabla descrita por la variable Table, no la base de datos.

Ejemplo El código siguiente utiliza **isTable** para averiguar si la tabla *Pedidos* existe en una base de datos dada. Este código se anexa al método estándar **pushButton** de *esteBotón*.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   bd           Database
   Compruébame        String
   CompruébameTambién       Table
   Tabla  TableView
endVar
bd.open()                          ; abre la base de datos por
defecto     Compruébame = "Pedidos.db"
if bd.isTable(Compruébame) then
   Tabla.open(Compruébame)
else
   message(Compruébame, " no es una tabla")
endIf

CompruébameTambién.attach("ventas.db")
if CompruébameTambién.isTable() then
   tot = CompruébameTambién.cSum("Total ventas")
   msgInfo("Total ventas:", tot)
endif
endmethod
```

Vea también isAssigned

open

Método Abre una base de datos.

Tipo Database

Sintaxis

1. open ( ) Logical

2. open (const ***nombreAlias*** String) Logical

3. open (const ***sesión*** Session) Logical

4. open (const ***nombreAlias*** String, const ***sesión*** Session) Logical

Descripción Abre una base de datos. En la sintaxis 1, donde no se dan argumentos, **open** abre la base de datos por defecto en la sesión actual. En la sintaxis 2, se abre la base de datos especificada en *nombreAlias* en la sesión actual. La sintaxis 3 permite abrir la base de datos por defecto en la sesión especificada en *sesión*. Utilice la sintaxis 4 para abrir una base de datos especificada en una sesión especificada.

Si utiliza las sintaxis 2 o la sintaxis 4, *nombreAlias* debe ser un alias válido en la sesión actual o en la sesión *sesión*. Los signos de dos puntos alrededor del nombre de alias son optativos.

Las sintaxis 3 y 4 requieren que se haya abierto una variable Session válida; en las sintaxis 1 y 2 se da por supuesta la sesión actual.

open devuelve True si logra abrir la base de datos especificada; de lo contrario, devuelve False.

ObjectPAL asocia la variable Database con la base de datos por defecto. **open** devuelve True si puede abrir la base de datos especificada; si no puede, devuelve False.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* abre tres bases de datos en la sesión actual.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   dBd, Bd, pBd Database
endVar

dBd.open()             ; conectamos dBd con la base de datos
por defecto     Bd.open("InfClien")    ; conectamos Bd con la
base de datos InfClien
                       ; (InfClien es un alias)
pBd.open("PERSONAL")   ; conectamos pBd con el directorio
PERSONAL

endmethod
```

Vea también close
Session

writeQBE

Método/ procedimiento Escribe una sentencia de consulta o una cadena de consulta en un archivo.

Tipo Database

Sintaxis

1. **writeQBE** (const ***varQBE*** Query, const ***nombreArchivo*** String) Logical
2. **writeQBE** (const ***cadena*** String, const ***nombreArchivo*** String) Logical

Descripción Escribe una sentencia o cadena de consulta definida previamente en el archivo especificado en *nombreArchivo*. Si *nombreArchivo* existe, se sustituye sin pedir confirmación. **writeQBE** devuelve True si logra escribir; en caso contrario, devuelve False.

Ejemplo En este ejemplo, supóngase que una ficha tiene un botón llamado *obtenNom*. Cuando se abre la ficha, este ejemplo averigua si el archivo OBTENNOM.QBE existe en el directorio actual (o en el directorio personal). Si no existe, el método estándar **open** de la ficha utiliza **writeQBE** para escribir una cadena de consulta en OBTENNOM.QBE. El método estándar **pushButton** de *obtenNom* ejecuta la consulta mediante **executeQBEFile** y muestra el resultado en un TableView. El código siguiente se anexa al método **open** de la ficha.

```
;estaFicha::open
method open(var eventInfo Event)
Var
   qs String    ; una cadena QBE
endVar

if eventInfo.isPreFilter() then
   ;el código fuente que haya aquí se ejecuta para cada objeto
de la Ficha     else
   ;el código fuente que haya aquí se ejecuta sólo para la
propia Ficha

   if not isfile("ObtDest.qbe") then  ; si el archivo de la
consulta no existe
                                      ; construimos una cadena
QBE

      qs = "Query\n\n" +
          ":AplTham:Destinos | Nombre Destino | Temperatura
Media (F) |\n" +
                              "| Check | Check 70 |\n\n" +
      "EndQuery"

; escribimos la cadena QBE en ObtenNom.qbe
writeQBE(qs, "ObtenNom.qbe")
endif

endif
endmethod
```

El código siguiente se anexa al método estándar **pushButton** del botón *obtenNom*. Este código no comprueba si OBTENNOM.QBE existe, porque el método **open** de la ficha garantiza que el archivo está disponible.

```
;obtenDestino::pushButton
method pushButton(var eventInfo Event)
var
   tv TableView
endVar

; ejecutamos el archivo de consulta y almacenar los resultados
en Destino.db
executeQBEFile("ObtenDes.qbe", "Destino.db")
; mostramos la tabla
tv.open("Destino")

endmethod
```

Otro uso de este método es emplear ObjectPAL para crear y almacenar una consulta que el usuario puede ejecutar interactivamente mediante el Editor de consultas.

Vea también executeQBE
executeQBEFile
executeQBEString

executeQBE

Método Ejecuta una consulta QBE.

Tipo Query

Sintaxis

1. executeQBE ( ) Logical

2. executeQBE (const ***tablaRespuesta*** String) Logical

3. executeQBE (const ***tablaRespuesta*** Table) Logical

4. executeQBE (var ***tablaRespuesta*** TCursor) Logical

Descripción Ejecutes una consulta creada en un método de ObjectPAL y escribe el resultado en *tablaRespuesta*. En la sintaxis 1, en que no se especifica *tablaRespuesta*, **executeQBE** escribe el resultado en SOLUCION.DB en el directorio personal del usuario. En la sintaxis 2, especifique la tabla de respuesta como una cadena; si no incluye una extensión de archivo, *tablaRespuesta* es una tabla de Paradox por defecto. En la sintaxis 3, en que *tablaRespuesta* es una variable Table, *tablaRespuesta* debe estar asignada y ser válida. Si emplea la sintaxis 4 para escribir el resultado de la consulta en un TCursor, el resultado sólo se almacena en la memoria del sistema; no se crea una tabla en el disco.

Si **executeQBE** es satisfactorio si se crea *tablaRespuesta* o SOLUCION.DB este método devuelve True (incluso si la tabla resultante está vacía); en caso contrario, devuelve False.

Una consulta comienza en un método con una variable Query, el signo = y la palabra clave Query seguida por una línea en blanco. A continuación, viene el cuerpo de la consulta y otra línea en blanco. La consulta termina con la palabra clave **EndQuery**.

Puesto que este tipo de consulta no es una cadena entrecomillada, puede contener variables de tilde (compare este método con **executeQBEString** en el tipo Database). Es posible utilizar alias o vías de acceso absolutas en la definición de la consulta.

Ejemplo En este ejemplo, el método **pushButton** del botón *obtenPendiente* construye una sentencia de consulta, la asigna a una variable Query y la ejecuta con **executeQBE**. La sentencia de consulta de este ejemplo es una consulta de inserción: recupera ciertos registros de CLIENTES.DB y PEDIDOS.DB y lo inserta en la tabla *Clien* existente. Los criterios de selección de este ejemplo emplea una variable de tilde *Provincia* y designan que se incluyan los clientes de Orense en el resultado. Puesto que OR es la abrevitatura de Orense, la variable *Provincia* debe evaluarse en una cadena entrecomillada para distinguirla de la expresión de consulta **OR**.

```
; obtenPendiente::pushButton
method pushButton(var eventInfo Event)
var
  qVar      Query
  Provincia String
  tv        TableView
endVar

; añadimos el alias ejemp para el directorio ejemplo de PdoxWin
addAlias("ejemp", "Standard", "c:\\pdoxwin\\ejemplo")

; OR es una palabra reservada, pero ya que es también la
abreviatura de
; Orense, debe estar entre comillas
Provincia = "\"OR\""

; ahora utilizamos Provincia como una variable tilde en esta
estructura de consulta
qVar = Query

     :ejemp:Clientes.db | N° de Cliente | Nombre  | Provincia
| Teléfono  |
                        | _clien        | _nombre | ~Provincia
| _Teléfono |

     :ejemp:Pedidos.db | N° de Cliente | Pendiente  |
                       | _clien        | 0, _BalDeb |

           Clien.db    | N° de Cliente | Nombre  | Pendiente |
Teléfono  |
              insert   | _clien        | _Nombre | _BalDeb   |
_Teléfono |


     EndQuery

executeQBE(qVar, "Clien.db")   ; escribe los resultados en
Clien.db
tv.open("Clien.db")            ; mostramos la tabla

endmethod
```

Vea también QUERY
writeQBE
Database::executeQBEFile
Database::executeQBEString

isAssigned

Método Informa de si se ha asignado un valor a una variable Query.

Tipo Query

Sintaxis **isAssigned**( ) Logical

Descripción Devuelve True si se ha asignado un valor a una variable Query; de lo contrario, devuelve False. Este método no comprueba la validez de la consulta asignada.

Ejemplo En el ejemplo siguiente, la llamada a **isAssigned** devuelve True, porque se ha asignado un valor a la variable Query *qVar*, aunque el valor no es una consulta válida.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  qVar Query
endVar

qVar = Query

       Esto no es una consulta

       endQuery

msgInfo("¿Está asignada?", qVar.isAssigned())    ; muestra True

endmethod
```

Vea también query

query

Palabra clave Comienza una sentencia de consulta.

Tipo Query

Sintaxis **query**

nombreTabla | nombreCampo | [nombreCampo |]*
| criterios | [criterios |]*

[nombreTabla | nombreCampo | [nombreCampo |]*
| criterios | [criterios |]*] *

endQuery

Descripción Marca el comienzo de una sentencia QBE. Una sentencia QBE extrae datos de una o más tablas según los campos especificados en nombreCampo y los criterios de selección, donde *criterios* pueden ser cualquier expresión QBE válida.

Una consulta comienza en un método con una variable Query, el signo = y la palabra clave **Query** seguida por una línea en blanco. A continuación, viene el cuerpo de la consulta y otra línea en blanco. La consulta termina con la palabra clave **EndQuery**.

No es necesario incluir todos los campos de la tabla. Es posible listar sólo los campos que afectan a la consulta, como en este ejemplo.

```
Var MiQBE Query endVar
MiQBE = Query

           Clientes | N° de Cliente | Nombre |
                    | Check         | A..     |

        endQuery
```

La sentencia de consulta anterior recupera, de la tabla *Clientes*, los números de cliente cuyo nombre comience por A (la tabla *Clientes* tiene más de dos campos, pero sólo se especifican dos en este ejemplo).

Las líneas vacías por encima y por debajo del cuerpo de la consulta son necesarias. No es necesario alinear los separadores de campo verticales (aunque la alineación aumenta la legibilidad). ObjectPAL interpreta el código siguiente exactamente igual que el código del ejemplo anterior.

```
Var MiQBE Query endVar
MiQBE = Query

Clientes | N° de Cliente           |Nombre |
|Check| A..   |

   endQuery
```

Si se construye una sentencia de consulta que incluye dos o más tablas, es necesario separar cada tabla con una línea en blanco, como sigue:

```
Var MiQBE Query endVar
MiQBE = Query

           Clientes | N° de cliente | Nombre | Teléfono |
                    | _x            | Check  | Check    |

           Pedidos  | N° de cliente | Pendiente |
                    | _x            | Check 0   |

       endQuery
```

Puesto que este tipo de consulta no es una cadena entrecomillada, puede contener variables de tilde (compare este tipo de consulta con una cadena de consulta en el tipo Database). Es posible utilizar alias o vías de acceso absolutas en la definición de la consulta.

Ejemplo En este ejemplo, el método **pushButton** del botón *obtenPendiente* construye una sentencia de consulta, la asigna a una variable Query y la ejecuta con **executeQBE**. La sentencia de consulta de este ejemplo es una consulta de inserción: recupera ciertos registros de CLIENTES.DB y PEDIDOS.DB y los inserta en la tabla *MiClien* existente. Los criterios de selección de este ejemplo utilizan la variable de tilde *Provincia* que designan la inclusión de los clientes de Orense en el resultado. Puesto que OR es la abreviatura de Orense, la variable *Provincia* debe evaluarse en una cadena entrecomillada para diferenciarla de la expresión de consulta **OR**.

```
; obtenPendiente::pushButton
method pushButton(var eventInfo Event)
var
  qVar      Query
  Provincia String
  tv        TableView
endVar

; añadimos el alias ejemp para el directorio ejemplos de
PdoxWin
addAlias("ejemp", "Standard", "c:\\pdoxwin\\ejemplo")

; OR es una palabra reservada, pero ya que también es la
abreviatura de
; Orense, debe escribirse entre comillas
Provincia = "\"OR\""

; ahora utilizamos Provincia como una variable tilde en esta
estructura de consulta
qVar = Query

   :ejemplo:Clientes.db | N° de cliente | Nombre  | Provincia
| Teléfono  |
                        | _clien        | _nombre | ~Provincia
| _Teléfono |

    :ejemp:Pedidos.db   | N° de Cliente | Pendiente  |
                        | _clien        | 0, _BalDeb |

          MiClien.db    | N° de Cliente | Nombre   | Pendiente
| Teléfono  |
              insert    | _clien        | _nombre  | _BalDeb
| _teléfono |


    EndQuery

executeQBE(qVar, "MiClien.db")   ; escribimos los resultados en
MiClien.db
tv.open("MiClien.db")            ; mostramos la tabla

endmethod
```

Vea también executeQBE
DataBase::executeQBEFile
DataBase::executeQBEString

writeQBE

Método Escribe una sentencia de consulta en un archivo especificado.

Tipo Query

Sintaxis **writeQBE** (const ***nombreArchivo*** String) Logical

Descripción Escribe una sentencia de consulta definida previamente en el archivo especificado en *nombreArchivo*. Si *nombreArchivo* existe, se sustituye sin pedir confirmación. **writeQBE** devuelve True si logra escribirlo; de lo contrario, devuelve False.

Ejemplo En este ejemplo, supóngase que una ficha tiene un botón llamado *obtnDest*. Cuando se abre la ficha, este ejemplo determina si el archivo OBTNDEST.QBE existe en el directorio actual. Si no existe, el método estándar **open** de *páginaUno* utiliza **writeQBE** para escribir una cadena de consulta en OBTNDEST.QBE. El método estándar **pushButton** de *obtnDest* ejecuta la consulta mediante **executeQBEFile** y, a continuación, abre la tabla. Este código supone que ya se ha definido el alias :THAM:. A continuación, se muestra el código anexado al método **open** de *páginaUno*:

```
; páginaUno::open
method open(var eventInfo Event)
Var
  qVar Query
endVar

; si el archivo de consulta ObtnDest.qbe no existe
if not isfile("ObtnDest.qbe") then

  ; construimos una consulta
  qVar = Query

      :AbtMAST:Dest | Nombre Destino | Tiempo medio (F) |
                    | Check          | Check 70         |

    EndQuery

  ; escribimos la estructura de la consulta en el archivo
ObtnDest.qbe
  writeQBE(qVar, "ObtnDest.qbe")

endif

endmethod
```

El código siguiente se anexa al método estándar **pushButton** del botón *obtnDest*. Este código no comprueba si OBTNDEST.QBE existe, porque el método **open** de la página garantiza que el archivo está disponible.

```
; obtenDest::pushButton
method pushButton(var eventInfo Event)
var
  tv TableView
endVar

; ejecutamos el archivo de la consulta y almacenamos los
resultados en MiDest.db
executeQBEFile("ObtnDest.qbe", "MiDest.db")
; abrimos la tabla
tv.open("MiDest")

endmethod
```

Otro utilidad de este método es emplear ObjectPAL para crear y guardar una consulta que el usuario puede ejecutar interactivamente mediante el Editor de consultas.

Vea también executeQBE
Database::executeQBEFile
Database::executeQBEString

add

Principiante

Método/ procedimiento Añade los datos de una tabla en otra.

Sintaxis

1. add (const ***nombreTablaDestino*** String
[, const ***añadir*** Logical [, const ***actualizar*** Logical]]) Logical

2. add (const ***varTablaDestino*** Table
[, const ***añadir*** Logical [, const ***actualizar*** Logical]]) Logical

Descripción

Añade los datos contenidos en una tabla en una tabla destino, que puede especificarse utilizando una variable String (nombreTablaDestino en la sintaxis 1) o una variable Table (varTablaDestino en la sintaxis 2). Si la tabla destino no existe, este método la crea. La tabla origen y la destino pueden ser del mismo tipo o de tipos distintos; en cualquier caso, las tablas deben tener estructuras de campos compatibles.

Los argumentos añadir y actualizar pueden ser True o False. Cuando es True, añadir añade registros al final de una tabla destino no indexada o en el lugar adecuado de una tabla destino indexada. Cuando es True, actualizar compara los registros de ambas tablas y, donde los valores de clave coincidan, sustituye los datos en la tabla destino. Cuando ambos son True, los registros con valores de clave coincidentes se actualizan, mientras que los demás se añaden. Estos argumentos son optativos, pero si se especifica actualizar, es necesario especificar también añadir. Si se omiten, ambos se consideran True.

Las violaciones de clave, si las hay, se enumeran en VIOLCLAV.DB que se halla en el directorio personal del usuario. Este método sustituye un VIOLCLAV.DB existente o crea uno, si es necesario. A continuación, se presentan algunas sentencias de ejemplo.

Cuando las tablas están indexadas, **add** utiliza los campos indexados para determinar qué registros actualizar y cuáles añadir. Cuando la tabla destino no está indexada, **add** falla si actualizar es True.

```
MiTabla.add (TuTabla, False, True) ; especifica actualizar
MiTabla.add (TuTabla)              ; especifica actualizar y
por defecto, añadir
```

Este método intenta, durante el periodo de reintento, aplicar bloqueos de escritura a las tablas origen y destino. Si no es posible aplicar alguno de los bloqueos, el método falla.

Ejemplo

Para este ejemplo, el método **pushButton** de *actualizarClient* ejecuta una consulta de un archivo existente y añade registros de la tabla *Respuest* en la tabla *Clientes*.

```
; actualizarClient::pushButton
method pushButton (var eventInfo Event)
var
  NuevoCliente Query
  TablaRespuesta Table
  TablaDestino   String
endVar
TablaDestino = "Clientes.db"
if executeQBEFile ( ObtenCli.qbe ) then            ; si la
consulta tiene éxito
  TablaRespuesta.attach ( Respuest.db )
  ; intentamos añadir registros de Respuest.db a Clientes.db
  if isTable (TablaDestino) then
    if NOT TablaRespuesta.add (TablaDestino) then
      msgStop ( Error ,"No puedo escribir en la tabla   +
TablaDestino +  .")
    endif
  else
    msgStop ( Error ,"No puedo encontrar   + TablaDestino
+  .")
  endif
else
  msgStop ( Error ,"Fallo en la consulta.")
endif

endmethod
```

Vea también copy
subtract

attach

Principiante

Método Asocia una variable Table con una tabla en disco.

Tipo Table

Sintaxis

1. attach (const ***nombreTabla*** String) Logical

2. attach (const ***nombreTabla*** String, const ***db*** Database) Logical

3. attach (const ***nombreTabla*** String, const ***tipoTabla*** String) Logical

4. attach (const ***nombreTabla*** String, const ***tipoTabla*** String, const ***db*** Database) Logical

Descripción Asocia una variable Table con los datos de nombreTabla. Los argumentos optativos tipoTabla y db especifican un tipo de tabla (Paradox o dBASE) y una base de datos, respectivamente. Si no se especifica tipoTabla, ObjectPAL intenta determinar el tipo de tabla por la extensión de archivo del nombre de la tabla. Si no se especifica db, ObjectPAL trabaja en la base de datos por defecto.

Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Nota: attach no comprueba que nombreTabla sea una tabla, ni siquiera que exista el archivo. Utilice el método **isTable** para comprobar su existencia.

Para liberar una variable Table por completo, utilice **unAttach**. Para asociar la variable Table con otra tabla, utilice **attach** de nuevo; **unAttach** se ejecuta automáticamente.

Ejemplo En este ejemplo, la variable Table *TablaOeste* se anexa a *Pedidos* para que **cSum** pueda emplearse con esa variable Table. Este ejemplo emplea **isTable** para determinar si *Pedidos* existe en la base de datos por defecto antes de realizar un cálculo sobre la tabla.

```
; obtenerTotalDeOeste::pushButton
method pushButton (var eventInfo Event)
  var
    TablaOeste Table
    TotalOeste Number
  endVar

  if isTable ( Pedidos.db ) then

    ; conectamos la tabla de Paradox Pedidos.db con la base de
datos por defecto
    TablaOeste.attach ( Pedidos.db ,  Paradox )
    ; obtener el total del campo Total Factura y almacenar el
resultado en
    ; TotalOeste
    TotalOeste = TablaOeste.cSum ( Total Factura )
    ; mostrar el total de facturas
    msgInfo ( Total Facturas , TotalOeste)

  else
    msgInfo ( Estado ,  No puedo encontrar la tabla Pedidos.db.
)
  endif

endmethod
```

Vea también create
unAttach

cAverage

Principiante

Método/ procedimiento Devuelve el valor medio de un campo (columna) de una tabla.

Tipo Table

Sintaxis **1. cAverage** (const ***nombreCampo*** String) Number

2. cAverage (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la media de los valores de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cAverage** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo Este ejemplo utiliza **cAverage** para calcular el tamaño medio de los pedidos de la tabla *Pedidos*. Este código se anexa al método **pushButton** del botón *obtenerMediaDeVentas*.

```
; obtenerMediaDeVentas::pushButton
method pushButton (var eventInfo Event)
  var
    TablaPedidos Table
    MediaVentas Number
  endVar
  TablaPedidos.attach ( Pedidos.db )
  MediaVentas = TablaPedidos.cAverage ( Total Factura )
     ; almacenar la media de los totales de factura en
MediaVentas
  msgInfo ( Tamaño medio de Pedido , MediaVentas)
     ; mostrar MediaVentas en una ventana de diálogo

endmethod
```

Vea también cCount
cMax
cMin
cSum
cStd

cCount

Principiante

Método/ procedimiento Devuelve el número de valores no vacíos de un campo (columna) de una tabla.

Tipo Table

Sintaxis **1. cCount** (const ***nombreCampo*** String) Number

2. cCount (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve el número de valores no vacíos existentes en la columna (campo) especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). **cCount** funciona con todos los tipos de campo. Si el campo es numérico, este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión. Si el campo no es numérico, **cCount** devuelve el número de valores de no vacíos de la columna de campos.

Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo En este ejemplo, el método **pushButton** de *infoSobreLineaDeItem* utiliza **cAverage** y **cCount** para realizar cálculos con el campo *Cant* de ARTICULO.DB. El ejemplo intenta aplicar un bloqueo de escritura sobre la tabla, de forma que otro usuario de la red no pueda realizar cambios en la tabla entre la ejecución de **cAverage** y **cCount**. Si el bloqueo no es satisfactorio, este código cancela la operación.

```
; infoSobreLineaDeItem::pushButton
method pushButton (var eventInfo Event)
  var
    TablaDeLaLinea Table
    CantidadMedia, NumeroArtic Number
  endVar
  if TablaDeLaLinea.attach ( articulo.db ) then
    if TablaDeLaLinea.Lock ( Write ) then
     ; si el bloqueo de escritura tiene éxito
      CantidadMedia = TablaDeLaLinea.cAverage ( Cant )
      NumeroArtic = TablaDeLaLinea.cCount (4) ; asume que Cant
es el campo 4.
      TablaDeLaLinea.unLock ( Write )              ;
desbloquear la tabla
      msgInfo ( Cantidad Media ,  Cantidad Media:   +
      String (CantidadMedia) +  \nbasado en   + String
(NumeroArtic) +   artículos. )
    else
      msgStop ( Error ,  No puedo bloquear la tabla articulo )
    endif
  else
    msgStop ( Lo siento ,  No puedo conectar con la tabla
artículo. )
  endif

endmethod
```

Vea también cAverage
cMax
cMin
cStd
cSum

cMax

Principiante

Método/ procedimiento Devuelve el valor máximo de un campo (columna) de una tabla.

Tipo Table

Sintaxis **1. cMax** (const ***nombreCampo*** String) Number

2. cMax (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve el valor máximo de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cMax** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo El código siguiente muestra la mayor cantidad del campo *Total Factura* de la tabla *Pedidos*.

```
; mostrarPedidoMáximo::pushButton
method pushButton (var eventInfo Event)
  var
    TablaDePedidos Table
  endVar

  if TablaDePedidos.attach ( Pedidos.db ) then
     ; mostrar el pedido máximo en una ventana de diálogo
    msgInfo ( El mayor pedido de la Historia ,
TablaDePedidos.cMax ( Total Factura ))
  else
    msgStop ( Lo siento ,  No puedo abrir la tabla Pedidos )
  endif

endmethod
```

Vea también cAverage
cMin
cStd
cSum
cCount

cMin

Principiante

Método/ procedimiento Devuelve el valor mínimo existente en un campo (columna) de una tabla.

Tipo Table

Sintaxis **1. cMin** (const ***nombreCampo*** String) Number

2. cMin (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve el valor mínimo de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cMin** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo El código siguiente muestra la menor cantidad del campo *Total Factura* de la tabla *Pedidos*.

```
; mostrarPedidoMínimo::pushButton
method pushButton (var eventInfo Event)
  var
    TablaDePedidos Table
  endVar

  if TablaDePedidos.attach ( Pedidos.db ) then
     ; muestra el pedido mínimo en un cuadro de diálogo
    msgInfo ( El menor pedido de la Historia ,
TablaDePedidos.cMin ( Total Factura ))
else
    msgStop ( Lo siento ,  No puedo abrir la tabla de pedidos )

endif

endmethod
```

Vea también cAverage
cCount
cStd
cSum
cMax

cNpv

Principiante

Método/ procedimiento Devuelve el valor presente neto de un campo (columna), en función de un tipo de interés o de descuento especificado.

Tipo Table

Sintaxis **1. cNpv** (const ***nombreCampo*** String, const ***tipoInterés*** AnyType) Number

2. cNpv (const ***númeroCampo*** SmallInt, const ***tipoInterés*** AnyType) Number

Descripción Devuelve el valor presente neto de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cNpv** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

El cálculo se basa en tipoInterés, expresado como un decimal (por ejemplo, 0,12 para el 12 por ciento). Este método calcula el valor presente neto utilizando la fórmula siguiente:

cNpv = *sum*(*p* = 1 to *n*) of *Vp* / (1 + *r*)*p*

donde

p = número de periodos, *Vp* = flujo de caja en el periodo *p*, e *i* = tipo de interés por periodo.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo El código siguiente define una variable Table para la tabla *BuenFond* y calcula el valor presente neto del campo *Retorno Esperado*. En este ejemplo, el valor presente neto se calcula en función de un tipo de interés mensual.

```
; calcularElVPN:: pushButton
method pushButton (var eventInfo Event)
  var
    Tabla Table
    BuenFondoVPN, CPA Number
  endVar
  CPA = 0,125     ; Coeficiente de Porcentaje Anual
  Tabla.attach ( BuenFond.db )

; calcular el Valor Presente Neto basado en el coeficiente de
interés mensual
  BuenFondoVPN = Tabla.cNpv ( Retorno Esperado , (CPA/12))
  msgInfo ( Valor Presente Neto , BuenFondoVPN)

endmethod
```

Vea también cAverage

cMax
cMin
cSum
cVar

compact

Principiante

Método Elimina registros borrados de una tabla de dBASE.

Tipo Table

Sintaxis **compact** ([const **regIndice** Logical]) Logical

Descripción Los registros borrados no se eliminan inmediatamente de una tabla de dBASE. En su lugar, se marcan como borrados y se mantienen en la tabla. **compact** elimina los registros borrados. El argumento optativo regIndice especifica si se regeneran los índices asociados con la tabla o sólo se fijan. Cuando regIndice es True, este método regenera todos los índices mantenidos que están asociados con la tabla: índices especificados por **usesIndexes** y el índice .MDX cuyo nombre coincide con el de la tabla. Cuando regIndice es False, sólo se regeneran los índices mantenidos. Si se omite, regIndice es True por defecto.

Cuando se borran registros de una tabla de Paradox, ya no pueden recuperarse se borran permanentemente . Sin embargo, el archivo de tabla (y los archivos de índice asociados) contienen espacio muerto en que el registro estaba almacenado originalmente. compact elimina el espacio muerto de los archivos de Paradox.

Este método falla si se han aplicado bloqueos a la tabla, o la tabla está abierta. Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente demuestra cómo **compact** afecta a los índices especificados por **usesIndexes**. En este ejemplo, la variable Table TablaPedidos se anexa a PEDIDOS.DBF y *TablaVentas* se anexa a VENTAS.DBF. Puesto que *TablaPedidos* utiliza INDICE1.NDX e INDICE2.NDX (especificados por **usesIndexes**), **compact** regenera INDICE1.NDX e INDICE2.NDX si *regIndice* es True. Para este ejemplo, *regIndice* se define como False por lo que **compact** afecta sólo a PEDIDOS.NDX.

```
; compactarTablas::pushButton
method pushButton(var eventInfo Event)
  var
    TablaPedidos, TablaVentas Table
  endVar
  TablaPedidos.usesIndexes( indice1.ndx ,  indice2.ndx )
  TablaPedidos.attach( Pedidos.dbf )
  TablaPedidos.compact(False)     ; elimina los registros
borrados y fija
    ; Pedidos.mdx
  TablaVentas.usesIndexes( indice3.mdx )
  TablaVentas.attach( Ventas.dbf )
  TablaVentas.compact()           ; elimina los registros
borrados y regenera todos
    ; los índices
endmethod
```

Vea también showDeleted
usesIndexes

copy

Principiante

Método/ procedimiento Copia una tabla.

Tipo Table

Sintaxis **1. copy** (const ***tablaDestino*** String) Logical

2. copy (const ***tablaDestino*** Table) Logical

Descripción Copia los registros de una tabla origen en una tabla destino (tablaDestino). La tabla origen y la tabla destino pueden ser de tipos diferentes, pero deben tener tipos de campos compatibles. **copy** falla si las tablas origen y destino tienen tipos de campos incompatibles.

Si la tabla destino ya existe, **copy** la sustituye sin pedir confirmación. Si la tabla destino está abierta, el método falla.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla origen y un bloque completo (exclusivo) sobre la tabla destino. Si no es posible aplicar alguno de los bloqueos, el método falla.

Ejemplo En este ejemplo, el método **pushButton** de copiaSeguridadClientes copia la tabla Clientes en ClientBK. Si ClientBK ya existe en el directorio actual, este método pide confirmación al usuario antes de sustituirla.

```
; copiaSeguridadClientes::pushButton
method pushButton(var eventInfo Event)
  var
    TablaOrigen  Table
    TablaDestino String
  endVar

  TablaDestino =  ClientBK.db
  TablaOrigen.attach( Clientes.db )

  if isTable(TablaDestino) then         ; si  ClientBK.db
existe, pedimos conformidad
    if msgQuestion( Copiar tabla ,  ¿Sobreescribir   +
TablaDestino +  ? ) <<>>
     Yes  then
      return
    endif
  endif
  TablaOrigen.copy(TablaDestino)        ; esto copia Clientes.db
en NuevoCli.db

endmethod
```

Vea también add
subtract
rename

create

Palabra clave Crea una tabla.

Tipo Table

Sintaxis **create** "*nombreTabla*" [**as** "*tipoTabla*"] [**database** *db*]

[[**like** *comoObjeto]*

[**with** "*nombreCampo*" : "*tipo*" [, "*nombreCampo*" : "*tipo*"] *]

[***where*** *descCampo* ***is*** *nuevoNombre* [*, descCampo* ***IS*** *nuevonombre*] *]

[***without*** *descCampo* [, *descCampo*] *]

[**struct** *tablaEstructCampo*]

[**indexStruct** *tablaEstructIndice*]

[**refIntStruct** *tablaEstructIntRef*]

[**secStruct** *tablaEstructSeg*]
] *

[**key** *descCampo* [, *descCampo*] *]
endCreate

Descripción Crea una tabla, donde nombreTabla es el nombre de la tabla que se creará. Por ejemplo:

```
Pedidos.db
```

Si *nombreTabla* existe, **create** intenta, durante el periodo de reintento, aplicar un bloqueo completo (exclusivo) sobre *nombreTabla*. Si no puede aplicarse el bloqueo, **create** falla.

AS tipoTabla especifica el formato de tabla. Ejemplo:

```
AS  Paradox
```

Si se omite **as**, *tipoTabla* es Paradox por defecto. **as** deduce *tipoTabla* de la extensión de *nombreArchivo*, si se da (.DB es una tabla de Paradox mientras que .DBF corresponde a una de dBASE).

database *db* es una variable de base de datos (abierta antes de crear la nueva tabla) que especifica dónde se almacenará la tabla. Por ejemplo:

```
DATABASE megaData
```

like *comoObjeto* especifica un TCursor, nombre de tabla o TableView abiertos desde los que se toman los nombres y tipos de campo. La cláusula **like** no toma los controles de validación, índices primarios ni secundarios, información de integridad referencial ni información de seguridad (utilice **struct**, **indexStruct**, **reFintStruct** y **secStruct** para tomar información más detallada).

Por ejemplo:

```
  like  Ventas.dbf          ; el nombre de la tabla como una
cadena
  like PedidosTC            ; una variable TCursor apuntando a
Pedidos.db
  like PedidosTV            ; una variable TableView apuntando a
Pedidos.db
```

with *nombreCampo* : *tipo* añade uno o más campos a la estructura de la tabla. Por ejemplo:

```
with  Apellido  :  A20",  Nombre"  A15",  Cantidad" :  N
```

Especifique en *tipo* el tipo de campo para *nombreCampo*. Los valores válidos para tipo varían dependiendo del tipo de tabla que se crea. La tabla siguiente muestra las especificaciones de campo válidas para las tablas de Paradox y dBASE.

Tipo de campo,	**Tabla de Paradox,**	**Tabla de dBASE**
Alfanumérico,	A*nn*,	(ninguno)
Carácter,	(ninguno),	C*nn*
Fecha,	D,	D
Entero,	S,	N
V. de coma flotante,	N,	No es posible crear un campo de coma flotante con create.
Imagen,	G,	(ninguno)
Lógico,	(ninguno),	L
Dinero,	$,	(ninguno)
Memo,	M*nn*,	M
Memo con formato,	F*nn*,	(ninguno)
Binario,	B*nn*,	(ninguno)
Objeto OLE,	O,	(ninguno)

where *descCampo* **is** *nuevoNombre* cambia el nombre de uno o más campos nombreCampo (nombre o número) a *nuevoNombre* (cadena). Ejemplo:

```
where  Apellido  is  Apellido cliente , 2 is  Nombre cliente
```

without *descCampo* elimina uno o más campos de la estructura. Ejemplo:

```
without 4,  Código del país
```

struct especifica en *tablaEstructCampo* un TCursor, nombre de tabla o TableView abiertos desde los que se toma la estructura a nivel de campo. A diferencia de la cláusula **like**, **struct** toma todos los controles de validación y la información de clave primaria. Utilice el método **enumFieldStruct** para generar *tablaEstructCampo* (o créela manualmente) antes de ejecutar **create**. Por ejemplo:

```
struct  CmpClien.db
```

indexStruct especifica en *tablaEstructIndice* un TCursor, nombre de tabla

o TableView abiertos desde los que se toma la información de índice secundario. Utilice el método **enumFieldStruct** para generar tablaEstructIndice (o créela manualmente) antes de ejecutar **create**. Por ejemplo:

```
indexStruct  IndClien.db
```

refIntStruct especifica un TCursor, nombre de tabla o TableView abiertos desde los que se toma la información de integridad referencial. Utilice el método **enumRefIntStruct** para generar *tablaEstructIntRef* (o créela manualmente) antes de ejecutar **create**. Por ejemplo:

```
refIntStruct  ClienRef.db
```

secStruct especifica en *tablaEstructSeg* un TCursor, nombre de tabla o TableView abiertos desde los que se toma la información de seguridad. Utilice el método **enumSecStruct** para generar *tablaEstructSeg* (o créela manualmente) antes de ejecutar **create**. Por ejemplo:

```
secStruct  ClienSeg.db
```

key *fieldID* especifica uno o más campos clave (sólo en tablas de Paradox). Es necesario especificar los campos clave por orden de izquierda a derecha. Por ejemplo:

```
key  Apellido ,  Nombre
```

Los campos se crean en el orden en que los especifique, ya sea explícitamente mediante una cláusula **with** o implícitamente por una o más cláusulas **like**. Las cláusulas **where** y **without** carecen de significado a menos que se les anteponga una cláusula **like**.

Nota: create no es un método, por lo que la notación de punto, como en la sentencia siguiente, es inapropiada:

```
TablaVar.create ()
```

En su lugar, utilice = para asignar la estructura **create** a una variable Table.

Ejemplo

El ejemplo siguiente crea la tabla de Paradox PARTES.DB. La tabla tiene tres campos: Número de Parte, Nombre Parte y Cantidad. Tiene un campo clave: Número de Parte.

```
; crearPartes::pushButton
method pushButton(var eventInfo Event)
  var
    PartesNuevas Table
    PartesTV TableView
  endVar
  if isTable( Partes.db ) then
    if msgQuestion( Confirme ,  Partes.db existe.
¿Reemplazarla? )   Yes  then
      return
    endif
  endif
  PartesNuevas = create  Partes.db
                 with  Número de Parte  :  A20", Nombre
Parte" :  A20",
                  Cantidad" :  S
                 key  Número de Parte
                 endCreate
  PartesTV.open( Partes.db )            ; abrimos la tabla nueva
endmethod
```

El ejemplo siguiente muestra dos formas de crear la tabla de dBASE NUEVENTA.DBF utilizando la misma estructura que en la tabla de dBASE VENTAS.DBF.

```
; versión 1
  var
    NuevasVentas Table
  endVar
 NuevasVentas = create  NueVenta.dbf
                like  Ventas.dbf
                endCreate


; versión 2
  var
    NuevasVentas Table
    VentasTC TCursor
  endVar
  VentasTC.open( Ventas.dbf )
  NuevasVentas = create
                 like VentasTC
                 endCreate
```

El ejemplo siguiente utiliza la opción **struct** para tomar información a nivel de campo, que incluye las claves primarias y los controles de validación, para su uso en una nueva tabla (para más información, consulte **enumFieldStruct**).

```
; crearNuevoCliente::pushButton
method pushButton(var eventInfo Event)
  var
    TablaClientes, NuevaTablaClientes Table
    ClienteTC TCursor
  endVar

  TablaClientes.attach( Clientes.db )
  if TablaClientes.isTable() then
      ; incluir desde Clientes los nombres de campo, las
condiciones de validez,
      ; y los campos clave en la nueva tabla CmpClien
    if TablaClientes.enumFieldStruct( CmpClien.db ) then

      ; definimos una variable TCursor para la tabla CmpClien
      ClienteTC.open( CmpClien.db )
      ClienteTC.edit()

      ; este bucle busca en la tabla CmpClien y cambia la
definición de las
      ; condiciones de validez de todos los campos
      scan ClienteTC :
        ClienteTC."_Valor Requerido" = 1    ; crear todos los
campos requeridos
      endScan

      ; ahora creamos NueClien.db y tomamos los nombres de
campo,
      ; las condiciones de validez y los campos clave de
CmpClien.db
      NuevaTablaClientes = create  NueClien.db
                             struct  CmpClien.db
                           endCreate
      ; NueClien requiere que se rellenen todos los campos

    else
      msgStop( Error ,  No puedo obtener la estructura de
campos de la tabla de Clientes. )
    endif

  else
    msgStop( Error ,  No encuentro la tabla de Clientes. )
  endif

endmethod
```

Vea también copy
enumFieldStruct
enumIndexStruct
enumRefIntStruct
enumSecStruct

cSamStd

Principiante

Método/ procedimiento Devuelve la desviación típica de muestra de un campo (columna) de una tabla.

Tipo Table

Sintaxis

1. cSamStd (const ***nombreCampo*** String) Number

2. cSamStd (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la desviación típica de la muestra de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cSamStd** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

El cálculo se basa en la varianza de muestra. La desviación típica de la muestra (en oposición a población) se calcula utilizando la fórmula:

sqrt((*sampVar*) * (*n*/*n*-1))

donde

sampVar = cVar(*nombreTabla*, *nombreCampo*)

n = cCount(*nombreTabla*, *nombreCampo*).

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo El ejemplo siguiente utiliza las dos formas de la sintaxis para calcular la desviación típica de la muestra de dos campos diferentes de la tabla *Respuest*. Este código se anexa al método **pushButton** de *mostrarEjemploEstándar*.

```
;mostrarEjemploEstándar::pushButton
method pushButton(var eventInfo Event)
  var
    TablaVacía Table
    NombreTabla String
    calcSalario, calcAños Number
  endVar
  NombreTabla =  Respuest

  TablaVacía.attach(NombreTabla)
  calcSalario = TablaVacía.cSamStd( Salario )       ; obtener
ejemplo de la
                              ; desviación Estándar de Salarios
  calcAños  = TablaVacía.cSamStd(2)                 ; asumir que
Años de servicio
                              ; es el campo 2
  msgInfo( Desviación Estandar Ejemplo ,            ; mostrar
información en una
                              ; ventana de diálogo
           Salarios :   + String(calcSalario,
           \nAños de servicio :  , calcAños))

endmethod
```

Vea también cSamVar
cAverage
cCount
cMax
cMin
cNpv
cSum
cVar
cStd

cSamVar

Principiante

Método/ procedimiento Devuelve la varianza de muestra de un campo (columna) de una tabla.

Tipo Table

Sintaxis **1. cSamVar** (const ***nombreCampo*** String) Number

2. cSamVar (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la varianza de muestra de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cSamVar** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

La varianza de la muestra (en oposición a población) se calcula utilizando la fórmula:

cVar(nombreTabla, nombrecampo) * (n/(n - 1))

donde

n = cCount(*nombreTabla, nombreCampo*)

Este método intenta, durante el periodo de reintento, aplicar un bloqueo sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo El ejemplo siguiente utiliza ambas formas de la sintaxis para calcular la varianza de la muestra de dos campos diferentes de la tabla *Respuest*. Este código se anexa al método **pushButton** de *mostrarEjemploVarianza*.

```
; mostrarEjemploVarianza::pushButton
method pushButton(var eventInfo Event)
  var
    TablaVacía Table
    NombreTabla String
    calcSalario, calcAños Number
  endVar
  NombreTabla =  Respuest

  TablaVacía.attach(NombreTabla)
  calcSalario = TablaVacía.cSamVar( Salario ) ; obtener la
varianza de Salarios
  calcAños  = TablaVacía.cSamVar(2)                ; asumir que
Años de servicio
                           ; es el campo 2
  msgInfo( Varianza Ejemplo ,                      ; mostrar
información en una
                           ; ventana de diálogo
          Salarios :   + String(calcSalario,
          \nAños de servicio :  , calcAños))

endmethod
```

Vea también cAverage
cCount
cMax
cMin
cNpv
cSamStd
cStd
cSum
cVar

cStd

Principiante

Método/ procedimiento Devuelve la desviación típica de un campo (columna) de una tabla.

Tipo Table

Sintaxis **1. cStd** (const ***nombreCampo*** String) Number

2. cStd (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la desviación típica de población de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. El cálculo se basa en la varianza; consulte **cVar**. Este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo Para este ejemplo, el método **pushButton** de *esteBotón* calcula la desviación típica de población de dos campos distintos y muestra los resultados en un cuadro de diálogo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    miTabla Table
    prueba1, prueba2 Number
  endVar
  MiTabla.attach( puntos.db )
  prueba1 = MiTabla.cStd( prueba1")
  prueba2 = MiTabla.cStd(2)                    ; asume que
prueba2 es el campo 2
  msgInfo( Desviación Estándar",
           resultado prueba1 :   + String(prueba1) +  \n  +
           resultado prueba2 :   + String(prueba2))
endmethod
```

Vea también cSamStd
cSamVar
cSum
cAverage
cCount
cMax
cMin
cNpv
cVar

cSum

Principiante

Método/ procedimiento Devuelve la suma de los valores de un campo (columna) de una tabla.

Tipo Table

Sintaxis **1. cSum** (const ***nombreCampo*** String) Number

2. cSum (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la suma de los valores de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cSum** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo Para este ejemplo, el método **pushButton** de *sumaDePedidos* utiliza ambas formas de la sintaxis de **cSum** para calcular totales para dos campos de PEDIDOS.DB.

```
; sumaDePedidos::pushButton
method pushButton(var eventInfo Event)
  var
    TablaPedidos Table
    TotalPedidos, CantidadPagada Number
    NombreTabla String
  endVar
  NombreTabla =  Pedidos


  TablaPedidos.attach(NombreTabla)
  TotalPedidos = TablaPedidos.cSum( Total Factura )
  CantidadPagada  = TablaPedidos.cSum(7)            ; asume que
la Cantidad Pagada es el
                      ; campo 7
  msgInfo( Totales de Pedidos ,
           Total Pedidos :   + String(TotalPedidos) +  \n  +
           Total Recibos :   + String(CantidadPagada))

endmethod
```

Vea también cAverage
cCount
cMax
cMin
cNpv
cSamStd
cSamVar
cStd
cVar

cVar

Principiante

Método/ procedimiento Devuelve la varianza de un campo de una tabla.

Tipo Table

Sintaxis 1. **cVar** (const ***nombreCampo*** String) Number

2. **cVar** (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la varianza de población de la columna de campos numéricos especificada por nombreCampo o númeroCampo (los campos se numeran de izquierda a derecha, comenzando con 1). Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cVar** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo Para este ejemplo, el método **pushButton** de *esteBotón* calcula la desviación de la varianza de población para dos campos distintos y muestra el resultado en un cuadro de diálogo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    miTabla Table
    prueba1, prueba2 Number
  endVar
  miTabla.attach( puntos.db )
  prueba1 = MiTabla.cVar( prueba1")
  prueba2 = MiTabla.cVar(2)                    ; asume que
prueba2 es el campo 2
  msgInfo( Varianza de Población",
           resultado Prueba1 :   + String(prueba1) +  \n  +
           resultado Prueba2 :   + String(prueba2))

endmethod
```

Vea también cAverage
cCount
cMax
cMin
cNpv
cSamVar
cSamStd
cSum

delete

Principiante

Método/ procedimiento Borra una tabla.

Tipo Table

Sintaxis **delete** ( ) Logical

Descripción Borra una tabla sin pedir confirmación. Compare este método con **empty**, que elimina datos de una tabla pero sin borrarla.

Este método falla si la tabla no está bloqueada.

Ejemplo El código siguiente borra RESPUEST.DB del directorio personal, si existe.

```
; borrarRespuesta::pushButton
method pushButton (var eventInfo Event)
  var
    Tabla Table
    NombreTabla String
  endVar

  NombreTabla = privDir() +  \\Respuest.db

  Tabla.attach (NombreTabla)
  if Tabla.isTable() then
    Tabla.delete()
    message (NombreTabla,   eliminada. )
  else
    message ( No puedo encontrar  , NombreTabla,  . )
  endif

endmethod
```

Vea también empty

dropIndex

Método Borra un archivo de índice asociado con una tabla.

Tipo Table

Sintaxis 1. (Tablas de Paradox) **dropIndex** ([const ***nombreIndice*** String]) Logical

2. (Tablas de dBASE) dropIndex ([const ***nombreIndice*** String [, const ***nombreEtiqueta*** String]]) Logical

Descripción Borra un archivo de índice o etiqueta de índice especificados.

Cuando se trabaja con una tabla de Paradox, nombreIndice es optativo. Si se omite nombreIndice, **dropIndex** borra el índice primario de la tabla (a menos que la tabla tenga un índice secundario, en cuyo caso el método falla).

Cuando se trabaja con una tabla de dBASE, es posible utilizar nombreIndice para especificar un archivo .NDX o emplear nombreIndice y nombreEtiqueta para especificar un archivo .MDX y una etiqueta de índice.

Este método precisa derechos exclusivos sobre la tabla si se está borrando un índice mantenido; en caso contrario, precisa un bloqueo de escritura.

dropIndex falla si el índice que se intenta borrar está siendo utilizado actualmente o si la tabla está abierta.

Ejemplo Para este ejemplo, el método **pushButton** de *esteBotón* borra la etiqueta *NomClien* de un archivo .MDX.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    TablaVentas Table
  endVar

  TablaVentas.attach( Ventas.dbf )            ; Ventas.dbf es
una tabla de dBASE.
  if isTable(TablaVentas) then          ; si TablaVentas es una
tabla

    ; eliminar la etiqueta NomClien del archivo indice2.mdx
    if TablaVentas.dropIndex( indice2.mdx ,  NomClien ) then
      msgInfo( Estado ,  Indice NomClien eliminado. )
    else
      msgInfo( Error ,  No puedo borrar NomClien de indice2. )
    endif

  else
    msgStop( ¡Atención! ,  No encuentro la tabla Ventas.dbf. )
  endif

endmethod
```

Vea también setIndex
usesIndexes

empty

Principiante

Método/ procedimiento Elimina todos los registros de una tabla de una base de datos.

Tipo Table

Sintaxis **empty** ( ) Logical

Descripción Elimina todos los registros de una tabla sin pedir confirmación. Esta operación no puede anularse. **empty** falla si la tabla está abierta.

empty elimina información de la tabla, pero no borra la tabla en sí. Compare este método con **delete**, que sí borra la tabla.

Este método intenta, en primer lugar, obtener derechos exclusivos sobre la tabla. Si los derechos exclusivos no son posibles, **empty** intenta aplicar un bloqueo de escritura sobre la tabla.

Si los derechos exclusivos son posibles, este método borra todos los registros de la tabla de una vez. Si sólo es posible un bloqueo de escritura, **empty** debe borrar los registros de uno en uno (lo cual puede ser prolongado en tablas grandes).

Si **empty** puede obtener derechos exclusivos sobre una tabla de dBASE, se borran todos los registros y la tabla se compacta (los registros se borran permanentemente). Si sólo es posible un bloqueo de escritura, este método marca todos los registros como borrados, pero no los elimina de la tabla (los registros pueden recuperarse de una tabla de dBASE si no se han eliminado con el método **compact**).

Ejemplo El ejemplo siguiente pide confirmación al usuario antes de borrar todos los registros de la tabla *Grabar*. Si el usuario no confirma la acción, este código usa **nRecords** para indicar cuántos registros existen en GRABAR.DB.

```
; tablaVacía::pushButton
method pushButton(var eventInfo Event)
  var
    NombreTabla String
    Tabla Table
  endVar
  NombreTabla =  Grabar.db

  Tabla.attach(NombreTabla)
  if isTable(NombreTabla) then
    if msgYesNoCancel( Confirme ,  ¿Vacío la tabla   +
NombreTabla +  ? ) =
     Yes  then
      if Tabla.empty() then
        message( Todos los registros de   + NombreTabla +   han
sido
        eliminados. )
      else
        message( Fallo al vaciar.  +
            NombreTabla +   tiene   + String(Tabla.nRecords())
+   registros. )
      endif
    endif
  else
    msgInfo( Error ,  No encuentro la tabla   + NombreTabla
+  . )
  endif

endmethod
```

Vea también delete

enumFieldNames

Método Rellena una matriz con los nombres de los campos de una tabla.

Tipo Table

Sintaxis **enumFieldNames** (var ***matrizCampos*** Array[] String) Logical

Descripción Rellena matrizCampos con los nombres de los campos de una tabla. Es necesario declarar matrizCampos como matriz redimensionable antes de activar este método. Si matrizCampos ya existe, este método la sustituye sin pedir confirmación.

Ejemplo Para este ejemplo, el método **pushButton** del botón *mostrarCampos* almacena nombres de campos en una matriz redimensionable y utiliza **view** para mostrar el contenido de la matriz.

```
; mostrarCampos::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    NombresDeCampo Array[] AnyType
  endVar

  Tabla.attach( Ventas.db )
  if Tabla.isTable() then
      Tabla.enumFieldNames("IndFecha", "porfecha",
"NombresDeCampo")
      ; muestra los nombres de los campos del índice
      ; de porfecha en IndFecha
     NombresDeCampo.view()
  else
      msgStop( Alto ,  No encuentro Ventas.db. )
  endIf

endmethod
```

Vea también enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct
enumRefIntStruct
enumSecStruct

enumFieldNamesInIndex

Método Rellena una matriz con los nombres de los campos existentes en el índice de una tabla.

Tipo Table

Sintaxis 1. (Tablas de Paradox) **enumFieldNamesInIndex** ([const ***nombreIndice*** String,]
var ***matrizCampos*** Array[] String) Logical

2. (Tablas de dBASE) **enumFieldNamesInIndex** [const ***nombreIndice*** String, [const ***nombreEtiqueta*** String,]]

var ***matrizCampos*** Array[] String) Logical

Descripción Rellena matrizCampos con los nombres de los campos del índice de una tabla, según se especifica en nombreIndice. Es necesario declarar matrizCampos como matriz redimensionable antes de activar este método. Si matrizCampos ya existe, este método la sustituye sin pedir confirmación.

Cuando se trabaja con una tabla de dBASE, el argumento nombreEtiqueta debe especificar una etiqueta de índice dentro de un archivo .MDX.

Si se omite, *nombreIndice* corresponde al índice que se utiliza actualmente.

Ejemplo Para este ejemplo, el método **pushButton** del botón *mostrarCamposDeIndice* almacena nombres de campos en una matriz redimensionable y emplea **view** para mostrar el contenido de la matriz.

```
; mostrarCamposDeIndice::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    NombresDeCampo Array[] AnyType
endVar

Tabla.attach( Ventas.dbf )
if Tabla.isTable() then
  Tabla.enumFieldNamesInIndex( IndFecha ,  porfecha ,
NombresDeCampo)
  ; muestra los nombres de los campos del índice de porfecha en
IndFecha
  NombresDeCampo.view()
else
  msgStop( Alto ,  No encuentro Ventas.dbf. )
endIf

endmethod
```

Vea también enumFieldNames
enumFieldStruct
enumIndexStruct
enumRefIntStruct
enumSecStruct

enumFieldStruct

Método Crea una tabla de Paradox que enumera la estructura de campos de una tabla.

Tipo Table

Sintaxis **enumFieldStruct** (const ***nombreTabla*** String) Logical

Descripción Crea la tabla de Paradox especificada en nombreTabla que enumera la estructura de una tabla. Si **nombreTabla** existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, este método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **struct** de una sentencia **create** para tomar su estructura de campos (incluyendo las claves primarias y controles de validación) para su uso en la nueva tabla. La estructura de *nombreTabla* se muestra a continuación.

Nombre de campo,	**Tipo, Tamaño**
Field Name,	A, 31
Type,	A, 31
Size,	S ,
Dec,	S ,
Key,	A, 1
_Required Value,	A, 1
_Min Value,	A, 255
_Max Value,	A, 255
_Default Value,	A, 255
_Picture Value,	A, 175
_Table Lookup,	A, 81
_Table Lookup Type,	A, 1
_Invariant Field ID,	S

Ejemplo Para este ejemplo, supóngase que se desea una nueva tabla llamada *NueClie* que es similar a la tabla *Clientes*. Sin embargo, se desea que todos los campos de *NueClie* sean campos necesarios. Para lograrlo, el código siguiente utiliza **enumFieldStruct** para cargar una nueva tabla (CMPCLIEN.DB) con la información a nivel de campos de *Cliente*. A continuación, el código explora *CmpClien* y modifica las definiciones de campos para que cada registro describa un campo que será necesario. *CmpClien* se suministra en la cláusula **struct** de una sentencia **create**.

```
; crearNuevoCliente::pushButton
method pushButton(var eventInfo Event)
  var
    TablaClientes, NuevaTablaClientes Table
    ClienteTC TCursor
  endVar

  TablaClientes.attach( Clientes.db )
  if TablaClientes.isTable() then

  ; incluir de Clientes los nombres de campo, las condiciones
de validez,
  ; y los campos clave en la nueva tabla CmpClien
    if TablaClien.enumFieldStruct( CmpClien.db ) then

      ; definimos una variable TCursor para la tabla CmpClien
      ClienteTC.open( CmpClien.db )
      ClienteTC.edit()

      ; este bucle busca en la tabla CmpClien y
      ; cambia la definición de las condiciones de validez de
todos los campos
      scan ClienteTC;
        ClienteTC."_Valor Requerido" = 1    ; crear todos los
campos requeridos
      endscan

      ; ahora creamos NUECLIEN.DB y tomamos los nombres de
campo,
      ; las condiciones de validez y los campos clave de
NUECLIEN.DB
      NuevaTablaClientes = create  NueClien.db
                               struct  CmpClien.db
                           endcreate

      ; NUECLIEN.DB requiere que se rellenen todos los campos

    else
      msgStop( Error ,  No puedo obtener la estructura de
campos de la tabla
      Clientes. )
    endif

  else
    msgStop( Error ,  No encuentro la tabla Clientes. )
  endif

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumIndexStruct
enumRefIntStruct
enumSecStruct

enumIndexStruct

Método Crea una tabla de Paradox que enumera la estructura de índices secundarios de una tabla.

Tipo Table

Sintaxis **enumIndexStruct** (const **NOMBRETABLA** String) Logical

Descripción Crea la tabla de Paradox especificada en nombreTabla listando la estructura de índices secundarios de una tabla. Si nombreTabla existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, este método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **indexStruct** de una sentencia **create** para tomar sus índices secundarios para su uso en la nueva tabla.

La estructura de *nombreTabla* se muestra a continuación.

Nombre de campo,	**Tipo,Tamaño**
infoHeader,	A, 1
szName,	A, 127
szTagName,	A, 31
szFormat,	A, 31
bPrimary,	A, 1
bUnique,	A, 1
bDescending,	A, 1
bMaintained,	A, 1
bCaseInsensitive,	A, 1
bSubset,	A, 1
bExpIdx,	A, 1
bKeyExpType,	N ,
szKeyExp,	A, 220
szKeyCond,	A, 220
FieldNo,	N ,
FieldName,	A, 31

Para las tablas de dBASE, *nombreTabla* incluye información para los índices que se utilizarían si la tabla estuviera abierta. Para especificar qué índices se asocian con una variable Table, utilice el método **usesIndexes** y, a continuación, ejecute **enumIndexStruct** para crear una tabla que liste esos índices.

Ejemplo Para este ejemplo, supóngase que se desea una nueva tabla llamada *NueClien* que sea similar a la tabla *Clientes*. Sin embargo, no se desea tomar información de integridad referencial ni de seguridad. Para lograrlo, el código siguiente emplea **enumFieldStruct** y **enumIndexStruct** para generar dos tablas: CMPCLIEN.DB e INDCLIEN.DB. *CmpClien* e *IndClien* se suministran entonces en las cláusulas **struct** e **indexStruct** de una

sentencia **create**.

```
; crearNuevoCliente::pushButton
method pushButton(var eventInfo Event)
  var
    TablaClientes, NuevaTablaClientes Table
  endVar

  TablaClientes.attach( Clientes.db )
  if TablaClientes.isTable() then

    TablaClientes.enumFieldStruct( CmpClien.db )
    TablaClientes.enumIndexStruct( IndClien.db )

    ; ahora creamos NUECLIEN.DB
    ; tomamos los nombres de campo, las condiciones de validez
y los campos
    ; clave de CMPCLIEN.DB
    ; toma el índice secundario de INDCLIEN.DB
    NuevaTablaClientes = create  NueClien.db
                              struct  CmpClien.db
                              indexstruct "CmpClien.db
                         endCreate
  else

    msgStop( Error ,  No encuentro la tabla Clientes. )
  endif

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumRefIntStruct
enumSecStruct

enumRefIntStruct

Método Crea una tabla de Paradox listando la información de integridad referencial de una tabla.

Tipo Table

Sintaxis **enumRefIntStruct** (const ***nombreTabla*** String) Logical

Descripción Escribe la información de integridad referencial de la tabla actual en nombreTabla. Si nombreTabla existe, este método la sustituye sin pedir confirmación. Si nombreTabla está abierta, el método falla. Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **refIntStruct** de una sentencia **create** para tomar su información de integridad referencial para su uso en la nueva tabla. La estructura de *nombreTabla* se muestra a continuación.

Nombre de campo,	**Tipo, Tamaño**
infoHeader,	A, 1
Ref Name,	A, 31
Other Table,	A, 81
Slave,	A, 1
Modify,	A, 1
Delete,	A, 1
FieldNo,	N ,
aiThisTabField,	A, 31
Other FieldNo,	N ,
aiOthTabField,	A, 31

Ejemplo Este ejemplo utiliza **enumRefIntStruct** para escribir la información de integridad referencial de CLIENTES.DB en la tabla *RefClien*. A continuación, el código suministra *RefClien* en la cláusula **refIntStruct** de una sentencia **create**.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla, Tabla2 Table
  endVar

  Tabla.attach( Clientes.db )
  Tabla.enumRefIntStruct( RefClien.db )            ; Escribe
información referencial de
                       ; integridad en RefClien
  Tabla.enumFieldStruct( CmpClien.db )       ; Escribe
estructura de campos en
                       ; CmpClien
  Tabla2 = create  NueClien.db
              struct  CmpClien.db            ; utiliza la
información de los campos
                       ; de CmpClien
              refIntStruct  RefClien.db      ; usa información
referencial de
                       ; integridad de RefClien
           endCreate

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct
enumSecStruct

enumSecStruct

Método Crea una tabla de Paradox listando la información de seguridad de una tabla.

Tipo Table

Sintaxis **enumSecStruct** (const ***nombreTabla*** String) Logical

Descripción Crea la tabla de Paradox especificada en *nombreTabla* listando la información de seguridad (derechos de acceso) de una tabla. Si *nombreTabla* existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, este método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **secStruct** de una sentencia **create** para tomar su información de seguridad para su uso en la nueva tabla. La estructura de *nombreTabla* se muestra a continuación.

Nombre de campo,	**Tipo, Tamaño**
infoHeader,	A, 1
iSecNum,	N ,
eprvTable	N ,
eprvTableSym,	N ,
iFamRights,	N ,
iFamRightsSym,	A, 10
szPassword,	A, 31
fldNum,	N ,
aprvFld,	N ,
aprvFldSym,	A, 10

Ejemplo Este ejemplo crea una nueva tabla a partir de la información de seguridad asociada con la tabla *Secretos*. El código emplea **enumSecStruct** para escribir información de seguridad en la tabla *SecInfo* y, entonces, emplea la tabla para crear la tabla *MisSecrs*.

```
; desvelarSecretos::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla,Tabla2 Table
  endVar

  Tabla.attach( Secretos.db )
  Tabla.enumSecStruct( SecInfo.db )

  Tabla2 = create  MisSecrs.db
             like  Secretos.db
             secStruct  Secretos.db
           endCreate

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct
enumRefIntStruct

familyRights

Método Comprueba si el usuario puede crear o modificar objetos en la familia de una tabla.

Tipo Table

Sintaxis **familyRights** (const ***derechos*** String) Logical

Descripción Devuelve True si el usuario tiene derechos sobre el tipo de objeto especificado en *derechos*; de lo contrario, devuelve False. *derechos* es una cadena de un solo carácter F (ficha), R (informe), S (valores de plantilla) o V (controles de validación) , que indica el tipo de objeto en que se está interesado.

Ejemplo Este ejemplo indica en un cuadro de diálogo si se tienen derechos F sobre PEDIDOS.DB.

```
; mostrarDerechosDeFamilia::pushButton
method pushButton(var eventInfo Event)
  var
    TablaClientes Table
  endVar

  TablaClientes.attach( Pedidos.db )
  if TablaClientes.isTable() then
    msgInfo( Derechos ,  Derechos de ficha :   +
             String(TablaClientes.familyRights( F )))
    ; muestra True si se tienen derechos de ficha para
Pedidos.db
  else
    msgStop( Error ,  No encuentro Pedidos.db. )
  endif

endmethod
```

Vea también tableRights

fieldName

Método/ procedimiento Devuelve el nombre de campo de una tabla, dado un número de campo.

Tipo Table

Sintaxis **fieldName** (const ***númeroCampo*** SmallInt) String

Descripción Devuelve el nombre del campo especificado en númeroCampo. Si *númeroCampo* es mayor que el número de campos de la tabla, fieldName devuelve una cadena vacía.

Ejemplo El ejemplo siguiente utiliza **fieldName** para mostrar el nombre del campo número dos de la tabla *Respuest*. Este código se anexa al método estándar **pushButton** de un botón.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    NombreCampo, NombreTabla String
    NumeroCampo SmallInt
  endVar
  NombreTabla =  Respuest.db
  NumeroCampo = 2

  Tabla.attach(NombreTabla)
  if isTable(Tabla) then
    NombreCampo = Tabla.fieldName(NumeroCampo)  ; guardar el
nombre del campo 2
                           ; en NombreCampo
    msgInfo( El nombre del campo   + String(NumeroCampo) +
es: , NombreCampo)
  else
    msgStop( Lo siento ,  No encuentro la tabla   + Nombretabla
+  . )
  endIf

endmethod
```

Vea también fieldNo

fieldNo

Método/ procedimiento Devuelve la posición de un campo en una tabla.

Tipo Table

Sintaxis **fieldNo** (const ***nombreCampo*** String) SmallInt

Descripción Devuelve la posición de nombreCampo en una tabla, o 0 si no se encuentra nombreCampo. Los campos están numerados de izquierda a derecha, comenzando con 1.

Ejemplo Este código muestra el número de campo del campo Fecha si existe en la tabla Pedidos.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Pedidos Table
    NumeroCampo SmallInt
  endVar
  Pedidos.attach( Pedidos.db )
  NumeroCampo = Pedidos.fieldNo( Fecha )

  if NumeroCampo = 0 then
    msgInfo( Tabla Pedidos ,  Fecha no es un campo de esta
tabla. )
  else
    msgInfo( Tabla Pedidos ,  Fecha es el campo número   +
String(NumeroCampo))
  endif

endmethod
```

Vea también fieldName

fieldType

Método/ procedimiento Devuelve el tipo de un campo de una tabla.

Tipo Table

Sintaxis **1. fieldType** (const ***nombreCampo*** String) String

2. fieldType (const ***númeroCampo*** SmallInt) String

Descripción Devuelve el tipo de datos de un campo. Si no se encuentra el campo, devuelve:

Tipo de campo,	**Tabla de Paradox,**	**Tabla de dBASE**
Alfanumérico,	ALPHA,	(ninguno)
Carácter,	(ninguno),	CHARACTER
Fecha,	DATE,	DATE
Entero,	SHORT,	(ninguno)
V. de coma flotante,	NUMERIC,	FLOAT (IV), NUMERIC (III+ or IV)
Imagen,	GRAPHIC,	(ninguno)
Lógico,	(ninguno),	BOOLEAN
Dinero,	MONEY,	(ninguno)
Memo,	MEMO,	MEMO
Memo con formato,	FMTMEMO,	(ninguno)
Binario,	BINARYBLOB,	(ninguno)
Objeto OLE,	OLEOBJ,	(ninguno)

Ejemplo Este ejemplo emplea una matriz dinámica para almacenar el tipo de cada campo de la tabla Vidamar y muestra el contenido de la matriz en un cuadro de diálogo.

```
; mostrarTiposDeCampos::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    i SmallInt
    TipoCampos DynArray[] AnyType
    NombreTabla String
  endVar

  NombreTabla =  Vidamar.db
  if isTable(NombreTabla) then
    Tabla.attach(NombreTabla)
      ; este bucle FOR carga el DynArray con los tipos de
campos de Vidamar
    for i from 1 to Tabla.nFields()
      TipoCampos[Tabla.fieldName(i)] = Tabla.fieldtype(i)
    endFor
      ; ahora mostramos el contendio del DynArray
    TipoCampos.view( Tipos de campos de   +NombreTabla)
  else
    msgStop( Lo siento ,  No encuentro la tabla   + NombreTabla
+  . )
  endif
endmethod
```

Vea también fieldNo

index

Palabra clave Crea un índice primario o secundario en los campos especificados de una tabla.

Tipo Table

Sintaxis

1. **index**

   [**maintained**] *descTabla* **on** *IDCampo*

   endIndex

2. **index** *descTabla*

[**maintained**] (Paradox)

[**descending**] (dBASE)

[**unique**] (dBASE)

[**primary**] (Paradox)

[**caseInsensitive**] (Paradox)

on

{ *descCampo* [, *descCampo*] [**to** *nombreIndice]*

|

{ exprClave

to *nombreArchivoNDX* | ***tag*** *nombreEtiqueta*
[**of** *nombreArchivoMDX*]

|

for condición } }

endIndex

Descripción Genera un índice secundario para un campo de una tabla. Paradox utiliza el índice para acelerar las consultas y búsquedas que acceden al campo.

Si se está creando un índice en una tabla con clave, la palabra clave **maintained** indica a Paradox que se desea que el índice se mantenga incrementalmente. El mantenimiento incremental implica que, después de guardar los cambios realizados en una tabla, sólo se actualiza la parte del índice afectada por los cambios. Los tablas con clave que tienen índices secundarios mantenidos incrementalmente, suelen producir un mejor rendimiento que los que no se configuran de esta forma, aunque existe el inconveniente oculto de que el mantenimiento del índice consume recursos.

Si se utiliza la palabra clave **maintained** para las tablas de Paradox y se especifica una tabla sin clave para su indexación, **index** falla. Sin la palabra clave **maintained**, no se realiza el mantenimiento automático de un índice secundario. Para ello, es necesario utilizar **reIndex** o **reIndexAll**. En las tablas de dBASE, todos los archivos de índice abiertos se mantienen automáticamente.

La cláusula **on**, que especifica qué campos se deben utilizar, tiene dos formas: una para las tablas de Paradox y otra para las de dBASE.

Para las tablas de Paradox, utilice

on *descCampo* [, *descCampo*] **to** *nombreIndice*

donde descCampo especifica uno o más nombres o números de campo, mientras que nombreIndice especifica el archivo de índice en que se escribirá. Para las tablas de Paradox, los archivos de índice secundario tienen extensiones .Xnn e .Ynn, donde nn referencia la posición estructural del campo que se indexa expresada en notación hexadecimal.

Para las tablas de dBASE, utilice

exprClave **to** *nombreArchivoNDX* | **tag** *nombreEtiqueta* [**of** *nombreArchivoMDX*]

que permite elegir entre un archivo .NDX o una etiqueta de un archivo .MDX. Si se omite *nombreArchivoMDX*, el nombre de archivo .MDX por defecto es el mismo que el de la tabla.

En aplicaciones multiusuario, **index** aplica automáticamente un bloqueo completo sobre la tabla mientras que la indexa. Si algún otro usuario o aplicación ha bloqueado la tabla, el comando se reitenta continuamente durante el periodo de reintento definido actualmente. Si no ha podido lograrse el bloqueo al final del periodo, **index** falla. Es posible utilizar el comando **lock** para confirmar que se puede bloquear la tabla antes de utilizar el comando **index**.

Puesto que las tablas con clave tienen índices primarios sobre sus campos clave, no es necesario crear un índice secundario para estos campos, ya que sólo se duplicaría el índice primario. Sin embargo, puede emplearse **index** para crear un índice secundario. Los campos Memo, OLE y de Imagen no pueden indexarse.

Es conveniente desarrollar las aplicaciones sin preocuparse de los índices y, posteriormente, introducirlos donde sea conveniente para acelerar las consultas y búsquedas.

En las situaciones siguientes, el comando **index** no se completará de forma satisfactoria:

Ya existen demasiados índices (máximo de 255 para una sola tabla).

Un índice que se está definiendo ya está en uso.

index no es un método, por lo que no es apropiada la notación de punto, como en

```
TablaVar.index()
```

En su lugar, se crea una estructura de índice para especificar cómo se indexa la tabla.

Ejemplo

El ejemplo siguiente crea un índice primario para una tabla de Paradox llamada CLIENTES.DB. Si no se encuentra la tabla *Clientes*, o no puede bloquearse, este método cancela la operación **index**. Si este código logra indexar la tabla, el código enumera los campos indexados en una matriz y muestra el contenido de la matriz en un cuadro de diálogo.

```
; nuevasClavesDeCliente::pushButton
method pushButton(var eventInfo Event)
  var
    TablaAIndexar String
    Tabla Table
    CamposIndexados Array[] String
  endVar
  TablaAIndexar =  Clientes.db

  if isTable(TablaAIndexar) then
    Tabla.attach(TablaAIndexar)
    if not Tabla.lock( Full ) then
      msgStop( ¡Atención! ,  No puedo bloquear la tabla   +
TablaAIndexar +  . )
      return
    endif
    index Tabla              ; creamos un índice primario nuevo
para Clientes.db
      primary
      on  N° de Cliente ,  Nombre ,  Dirección
    endIndex

    ; ahora se muestran los campos clave en una ventana de
diálogo
    Tabla.enumFieldNamesInIndex(CamposIndexados)
    CamposIndexados.view( Campos clave primarios de   +
TablaAIndexar)

  else
    msgStop( ¡Alto! ,  No Encuentro la tabla   + TablaAIndexar
+  . )
  endif

endmethod
```

Paradox CLIENTES.DB. Si lo logra, este código enumera los nombres de los campos indexados en una matriz y los muestra en un cuadro de diálogo.

```
; índiceDelEstadoDeLaCiudad::pushButton
method pushButton(var eventInfo Event)
var
  TablaAIndexar           String
  Tabla                   Table
  CamposIndexados Array[] String
  tv                      TableView
endVar
TablaAIndexar =  Clientes.db

if isTable(TablaAIndexar) then
  Tabla.attach(TablaAIndexar)
  if not Tabla.lock( Full ) then
    msgStop( ¡Atención! ,  No puedo bloquear la tabla   +
TablaAIndexar +  . )
    return
  endif

  index Tabla                     ; creamos un índice secundario
de Clientes.db
    maintained                    ; mantener el índice
incrementalmente
    on  Ciudad ,  Provincia       ; indexar sobre estos dos
campos
    to  EstadoCiudad        ; llamar al índice  EstadoCiudad

  endIndex

    ; mostramos los campos clave de Clientes en una ventana de
diálogo
  Tabla.enumFieldNamesInIndex( EstadoCiudad , CamposIndexados)

  CamposIndexados.view( Campos del índice EstadoCiudad )

  else msgStop( ¡Atención! ,  No encuentro la tabla   +
TablaAIndexar +  . )  endif

endmethod
```

Vea también create
enumIndexStruct
setIndex
usesIndexes

isAssigned

Método Informa de si se ha asignado un valor a una variable Table.

Tipo Table

Sintaxis **isAssigned** ( ) Logical

Descripción Devuelve True si se ha asignado un valor a una variable Table; en caso contrario, devuelve False. Es posible asignar un valor a una variable Table utilizando **create** o **attach**.

Nota: El valor devuelto True no garantiza que la variable esté anexada a una tabla, o, si lo está, que la tabla exista. Por ejemplo, el código siguiente muestra True en un cuadro de diálogo:

```
var
  tabla table
endVar
tabla.attach( zxcv.qw )                              ; conectada
a algún archivo sin sentido
msgInfo( ¿Asignada? ,tabla.isAssigned())      ; visualiza True
```

Ejemplo Este ejemplo comprueba si se ha asignado un valor a la variable Table *Tabla* antes de anexarla a una tabla. El código siguiente va en la ventana Var de la ficha *estaFicha*:

```
; estaFicha::var
  var
    Tabla table
  endVar
```

El código siguiente se anexa al método **pushButton** del botón *esteBotón*. En este código, si *Tabla* no está asignada aún, se anexa a la tabla *Pedidos*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)

if NOT Tabla.isAssigned() then
  Tabla.attach( Pedidos.db )
else
  msgStop( Error ,  No puedo conectar Tabla a Pedidos.db )
endif

endmethod
```

Vea también attach
isTable

isEmpty

Método/ procedimiento Informa de si una tabla contiene algún registro. Puede ser una operación prolongada con tablas de dBASE.

Tipo Table

Sintaxis **isEmpty** ( ) Logical

Descripción Devuelve True si no hay registros en una tabla; en caso contrario, devuelve False.

Ejemplo Para este ejemplo, el método **pushButton** del botón *infRegNum* muestra el número de registros de la tabla *Pedidos*. Si *Pedidos* está vacía, este código alerta al usuario de que la tabla está vacía.

```
; infRegNum::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    NombreTabla String
  endVar
  NombreTabla =  Pedidos.db
  if isTable(NombreTabla) then
    Tabla.attach(NombreTabla)
    if Tabla.isEmpty() then      ; si la tabla Pedidos.db está
vacía
      msgStop( ¡Atención! ,  La tabla   + NombreTabla +   está
vacía. )
    else
      msgInfo( La tabla   + NombreTabla +   tiene  ,
String(Tabla.nRecords()) +
                registros. )
    endIf
  else
    msgStop( Lo Siento ,  No puedo abrir la tabla   +
NombreTabla +  . )
  endif
endmethod
```

Vea también empty
isTable

isEncrypted

Método Informa de si una tabla está cifrada.

Tipo Table

Sintaxis **isEncrypted** ( ) Logical

Descripción Devuelve True si una tabla está protegida mediante contraseña; en caso contrario, devuelve False. Un TCursor no puede abrirse en una tabla cifrada hasta que se utilice el método **addPassword** del tipo Session para presentar la contraseña requerida. Este método no informa de si un usuario tiene derechos de acceso a la tabla para ello, emplee **tableRights** .

Ejemplo Este ejemplo utiliza **isEncrypted** para averiguar si la tabla Secretos está protegida (cifrada); si lo está, el código ejecuta **unProtect** para eliminar permanentemente de la tabla la protección mediante contraseña.

```
; desencrip::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla table
    NombreTabla String
  endVar
  NombreTabla = "Secretos.db"
  Tabla.attach(NombreTabla)
  if Tabla.isEncrypted() then
    Tabla.unprotect( coge007")  ; borra la contraseña
permanentemente
                                ; se asume que coge007 es la
contraseña maestra
  endif

endmethod
```

Vea también protect
tableRights
Session**::**addPassword
Session**::**removePassword

isShared

Método/ procedimiento Informa de si una tabla está compartida actualmente.

Tipo Table

Sintaxis **isShared** ( ) Logical

Descripción Devuelve True si una tabla es compartida por otro usuario de una red; en caso contrario, devuelve False. **isShared** no informa de si una tabla está siendo compartida por otra sesión.

Ejemplo En este ejemplo, una variable Table se anexa a la tabla *Clientes*. Este código utiliza **setExclusive** para dar al usuario derechos exclusivos sobre *Clientes* y emplea **isShared** para demostrar el efecto que **setExclusive** tiene sobre las tablas en un entorno multiusuario.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
     Tabla Table
    NombreTabla String
  endVar
  NombreTabla =  Clientes.db

  Tabla.attach(NombreTabla)
  Tabla.setExclusive(True)        ; proporciona al usuario
derechos exclusivos
          ; sobre Clientes.db
  if Tabla.isShared() then        ; esto nunca es cierto
                                        ; las tablas exclusivas
no se pueden compartir
    msgStop(  ,  ¡Este mensaje nunca aparece! )
  else
    msgInfo( Estado Multiusuario ,"La Tabla   + NombreTabla +
no está
             compartida.")
  endif

endmethod
```

Vea también tableRights

Syntax **isShared** (const ***tableName*** String) Logical

isTable

Método/ procedimiento Informa de si existe una tabla en una base de datos.

Tipo Table

Sintaxis **isTable** ( ) Logical

Descripción Devuelve True si la variable Table representa una tabla que puede abrirse; en caso contrario, devuelve False.

Ejemplo Este ejemplo utiliza **isTable** para comprobar que la tabla *Clientes* existe antes de realizar alguna operación con la tabla. Si *Clientes* existe en la base de datos por defecto, este código almacena los nombres de campos de *Clientes* en una matriz y, entonces, muestra el contenido de la matriz en un cuadro de diálogo.

```
; mostrarCamposDeClientes::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    NombreTabla String
    NombresDeCampo Array[] AnyType
  endVar
  NombreTabla =  Clientes.db

  Tabla.attach(NombreTabla)
  if isTable(Tabla) then
    Tabla.enumFieldNames(NombresDeCampo)
    NombresDeCampo.view( Campos de   + NombreTabla)
  else
    msgStop( ¡Atención! ,  No encuentro la tabla   +
NombreTabla +  . )
  endif

endmethod
```

Vea también isAssigned

lock

Principiante

Método Bloquea una tabla especificada.

Tipo Table
Principiante

Sintaxis **lock** (const ***tipoBloqueo***) String Logical

Descripción Intenta aplicar un bloqueo sobre la tabla, donde *tipoBloqueo* es uno de los siguientes valores String: Write (escritura), Read (lectura), Full (Completo) o Any (Cualquiera). Si lo logra, este método devuelve True; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente anexa una variable Table a *Clientes*, aplica un bloqueo de escritura sobre la tabla y utiliza **reIndex** para reconstruir el índice *Teléfono*. Una vez que se reconstruye el índice, este código desbloquea *Clientes* para que otros usuarios de una red puedan obtener acceso a la tabla.

```
; reindexarClientes::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    TablaParadox String
  endVar
  TablaParadox =  Clientes.db

  if isTable(TablaParadox) then
    Tabla.attach(TablaParadox)
    if Tabla.lock( Full ) then          ; intentamos conseguir
acceso exclusivo
      Tabla.reIndex( Teléfono )         ; reconstruimos el
índice sobre Teléfono
      Tabla.unLock( Full )              ; desbloqueamos la tabla
    else
      msgStop( Lo Siento ,  No puedo bloquear la tabla   +
TablaParadox +  . )
    endif
  else
    msgStop( Lo Siento ,  No encuentro la tabla   +
TablaParadox +  . )
  endif
endmethod
```

Ver también unlock
Session::lock
Session::unlock

nFields

Método/ procedimiento Devuelve el número de campos de una tabla.

Tipo Table

Sintaxis **nFields** ( ) LongInt

Descripción Devuelve el número de campos de una tabla.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* muestra el número de campos de la tabla *Vidamar*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
  endVar
  Tabla.attach( Vidamar.db )
  msgInfo( Vidamar ,  Vidamar tiene   + String(Tabla.nFields())
+   campos. )
endmethod
```

Vea también nKeyFields
nRecords

nKeyFields

Método/ procedimiento Devuelve el número de campos existentes en el índice primario o actual de una tabla.

Tipo Table

Sintaxis **nKeyFields** ( ) LongInt

Descripción Devuelve el número de campos del índice actual de una tabla. Cuando se utiliza con una tabla de Paradox, este método trabaja con el índice primario; cuando se emplea con una de dBASE, trabaja con el índice especificado por el método **setIndex**.

Ejemplo Este ejemplo informa del número de campos clave primarios de una tabla de Paradox (PEDIDOS.DB) y el número de campos clave primarios del índice APELLIDO.MDX de una tabla de dBASE (PUNTOS.DBF).

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    TablaParadox, TabladBase Table
    ncc LongInt
  endVar

  TablaParadox.attach( Pedidos.db )
  ncc = TablaParadox.nKeyFields()       ; número de campos clave
del índice
                ; primario
  msgInfo( Pedidos ,  Pedidos.db tiene   + String(ncc) +
campos clave. )
  TabladBase.attach( Puntos.dbf )
  TabladBase.setIndex( Apellido.MDX )   ; número de campos clave
del índice
                ; Apellido.MDX
  ncc = TabladBase.nKeyFields()
  msgInfo( Puntos.dbf ,  Puntos.dbf tiene   + String(ncc) +
campos clave. )

endmethod
```

Vea también nFields

nRecords

Método/ procedimiento Devuelve el número de registros de una tabla.

Tipo Table

Sintaxis **nRecords** ( ) LongInt

Descripción Devuelve el número de registros de una tabla. En las tablas de dBASE, puede ser un proceso largo.

Ejemplo El ejemplo siguiente pide al usuario confirmación antes de borrar todos los registros de la tabla *Grabar*. Si el usuario no confirma la acción, el código emplea **nRecords** para indicar cuántos registros hay en GRABAR.DB.

```
; tablaVacía::pushButton
method pushButton(var eventInfo Event)
var
  NombreTabla String
  Tabla Table
endVar
NombreTabla =  Grabar.db
Tabla.attach(NombreTabla)

if isTable(NombreTabla) then
  if msgYesNoCancel( Confirmación ,  ¿Desea vaciar la tabla   +
NombreTabla +
                     ? ) =  Yes  then
    Tabla.empty()
    message( Se han borrado todos los registros de   +
NombreTabla +  . )
  else
    message(NombreTabla +   tiene   + String(Tabla.nRecords())
+   registros. )
  endif
else
  msgInfo( Error ,  No Encuentro la tabla   + NombreTabla
+  . )
endif

endmethod
```

Vea también nFields
nKeyFields

protect

Método/ procedimiento Cifra una tabla y le asigna una contraseña de propietario.

Tipo Table

Sintaxis **protect** ([const ***contraseña*** String]) Logical

Descripción Asigna una contraseña de propietario a una tabla. Una tabla protegida está cifrada, y no es posible acceder a ella sin presentar la contraseña especificada en contraseña. Si se omite el argumento optativo contraseña, este método elimina permanentemente la contraseña de propietario. Si la tabla ya tiene una contraseña, **protect** falla.

Una vez que una tabla está protegida, es posible utilizar el método **addPassword** para presentar la contraseña de una tabla protegida y el método **removePassword** para retirar la contraseña y reproteger la tabla. Se diferencia el uso de mayúsculas o minúsculas en *contraseña*: una tabla protegida con Sésamo no se abrirá con sésamo .

No confunda **protect** con **lock**: **protect** cifra tablas, mientras que **lock** controla el acceso simultáneo a las tablas.

Ejemplo En este ejemplo, el método **pushButton** de *protegerSecretos* protege mediante contraseña la tabla *Secretos* de la base de datos por defecto.

```
; protegerSecretos::pushButton
method pushButton(var eventInfo Event)
  var
    DatosSecretos Table
  endVar
  DatosSecretos.attach( Secretos.db )
  if not DatosSecretos.isEncrypted() then
    DatosSecretos.protect( coge007")       ; Proteger la tabla
con la clave  coge007"
  endIf

endmethod
```

Vea también isEncrypted
Session**::**addPassword
Session**::**removePassword

reIndex

Método Reconstruye los archivos de índice especificados.

Tipo Table

Sintaxis **1**. (Tablas de Paradox) **reIndex** (const ***nombreIndice*** String) Logical

2. (Tablas de dBASE) **reIndex** (const ***nombreIndice*** String [, const ***nombreEtiqueta*** String]) Logical

Descripción Reconstruye un índice (o etiqueta de índice) que no se mantiene automáticamente. Cuando trabaje con una tabla de Paradox, utilice nombreIndice para especificar un índice. Cuando trabaje con una de dBASE, utilice nombreIndice para especificar un archivo .NDX, o nombreIndice y nombreEtiqueta para especificar una etiqueta de índice de un archivo .MDX. Este método precisa el acceso exclusivo a la tabla.

Ejemplo El ejemplo siguiente anexa una variable Table a Clientes (una tabla de Paradox), aplica un bloqueo de escritura sobre la tabla y utiliza **reIndex** para reconstruir el índice *Teléfono*.

```
; reindexarClientes::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    TablaParadox String
  endVar
  TablaParadox =  Clientes.db

  Tabla.attach(TablaParadox)
  if Tabla.lock( Full ) then      ; intentamos conseguir acceso
exclusivo
    Tabla.reIndex( Teléfono )     ; reconstruir el índice sobre
Teléfono
    Tabla.unLock( Full )          ; desbloquear la tabla
  else
    msgStop( Lo siento ,  No puedo bloquear la tabla   +
TablaParadox +  . )
  endif

endmethod
```

Vea también reIndexAll

reIndexAll

Método Reconstruye todos los archivos de índice asociados con una tabla.

Tipo Table

Sintaxis **reIndexAl**l ( ) Logical

Descripción Reconstruye todos los archivos de índice asociados con un tabla. Este método precisa derechos exclusivos sobre la tabla para reconstruir un índice mantenido, y precisa un bloqueo de escritura para reconstruir un índice no mantenido.

Ejemplo Para este ejemplo, el método **pushButton** de un botón intenta aplicar un bloqueo completo sobre la tabla *Clientes*. Si **lock** es satisfactorio, este código reconstruye todos los índices de la tabla *Clientes* y la desbloquea.

```
; reindexarLosIndicesDeClientes::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    TablaParadox String
  endVar
  TablaParadox =  Clientes.db

  Tabla.attach(TablaParadox)
  if Tabla.lock( Full ) then      ; intentamos conseguir acceso
exclusivo
    Tabla.reIndexAll()            ; reconstruir todos los
índices
    Tabla.unLock( Full )          ; desbloquear la tabla
  else
    msgStop( Lo siento ,  No puedo bloquear la tabla   +
TablaParadox +  . )
  endif

endmethod
```

Vea también reIndex

rename

Principiante

Método/ procedimiento Renombra una tabla.

Tipo Table

Sintaxis **rename** (const ***nombreTablaDestino*** String) Logical

Descripción Cambia el nombre de una tabla a nombreTablaDestino. Si la tabla mencionada en nombreTablaDestino existe, se produce un error.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo completo sobre la tabla. Si no es posible aplicar el bloqueo, se produce un error.

Ejemplo El código siguiente renombra CLIENTES.DB a ANTCLIEN. Si AntClien existe, este ejemplo ofrece al usuario una oportunidad de cancelar la operación.

```
; renombrarClientes::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    NombreAnterior, NombreNuevo String
  endVar

  NombreAnterior =  Clientes.db
  NombreNuevo =  AntClien.db

  Tabla.attach(NombreAnterior)
  if Tabla.isTable() then
    if isTable(NombreNuevo) then
      if msgQuestion( Confirme , NombreNuevo +   ya existe.
¿Desea reemplazarlo? ) <<>>  Yes  then
        message( Operación cancelada. )
        return
      endif
    endif
    Tabla.rename(NombreNuevo)
    message( Se ha cambiado el nombre de   + NombreAnterior +
por   + NombreNuevo)
  else
    msgStop( ¡Atención! ,  No encuentro la tabla   +
NombreAnterior +  . )
  endif
endmethod
```

Vea también copy

setExclusive

Método Especifica si se conceden al usuario derechos exclusivos sobre una tabla cuando se abre.

Tipo Table

Sintaxis **setExclusive** ([const ***yesNo*** Logical])

Descripción Especifica en *yesNo* si se abre una tabla con derechos exclusivos o compartidos. Este método no aplica ningún bloqueo sobre la tabla un bloqueo exclusivo se aplica sobre la tabla sólo cuando se abre .

Por defecto, las tablas se abren en modo compartido. El argumento optativo *yesNo* especifica si se definen derechos exclusivos: el valor Yes concede derechos exclusivos, mientras que No permite que la tabla se abra en modo compartido. Si se omite este argumento, se toma Yes por defecto.

Ejemplo Este ejemplo demuestra cómo **setExclusive** afecta a los derechos de acceso de una tabla. El código define una variable Table para la tabla *Clientes* y ejecuta **setExclusive** de forma que *Clientes* se abra exclusivamente. A continuación, se abre un TCursor para *Clientes*. Si el TCursor se abre satisfactoriamente, tiene derechos exclusivos sobre la tabla y el código ejecuta el método **lockStatus** de TCursor para indicar que se ha aplicado un bloqueo exclusivo a *Clientes*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    tc    TCursor
  endvar

  Tabla.attach( Clientes.db )
  if Tabla.isTable() then
    Tabla.setExclusive()
    ; asignamos derechos exclusivos a la variable de la Tabla

    ; intentamos abrir un TCursor en Clientes.db
    ; si tenemos éxito, tc tendrá derechos exclusivos sobre
Clientes.db
    if tc.open(Tabla) then

      ; si tc.open tuvo éxito, este mensaje indica
      ; que tc tiene un bloqueo exclusivo sobre Clientes.db
      msgInfo( Estado de bloqueo , tc.lockStatus( Exclusive ))

    else
      ; si no se indica que falló la apertura
      msgInfo( Estado ,  No puedo abrir Clientes.db )
    endif

  else
    msgInfo( Estado ,  No encuentro la tabla Clientes.db. )
  endif
  if tc.isAssigned() then   ; si el TCursor fue abierto
    tc.close()              ; cerrar tc ahora Clientes.db no
está bloqueado
                      ; y no puede abrirlo ningún otro usuario
  endif

endmethod
```

Vea también attach
setIndex
setReadOnly

setFilter

Método Especifica un rango de registros para su inclusión.

Tipo Table

Sintaxis **setFilter** ([const ***valorCoincExacta*** AnyType,] *
const ***valorMin*** AnyType, const ***valorMax*** AnyType) Logical

Descripción Especifica condiciones para la inclusión de un rango de registros. Los que cumplen las condiciones se incluyen al abrir la tabla; los que no las cumplen, no se incluyen.

Este método compara los criterios especificados con los valores de los campos correspondientes del índice de una tabla; **setFilter** falla si la tabla no está indexada. Para filtrar los registros por el valor de un solo campo, especifique los valores en valorMin y valorMax. Por ejemplo, la sentencia siguiente comprueba los valores del primer campo del índice de cada registro.

```
Tabla.setFilter (14, 88)
```

Si un valor es menor que 14 o mayor que 88, ese registro se descarta. Para especificar una coincidencia exacta en un solo campo, asigne el mismo valor a valorMin y valorMax. Por ejemplo, la sentencia siguiente descarta todos los valores excepto 55.

```
Tabla.setFilter (55, 55)
```

Es posible filtrar registros por los valores de más de un campo. Para ello, especifique coincidencias exactas excepto para el último elemento de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas de Borland y Paradox (suponiendo que son los primeros campos del índice) y valores entre 100 y 500, ambos incluidos, en el tercer campo.

```
TcVar.setFilter( Borland ,"Paradox", 100, 500)
```

La ejecución de **setFilter** sin argumentos redefine el filtro para incluir toda la tabla.

Ejemplo En este ejemplo, supóngase que el campo clave de Artículo es *Pedido Nº* y que se desea saber el total del pedido número 1005. El código siguiente anexa una variable Table a la tabla *Artículo*, limita el rango de registros a los que contengan 1005 en el primer campo del índice primario y utiliza **cSum** para calcular el total del pedido 1005.

```
; obtenerSumaDetalles::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    NombreTabla String
  endVar
  NombreTabla =  articulo.db
  Tabla.attach(NombreTabla)

    ; se limitan los registros que han de aparecer a aquellos
que contengan
    ; el valor 1005 en el primer campo del índice primario
  Tabla.setFilter(1005, 1005)

    ; ahora mostramos el total de Pedido N° 1005
  msgInfo( Total del Pedido 1005", Tabla.cSum( Total"))

endmethod
```

Vea también setIndex

setIndex

Método Especifica un índice para una tabla.

Tipo Table

Sintaxis **1.** (Tablas de Paradox) **setIndex** (const ***nombreIndice*** String) Logical

2. (Tablas de dBASE) **setIndex** (const ***nombreIndice*** String [const ***nombreEtiqueta*** String]) Logical

Descripción Especifica un índice que se utilizará cuando se abra una tabla.

Cuando se trabaja con una tabla de Paradox, utilice nombreIndice para especificar un índice. Cuando se trata de una tabla de dBASE, es posible utilizar nombreIndice para especificar un archivo .NDX, o nombreIndice y nombreEtiqueta para indicar una etiqueta de índice de un archivo .MDX.

Ejemplo En este ejemplo, supóngase que la tabla de Paradox *Clientes* tiene un índice secundario llamado EstadoCiudad. El código siguiente especifica EstadoCiudad con **setIndex** para definir una llamada a **setFilter**. Cuando el filtro designado se define para *Clientes*, este ejemplo carga un DynArray con información de la tabla filtrada y muestra el contenido del DynArray en un cuadro de diálogo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    TablaClientes Table
    tc TCursor
    dy DynArray[] Anytype
  endVar

  TablaClientes.attach( Clientes.db )
  if isTable(TablaClientes) then

    ; ahora utilizamos el índice secundario cuyo nombre es
EstadoCiudad
    TablaClientes.setIndex( EstadoCiudad )

    ; filtramos todo excepto Madrid
    TablaClientes.setFilter( Madrid ,  Madrid )

    ; abrimos un TCursor para el resultado de filtrar la tabla
Clientes
    if tc.open(TablaClientes) then

      ; buscamos en la tabla y cargamos el DynArray con
      ; nombres de compañia (Nombre) y números de teléfono
      scan tc:
        dy[tc.Nombre] = tc.Teléfono
      endScan
      ; mostramos el contenido del DynArray
      dy.view( Numeros de teléfono de Madrid )

    else
      msgStop( Error ,  No puedo abrir el TCursor. )
    endif

  else
    msgStop( Error ,  No encuentro Clientes.db )
  endif

endmethod
```

Vea también setFilter

setReadOnly

Método Especifica si se conceden al usuario derechos de sólo lectura sobre una tabla al abrirla.

Tipo Table

Sintaxis **setReadOnly** ([const ***yesNo*** Logical])

Descripción Especifica si se conceden al usuario derechos de sólo lectura sobre una tabla cuando se abre. Este método falla si la tabla ha sido bloqueada por otro usuario o si está abierta.

El argumento optativo yesNo especifica si se definen derechos de sólo lectura: el valor Yes concede derechos de sólo lectura, mientras que No concede derechos completos sobre la tabla. Si se omite este argumento, se toma Yes por defecto.

Ejemplo El código siguiente anexa una variable Table a la tabla Pedidos, ejecuta **setReadOnly** para limitar los derechos de usuario y abre un TCursor para *Pedidos*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    tc TCursor
  endVar

  Tabla.attach( Pedidos.db )      ; conectar Tabla con
Pedidos.db
  Tabla.setReadOnly()             ; acceder a Tabla sólo para
lectura
  tc.open(Tabla)                  ; abrir un TCursor para
Pedidos.db

endmethod
```

Vea también setExclusive

showDeleted

Método Especifica si se muestran los registros borrados de una tabla de dBASE.

Tipo Table

Sintaxis **showDeleted** ([const ***YesNo*** Logical]) Logical

Descripción Los registros borrados en una tabla de dBASE no se eliminan inmediatamente, sino que se marcan para su borrado y permanecen en la tabla. **showDeleted** especifica si se muestran estos registros cuando se abre la tabla. **showDeleted** sólo es relevante para las tablas de dBASE.

El argumento optativo yesNo especifica lo que se hará: el valor Yes muestra los registros borrados, mientras que No los oculta. Si se omite este argumento, se toma Yes por defecto. Si no se ejecuta este método antes de utilizar la variable Table asociada con la tabla, no se muestran los registros borrados.

Ejemplo Para este ejemplo, el método **pushButton** anexado al botón *mostrarRegistrosBorrados* instruye que se muestren los registros borrados de una variable Table.

```
; mostrarRegistrosBorrados::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
  endVar

  Tabla.attach( Pedidos.db )
  if isTable(Tabla) then

    ; mostrar los registros borrados de Pedidos.db
    Tabla.showDeleted(Yes)

    ; mostrar la suma de todos los registros (borrados y no
borrados)
     msgInfo( Total de Registros , Tabla.nRecords())
  else
    msgStop( Error ,  No encuentro la tabla Pedidos. )
  endif

endmethod
```

Vea también compact
delete

sort

Palabra clave Ordena una tabla.

Tipo Table

Sintaxis **sort** *tablaOrigen* [**on** *listaNombresCampos* [D]] [**to** *tablaDestino*] **endSort**

Descripción Rellena una ficha de ordenación para la tabla especificada en tablaOrigen y realiza la ordenación.

tablaOrigen puede ser de los tipos Table, TCursor o String, mientras que tablaDestino puede ser de los tipos Table o String.

Si se incluye la cláusula optativa **on**, la tabla se ordena por el primer campo especificado en listaNombresCampos. Se utiliza cada campo subsiguiente, uno tras otro, para establecer enlaces con los campos anteriores. Un argumento optativo **D** después de un nombre de campo especifica una ordenación descendente. Si se omite la cláusula **on**, los registros se ordenan ascendentemente, de izquierda a derecha en todos los campos.

Si se incluye la cláusula optativa **to**, el resultado de la ordenación se escribe en la tabla descrita en tablaDestino. Si ya existe dicha tabla, este método la sustituye sin pedir confirmación. Si se omite la cláusula **to**, los registros ordenados se sitúan de nuevo en *tablaOrigen* (esto falla si la tabla está abierta). Es necesario especificar la cláusula **to** si la tabla origen tiene clave.

sort aplica automáticamente un bloqueo completo sobre las tablas que se ordenan si el resultado se escribirá en la misma tabla. En caso contrario, es necesario un bloqueo de escritura para la tabla origen y uno completo para la tabla destino.

sort no es un método, por lo que es inapropiada la notación de punto, como en la sentencia siguiente.

```
Tabla.sort()
```

En su lugar, cree una estructura que especifique cómo ordenar la tabla.

Ejemplo Este ejemplo ordena *Clientes* por los campos Nombre y Ciudad y sitúa los resultados en la tabla *OrdClien*.

```
; ordenarTablaDeClientes::pushButtton
method pushButton(var eventInfo Event)
  var
    TablaClientes Table
    tv TableView
  endVar

  TablaClientes.attach( Clientes.db )

  sort TablaClientes
    on  Nombre  D,  Ciudad  D     ; ordenamos en sentido
descendente
    to  OrdClien.db
  endSort

  tv.open( OrdClien.db )          ; abrimos la tabla una vez
ordenada

endmethod
```

Vea también index

TCursor::sortTo

subtract

Principiante

Método/ procedimiento Extrae los registros de una tabla en otra tabla.

Tipo Table

Sintaxis **1. subtract** (const ***nombreTablaDestino*** String) Logical

2. **subtract** (const ***nombreTablaDestino*** Table) Logical

Descripción Comprueba si hay registros en la tabla origen que también se hallen en nombreTablaDestino. Si es así, **subtract** los borra de nombreTablaDestino sin pedir confirmación.

Si nombreTablaDestino no tiene clave, **subtract** borra todos los registros que coincidan exactamente con cualquier registro de la tabla origen. Si nombreTablaDestino tiene claves, **subtract** borra todos los registros con claves que coincidan exactamente con los valores de los campos clave correspondientes de la tabla origen. Tanto si nombreTablaDestino tiene claves como si no, este método sólo considera los campos que podrían tener claves. Por ejemplo, se consideran los campos numéricos, pero no los campos memo con formato.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo completo sobre ambas tablas. Si no es posible aplicarlos, se produce un error.

Ejemplo El código siguiente extrae de *Clientes* los registros que coincidan con los existentes en la tabla *Insertar* que se halla en el directorio personal.

```
; eliminarClientes::pushButton
method pushButton(var eventInfo Event)
  var
    TablaIns, TablaClien Table
    fs FileSystem
    NombreTabla String
  endVar
  NombreTabla = privDir() +  \\Insertar.db

  TablaIns.attach(NombreTabla)
  if TablaIns.isTable() then
    TablaIns.subtract(TablaClien)       ; eliminar los registros
coincidentes de
          ; TablaClien y TablaIns
  else
    msgInfo( Lo siento ,  No encuentro la tabla   + NombreTabla
+  . )
  endif

endmethod
```

Vea también add

tableRights

Método/ procedimiento Especifica si el usuario tiene derechos para realizar ciertas operaciones sobre una tabla.

Tipo Table

Sintaxis **tableRights** (const ***derechos*** String) Logical

Descripción Informa acerca de los derechos de un usuario sobre una tabla, donde derechos es uno de los siguientes:

ReadOnly (leer en la tabla, pero sin modificarla)

Modify (introducir o modificar datos)

Insert (añadir nuevos registros)

InsDel (añadir y borrar registros)

All (realizar todas las operaciones)

Ejemplo Este ejemplo informa de si el usuario tiene todos los derechos (All) sobre la tabla *Pedidos*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Derechos Logical
    Tabla Table
  endVar

  Tabla.attach( Pedidos.db )
  if Tabla.isTable() then
    Derechos = Tabla.tableRights( All )

    ; esta sentencia muestra True si se tienen todos los
derechos sobre
    ; Pedidos.db
    msgInfo( ¿Todos los derechos? , Derechos)

  else
    message( No encuentro la tabla Pedidos. )
  endif

endmethod
```

Vea también familyRights

type

Método Devuelve el tipo de una tabla.

Tipo Table

Sintaxis **type** ( ) String

Descripción Devuelve el valor de cadena Paradox o dBASE para informar del tipo de una tabla.

Ejemplo En este ejemplo, supóngase que una ficha tiene un botón llamado compactar y un marco de tabla asociado con la tabla Pedidos. El código siguiente (anexado al método **pushButton**) compacta (elimina los registros borrados) de la tabla si **type** devuelve dBASE; en caso contrario, un mensaje informa al usuario de que Pedidos es una tabla de Paradox

```
; compactar::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
  endVar
  Tabla.attach( PEDIDOS )
  if Tabla.type() =  dBASE  then
    Tabla.compact()
  else
    msgStop( ¡Atención! ,  Pedidos es una tabla de tipo   +
Tabla.type() +  . )
  endif

endmethod
```

Vea también attach

unAttach

Método Termina la asociación entre una variable Table y una tabla en disco.

Tipo Table

Sintaxis **unAttach** ( ) Logical

Descripción Termina la asociación (creada mediante **attach**) entre una variable Table y una tabla de un archivo. No es necesario terminar la asociación entre una variable Table y una tabla para anexar la misma variable a otra tabla **unAttach** se ejecuta automáticamente cuando se asigna otra tabla a una variable Table .

Ejemplo En este ejemplo, se utiliza una sola variable Table para resumir la información de ventas de dos tablas diferentes. Una vez que la variable Table (*Tabla*) ya no es necesaria, este código ejecuta **unAttach** para terminar la asociación entre *Tabla* y la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla   Table
    q1, q2  Number
    mensg   String
  endVar

  Tabla.attach( q1_venta.db )           ; conectamos la tabla
q1_ventas
  q1 = Tabla.cSum( Cantidad )           ; obtenemos el sumatorio
  Tabla.attach( q2_venta.db )           ; no necesitamos
desconectar
  q2 = Tabla.cSum( Cantidad )           ; obtenemos el sumatorio
de q2_ventas
  Tabla.unAttach()                      ; no necesitamos Tabla
para nada más
                                  ; de modo que finalizamos la
asociación con
          ; q2_ventas
  if q1 = q2 then
    mensg =  Las ventas deben incrementarse.
  else
    mensg =  Las ventas son altas.
  endIf
  msgInfo( Ventas , mensg)
endmethod
```

Vea también attach

unlock

Principiante

Método Desbloquea una tabla especificada.

Tipo Table

Sintaxis **unlock** (const ***tipoBloqueo*** String) Logical

Descripción Intenta eliminar los bloqueos aplicados explícitamente sobre una tabla. *tipoBloqueo* debe ser una expresión que se evalúe en uno de los valores de cadena siguientes: Write (escritura), Read (lectura), Full (Completo) o Any (Cualquiera). Si lo consigue, este método devuelve True; en caso contrario, devuelve False.

Ejemplo En este ejemplo, el método pushButton de *ActualizarClientes* ejecuta una consulta desde un archivo existente y añade registros de la tabla *Respuest* en la tabla *Clientes*. Este código intenta aplicar un bloqueo de escritura sobre la tabla *Clientes* antes de añadir registros en ella. Si se logra el bloqueo, este código añade registros de *Clientes* y utiliza **unlock** para desbloquear *Clientes*.

```
; ActualizarClientes::pushButton
method pushButton(var eventInfo Event)
  var
    NuevoCliente Query
    Tabla Table
    TablaDestino String
  endVar
  TablaDestino =  Clientes.db

  if executeQBEFile( obtClien.qbe ) then      ; si la consulta
tiene éxito
    Tabla.attach( Respuest.db )
    if TablaDestino.lock( Write ) then        ; intentamos
bloquear la tabla para
                       ; escritura
      Tabla.add(TablaDestino)                 ; añadimos
registros incluidos en
                       ; Respuest.db
      TablaDestino.unLock( Write )            ; desbloqueamos la
tabla
     else
        msgStop( ¡Atención! ,  No puedo escribir en la tabla
+ TablaDestino +   bloqueada. )
    endif
  else
    msgStop( ¡Atención! ,  Falló la Consulta. )
  endIf

endmethod
```

Vea también lock

unProtect

Método/procedimiento Descifra y elimina una contraseña de propietario de una tabla.

Tipo Table

Sintaxis **1**. (Procedimiento) **unProtect** (const ***nombreTabla*** String [const ***Contraseña*** String])

2. (Método) **unProtect** ([const ***contraseña*** String])

Descripción Elimina permanentemente una contraseña de propietario de una tabla. Una tabla protegida está cifrada y no es posible acceder a ella sin presentar la contraseña especificada en contraseña. Si ya se ha emitido la contraseña principal para una tabla, no es necesario el argumento contraseña.

Ejemplo Este ejemplo elimina permanentemente la protección mediante contraseña de la tabla *Secretos*.

```
; desencriptar::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla table
    NombreTabla String
  endVar

  NombreTabla="Secretos.db"
  Tabla.attach(NombreTabla)
  if Tabla.isEncrypted() then
    Tabla.unprotect( coge007")    ; borra la contraseña
permanentemente
                                  ; se asume que coge007 es la
contraseña maestra
  endif

endmethod
```

Vea también isEncrypted
protect
Session**::**addPassword
Session**::**removePassword

usesIndexes

Método Especifica los archivos de índice que se utilizarán y mantendrán con una tabla de dBASE.

Tipo Table

Sintaxis **usesIndexe**s (const ***nombreArchivoIndice*** String [const ***nombreArchivoIndice*** String]* Logical

Descripción Especifica en nombreArchivoIndice uno o más archivos de índice (.NDX y .MDX) para su uso con una tabla de dBASE. Este método no abre la tabla, pero especifica los archivos de índice que se abren cuando se abre la tabla. No es necesario utilizar este método para abrir el archivo de producción (por ejemplo, archivos .MDX) para una tabla de dBASE estos archivos se abren automáticamente .

Este método falla si no existen todos los archivos de índice especificados.

Ejemplo Este ejemplo ejecuta **usesIndexes** para especificar dos índices diferentes de la tabla *Pedidos* y abre un TCursor para la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
  var
    Tabla Table
    tc     TCursor
  endvar

  Tabla.attach( Pedidos.dbf )
  if Tabla.isTable() then

    ; especificamos los índices NomEstad y NomPedid
    Tabla.usesIndexes( NOMESTAD.NDX ,  NOMPEDID.NDX )

    ; ahora intentamos abrir la tabla utilizando los índices
especificados
    if tc.open(Tabla) then
      if tc.locate( Estado ,  ES ,  Contacto ,  Pérez ) then
        msgInfo( Fecha del Pedido , tc."Fecha Pedido")
      else
        msgStop( Error ,  No encuentro los valores. )
      endif
    endif
  else
    msgStop( Error ,  No encuentro la tabla Pedidos.dbf. )
  endif

endmethod
```

Vea también reIndex
reIndexAll
setIndex

add

Método Añade los registros de una tabla en otra.

Tipo TCursor

Sintaxis **1. add** (const ***tablaDestino*** String [, const ***añadir*** Logical [, const ***actualizar*** Logical]]) Logical

2. add (const ***tablaDestino*** Table [, const ***añadir*** Logical [, const ***actualizar*** Logical]]) Logical

3. add (const ***tablaDestino*** TCursor [, const ***añadir*** Logical [, const ***actualizar*** Logical]]) Logical

Descripción Añade los registros indicados por un TCursor en la tabla destino especificada en *tablaDestino*. Si ésta no existe, el método la crea. Las tablas origen y destino pueden ser del mismo tipo o de tipos distintos; en cualquier caso, las tablas deben tener estructuras de campos compatibles.

Los argumentos *añadir* y *actualizar* pueden ser True o False. Cuando es True, *añadir* añade registros al final de una tabla no indexada o en los lugares adecuados de una tabla indexada. Cuando es True, *actualizar* compara los registros de ambas tablas y, cuando los valores clave coinciden, sustituye los datos de la tabla destino. Cuando ambos son True, los registros con valores de clave coincidentes se actualizan, y los demás se añaden. Estos argumentos son optativos, pero si se especifica *actualizar*, es necesario especificar también *añadir*. Si se omiten, ambos son True. A continuación, se presentan algunas sentencias de ejemplo:

```
MiTCursor.add (TuTabla, False, True)  ; especificamos
actualizar
MiTCursor.add (TuTabla)               ; especificamos
actualizar y añadir ; por defecto
```

Cuando las tablas están indexadas, **add** utiliza los campos indexados para determinar qué registros actualizar y cuáles añadir. Cuando la tabla destino no está indexada, **add** falla si *actualizar* es True. Las violaciones de clave, si las hay, se enumeran en VIOLCLAV.DB en el directorio personal del usuario. Este método sustituye un VIOLCLAV.DB existente o crea uno, si es necesario. **add** respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro.

Este método intenta, durante el periodo de reintento, aplicar bloqueos de escritura a las tablas origen y destino. Si no es posible aplicar alguno de los bloqueos, el método falla.

Ejemplo En este ejemplo, supóngase que las tablas *AntClien* y *NueClien* existen en el directorio actual. El código siguiente asocia un TCursor con cada una de las tablas, añade los registros de *NueClien* en *AntClien* y añade todos los registros en una tabla llamada *MiClien*. Si *MiClien* no existe en el directorio actual, **add** la crea. Este código se anexa al método **pushButton** de un botón.

```
; obtenerMiCliente::pushButton
method pushButton(var eventInfo Event)
var
  dTC, sTC TCursor
```

```
endVar

if sTC.open("AntClien.db") and
   dTC.open("NueClien.db") then  ; si pueden asociarse ambos
objetos TCursor
   dTC.add(sTC, True)            ; añadimos registros de
AntClien a NueClien
                                 ; ahora sTC tiene registros de
ambas tablas

   sTC.add("MiClient.db", True)  ; añadimos sTC a la tabla
MICLIEN

   sTC.close()                   ; cerramos ambos objetos
TCursor
   dTC.close()
else     msgStop("¡Atención!", "No puedo abrir una o más
tablas.")
endif

endmethod
```

Vea también copy
subtract

atFirst

Método Informa de si el TCursor señala al primer registro de una tabla.

Tipo TCursor

Sintaxis **atFirst** ( ) Logical

Descripción Devuelve True si el TCursor señala al primer registro de una tabla; si no es así, devuelve False.

Ejemplo Este ejemplo supone que una ficha tiene un botón llamado *desplazarseAlOrigen* y un objeto multirregistro asociado con Pedidos.DB. El código anexado al método **pushButton** de *desplazarseAlOrigen* utiliza **atFirst** para determinar si el TCursor está en el primer registro. Si no lo está, este código se desplaza al primer registro.

```
; desplazarseAlOrigen::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar

tc.attach(Pedidos)           ; Pedidos es un objeto
multirregistro
if not tc.atFirst() then     ; si no estamos en el primer
registro
  tc.home()                  ; nos desplazamos a él.
  Pedidos.moveToRecord(tc)   ; apuntamos al primer registro
else
  msgStop("Actualmente estamos en el registro " +
String(tc.recNo()),                   "Estás en el primer
registro")
endif
endmethod
```

Vea también atLast
bot
eot

atLast

Método Informa de si el TCursor señala al último registro de una tabla.

Tipo TCursor

Sintaxis **atLast** ( ) Logical

Descripción Devuelve True si el TCursor señala al último registro de una tabla; en caso contrario, devuelve False.

Ejemplo Este ejemplo supone que una ficha tiene un botón llamado *desplazarseAlFinal* y un objeto multirregistro asociado con PEDIDOS.DB. El código anexado al método **pushButton** de *desplazarseAlFinal* utiliza **atLast** para determinar si el TCursor está en el último registro. Si no lo está, este código se desplaza al último registro.

```
; desplazarseAlFinal::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar

tc.attach(Pedidos)
if not tc.atLast() then     ; si no estamos en el último
registro
  tc.end()                  ; nos desplazamos a él.
  Pedidos.moveToRecord(tc)  ; apuntamos al último registro
else
  msgStop("Actualmente estamos en el registro " +
String(tc.recNo()),
          "Estás en el último registro")
endif
endmethod
```

Vea también atFirst
bot
eot

attach

Método Asocia un TCursor con un UIObject.

Tipo TCursor

Sintaxis

1. attach (const ***objecto*** UIObject) Logical

2. attach (const TCursor Origen TCursor) Logical

3. attach (const ***tv*** TableView) Logical

Descripción Asocia un TCursor con un UIObject (objeto<DC255> en la sintaxis 1), con otro TCursor (*TCursorOrigen* en la sintaxis 2) o con un objeto TableView (*tv* en la sintaxis 3). Los datos proceden de la tabla subyacente el TCursor no obtiene datos de los registros que no se han consignado (por ejemplo, porque el registro se está editando o acaba de añadirse) . **attach** devuelve True si es satisfactorio; en caso contrario, devuelve False.

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla asociado con PEDIDOS.DB, y otro marco de tabla asociado con ARTICULO.DB. La tabla *Pedidos* tiene un enlace con *Articulo* del tipo uno-varios. También hay un botón llamado *encontrarDetalles* en la ficha. Supóngase que se desea que el usuario pueda buscar en toda la tabla *Articulo* no sólo en los registros enlazados con el pedido actual . En este caso, el método **pushButton** de *encontrarDetalles* busca pedidos que incluyan el número de parte actual.

Este código se anexa a la ventana Var para el botón *encontrarDetalles*:

```
; encontrarDetalles::Var
Var
  lineaTC TCursor ; ejemplo de artículo para búsquedas
endVar
```

El código siguiente se anexa al método **open** del botón *encontrarDetalles*. Este código asocia el TCursor *lineaTC* con Articulo.DB.

```
;encontrarDetalles::open
method open(var eventInfo Event)
lineaTC.open("articulo.db")
endmethod
```

El código siguiente se anexa al método **pushButton** de *encontrarDetalles*:

```
;encontrarDetalles::pushButton
method pushButton(var eventInfo Event)
var
  cantidad    Number
  PedidoTC    TCursor
  PedidoNum   Number
endVar

cantidad = ARTICULO.cantidad ; obtener la cantidad de la línea
actual
; lineaTC fue declarada en Var y fue abierta por el método open
if NOT lineaTC.locateNext("cantidad", cantidad) then
  lineaTC.locate("cantidad", cantidad)
endif
```

```
PedidoTC.attach(PEDIDOS)       ; conectar el TCursor con la
tabla
PedidoTC.locate("N° de Pedido", lineaTC."N° de Pedido")
PEDIDOS.moveToRecord(PedidoTC) ; desplazarse a CLIENTES y
                               ; sincronizar con el TCursor
ARTICULO.moveTo()              ; desplazar el TCursor al
detalle de ARTICULO
; desplazar el TCursor al registro equivalente
ARTICULO.locate("N° de Pedido", cantidad)
endmethod
```

Este código se anexa al método **close** de *encontrarDetalles*:

```
; encontrarDetalles::close
method close(var eventInfo Event)
lineaTC.close()   ; cerrar el TCursor de articulo
endmethod
```

Vea también isValid

attachToKeyViol

Método Anexa un TCursor al registro existente que tiene la misma clave que el registro que se ha intentado consignar.

Tipo TCursor

Sintaxis **attachToKeyViol** (const ***TCantiguo*** TCursor) Logical

Descripción Después de que se produce una violación de clave, **attachToKeyViol** anexa un TCursor con el registro existente el registro que existía antes de que se produjera la violación de clave . Especifique en *TCAntiguo* el TCursor que señala al registro que provocó la violación de clave (el registro nuevo, sin almacenar).

Este método proporciona una forma de comparar registros conflictivos antes de sustituir o desechar un cambio en un registro existente. *TCAntiguo* debe estar señalando ya al nuevo registro (no almacenado aún).

Ejemplo Este ejemplo demuestra cómo puede emplearse **attachToKeyViol** después de una violación de clave. El código declarar dos TCursor: *violClaveTC* y *regOriginalTC*. El código abre *violClaveTC* para la tabla *Pedidos* e inserta deliberadamente un registro cuyo valor de clave entra en conflicto con otro registro. Entonces, el ejemplo intenta almacenar el nuevo registro en la tabla, lo cual fuerza una violación de clave. En este momento, si el usuario elige ver el registro existente, el código ejecuta **attachToKeyViol**, anexa el segundo TCursor (*regOriginalTC*) con el registro original y muestra el registro en un cuadro de diálogo mediante **view**. Si el usuario elige actualizar el registro original con los datos del nuevo registro, este ejemplo ejecuta el método **updateRecord** para hacerlo; en caso contrario, el código no realiza cambios.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  ViolClaveTC, RegOriginalTC TCursor
  Reg DynArray[] AnyType
endvar

ViolClaveTC.open("Pedidos.db")    ; abrimos un TCursor para
Pedidos
ViolClaveTC.edit()                ; activamos el modo de
Edición para el TCursor
ViolClaveTC.insertRecord()        ; insertamos un nuevo
registro
ViolClaveTC."N° de Pedido" = 1011 ; 1011 es una clave duplicada

; si se intenta almacenar un nuevo registro, la orden falla
if NOT ViolClaveTC.postRecord() then

; conectamos RegOriginalTC con el registro existente
RegOriginalTC.attachToKeyViol(ViolClaveTC)

; damos al usuario la opción de ver el registro existente
if msgQuestion("!La clave existe!",
   "¿Desea ver el registro?") = "Yes" then

   RegOriginalTC.copyToArray(Reg)  ; copiamos el registro
existente en Reg
```

```
   Reg.view("Registro Original")   ; mostramos Reg en un cuadro
de diálogo

endif

; damos al usuario la opción de reemplazar el registro
existente
if msgQuestion("Confirme la Actualización",
   "¿Desea reemplazar el registro existente?") = "Yes" then

  ; forzamos el almacenamiento del nuevo registro
  ViolClaveTC.updateRecord(True)
  else
    message("Se ha dejado intacto el registro original.")
    sleep(1500)
  endif
else
  message("Almacenado con el número 1011.")
endif

endmethod
```

Vea también postRecord
updateRecord

bot

Método Comprueba si se intenta un desplazamiento más allá del principio de una tabla.

Tipo TCursor

Sintaxis **bot** ( ) Logical

Descripción Devuelve True si un comando intenta ir más allá del primer registro de una tabla; en caso contrario, devuelve False. **bot** se restaura en la siguiente operación de desplazamiento.

Ejemplo Este ejemplo desplaza un TCursor hacia atrás en una tabla y, entonces, muestra un mensaje. Este código se anexa al método **pushButton** de un botón.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  MiTabla TCursor
endVar
  MiTabla.open("lugares.db")
  MiTabla.end()              ; nos desplazamos al final de la
tabla
while MiTabla.bot() = False  ; repetimos hasta llegar al
principio
  MiTabla.priorRecord()      ; nos desplazamos hacia atrás a
través
                             ; de la tabla
endWhile
  msgInfo("Principio", "Estás en el principio.")

endmethod
```

Vea también atFirst
atLast
end
eot
home

cancelEdit

Principiante

Método Termina el modo editar sin almacenar los cambios en el registro actual.

Tipo TCursor

Sintaxis **cancelEdit** ( ) Logical

Descripción Termina el modo editar para un TCursor sin guardar cambios en el registro actual. Es necesario utilizar **cancelEdit** antes de desplazar el TCursor desde el registro actual, o consignar o desbloquear de alguna otra forma el registro. Una vez que se desplaza el TCursor, se consignan los cambios en el registro.

Ejemplo El código de este ejemplo se anexa al método **pushButton** del botón *cambiarClave*. Este ejemplo asocia un TCursor con la tabla *Clientes* e intenta cambiar un valor de un campo con clave. Si el registro no puede almacenarse satisfactoriamente (por ejemplo, por una violación de clave), este ejemplo muestra un mensaje de error y ejecuta **cancelEdit** para restaurar el registro a los valores originales y terminar el modo editar.

```
; cambiarClave::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Reg Array[] AnyType
endVar

tc.open("Clientes.db")                    ; abrimos un TCursor
para Clientes
if tc.locate("N° de Cliente", 1231) then ; si existe 1231 en el
campo
                                          ; N° de Cliente
  tc.edit()                               ; activamos el modo de
edición
  tc."N° de Cliente" = 1221               ; intentamos cambiar
el valor clave
  if not tc.endEdit() then                ; si endEdit falla
    msgStop("Error", "No se puede completar la operación.")
    tc.cancelEdit()                       ; restauramos el
registro y; abandonamos la edición
    message("Se ha dejado intacto el registro.")
  else
    message("Se ha cambiado el valor de la clave.")
  endif
else
  msgStop("Error", "No encuentro el Cliente 1231")
endif

endmethod
```

Vea también edit
endEdit

cAverage

Método Devuelve el valor medio de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cAverage** (const ***nombreCampo*** String) Number

2. cAverage (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la media de los valores de la columna de campos especificada por *nombreCampo* o *númeroCampo*. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cAverage** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura a la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo Este ejemplo emplea **cAverage** para calcular el tamaño medio de los pedidos de la tabla *Pedidos*. Este código se anexa al método **pushButton** del botón *obtenMediaVentas*.

```
; obtenMediaVentas::pushButton
method pushButton(var eventInfo Event)
var
  PedidoTC TCursor
  MediaVentas, MediaPedidos Number
endVar

; abrimos un TCursor para la tabla PEDIDOS
PedidoTC.open("Pedidos.db")
; almacenamos la media de Total Factura en la variable
MediaVentas
MediaVentas = PedidoTC.cAverage("Total Factura")
; mostrar MediaVentas en un cuadro de diálogo
msgInfo("Tamaño medio de pedido", MediaVentas)

endmethod
```

Vea también cCount
cSum
cMax
cMin
cStd
blankAsZero del tipo Session

cCount

Método Devuelve el número de valores de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cCount** (const ***nombreCampo*** String) Number

2. cCount (const ***nombreCampo*** SmallInt) Number

Descripción Devuelve el número de valores existentes en la columna (campo) especificada por *nombreCampo* o *númeroCampo*. **cCount** funciona para todos los tipos de campos. Si el campo es numérico, este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión. Si el campo no es numérico, **cCount** devuelve el número de valores no vacíos de la columna de campos.

Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

cCount es útil para devolver el número de entradas utilizadas por otra función de columna.

Ejemplo Este ejemplo abre un TCursor para una tabla y utiliza **cCount** para mostrar el número de registros del TCursor. Este código se anexa al método **pushButton** del botón *informaciónDeArtículo*.

```
; informaciónDeArtículo::pushButton
method pushButton(var eventInfo Event)
var
  NumerosTC TCursor
  CantidadMedia, NumRegs Number
endVar
NumerosTC.open("articulo.db")
CantidadMedia = NumerosTC.cAverage("Cant")
NumRegs = NumerosTC.cCount(4)       ; asumimos que Cantidad es
el registro 4
msgInfo("Cantidad Media", "Cantidad Media: " +
String(CantidadMedia) + " \nbasado en " + String(NumRegs) +
"registros.")

endmethod
```

Vea también cAverage
cSum
cMax
cMin
cStd
blankAsZero del tipo Session

close

Principiante

Método Cierra una tabla.

Tipo TCursor

Sintaxis **close** ( ) Logical

Descripción Cierra un TCursor y deja sin asignación a la variable TCursor. Aunque el registro actual no pueda consignarse, **close** cierra el TCursor, pero desecha los cambios realizados en el registro.

Ejemplo Este ejemplo abre un TCursor para una tabla, muestra la información que encuentra en el último registro y cierra el TCursor. En este ejemplo, el código fuente muestra un mensaje que indica si la variable TCursor aún está asignada después de que se cierra el TCursor. Este código se anexa método estándar **pushButton** del botón *esteBotón*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar

tc.open("Pedidos.db")          ; abrimos un TCursor para la
tabla Pedidos
tc.end()                       ; nos desplazamos al final de la
tabla
                                                              ;
mostramos información del último registro
msgInfo("Ultimo Pedido", "Número del Pedido: " + String(tc."N°
de Pedido") +          " \nFecha del pedido: " +
String(tc."Fecha de venta"))

tc.close()                                    ; cerramos el
TCursor tc
msgInfo("¿Está asignado tc?", tc.isAssigned()) ; muestra False

endmethod
```

Vea también isAssigned
open

cMax

Método Devuelve el valor máximo de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cMax** (const ***nombreCampo*** String) Number

2. cMax (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve el valor máximo de la columna de campos numéricos especificada por *nombreCampo* o *númeroCampo*. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cMax** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicarlo, el método falla.

Ejemplo En el ejemplo siguiente, supóngase que una ficha tiene un botón, *obtenerBalanceMáximo*, y un marco de tabla asociado con la tabla *Pedidos*. En este código, el método **pushButton** de *obtenerBalanceMáximo* asocia el marco de tabla con un TCursor y, entonces, busca el mayor saldo debido en la tabla *Pedidos*:

```
; obtenerBalanceMáximo::pushButton
method pushButton(var eventInfo Event)
var
  PedidoTC TCursor
endVar

PedidoTC.attach(PEDIDOS)    ; PEDIDOS es una tabla de la Ficha
; localizamos el máximo valor en el campo Pendiente
PedidoTC.locate("Pendiente", PedidoTC.cMax("Pendiente"))
; sincronizamos la tabla con el TCursor
PEDIDOS.moveToRecord(PedidoTC)

endmethod
```

Vea también cAverage
cMin
cSum
cCount
cStd
blankAsZero del tipo Session

cMin

Método Devuelve el valor mínimo existente en un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cMin** (const ***nombreCampo***String) Number

2. cMin (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve el valor mínimo de la columna de campos especificada por *nombreCampo* o *númeroCampo*. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cMin** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo Este ejemplo utiliza ambas formas de la sintaxis para calcular valores mínimos en la tabla PEDIDOS.DB:

```
; mostrarMinimos::pushButton
method pushButton(var eventInfo Event)
var
  PedidoTC TCursor
  MinPendiente, MinPedido Number
endVar
PedidoTC.open("Pedidos.db")
MinPendiente = PedidoTC.cMin("Pendiente") ; obtenemos el valor
mínimo de
                                          ; Pendiente
MinPedido = PedidoTC.cMin(6)              ; asumimos que "Total
Factura" es
                                          ; el campo 6

; visualizamos los resultados en un cuadro de diálogo
msgInfo("Minimos", "Minimo Pendiente: " + String(MinPendiente)
+ "\n" +
                   "Minimo pedido: " + String(MinPedido))
endmethod
```

Vea también cMax
cAverage
cCount
cStd
cSum
blankAsZero del tipo Session

cNpv

Método Devuelve el valor presente neto de un campo (columna), en función de un tipo de interés o de descuento especificado.

Tipo TCursor

Sintaxis **1. cNpv** (const ***nombreCampo*** String, const ***discRate*** Number) Number

2. cNpv (const ***númeroCampo*** SmallInt, const ***discRate*** Number) Number

Descripción Devuelve el valor presente neto de las entradas no vacías de una columna de campos. El cálculo se basa en el tipo de interés o de descuento *tipoInterés* donde *tipoInterés* es un número decimal (por ejemplo, 12 por ciento se expresa como 0,12). Este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Este método calcula el valor presente neto utilizando la fórmula siguiente:

cNpv = sum(p=1 to n) of $Vp/(1+i)p$

donde n = número de periodos y Vp = flujo de caja en el periodo p.

i = tipo de interés por periodo.

Ejemplo El ejemplo siguiente asocia un TCursor con la tabla *BuenFond* y calcula el valor presente neto del campo *Retorno Esperado*. En este ejemplo, el valor presente neto se calcula en función de un tipo de interés mensual. Este código se anexa al método **pushButton** del botón *calcularElVPN*.

```
; calcularElVPN::pushButton
method pushButton(var eventInfo Event)
var
  SalvarTC TCursor
  BuenFondoVPN, apr Number
endVar
SalvarTC.open("BuenFond.db") ; asociamos el TCursor con la
tabla BuenFond
apr = ,125                   ; Coeficiente de Porcentaje Anual

; ahora calculamos el Valor Presente Neto basado en el
coeficiente de interés
                             ; mensual
BuenFondoVPN = SalvarTC.cNpv("Retorno Esperado", (apr / 12))
msgInfo("Valor Presente Neto", BuenFondoVPN)
endmethod
```

Vea también cAverage
cSum

compact

Método Elimina registros borrados de una tabla.

Tipo TCursor

Sintaxis **compact** ([const ***regIndice*** Logical]) Logical

Descripción Elimina los registros borrados de una tabla. Los registros borrados no se eliminan inmediatamente de una tabla de dBASE. En su lugar, se marcan como borrados y se mantienen en la tabla. El argumento optativo *regIndice* se utiliza para especificar si se regeneran los índices asociados con la tabla o sólo se fijan. Cuando *regIndice* es True, este método regenera todos los índices mantenidos asociados con el TCursor y libera el espacio no utilizado en los índices. Si se omite, *regIndice* es True por defecto.

Si la tabla en que se trabaja tiene índices mantenidos, este método precisa acceso exclusivo; en caso contrario, precisa un bloqueo de escritura.

Cuando los registros se borran de una tabla de Paradox, es imposible recuperarlos se eliminan permanentemente . Sin embargo, el archivo de tabla (y sus archivos de índice asociados) contienen espacio muerto donde se hallaba originalmente el registro. **compact** elimina el espacio muerto de archivos de Paradox.

Ejemplo El ejemplo siguiente inspecciona cada registro de una tabla y los registros borrados en el que el valor del campo Date tiene más de 30 días de antigüedad. Compacta la tabla después de haber borrado los registros adecuados. Este código se anexa al método **pushButton** del botón *purgarTabla*.

```
; purgarTabla::pushButton
method pushButton(var eventInfo Event)
var
  Tabla Table
  tc TCursor
  RegsBorrados SmallInt
endVar
Tabla.attach("AntDatos.dbf")
Tabla.setExclusive()           ; obtenemos derechos exclusivos
sobre la tabla
tc.open(Tabla)                 ; asociamos el TCursor con la
tabla AntDatos
tc.edit()
scan tc for tc.Date (today() - 30) :
  tc.deleteRecord()            ; eliminamos los registros
antiguos
endScan
tc.compact()                   ; borramos todos los registros
eliminados
tc.close()                     ; cerramos el TCursor
message("Registros antiguos eliminados.")
endmethod
```

Vea también deleteRecord
showDeleted

copy

Método Copia una tabla.

Tipo TCursor

Sintaxis **1. copy** (const ***nombreTabla*** String) Logical

2. copy (const ***nombreTabla*** Table) Logical

Descripción Copia una tabla en la tabla destino nombreTabla. Si *nombreTabla* no existe, **copy** la crea. Si existe ya, este método la sustituye sin pedir confirmación.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla origen y un bloqueo completo (exclusivo) sobre la tabla destino. Si no es posible aplicar alguno de los bloqueos o si la tabla de destino está abierta, el método falla.

Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro.

Ejemplo Este ejemplo copia la tabla *Clientes* en la tabla *NueClien*.

Este código emplea el método **isTable** (del tipo DataBase) para comprobar si *NueClien* existe; si es así, se solicita al usuario que confirme la acción antes de que se sustituya *NueClien*:

```
; copiarCliente::pushButton
method pushButton(var eventInfo Event)
var
  OrigenTC TCursor
  TablaDestino Table
endVar
TablaDestino.attach("NueClien.db")
OrigenTC.open("Clientes.db")

; si existe NueClien, pedimos conformidad
if isTable(TablaDestino) then
  if msgYesNoCancel("Copiar tabla", "¿Sobreescribir
NueClient.db?") = "Yes"     then

    ; copiar los registros de Clientes.db en NueClien.db
    OrigenTC.copy(TablaDestino)
  endIf
endIf

endmethod
```

Vea también add
copyRecord

copyFromArray

Método Copia datos de una matriz en los campos del registro actual.

Tipo TCursor

Sintaxis **1. copyFromArray** (const ***mat*** Array[] AnyTpe) Logical

2. copyFromArray (const ***mat*** DynArray[] AnyType) Logical

Descripción Copia los elementos de la matriz *mat* en el registro señalado por un TCursor, que debe estar en modo Editar. El primer elemento de la matriz se copia en el primer campo, el segundo elemento en el segundo campo, y así sucesivamente hasta que se termine la matriz o el registro esté lleno.

El método falla si se intenta copiar un elemento no asignado de una matriz o si las estructuras no coinciden (es imposible que suceda esto si la matriz se creó mediante **copyToArray**, porque **copyToArray** asigna un valor vacío si un campo está vacío). Si hay más elementos en la matriz que campos en el registro, los elementos sobrantes no se copian. Para copiar un nuevo registro en una tabla vacía, utilice **insertRecord** para insertar un registro vacío antes de emplear **copyFromArray**.

Ejemplo En este ejemplo, supóngase que NOMCLIEN.DB tiene tres campos: Apellidos, A20; Nombre, A20, y Teléfono, A12. Este método asocia un TCursor con la tabla *NomClien*, crea una matriz con tres elementos, inserta un nuevo registro en la tabla y utiliza **copyFromArray** para copiar los datos de la matriz en el nuevo registro.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  aa Array[3] AnyType
endVar
aa[1] = "Borland"
aa[2] = "Paco"
aa[3] = "555-1212"
if tc.open("NomClien.db") then ; abrimos la tabla
  tc.edit()                    ; copyFromArray sólo funciona en
modo Edición
  tc.insertRecord()            ; insertamos un nuevo registro
  tc.copyFromArray(aa)         ; copiamos de la matriz a la
tabla
  tc.endEdit()
else
  msgStop("¡Atención!", "No puedo abrir NomClien.db.")
endIf
endmethod
```

Vea también copyRecord
copyToArray
insertRecord
insertAfterRecord
insertBeforeRecord

copyRecord

Método Copia un registro señalado por un TCursor en el registro señalado por otro TCursor.

Tipo TCursor

Sintaxis **copyRecord** (const ***puntero*** TCursor) Logical

Descripción Copia el registro señalado por un TCursor en el registro señalado por otro TCursor. Por ejemplo, el código siguiente copia el registro señalado por el TCursor *copiarDeEsteTC* en el registro señalado por el TCursor *ApuntadorTC*:

```
copiarDeEsteTC.copyRecord(ApuntadorTC)
```

El TCursor especificado en *ApuntadorTC* no tiene que estar en modo editar; el TCursor al que se copia sí tiene que estarlo.

No es posible utilizar **copyRecord** para copiar un registro en una tabla vacía. Para copiar un nuevo registro en una tabla vacía, emplee **insertRecord**.

Ejemplo Este ejemplo utiliza un TCursor para buscar en la tabla *Pedidos* ventas almacenadas en los últimos 10 días y las copia a la tabla *NuePedid* del directorio actual. Este código se anexa al método **pushButton** del botón *obtenNuevosPedidos*.

```
; obtenNuevosPedidos::pushButton
method pushButton(var eventInfo Event)
var
  PedidoTC, NuevoPedidoTC TCursor
  IU TableView
endVar

PedidoTC.open("Pedidos.db")
NuevoPedidoTC.open("NuePedid.db")
NuevoPedidoTC.edit()                ; copyRecord sólo funciona
en modo Edición

; buscamos en la tabla Pedidos.db aquellos registros fechados
10 días atrás
                                    ; y posteriores
scan PedidoTC for PedidoTC."Fecha de venta" = today() - 10:
  NuevoPedidoTC.insertRecord()      ; insertamos un nuevo
registro en NuePedid.db
  NuevoPedidoTC.copyRecord(PedidoTC) ; copiamos el registro de
Pedidos.db en
                                    ; NuePedid.db
endScan
NuevoPedidoTC.endEdit()             ; salimos del modo Edición
del TCursor

IU.open("NuePedid.db")
endmethod
```

Vea también copyToArray
insertAfterRecord
insertBeforeRecord
insertRecord

copyToArray

Método Copia los campos del registro actual en una matriz.

Tipo TCursor

Sintaxis **1. copyToArray** (var ***ar*** Array[] AnyType) Logical

2. copyToArray (var ***ar*** DynArray[] AnyType) Logical

Descripción Copia los campos del registro actual en los elementos de la matriz especificada en *mat*. Es necesario declarar que la matriz sea del tipo AnyType o de un tipo que coincida con cada campo de la tabla.

En la sintaxis 1, donde *mat* es una matriz fija o redimensionable, el valor del primer campo se copia en el primer elemento de la matriz, el valor del segundo campo en el segundo elemento, y así sucesivamente. Si la matriz es redimensionable, crece automáticamente para albergar el número de campos del registro. Si no lo es, sólo recibe tantos campos como pueda, desechándose el resto.

En la sintaxis 2, donde *mat* es un DynArray, los valores de índice corresponden a los nombres de los campos y los valores del DynArray corresponden a los valores de los campos:

mat [*nombreCampo*] = valorCampo

El tamaño de la matriz es igual al número de campos del registro (a menos que *mat* sea una matriz fija). El campo de número de registro y cualquier campo de sólo visualización o calculado que aparezca en una vista de ficha de la tabla no se copian a la matriz.

Ejemplo En este ejemplo, supóngase que una ficha tiene un marco de tabla, CLIENTES, asociado con CLIENTES.DB. Cuando el usuario intenta borrar un registro de CLIENTES, este código (anexado al método estándar **action**) emplea **copyToArray** y **copyFromArray** para copiar el registro a una tabla archivada, ARCCLIEN.DB. Si ARCCLIEN.DB no puede abrirse, este método informa al usuario y no borra el registro.

```
; CLIENTES::action
method action(var eventInfo ActionEvent)
var
  tcOrig, tcArchivo TCursor
  RegArchivo Array[] AnyType
endVar

if eventInfo.id() = DataDeleteRecord then  ; cuando el usuario
elimina un
                                           ; registro
  if EstaFicha.Editing = True then         ; si la Ficha está
en modo Edición
    disableDefault                         ; no se elimina el
registro

                                           ; pedimos
conformidad
    if msgQuestion("Confirme", "¿Eliminar el registro?") =
"Yes" then
```

```
      tcOrig.attach(CLIENTES)              ; sincronizamos el
TCursor con
                                           ; el UIObject
      tcOrig.copyToArray(RegArchivo)       ; almacenamos el
registro en: RegArchivo
      if tcArchivo.open("ArcClien.db") then; Cierto si
tcArchivo puede abrir
                                           ; ArcClien
      tcArchivo.edit()                     ; copyFromArray
requiere modo
                                           ; Edición
      tcArchivo.insertAfterRecord()        ; creamos un nuevo
registro
      tcArchivo.copyFromArray(RegArchivo)  ; copiar RegArchivo
al nuevo
                                           ; registro
      enableDefault                        ; eliminar el
registro en Clientes
    else                                   ; no se puede abrir
el TCursor
                                           ; Clientes
      msgStop("¡Atención!", "No puedo archivar el registro.")
    endif
  else                                     ; el usuario no
confirmó el cuadro
                                           ; de diálogo
    message("No se ha eliminado el registro.")
  endif

  else                                     ; no estábamos en
modo Edición
  msgStop("¡Atención!", "Pulse F9 para pasar al modo de Edición
de datos.")
  endif
endif
endmethod
```

Vea también copyFromArray

cSamStd

Método Devuelve la desviación típica de muestra de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cSamStd** (const ***nombreCampo*** String) Number

2. cSamStd (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la desviación típica de muestra de los valores de una columna de campos numéricos. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. El valor devuelto se basa en la varianza de muestra. Este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

La desviación típica de la muestra (en oposición a población) se calcula utilizando la fórmula:

sqrt(*TCursor*.cVar(*nombreCampo*) * (*n*/(*n* - 1)))

donde

varianza = *TCursor*.cVar(*nombreCampo*)

n = *TCursor.*cCount(*nombreCampo*)

Ejemplo El ejemplo siguiente utiliza ambas formas de la sintaxis para calcular la desviación típica de muestra de dos campos distintos de la tabla *Respuest*. Este código se anexa al método **pushButton** de *mostrarEjemploEstándar*:

```
; mostrarEjemploEstándar::pushButton
method pushButton(var eventInfo Event)
var
  EmpleadoTC TCursor
  NombreTabla String
  CalcSalario, CalcAños Number
endVar
NombreTabla = "Respuest"
if EmpleadoTC.open(NombreTabla) then
  CalcSalario = EmpleadoTC.cSamStd("Salario") ; obtenemos la
desviación Estándar
                                              ; de Salarios
  CalcAños  = EmpleadoTC.cSamStd(2)           ; asumimos que
Años de
                                              ; servicio  es el
campo 2
  msgInfo("Desviación Estandar Ejemplo",      ; mostramos
información en un
                                              ; cuadro de
diálogo
          "Salarios : " + String(CalcSalario) + "\n" +
          "Años de servicio : " + String(CalcAños))
else
```

```
  msgInfo("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endif
endmethod
```

Vea también

cAverage
cCount
cMax
cMin
cNpv
cSamVar
cStd
cSum
cVar

cSamVar

Método Devuelve la varianza de muestra de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cSamVar** (const ***nombreCampo*** String) Number

2. cSamVar (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la varianza de muestra de los valores de una columna de campos. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. Este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

La varianza de la muestra (en oposición a población) se calcula utilizando la fórmula:

TCursor.cVar(*nombreCampo*) * (*n*/(*n*-1))

donde

n = *TCursor*.cCount(*nombreCampo*)

Ejemplo El ejemplo siguiente utiliza ambas formas de la sintaxis para calcular la varianza de muestra de dos campos distintos de la tabla *Respuest*. Este código se anexa al método **pushButton** de *mostrarVarianza*.

```
; mostrarVarianza::pushButton
method pushButton(var eventInfo Event)
var
  EmpleadoTC TCursor
  NombreTabla String
  CalcSalario, CalcAños Number
endVar
NombreTabla = "Respuest"
if EmpleadoTC.open(NombreTabla) then
  CalcSalario = EmpleadoTC.cSamVar("Salario") ; obtenemos la
varianza de
                                              ; Salarios
  CalcAños  = EmpleadoTC.cSamVar(2)           ; asumimos que
Años de servicio
                                              ; es el campo 2
  msgInfo("Ejemplo de Varianza",              ; mostramos
información en un
                                              ; cuadro de
diálogo
          "Salarios : " + String(CalcSalario) + "\n" +
          "Años de servicio : " + String(CalcAños))
else
  msgInfo("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endif
endmethod
```

Vea también cAverage

cCount
cMax
cMin
cNpv
cSamStd
cStd
cSum
cVar

cStd

Método Devuelve la desviación típica de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cStd** (const ***nombreCampo*** String) Number

2. cStd (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la desviación típica de los valores de una columna de campos. El cálculo se basa en la varianza. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. Este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* calcula la desviación típica de población de dos campos distintos y muestra el resultado en un cuadro de diálogo:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  miTabla TCursor
  prueba1, prueba2 Number
endVar
miTabla.open("puntos.db")
prueba1 = MiTabla.cStd("prueba1")
prueba2 = MiTabla.cStdr(2)    ; asumimos que prueba2 es el
campo 2

; mostramos los resultados en un cuadro de diálogo
msgInfo("Desviación Estándar",
        "resultado prueba1 : " + String(prueba1) + "\n" +
        "resultado prueba2 : " + String(prueba2))
endmethod
```

Vea también cAverage
cCount
cMax
cMin
cNpv
cSamStd
cSamVar
cSum
cVar

cSum

Método Devuelve la suma de los valores de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cSum** (const ***nombreCampo*** String) Number

2. cSum (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la suma de los valores de una columna de campos numéricos. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. Este método gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo En este ejemplo, el método **pushButton** de *sumaPedidos* utiliza ambas formas de la sintaxis de **cSum** para calcular totales de dos campos de PEDIDOS.DB:

```
; sumaPedidos::pushButton
method pushButton(var eventInfo Event)
var
  PedidoTC TCursor
  PedidoTotal, CantidadPagada Number
  NombreTabla String
endVar
NombreTabla = "Pedidos"
if PedidoTC.open(NombreTabla) then
  PedidoTotal = PedidoTC.cSum("Total Factura") ; obtenemos la
suma del campo
                                               ; Total Factura
  CantidadPagada = PedidoTC.cSum(7)            ; asumimos que
Pagado es el
                                               ; campo 7
  msgInfo("Totales de los Pedidos",
          "Total Pedidos: " + String(PedidoTotal) + "\n" +
          "Total Recibos: " + String(CantidadPagada))
else
  msgInfo("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endif
endmethod
```

Vea también cAverage
cCount
cMax
cMin
cNpv
cSamStd
cSamVar
cStd
cVar

currRecord

Método Lee el registro actual en la memoria intermedia de registro.

Tipo TCursor

Sintaxis **currRecord** ( ) Logical

Descripción Lee valores del registro actual en la memoria intermedia de registro. Este método garantiza que se trabaja con la versión actualizada más reciente del registro, especialmente en una red.

Ejemplo Este ejemplo copia un registro de la tabla *ClienSeg* y lo inserta en la tabla *Clientes*. Antes de almacenar el nuevo registro *Clientes*, el código da al usuario la oportunidad de confirmar los cambios o cancelarlos.

```
; actualizarDesdeCopiaDeSeguridad::pushButton
method pushButton(var eventInfo Event)
var
  ClienBakTC, ClienTC TCursor
endVar

ClienBakTC.open("ClienSeg.db")
ClienTC.open("Clientes.db")

if ClienBakTC.locate("N° de Cliente", 6312) then
  ClienTC.edit()
  ClienTC.insertRecord(ClienBakTC)
  if msgYesNoCancel("Confirme",
     "¿Desea sobreescribir la información en Clientes?") =
"Yes" then
     ; el usuario da su conformidad, por tanto reemplazamos el
registro
     ; existente
     ClienTC.updateRecord()
     ClienTC.endEdit()
   else
     ; el usuario no da su conformidad, por tanto se cancelan
los cambios
     ; del registro
     ClienTC.currRecord()
   endif
else
   msgStop("Error", "No puede encontrar el Cliente 6312")
endif

endmethod
```

Vea también home
end
nextRecord
priorRecord
skip
moveToRecord

cVar

Método Devuelve la varianza de un campo (columna) de una tabla.

Tipo TCursor

Sintaxis **1. cVar** (const ***nombreCampo*** String) Number

2. cVar (const ***númeroCampo*** SmallInt) Number

Descripción Devuelve la varianza de población de los valores de una columna de campos. Este método respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro. **cVar** gestiona los valores vacíos según se especifique en el valor de **blankAsZero** para la sesión.

Este método intenta, durante el periodo de reintento, aplicar un bloqueo de escritura sobre la tabla. Si no es posible aplicarlo, el método falla.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* calcula la desviación de la varianza de población de dos campos distintos y muestra el resultado en un cuadro de diálogo:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  miTabla TCursor
  prueba1, prueba2 Number
endVar
miTabla.open("puntos.db")
prueba1 = MiTabla.cVar("prueba1") ; obtenemos la varianza de
prueba1
prueba2 = MiTabla.cVar(2)         ; asumimos que prueba2 es el
campo 2
msgInfo("Varianza de población",
        "resultado prueba1 : " + String(prueba1) + "\n" +
        "resultado prueba2 : " + String(prueba2))
endmethod
```

Vea también cAverage
cCount
cMax
cMin
cNpv
cSamVar
cSamStd
cStd
cSum

deleteRecord

Principiante

Método Borra el registro señalado por un TCursor.

Tipo TCursor

Sintaxis **deleteRecord** ( ) Logical

Descripción Borra el registro señalado por un TCursor sin pedir confirmación. Esta operación no puede anularse. La tabla debe hallarse en modo editar.

Esta operación falla si el registro está bloqueado o ya ha sido borrado por otro usuario (en una tabla de dBASE).

Ejemplo En este ejemplo, el método **pushButton** del botón *comprobarIOU* determina si una deuda en particular se ha marcado como pagada; si es así, este código emplea **deleteRecord** para borrar el registro:

```
; comprobarIOU::pushButton
method pushButton(var eventInfo Event)
var
  deben TCursor
  BuscarNombre String
endVar
BuscarNombre = "Pedro"
deben.open("iou.db")
deben.edit()
if deben.locate("Nombre", BuscarNombre) then
  if deben."Pagado" = "Yes" then
    deben.deleteRecord()   ; eliminar el registro actual
    message(BuscarNombre + " eliminado")
  else
    EnviarFactura()        ; ejecutar procedimiento del usuario
  endIf
else
    msgStop("¡Atención!", "No encuentro " + BuscarNombre)
endIf
endmethod
```

Vea también empty

didFlyAway

Método Informa de si el registro actual se ha desplazado a otra posición como resultado del cambio de un valor de clave.

Tipo TCursor

Sintaxis **didFlyAway** ( ) Logical

Descripción Devuelve True si la ejecución más reciente de **unlockRecord** provocó que el registro se desplazase a otra posición en la tabla; en caso contrario, devuelve False. Este método sólo es relevante si el método **setFlyAwayControl** se ha definido como True; de lo contrario, **didFlyAway** devuelve False (aunque el registro se haya desplazado a su posición ordenada).

Ejemplo El ejemplo siguiente demuestra cómo **setFlyAwayControl** afecta a la posición de un TCursor después de una llamada a **unlockRecord** y bajo qué circunstancias **didFlyAway** devuelve True.

```
; botónDemostración::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar

tc.open("MiTabla.db")

; asumimos que MiTabla.db tiene los siguientes
; valores en su único campo clave, "N° de Cliente" :
;  Registro°    N° de Cliente
;     1            110
;     2            120   ; el programa cambia este valor por
145.
;     3            130
;     4            140
                         ; que mueve el registro a esta
posición
;     5            150

  tc.setFlyAwayControl(Yes)
; ahora, una llamada a unlockRecord conseguirá que tc apunte
; a un registro que "voló"

if tc.locate("Campo Clave", 120) then
  tc.edit()

  ; cambiamos el valor de la clave de modo que el registro
  ; cambia su posición relativa
  tc."Campo Clave" = 145

  ; lo desbloqueamos. Como setFlyAwayControl tiene el valor
Yes,
  ; tc aún apunta al registro
  tc.unlockRecord()

  ; visualiza True porque el nuevo valor de la clave
  ; varía la posición relativa del registro en la tabla
```

```
  msgInfo("¿Voló 145?", tc.didFlyAway())

else
  message("No he encontrado 120.")
endif

endmethod
```

Vea también setFlyAwayControl
unLockRecord

dropIndex

Método Borra un archivo de índice asociado con una tabla.

Tipo TCursor

Sintaxis **1.** (Tablas de Paradox) **dropIndex** ([const ***nombreIndice*** String]) Logical

2. (dBASE tables) **dropIndex** (const ***nombreIndice*** String [, const ***nombreEtiqueta*** String]) Logical

Descripción Borra un archivo de índice o etiqueta de índice especificados.

Cuando se trabaja con una tabla de Paradox, *nombreIndice* es optativo. Si se omite *nombreIndice*, **dropIndex** borra el índice primario de la tabla (a menos que la tabla tenga un índice secundario, en cuyo caso el método falla).

Cuando se trabaja con una tabla de dBASE, es posible utilizar *nombreIndice* para especificar un archivo .NDX o emplear *nombreIndice* y *nombreEtiqueta* para especificar un archivo .MDX y una etiqueta de índice.

Este método precisa derechos exclusivos sobre la tabla si se está borrando un índice mantenido; en caso contrario, precisa un bloqueo de escritura.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* borra la etiqueta *NomClien* de un archivo .MDX y el índice primario de una tabla de Paradox:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc1, tc2 TCursor
  Tabla Table
endVar

if isTable("Ventas.dbf") then
  Tabla.attach("Ventas.dbf")               ; Ventas.dbf es una
tabla de dBASE
  tc1.open(Tabla)
  tc1.dropIndex("indice2.mdx", "NomClien") ; elimina la
etiqueta NomClien
                                           ; de indice2
  msgInfo("", "NomClien eliminada")
else
  msgStop("¡Atención!", "No encuentro la tabla Ventas.dbf.")
endif

if isTable("Pedidos.db") then
  tc2.open("Pedidos.db")                   ; Pedidos.db es una
tabla de Paradox
  if tc2.dropIndex() then
    msgInfo("", "Se ha eliminado el índice primario de
Pedidos.db.")
  else
    msgStop("", "No he podido eliminar el índice primario de
Pedidos.db")
  endif
else
```

```
  msgStop("¡Atención!", "No encuentro la tabla Pedidos.db.")
endif
endmethod
```

Vea también switchIndex

edit

Principiante

Método Pone un TCursor en modo editar.

Tipo TCursor

Sintaxis **edit** ( ) Logical

Descripción Pone un TCursor en modo editar por lo que es posible realizar cambios en el registro actual. Tras la edición, si desea permanecer en modo editar, salga del registro o utilice **postRecord** para aceptar los cambios en el registro. Si desea salir del modo editar, utilice **cancelEdit** para cancelar los cambios en el registro o **endEdit** para aceptarlos.

Ejemplo El ejemplo siguiente crea una matriz y utiliza **copyFromArray** para copiar el contenido de la matriz en un nuevo registro de la tabla *NomClien*. Puesto que el TCursor debe estar en modo editar antes de que se inserte el nuevo registro, este código utiliza **edit** para comenzar a editar la tabla. Después de que se haya insertado el nuevo registro, este código emplea **endEdit** para terminar el modo editar y aceptar los cambios.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  aa Array[3] AnyType
endVar
aa[1] = "Borland"
aa[2] = "Paco"
aa[3] = "555-1212"
if tc.open("NomClien.db") then ; abrimos la tabla
  tc.edit()                    ; ponemos el TCursor en modo
Edición
  tc.insertRecord()            ; insertamos un registro nuevo
  tc.copyFromArray(aa)         ; copiamos del array a la tabla
  tc.endEdit()                 ; terminamos el modo Edición
else
  msgStop("¡Atención!", "No se puede abrir NomClien.db.")
endIf
endmethod
```

Vea también cancelEdit
endEdit
postRecord

empty

Método Elimina todos los registros de una tabla.

Tipo TCursor

Sintaxis **empty** ( ) Logical

Descripción Elimina todos los registros de una tabla sin pedir confirmación. No es necesario que la tabla se encuentre en modo editar, pero sí es preciso un bloqueo de escritura. Esta operación no puede anularse.

Ejemplo El ejemplo siguiente pide al usuario confirmación antes de borrar todos los registros de la tabla *Grabar*. Si el usuario no confirma la acción, este código emplea **nRecords** para indicar cuántos registros hay en GRABAR.DB.

```
; tablaVacía::pushButton
method pushButton(var eventInfo Event)
var
  NombreTabla String
  tc TCursor
endVar

NombreTabla = "Grabar.db"
if isTable(NombreTabla) then
  tc.open(NombreTabla)
  if msgQuestion("Confirme", "¿Vacío la tabla " + NombreTabla +
"?") = "Yes"      then
    tc.empty()
    message("He eliminado todos los registros de " +
NombreTabla + ".")
  else
    message(NombreTabla + " tiene " + String(tc.nRecords()) +
"registros.")
  endif
else
  msgInfo("Error", "No encuentro la tabla " + NombreTabla +
".")
endif
endmethod
```

Vea también deleteRecord
nRecords

end

Principiante

Método Desplaza un TCursor al último registro de una tabla.

Tipo TCursor

Sintaxis **end** ( ) Logical

Descripción Define el último registro de una tabla como registro actual (y memoria intermedia de registro).

Ejemplo Este ejemplo emplea **end** para desplazar un TCursor al último registro de la tabla *Pedidos* y muestra en un cuadro de diálogo la información del último registro.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Pedidos.db")        ; Abrimos tc para la tabla Pedidos
tc.end()                     ; nos desplazamos al último
registro de la tabla
                             ; mostramos la información del
último registro
msgInfo("N° de Cliente " + tc."N° de Cliente",
        "Balance pendiente: " + tc."Pendiente")

endmethod
```

Vea también home
nextRecord
priorRecord
currRecord
skip
moveToRecord

endEdit

Principiante

Método Sale del modo editar y acepta los cambios realizados en el registro actual.

Tipo TCursor

Sintaxis **endEdit** ( ) Logical

Descripción Acepta los cambios realizados en el registro actual y sale del modo editar. No cierra el TCursor (los cambios en los registros anteriores se consignan o se cancelan conforme el usuario navega por la tabla).

Ejemplo El ejemplo siguiente crea una matriz y utiliza **copyFromArray** para copiar el contenido de la matriz en un nuevo registro de la tabla *NomClien*. Puesto que *NomClien* debe estar en modo editar antes de que se inserte el nuevo registro, este código emplea **edit** para comenzar a editar la tabla. Después de que se inserte el nuevo registro, este código emplea **endEdit** para salir del modo editar.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  aa Array[3] AnyType
endVar
aa[1] = "Borland"
aa[2] = "Paco"
aa[3] = "555-1212"
if tc.open("NomClien.db") then    ; abrimos la tabla
  tc.edit()                       ; ponemos TCursor en modo
Edición
  tc.insertRecord()               ; insertamos un nuevo
registro
  tc.copyFromArray(aa)            ; copiamos del array a la
tabla
  tc.endEdit()                    ; terminamos el modo Edición
else
  msgStop("¡Atención!", "No puede abrir NomClien.db.")
endIf
endmethod
```

Vea también cancelEdit
edit

enumFieldNames

Método Rellena una matriz con los nombres de los campos de una tabla.

Tipo TCursor

Sintaxis **enumFieldNames** (const ***matrizCampo*** Array[] String) Logical

Descripción Rellena *matrizCampo* con los nombres de los campos de una tabla. Si la matriz es redimensionable, crece automáticamente para albergar los nombres de los campos; si no lo es, almacena tantos como pueda y desecha el resto. Si *matrizCampo* ya existe, este método la sustituye sin pedir confirmación.

Ejemplo Para este ejemplo, el método **pushButton** del botón *enumerarCampos* almacena nombres de campos en una matriz redimensionable y utiliza **view** para mostrar el contenido de la matriz.

```
; enumerarCampos::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombresDeCampo Array[] String     ; los nombres de los campos
son siempre
                                    ; de tipo String
endVar
if tc.open("Pedidos.db") then
  tc.enumFieldNames(NombresDeCampo) ; cargamos la matriz con
los nombres
                                    ; de los campos
  NombresDeCampo.view()             ; mostramos los nombres de
los campos
                                    ; en un cuadro de diálogo
else
  msgStop("¡Atención!", "No puedo abrir Pedidos.db.")
endIf

endmethod
```

Vea también enumFieldNamesInIndex

enumFieldNamesInIndex

Método Rellena una matriz con los nombres de los campos existentes en el índice de una tabla.

Tipo TCursor

Sintaxis **1.** (Tablas de Paradox) **enumFieldNamesInIndex** ([const ***nombreIndice*** String,] ***matrizCampo*** Array[] String) Logical

2. (dBASE tables) **enumFieldNamesInIndex** ([const ***nombreIndice*** String [, const ***nombreEtiqueta*** String ,] ***matrizCampo*** Array[] String) Logical

Descripción Rellena *matrizCampo* con los nombres de los campos del índice de una tabla, especificado en *nombreIndice*. Si se omite *nombreIndice*, este método emplea el índice actual. Si *matrizCampo* es redimensionable, crece automáticamente para albergar los nombres de los campos; si no lo es, almacena tantos como pueda y desecha el resto. Si *matrizCampo* ya existe, este método la sustituye sin pedir confirmación.

Cuando se trabaja con una tabla de dBASE, el argumento *nombreEtiqueta* puede utilizarse para especificar una etiqueta de índice dentro de un archivo .MDX.

Ejemplo En este ejemplo, el método **pushButton** del botón *enumerarIndice* almacena nombres de campos en una matriz redimensionable y emplea **view** para mostrar el contenido de la matriz:

```
; enumerarIndice::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombresDeCampo Array[] String
endVar
if tc.open("Ventas.dbf") then
  ; cargamos el array con los nombres de los campos del índice
PorFecha
  tc.enumFieldNamesInIndex("IndFecha", "PorFecha",
NombresDeCampo)
  ; mostramos los nombres de campo del índice ordenados por
PorFecha
  ;según IndFecha
  NombresDeCampo.view()
else
  msgStop("¡Atención!", "No puedo abrir Ventas.dbf.")
endIf
endmethod
```

Vea también enumIndexStruct

enumFieldStruct

Método Crea una tabla de Paradox que enumera la estructura de un TCursor.

Tipo TCursor

Sintaxis **enumFieldStruct** (const ***nombreTabla*** String) Logical

Descripción Crea la tabla de Paradox especificada en *nombreTabla* listando la estructura de un TCursor. Si *nombreTabla* existe, este método la sustituye sin pedir confirmación.

Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **struct** de una sentencia **create** para tomar la estructura de campos de una tabla (incluyendo sus claves primarias y controles de validación) para su uso en la nueva tabla.

La estructura de la tabla de Paradox se enumera en la tabla siguiente:

Nombre de campo	Tipo de campo
FieldName	A31
Type	A31
Size	S
Dec	S
Key	A1
_Required Value	A1
_Min Value	A255
_Max Value	A255
_Default Value	A255
_Picture Value	A175
_Table Lookup	A81
_Table Lookup Type	A1
_Invariant Field ID	S

Ejemplo Para este ejemplo, supóngase que se desea que una nueva tabla llamada *NueClien* sea similar a la tabla *Clientes*. Sin embargo, se desea que todos los campos de *NueClien* sean campos necesarios. Para lograrlo, el código siguiente emplea **enumFieldStruct** para cargar una nueva tabla (*CmpClien.DB*) con la información a nivel de campo de *Clientes*. Entonces, el código explora toda la tabla *CmpClien* y modifica las definiciones de los campos para que cada registro describa un campo que será necesario. Entonces, se suministra *CmpClien* en la cláusula **struct** de una sentencia **create**.

```
; crearNuevosClientes::pushButton
method pushButton(var eventInfo Event)
var
  NueClienTabla Table
  tc  TCursor
  estructura, NombreOrigen String
endVar

estructura = "CmpClien.db"
NombreOrigen = "Clientes.db"
```

```
if tc.open(NombreOrigen) then

  ; incluimos los nombres de campo, condiciones de validez, y
campos clave
  ; en la nueva tabla CmpClien a partir de Clientes
  tc.enumFieldStruct(estructura)

  ; ahora apuntamos el TCursor a la tabla CmpClien
  tc.open(estructura)
  tc.edit()

  ; este bucle busca a través de la tabla CmpClien y
  ; cambia las definiciones de las condiciones de validez de
todos los campos
  scan tc :
    tc."_Valor Requerido" = 1    ; crear todos los campos
requeridos
  endscan

  ; ahora creamos NueClien.db y tomamos los nombres de campo,
  ; las condiciones de validez y los campos clave de
CmpClien.db
  NueClienTabla = create "NueClien.db"
                    struct estructura
                  endCreate

  ; NueClien.db requiere que se rellenen todos los campos

else
  msgStop("Error", "No puedo obtener la estructura de campos de
la tabla Clientes.")
endif

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumIndexStruct
enumRefIntStruct
enumSecStruct

enumIndexStruct

Método Crea una tabla de Paradox que enumera la estructura de índices secundarios de un TCursor.

Tipo TCursor

Sintaxis **enumIndexStruct** (const ***tableName*** String) Logical

Descripción Crea la tabla de Paradox especificada en *nombreTabla* listando la estructura de índices secundarios de una tabla. En las tablas de dBASE, este método enumera la estructura de los índices asociados con la tabla por el método **usesIndexes**. Si *nombreTabla* existe, este método pide confirmación al usuario antes de sustituirla.

Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **indexStruct** de una sentencia **create** para tomar sus índices secundarios para su uso en la nueva tabla.

La estructura de *nombreTabla* se muestra a continuación.

Nombre de campo	Tipo de campo
infoHeader	A1
szName	A127
szTagName	A31
szFormat	A31
bPrimary	A1
bUnique	A1
bDescending	A1
bMaintained	A1
bCaseInsensitive	A1
bSubset	A1
bExpIdx	A1
bKeyExpType	N
szKeyExp	A220
szKeyCond	A220
FieldNo	N
FieldName	A31

Ejemplo Para este ejemplo, supóngase que se desea que una nueva tabla llamada *NueClien* sea similar a la tabla *Clientes*. Sin embargo, no se desea tomar su información de integridad referencial ni de seguridad. Para lograrlo, el código siguiente emplea **enumFieldStruct** y **enumIndexStruct** para generar dos tablas: CMPCLIEN.DB e INDCLIEN.DB. *CmpClien* e *IndClien* se suministran entonces en las cláusulas **struct** e **indexStruct** de una sentencia **create**.

```
; crearNuevosClientes::pushButton
method pushButton(var eventInfo Event)
var
  NueClienTC Table
  ClienTC   TCursor
endVar

if ClienTC.open("Clientes.db") then
```

```
  ; escribimos la información de los campos en CMPCLIEN.DB
  ClienTC.enumFieldStruct("CmpClien.db")

  ; escribimos la información del índice secundario en
INDCLIEN.DB
  ClienTC.enumIndexStruct("IndClien.db")

  ; ahora creamos NUECLIEN.DB
  ; tomamos los nombres de campo, condiciones de validez y
campos
  ; clave de CMPCLIEN.DB
  ; tomamos los índices secundarios de INDCLIEN.DB
  NueClienTC = create "Nueclien.db"
                 struct "Cmpclien.db"
                 indexStruct "Indclien.db"
               endCreate

else
  msgStop("Error", "No encuentro la tabla Clientes.")
endif

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumRefIntStruct
enumSecStruct

enumLocks

Método Crea una tabla de Paradox que enumera los bloqueos aplicados actualmente a un TCursor.

Tipo TCursor

Sintaxis **enumLocks** (const ***nombreTabla*** String) LongInt

Descripción Crea la tabla de Paradox especificada en *nombreTabla*, que lista los bloqueos aplicados actualmente a la tabla señalada por un TCursor. Si *nombreTabla* existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, este método falla. En las tablas de dBASE, este método sólo enumera el bloqueo que se ha aplicado (no todos los bloqueos actuales aplicados a la tabla). Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de *nombreTabla* se muestra a continuación:

Nombre de campo	Tipo de campo
UserName	A15
Lock Type	A32
Net Session	N
Session	N
Record Number	N

Ejemplo En este ejemplo, el método estándar **pushButton** del botón *mostrarBloqueoDePedidos* crea una tabla que enumera los bloqueos aplicados actualmente en PEDIDOS.DB y abre la tabla recién creada.

```
; mostrarBloqueoDePedidos::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  tv TableView
endVar
if tc.open("Pedidos.db") then
  tc.enumLocks("BloqPedi.db")  ; almacenamos los bloqueos de
Pedidos.db
                               ; en BloqPedi.db
  tv.open("BloqPedi.db")       ; abrimos BloqPedi.db
else
  msgStop("¡Alto!", "No puedo abrir la tabla Pedidos")
endif
endmethod
```

Vea también lock
lockStatus

enumRefIntStruct

Método Crea una tabla de Paradox que enumera información sobre integridad referencial para un TCursor.

Tipo TCursor

Sintaxis **enumRefIntStruct** (const ***nombreTabla*** String) Logical

Descripción Escribe información de integridad referencial para un TCursor en la tabla especificada en *nombreTabla*. Si *nombreTabla* existe, este método pide confirmación al usuario antes de sustituir la tabla. Si *nombreTabla* está abierta, este método falla. Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **refIntStruct** de una sentencia **create** para tomar su información de integridad referencial para su uso en la nueva tabla.

La estructura de *nombreTabla* se muestra a continuación:

Nombre de campo	Tipo de campo
infoHeader	A1
RefName	A31
Other Table	A81
Slave	A1
Modify	A1
Delete	A1
FieldNo	N
aiThisTabField	A31
Other FieldNo	N
aiOthTabField	A31

Ejemplo Este ejemplo emplea **enumRefIntStruct** para escribir la información de integridad referencial de CLIENTES.DB en la tabla *RefClien.db*. Entonces, el código suministra *RefClien.db* en la clásula **refIntStruct** de una sentencia **create**.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc  TCursor
  tbl Table
endVar

tc.open("Clientes.db")

; escribimos la información referencial de integridad en
RefClien.db
tc.enumRefIntStruct("RefClien.db")

; escribimos la información de campos en CmpClien
tc.enumFieldStruct("CmpClien.db")

; ahora creamos NUECLIEN.DB
; tomamos la información de los campos de CMPCLIEN.DB     ;
tomamos la información referencial de integridad de REFCLIEN.DB
```

```
tbl = create "NueClien.db"
        struct "CmpClien.db"
        refIntStruct "RefClien.db"
      endCreate

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct
enumSecStruct
Form::enumTableLinks

enumSecStruct

Método Escribe información de seguridad de una tabla en una tabla de Paradox.

Tipo TCursor

Sintaxis **enumSecStruct** (const ***nombreTabla*** String)

Descripción Crea la tabla de Paradox especificada en *nombreTabla* listando la información de seguridad (derechos de acceso) de una tabla. Este método devuelve un valor de Logical. Si *nombreTabla* existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, el método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Es posible suministrar *nombreTabla* en la opción **secStruct** de una sentencia **create** para tomar su información de seguridad para su uso en la nueva tabla.

La estructura de *nombreTabla* se muestra a continuación.

Nombre campo	Tipo campo
TableName	A32
PropertyName	A64
PropertyValue	A255

Ejemplo Este ejemplo crea una nueva tabla en función de la información de seguridad asociada con la tabla *Secretos*. El código emplea **enumSecStruct** para escribir su información de seguridad en la tabla *InfoSec* y emplea la tabla para crear la tabla *MisSecrs*.

```
; obtenSecretos::pushButton
method pushButton(var eventInfo Event)
var
  tc  TCursor
  Tabla Table
endVar

tc.open("Secretos.db")

; escribimos la información de seguridad en SECRETOS.DB
tc.enumSecStruct("Secretos.db")

; ahora creamos MISSECRS.DB
; tomamos los nombres de los campos y sus tipos de SECRETOS.DB
; tomamos la información de seguridad de MISSECRS.DB
Tabla = create "MisSecrs.db"
          like "Secretos.db"
          secStruct "Secretos.db"
        endCreate

endmethod
```

Vea también enumFieldNames
enumFieldNamesInIndex
enumFieldStruct
enumIndexStruct

enumRefIntStruct

enumTableProperties

Método Escribe las propiedades de un TCursor en una tabla de Paradox.

Tipo TCursor

Sintaxis **enumTableProperties** (const ***nombreTabla*** String) Logical

Descripción Escribe las propiedades de una tabla asociada con un TCursor en la tabla especificada en *nombreTabla*. Si *nombreTabla* existe, este método pide confirmación al usuario antes de sustituirla. Si *nombreTabla* está abierta, el método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de *nombreTabla* se muestra a continuación.

Nombre de campo	**Tipo de campo**
TableName	A32
PropertyName	A64
PropertyValue	A255

Ejemplo El ejemplo siguiente emplea **enumTableProperties** para escribir las propiedades de PEDIDOS.DB en PROPPEDI.DB. Si PROPPEDI.DB existe, este código pide confirmación al usuario antes de sustituir la tabla.

```
; mostrarPropiedadesDeTabla::pushButton
method pushButton(var eventInfo Event)
var
  NombreTabla, PropiedTabla String
  tc TCursor
  tv TableView
endVar
NombreTabla = "Pedidos.db"
PropiedTabla = "PropPedi.db"

if tc.open(NombreTabla) then
  if isTable(PropiedTabla) then
    if msgYesNoCancel("Confirme",
       PropiedTabla + " existe. ¿Desea sobreescribir?") <>
"Yes" then
      return
    endif
  endif
  ; escribimos las propiedades de Pedidos.db en PropPedi.db
  tc.enumTableProperties(PropiedTabla)
  ; abrimos la tabla nuevamente creada PropPedi.db
  tv.open(PropiedTabla)
else
  msgStop("¡Atención!", "No puedo abrir la tabla " +
NombreTabla+ ".")
endif

endmethod
```

Vea también enumFieldNames

eot

Método Comprueba si se intenta un desplazamiento más allá del final de una tabla.

Tipo TCursor

Sintaxis **eot** ( ) Logical

Descripción Devuelve True si un comando intenta ir más allá del último registro de una tabla; en caso contrario, devuelve False. **eot** se restaura en la siguiente operación de desplazamiento.

eot (y **bot**) también devuelve True si un comando fuerza el TCursor para señalar a un registro no existente. Por ejemplo, supóngase que la tabla *Clientes* tiene valores en el primer campo clave que se hallan entre 1000 y 10.000. Si se ejecuta **setFilter** de forma que el TCursor señale a los valores clave de 1 a 10 (fuera del rango posible de los valores de *Clientes*), el TCursor señala a un registro no existente. El fragmento de código siguiente demuestra que **setFilter** puede afectar a **eot** y a **bot**:

```
var  tc TCursor endvar
tc.open("Clientes.db")
tc.setFilter(1, 10)           ; filtra los rangos desde 1 hasta
10
  ; en este punto tc.eot() y tc.bot() valen True
```

De forma similar, si una llamada a **switchIndex** fuerza que el TCursor señale a un registro no existente, los métodos **eot** y **bot** devuelven True.

Ejemplo En este ejemplo, un bucle **while** controla el movimiento de un TCursor por la tabla *Pedidos*. Cuando el código dentro del bucle intenta ir más allá del final de la tabla, **eot** devuelve True y el bucle termina.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombreTabla String
  ValCampo AnyType
endVar
NombreTabla = "Clientes.db"
if tc.open(NombreTabla) then
  while tc.eot() = False        ; mientras las siguientes
sentencias no
                                ; desplacen el apuntador tras
el último registro
                                ; de la tabla
    message(tc."N° de Cliente") ; mostramos el valor del campo
N° de Cliente
    sleep(250)                  ; hacemos una pausa para el
mensaje
    tc.nextRecord()             ; y saltamos al siguiente
registro
  endWhile
  msgInfo("Fin", "¡Eso es todo amigos!")
else
  msgStop("¡Atención!", "No puedo abrir la tabla " +
NombreTabla + ".")
```

```
endif
endmethod
```

Vea también bot
end
home

familyRights

Método Comprueba si el usuario puede crear o modificar objetos en la familia de una tabla.

Tipo TCursor

Sintaxis **familyRights** (const ***derechos*** String) Logical

Descripción Devuelve True si el usuario tiene derechos sobre el tipo de objeto especificado en *derechos*; de lo contrario, devuelve False. *derechos* es una cadena de un solo carácter F (ficha), R (informe), S (valores de plantilla) o V (controles de validación) que indica el tipo de objeto en que se está interesado.

Ejemplo Este ejemplo indica en un cuadro de diálogo si se tienen derechos "F" sobre CLIENTES.DB.

```
; mostrarDerechosDeFamilia::pushButton
method pushButton(var eventInfo Event)
var
  ClienTC TCursor
endVar

ClienTC.open("Clientes.db")
msgInfo("Derechos", "Derechos de Familia: " +
String(ClienTC.familyRights("F")))
; visualiza True si se tienen derechos de Familia para
Clientes.db

endmethod
```

Vea también tableRights

fieldName

Método Devuelve el nombre de un campo.

Tipo TCursor

Sintaxis **fieldName** (const ***númeroCampo*** SmallInt) String

Descripción Devuelve el nombre del campo *númeroCampo*. Los campos se numeran de izquierda a derecha, comenzando en 1.

Ejemplo El ejemplo siguiente utiliza **fieldName** para mostrar el nombre del campo número dos de la tabla *Respuest*. Este código se anexa al método estándar **pushButton** de un botón.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombreCampo, NombreTabla String
endVar

NombreTabla = "Respuest.db"

if tc.open(NombreTabla) then
  NombreCampo = tc.fieldName(2)  ; almacena el nombre del campo
2 en
                                 ; NombreCampo
  msgInfo("Nombre del campo",    ; muestra el nombre del campo
2
          "El nombre de campo del campo 2 es\n" + NombreCampo)
else
  msgStop("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endIf

endmethod
```

Vea también fieldNo
fieldType
fieldValue

fieldNo

Método Devuelve la posición de un campo en una tabla.

Tipo TCursor

Sintaxis **fieldNo** (const ***nombreCampo*** String) SmallInt

Descripción Devuelve la posición del campo *nombreCampo* en una tabla. Los campos se numeran de izquierda a derecha, comenzando en 1.

Ejemplo El código siguiente se anexa al método **pushButton** de *esteBotón*. Cuando se pulsa *esteBotón*, este ejemplo utiliza **fieldName** para mostrar la posición del campo *Nombre común* en la tabla *VidaMar*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombreCampo, NombreTabla String
endVar

NombreTabla = "Respuest.db"

if tc.open(NombreTabla) then
  NombreCampo = tc.fieldName(2)  ; almacena el nombre del campo
2 en
                                 ; NombreCampo
  msgInfo("Nombre del campo",    ; muestra el nombre del campo
2
          "El nombre de campo del campo 2 es\n" + NombreCampo)
else
  msgStop("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endIf

endmethod
```

Vea también fieldValue

fieldRights

Método Informa de si un usuario tiene derechos para leer o modificar un campo en una tabla.

Tipo TCursor

Sintaxis **1. fieldRights** (const ***nombreCampo*** String, const ***derechos*** String) Logical

2. fieldRights (const ***númeroCampo*** SmallInt, const ***derechos*** String) Logical

Descripción Devuelve True si el usuario tiene *derechos* sobre el campo especificado en *nombreCampo* o *númeroCampo*; en caso contrario, devuelve False. El valor de *derechos* debe ser una expresión que se evalúa en una de las cadenas siguientes: ReadOnly (sólo lectura), ReadWrite (lectura/escritura) o All (Cualquiera). Los derechos se obtienen empleando el método **addPassword** del tipo Session; los derechos no pueden conseguirse después de que se haya abierto la tabla.

Ejemplo Este ejemplo emplea **fieldRights** para determinar si un TCursor tiene derechos de campo adecuados antes de intentar modificar el valor del campo.

```
; actualizarClientes::pushButton
method pushButton(var eventInfo Event)
var
  ClienTC TCursor
endVar
ClienTC.open("Clientes.db")
if ClienTC.locate("Nombre", "Unisco") then
  ; si no tenemos derechos suficientes para cambiar el campo
Nombre
  if NOT ClienTC.fieldRights("Nombre", "ReadWrite") then
    ; aparece un mensaje de error y se suspende la operación
    msgStop("Error", "No tiene derechos suficientes para
cambiar el campo Nombre")
  else
    ; de otro modo, tenemos derechos para realizar los cambios
en el campo
    ClienTC.edit()
    ClienTC.Nombre = "Unisco Internacional Inc."
    message("He cambiado Unisco por Unisco Internacional Inc.")
    ClienTC.endEdit()
  endif
else
  msgStop("Error", "No encuentro Unisco")
endif

endmethod
```

Vea también tableRights

fieldSize

Método Devuelve el tamaño de un campo.

Tipo TCursor

Sintaxis **1. fieldSize** (const ***nombreCampo*** String) SmallInt

2. fieldSize (const ***númeroCampo*** SmallInt) SmallInt

Descripción Devuelve el tamaño de un campo según se definió cuando se creó la tabla. El valor devuelto puede representar el número máximo de caracteres que puede contener un campo; por ejemplo, dado un campo definido como Alpha20, **fieldSize** devuelve 20. O bien, el valor devuelto puede representar la cantidad máxima de datos que pueden visualizarse; por ejemplo, cuando se crea una tabla y se define un campo Memo, es posible especificar el número de caracteres que se muestran. **fieldSize** devolvería ese número.

Los campos numéricos de las tablas de dBASE pueden especificar el número de dígitos que se mostrarán a cada lado de la coma decimal; por ejemplo, un número definido como Number 8,2 podría mostrar un total de 8 dígitos, con 6 dígitos a la izquierda de la coma y 2 a la derecha. **fieldSize** devuelve la primera parte de la definición; es decir, el número de dígitos situados a la izquierda de la coma decimal. Para obtener la segunda parte, utilice **fieldUnits2**.

En los tipos de campos que no muestran caracteres o números (como OLE, binario, imagen, etc.), este método devuelve 0.

Ejemplo Este ejemplo emplea una matriz dinámica para almacenar el tamaño de cada campo de la tabla *VidaMar* y, a continuación, muestra el contenido de la matriz dinámica en un cuadro de diálogo.

```
; mostrarTamañoDeCampos::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  i SmallInt
  TamañoCampos DynArray[] AnyType
  NombreTabla String
endVar
NombreTabla = "VidaMar.db"

if tc.open(NombreTabla) then
   ; este bucle FOR carga el DynArray con los tamaños de los
campos
   ; de VidaMar.db
  for i from 1 to tc.nFields()
    TamañoCampos[tc.fieldName(i)] = tc.fieldSize(i)
  endFor
   ; ahora mostramos el contenido del DynArray
  TamañoCampos.view("Tamaño de los campos de " + NombreTabla +
".")
else
  msgStop("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endif
endmethod
```

Vea también fieldName
fieldType
fieldUnits2

fieldType

Método Devuelve el tipo de datos de un campo.

Tipo TCursor

Sintaxis **1. fieldTipo** (const ***nombreCampo*** String) String

2. fieldTipo (const ***númeroCampo*** SmallInt) String

Descripción Devuelve el tipo de datos de un campo. Desconocido si no lo encuentra. La tabla siguiente enumera los valores devueltos posibles:

Tipo de campo	Tabla de Paradox	Tabla de dBASE
Alfanumérico	ALPHA	(ninguno)
Carácter	(ninguno)	CHARACTER
Fecha	DATE	DATE
Entero	SHORT	(ninguno)
Valor de coma flotante	NUMERIC	FLOAT (IV) NUMERIC (III+ o IV)
Imagen	GRAPHIC	(ninguno)
Lógico	(ninguno)	BOOLEAN
Dinero	MONEY	(ninguno)
Memo	MEMO	MEMO
Memo con formato	FMTMEMO	(ninguno)
Binario	BINARYBLOB	(ninguno)
Objeto OLE	OLEOBJ	(ninguno)

Ejemplo Este ejemplo emplea una matriz dinámica para almacenar el tipo de cada campo de la tabla *VidaMar* y, a continuación, muestra el contenido de la matriz en un cuadro de diálogo.

```
; mostrarTiposDeCampos::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  i SmallInt
  TipoCampos DynArray[] AnyType
  NombreTabla String
endVar
NombreTabla = "VidaMar.db"

if tc.open(NombreTabla) then
   ; este bucle FOR carga el DynArray con los tipos de los
campos de VidaMar.db
  for i from 1 to tc.nFields()
    TipoCampos[tc.fieldName(i)] = tc.fieldtype(i)
  endFor
   ; ahora mostramos el contenido del DynArray
  TipoCampos.view("Tipo de los campos de " + NombreTabla + ".")
else
  msgStop("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endif
endmethod
```

Vea también fieldNo

fieldUnits2

Método Devuelve el número de cifras decimales definido para un campo númerico de una tabla de dBASE.

Tipo TCursor

Sintaxis **1. fieldUnits2** (const ***nombreCampo*** String) SmallInt

2. fieldUnits2 (const ***númeroCampo*** SmallInt) SmallInt

Descripción Devuelve el número de cifras decimales definido para un campo numérico de una tabla de dBASE. Los campos numéricos de las tablas de dBASE pueden especificar el número de dígitos que se muestran a cada lado de la coma decimal; por ejemplo, un campo definido como Number 8,2 podría mostrar un total de 8 dígitos, con 6 dígitos a la izquierda de la coma decimal y 2 dígitos a la derecha. **fieldUnits2** devuelve la segunda parte de la definición, es decir, el número de dígitos a la derecha de la coma decimal. Para obtener la primera parte, utilice **fieldSize**. Este método devuelve 0 para los tipos de campos no numéricos, como alfanumérico, booleano, fecha, etc.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* concatena los valores devueltos por **fieldSize** y **fieldUnits2**, de forma que ambos lados de la coma decimal se expresen en un solo número. Por ejemplo, si el tamaño de un campo es 11 y se define que tenga 2 cifras decimales, este método concatena los valores en 11,2. Este código emplea un DynArray para almacenar los valores concatenados para cada campo de PUNTOS.DBF y, entonces, muestra el contenido de la matriz en un cuadro de diálogo.

```
; mostrarTamañoDeCampos::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  i SmallInt
  TamañoCampos DynArray[] AnyType
  NombreTabla String
  TamañoTotal Number
endVar
NombreTabla = "Puntos.dbf"

if tc.open(NombreTabla) then
   ; este bucle FOR carga el DynArray con las especificaciones
de los campos
   ; por ejemplo if fieldSize(1) = 11 and fieldUnits2(1) = 2,
   ; un valor del DynArray sería 11,2
  for i from 1 to tc.nFields()
    TamañoTotal = numVal(String(tc.fieldsize(i)) + "." +
                         String(tc.fieldUnits2(i)))
    TamañoCampos[tc.fieldName(i)] = TamañoTotal
  endFor
   ; ahora mostramos el contenido del DynArray
  TamañoCampos.view("Tamaños totales de los campos de " +
NombreTabla + ".")
else
  msgStop("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endif
```

```
endmethod
```

Vea también fieldName
fieldNo
fieldSize
fieldType

fieldValue

Método Lee el valor de un campo especificado.

Tipo TCursor

Sintaxis **1. fieldValue** (const ***nombreCampo*** String, var ***resultado*** AnyType) Logical

2. fieldValue (const ***númeroCampo*** SmallInt, var ***resultado*** AnyType) Logical

Descripción Obtiene el valor de un campo especificado (*nombreCampo* o *númeroCampo*) del registro actual y lo asigna a la variable *resultado*. Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Es posible obtener la misma información utilizando la notación de punto. Por ejemplo, esta sentencia emplea la notación de punto para asignar a la variable *miCoste* los datos del campo Ultima oferta:

```
miCoste = tcVar."Ultima oferta"
```

La sentencia siguiente emplea **fieldValue** para obtener el mismo resultado:

```
tcVar.fieldValue ("Ultima oferta",miCoste)
```

Ejemplo Para este ejemplo, supóngase que una ficha tiene al menos un campo, llamado *campoDePago*. Cuando el usuario hace clic con el botón derecho del ratón sobre *CampoDePago*, el código de este ejemplo presenta un menú emergente que lista los valores posibles del campo. Cuando se elige una opción en el menú, ese valor se inserta en el campo.

El código siguiente se anexa a la ventana Var del campo:

```
; campoDePago::Var
Var
  Tabla String
  MatrizMenú Array[] String
  ValorCampo AnyType
  p1 PopUpMenu
  tc TCursor
endVar
```

El código siguiente se anexa al método **open** del campo. Cuando se abre el campo, este código explora toda la tabla *MtodPago* y carga la matriz *MatrizMenú* con los valores del campo *Método de Pago*.

```
; campoDePago::open
method open(var eventInfo Event)

tabla = "MtodPago.db"
tc.open(tabla)
scan tc :                                        ; buscamos a
través de la tabla
  tc.fieldValue("Método de Pago",ValorCampo) ; almacenamos el
valor del campo
                                                 ; en ValorCampo
```

```
  MatrizMenú.addLast(ValorCampo)              ; añadimos un
nuevo elemento
                                              ; a MatrizMenú
endScan
p1.addStaticText("Valores Posibles")          ; ponemos texto al
principio
                                              ; del menú
p1.addSeparator()                             ; añadimos una
barra horizontal
                                              ;debajo del texto
p1.addArray(MatrizMenú)                       ; añadimos la
matriz al menú

endmethod
```

El código siguiente se anexa al método **mouseRightUp** del campo. Cuando el usuario haga un clic derecho en el campo, este código presenta un menú emergente y el campo toma el valor elegido por el usuario en el menú.

```
; campoDePago::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)

disableDefault                  ; no se muestra el menú por
defecto
opción = p1.show()              ; mostramos el menú desplegable
if NOT isBlank(opción) then     ; si el usuario no pulsó ESC
  self.value = opción           ; introducimos la opción en el
campo
endif

endmethod
```

Vea también setFieldValue

getLanguageDriver

Método Devuelve el nombre de la tabla de idioma actual de una tabla.

Tipo TCursor

Sintaxis **getLanguageDriver** ( ) String

Descripción Devuelve un valor de String que indica el nombre de la tabla de idioma actual de una tabla.

Ejemplo Este ejemplo muestra en un cuadro de diálogo la tabla de idioma de *Clientes*:

```
; obtenControlador::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Clientes.db")
msgInfo("Controlador", tc.getLanguageDriver())  ; visualiza el
mensaje "ascii"
endmethod
```

Vea también getLanguageDriverDesc

getLanguageDriverDesc

Método Devuelve el nombre de la descripción de tabla de idioma actual de una tabla.

Tipo TCursor

Sintaxis **getLanguageDriverDesc** ( ) String

Descripción Devuelve un valor de String que indica la descripción de tabla de idioma actual de una tabla.

Ejemplo Este ejemplo muestra en un cuadro de diálogo la descripción de tabla de idioma de la tabla *Clientes*.

```
; obtenDescripciónDelControlador::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Clientes.db")
msgInfo("Descripción", tc.getLanguageDriverDesc())
; visualiza el mensaje
; "Paradox ascii"
endmethod
```

Vea también getLanguageDriver

home

Principiante

Método Se desplaza al primer registro de una tabla.

Tipo TCursor

Sintaxis **home** ( ) Logical

Descripción Define el primer registro de una tabla como registro actual (y memoria intermedia de registro).

Ejemplo En este ejemplo, el método **pushButton** asocia un TCursor con la tabla *Pedidos* y carga en una matriz los valores de campo mediante un bucle **scan**. Cuando termina el bucle, el TCursor se sitúa en el último registro de la tabla. Este método emplea **home** para devolver el TCursor al primer registro de la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  matrizCampos Array[] AnyType
  ValorCampos AnyType
endVar
tc.open("Pedidos.db")
matrizCampos.grow(tc.nRecords())
scan tc:
; busca en la tabla y almacena el número de pedido en
matrizCampos
  tc.fieldValue(1, ValorCampos)
  matrizCampos[tc.recNo()] = tc."N° de Pedido"
endScan
; después del bucle de búsqueda, TCursor queda situado en el
último registro

matrizCampos.view()    ; muestra el contenido de la matriz

tc.home()              ; desplaza el TCursor al primer registro
endmethod
```

Vea también end
moveToRecord
moveToRecNo
nextRecord
priorRecord
skip

initRecord

Método Vacía la memoria intermedia de registro.

Tipo TCursor

Sintaxis **initRecord** ( ) Logical

Descripción Inicializa la memoria intermedia de registro rellenándola con blancos (*no* espacios). Si se han definido valores por defecto para los campos de la tabla, **initRecord** inicializa esos campos con los valores por defecto.

Vea también copyRecord
currRecord

insertAfterRecord

Método Inserta un registro en una tabla después del registro actual.

Tipo TCursor

Sintaxis **insertAfterRecord** ([const ***puntero*** TCursor]) Logical

Descripción Inserta un registro después del registro actual. Este método es útil para insertar un nuevo registro después del último registro de una tabla. Es posible emplear el argumento optativo *puntero* para insertar el registro actual señalado por otro TCursor, u omitir el argumento para insertar un registro vacío.

Si la tabla esta indexada, el registro se sitúa en su posición ordenada cuando se consigna el registro; si no lo está, se inserta después del registro actual.

Este método falla si la tabla no está en modo editar. También falla si el registro actual no puede consignarse (por ejemplo, por una violación de clave).

Ejemplo Para este ejemplo, supóngase que una ficha tiene un marco de tabla, *CLIENTES*, asociado con CLIENTES.DB. Cuando el usuario intenta borrar un registro, el método estándar **action** de *CLIENTES* desplaza el registro a ARCCLIEN.DB antes de borrar el registro de *CLIENTES*. Podría utilizarse **copyFromArray** y **copyToArray** para lograr el mismo resultado, pero si se emplea **insertAfterRecord** no es necesario almacenar el registro en una matriz para copiarlo. Como demuestra este código, es posible emplear el argumento optativo *puntero* para insertar el registro señalado por un TCursor.

```
; CLIENTES::action
method action(var eventInfo ActionEvent)
var
  ClienTC, ArchivoTC TCursor
endVar
if eventInfo.id() = DataDeleteRecord then     ; si el usuario
intenta eliminar
                                              ; un registro
  if EstaFicha.Editing = True then            ; y la Ficha está
en modo edición
    disableDefault                            ; todavía no se
procesa
                                              ;
DataDeleteRecord
    if msgYesNoCancel("Confirme",             ; si el usuario
confirma el borrado
       "¿Desea eliminar el registro actual?") = "Yes" then
       ClientTC.attach(CLIENTES)              ; sincronizamos
TCursor con el
                                              ; apuntador de
CLIENTES
       if ArchivoTC.open("ArcClien.db") then
         ArchivoTC.edit()
         ArchivoTC.end()                      ; saltamos al
final de la tabla
         ArchivoTC.insertAfterRecord(ClienTC) ; insertamos el
registro actual
                                              ; de CLIENTES
después del último
```

```
                                                    ; registro de
ArcClien.db
         doDefault                                  ; procesamos
DataDeleteRecord
       else
         msgStop("¡Atención!", "No puedo archivar el
registro.")
       endif
     else                                           ; el usuario no
confirmó el borrado
       message("No he eliminado el registro.")
     endif
   else                                             ; la Ficha no
está en modo Edición
     msgStop("¡Atención!", "Pulse F9 para activar el modo
edición.")
   endif
endif
endmethod
```

Vea también insertRecord
insertBeforeRecord

insertBeforeRecord

Método Inserta un registro en una tabla antes del registro actual.

Tipo TCursor

Sintaxis **insertBeforeRecord** ([const ***puntero*** TCursor]) Logical

Descripción Inserta un registro antes del registro actual (igual que **insertRecord**). Es posible emplear el argumento optativo *puntero* para insertar el registro señalado por otro TCursor, u omitir el argumento para insertar un registro vacío.

Si la tabla esta indexada, el registro se sitúa en su posición ordenada cuando se consigna el registro; si no lo está, se inserta antes del registro actual.

Este método falla si la tabla no está en modo editar. También falla si el registro actual no puede consignarse (por ejemplo, por una violación de clave).

Ejemplo Para este ejemplo, supóngase que una ficha tiene un marco de tabla, *CLIENTES*, asociado con CLIENTES.DB. Cuando el usuario intenta borrar un registro, el método estándar **action** de *CLIENTES* desplaza el registro a ARCCLIEN.DB antes de borrar el registro de *CLIENTES*. Podría utilizarse **copyFromArray** y **copyToArray** para lograr el mismo resultado, pero si se emplea **insertBeforeRecord** no es necesario almacenar el registro en una matriz para copiarlo. Como demuestra este código, es posible emplear el argumento optativo *puntero* para insertar el registro señalado por un segundo TCursor.

```
; CLIENTES::action
method action(var eventInfo ActionEvent)
var
  ClienTC, ArchivoTC TCursor
endVar
if eventInfo.id() = DataDeleteRecord then ; si el usuario
intenta eliminar
                                          ; un registro
  if EstaFicha.Editing = True then        ; si la Ficha está en
modo
                                          ; Edición
    disableDefault                        ; todavía no se
procesa
                                          ; DataDeleteRecord
    if msgYesNoCancel("Confirme",         ; si el usuario
confirma el borrado
       "¿Desea eliminar el registro actual?") = "Yes" then
      ClienTC.attach(CLIENTES)            ; sincronizamos
TCursor con el
                                          ; apuntador de
CLIENTES
      if ArchivoTC.open("ArcClien.db") then
        ArchivoTC.edit()
        ArchivoTC.end()                   ; saltamos al final
de la tabla
        ArchivoTC.insertBeforeRecord(ClienTC) ; insertamos el
registro actual
                                              ; de CLIENTES
antes del último
```

```
                                                    ; registro de
ArcClien.db
        doDefault                                   ; procesamos
DataDeleteRecord
      else
        msgStop("¡Atención!", "No puedo archivar el registro.")
      endif
    else                                            ; el usuario no
confirmó el borrado
      message("No he elimiando el registro.")
    endif
  else                                              ; la Ficha no
está en modo Edición
    msgStop("¡Atención!", "Pulse F9 para activar el modo
edición.")
  endif
endif
endmethod
```

Vea también insertRecord
insertAfterRecord

insertRecord

Principiante

Método Inserta un registro en una tabla.

Tipo TCursor

Sintaxis **insertRecord** ([const ***puntero*** TCursor]) Logical

Descripción Inserta un registro en una tabla antes del registro actual (igual que **insertBeforeRecord**). Es posible emplear el argumento optativo *puntero* para insertar el registro señalado por otro TCursor, u omitir el argumento para insertar un registro vacío.

Si la tabla está indexada, el registro se sitúa en su posición ordenada cuando se consigna el registro; si no lo está, se inserta antes del registro actual.

Este método falla si la tabla no está en modo editar. También falla si el registro actual no puede consignarse (por ejemplo, por una violación de clave).

Ejemplo Para este ejemplo, supóngase que una ficha tiene un marco de tabla, *CLIENTES*, asociado con CLIENTES.DB. Cuando el usuario intenta borrar un registro, el método estándar **action** de *CLIENTES* desplaza el registro a ARCCLIEN.DB antes de borrar el registro de *CLIENTES*. Podría utilizarse **copyFromArray** y **copyToArray** para lograr el mismo resultado, pero si se emplea **insertRecord** no es necesario almacenar el registro en una matriz para copiarlo. Como demuestra este código, es posible emplear el argumento optativo *puntero* para insertar el registro señalado por otro TCursor.

```
; CLIENTES::action
method action(var eventInfo ActionEvent)
var
  ClienTC, ArchivoTC TCursor
endVar
if eventInfo.id() = DataDeleteRecord then ; si el usuario
intenta eliminar
                                          ; un registro
  if EstaFicha.Editing = True then        ; si la Ficha está en
modo
                                          ; Edición
    disableDefault                        ; todavía no se
procesa
                                          ; DataDeleteRecord
    if msgYesNoCancel("Confirme",         ; si el usuario
confima el borrado
       "¿Desea eliminar el registro actual?") = "Yes" then
      ClienTC.attach(CLIENTES)            ; sincronizamos
TCursor con el
                                          ; puntero de CLIENTES
      if ArchivoTC.open( ArcClien.db ) then
        ArchivoTC.edit()
        ArchivoTC.insertRecord(ClienTC)   ; insertamos el
registro actual de
                                          ; CLIENTES antes del
último
                                          ; registro de
ArcClien.db
```

```
        doDefault                          ; ahora procesamos
DataDeleteRecord
      else
        msgStop("¡Atención!", "No puedo archivar el registro.")
      endif
    else                                   ; el usuario no
confirmó el borrado
      message("No he eliminado el registro.")
    endif
  else                                     ; la Ficha no está en
modo Edición
    msgStop("¡Atención!", "Pulse F9 para activar el modo
edición.")
  endif
endif
endmethod
```

Vea también insertAfterRecord
insertBeforeRecord

isAssigned

Método Informa de si se ha asignado un valor a una variable TCursor.

Tipo TCursor

Sintaxis **isAssigned** ( ) Logical

Descripción Devuelve True si se ha asignado un valor a una variable TCursor mediante **open** o **attach**; en caso contrario, devuelve False.

Ejemplo Este ejemplo asocia un TCursor con una tabla, muestra información encontrada en el último registro y cierra el TCursor. En este ejemplo, el código muestra un mensaje que indica si la variable TCursor todavía mantiene su asignación después de que se halla cerrado el TCursor. Este código se anexa al método estándar **pushButton** de *esteBotón*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Pedidos.db")         ; abrimos un TCursor para
Pedidos.db
tc.end()                      ; saltamos al final de la tabla

; mostramos la información del último registro
msgInfo("Ultimo Pedido", "Número del Pedido: " +
String(tc."N° de Pedido") + "\nFecha del pedido: " +
String(tc."Fecha de venta"))

tc.close()                    ; intentamos cerrar el TCursor

; si tenemos éxito al cerrar, se visualiza False (tc no se
vuelve a asignar)
; de otro modo, se muestra True (tc aún está asignado si hay
algún fallo ; al cerrar)
msgInfo("¿Está tc asignado?", tc.isAssigned())

endmethod
```

Vea también open
close

isEdit

Método Informa de si un TCursor está en modo editar.

Tipo TCursor

Sintaxis **isEdit** ( ) Logical

Descripción Devuelve True si el TCursor está en modo editar; si no es así, devuelve False. Si se anexa un TCursor con un gestor de visualización (como un UIObject o un TableView), y ese objeto está en modo editar, el TCursor también estará en dicho modo.

Ejemplo Para este ejemplo, supóngase que una ficha tiene un marco de tabla asociado con la tabla *Clientes* y un botón. El código anexado al método **pushButton** de *esteBotón* anexa un TCursor con el marco de tabla y, a continuación, utiliza **isEdit** para determinar si el TCursor se halla en modo editar. Si el marco de tabla estaba en modo editar cuando se anexó el TCursor, éste también lo estará.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar

; conectamos con la tabla
tc.attach(CLIENTES)

; si CLIENTES está en modo Edición, tc también estará en modo
Edición

if NOT tc.isEdit() then        ; comprobamos si tc está en modo
Edición
  tc.edit()
endif

if tc.locate("Nombre", "Club Acción") then
  tc.teléfono = "91-555-1234"
else
  msgStop("Lo siento", "No puedo encontrar el Club Acción")
endif

endmethod
```

Vea también edit
endEdit

isEmpty

Método Determina si una tabla contiene algún registro.

Tipo TCursor

Sintaxis **isEmpty** ( ) Logical

Descripción Devuelve True si no hay registros en la tabla asociada con el TCursor; en caso contrario, devuelve False.

Ejemplo En este ejemplo, el método **pushButton** del botón *mostrarNúmeroDeRegistros* muestra el número de registros de la tabla *Pedidos*. Si la tabla está vacía, este código alerta al usuario al respecto.

```
; mostrarNúmeroDeRegistros::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombreTabla String
endVar
NombreTabla = "Pedidos.db"

if tc.open(NombreTabla) then
  if tc.isEmpty() then           ; si la tabla Pedidos.db está
vacía
msgStop("¡Atención!",
               "La tabla"  + NombreTabla "está vacía.")
  else
                                 ; mostramos el número de
registros
  msgInfo( "La tabla" + NombreTabla +   "tiene" ,
            String(tc.nRecords()) + "registros.")
  endIf
else
  msgStop("Lo siento", "No puedo abrir la tabla"  + NombreTabla
+ ".")
endif
endmethod
```

Vea también empty
nRecords
insertRecord

isEncrypted

Método Informa de si una tabla está protegida mediante contraseña.

Tipo TCursor

Sintaxis **isEncrypted** ( ) Logical

Descripción Devuelve True si una tabla está protegida mediante contraseña (cifrada); en caso contrario, devuelve False. Un TCursor no puede abrirse en una tabla cifrada hasta que se utilice el método **addPassword** del tipo Session para presentar la contraseña requerida. Este método no informa de si un usuario tiene derechos de acceso a la tabla; para ello, utilice **tableRights**.

Ejemplo Este ejemplo comprueba si la tabla *Clientes* está cifrada.

```
; esteBotón::pushButton
method open(var eventInfo Event)
var
  tc TCursor
  endvar

if tc.open("Clientes.db") then
  if tc.isEncrypted() then
    msgInfo("La tabla está protegida", "Se ha utilizado una
clave aceptable.")
endIf
else
    msgStop("Error", "No puedo abrir la tabla Clientes.")
endif
endmethod
```

Vea también tableRights
Session::addPassword
Table::protect

isRecordDeleted

Método Informa de si se ha borrado el registro actual (sólo en tablas de dBASE).

Tipo TCursor

Sintaxis **isRecordDeleted** ( ) Logical

Descripción Informa de si se ha borrado el registro actual. **isRecordDeleted** sólo funciona con tablas de dBASE porque un registro de Paradox borrado no puede mostrarse. Este método devuelve True si el registro actual se ha borrado; en caso contrario, devuelve False..

Por defecto, los registros borrados de una tabla de dBASE no se muestran. Para que **isRecordDeleted** funcione correctamente, es necesario ejecutar **showDeleted** para mostrar los registros borrados de la tabla; de lo contrario, los registros borrados no son visibles para **isRecordDeleted**.

Ejemplo Este ejemplo abre un TCursor para la tabla de dBASE PUNTOS.DBF y utiliza **showDeleted** para mostrar todos los registros borrados. Entonces, el código intenta encontrar un específico de la tabla. Este ejemplo emplea **isRecordDeleted** para determinar si el registro se ha borrado; si se ha borrado, se recupera mediante **undeleteRecord**. El código siguiente se anexa al método **pushButton** de *esteBotón*:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Puntos.dbf")                ; abrimos un TCursor para
una tabla de dBASE
tc.showDeleted()                     ; mostramos los registros
eliminados
if tc.locate("Nombre", "Pedro") then ; si encontramos a Pedro
en el campo Nombre
if tc.isRecordDeleted() then         ; si el registro ha sido
eliminado
tc.edit()                            ; activamos el modo de
edición
tc.undeleteRecord()                  ; y recuperamos el
registro eliminado
message("El registro de Pedro ha sido recuperado")
  endif
else
msgStop("Error", "No encuentro a Pedro.")
endif
endmethod
```

Vea también isShowDeletedOn
showDeleted

isShared

Método Informa de si una tabla está compartida actualmente.

Tipo TCursor

Sintaxis **isShared** ( ) Logical

Descripción Devuelve True si otro usuario ha abierto la tabla señalada por un TCursor; en caso contrario, devuelve False.

Ejemplo En este ejemplo, el método estándar **open** de una ficha determina si CLIENTES.DB es compartido actualmente por otro usuario; si es así, se advierte al usuario y se le da la opción de continuar o cancelar.

```
; estaPágina::open
method open(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Clientes.db")            ; abrimos un TCursor para
Clientes
if tc.isShared() then             ; si la tabla está
actualmente compartida
if msgYesNoCancel("¿Continuamos?",; pedimos conformidad
   "La tabla Clientes se encuentra compartida en este momento.\
n" +
   "¿Desea continuar?") <> "Yes" then
  close()                         ; cerramos la Ficha
  endif
endif
endmethod
```

Vea también isAssigned
isValid

isShowDeletedOn

Método Informa de si se muestran los registros borrados de una tabla de dBASE.

Tipo TCursor

Sintaxis **isShowDeletedOn** ( ) Logical

Descripción Informa de si la tabla señalada por un TCursor muestra actualmente registros borrados. Es posible emplear el método **showDeleted** para especificar si se muestran o no los registros borrados y, a continuación, utilizar **isShowDeletedOn** para determinar su estado. **isShowDeletedOn** sólo es válido en las tablas de dBASE.

Ejemplo En este ejemplo, si **isShowDeletedOn** devuelve False, el código ejecuta **showDeleted** para mostrar los registros borrados de PEDIDOS.DBF.

```
; mostrarRegistrosEliminados::pushButton
method pushButton(var eventInfo Event)
var
  fbdTC TCursor
endVar
if fbdTC.open("Pedidos.dbf") then
  if NOT fbdTC.isShowDeletedOn() then ; si no se muestran los
registros eliminados
    fbdTC.showDeleted(Yes)            ; mostramos los registros
eliminados
  endif
else
  msgStop("Lo siento", "No puedo abrir la tabla Pedidos.dbf.")
endIf
endmethod
```

Vea también compact
isRecordDeleted
showDeleted

isValid

Método Informa de si el contenido de un campo es legítimo y está completo.

Tipo TCursor

Sintaxis **1. isValid** (const ***nombreCampo*** String, const ***valor***AnyType) Logical

2. isValid (const ***númeroCampo*** SmallInt, const ***valor*** AnyType) Logical

Descripción Informa de si el valor especificado en *valor* corresponde al tipo de campo y controles de validación del campo especificado en *númeroCampo* o *nombreCampo*. Este método proporciona la oportunidad de comprobar si un nuevo valor es válido para un campo antes de que se intente almacenar el registro.

isValid devuelve True si *valor* es conforme con el tipo de campo y controles de validación; en caso contrario, devuelve False.

Ejemplo Este ejemplo utiliza **isValid** para comprobar si un valor dado es válido para un campo de fecha. Si el valor no es válido, este código advierte al usuario del error; si lo es, el valor se introduce en el campo. El código siguiente se anexa al método **pushButton** de *esteBotón*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  valor Number
endVar
valor = 100
tc.open("Pedidos.db")
if NOT tc.isValid("Fecha de Venta", Valor) then  ; 100 no es
una fecha válida
  msgStop("Error",
  String(valor) +  "no es válido para este campo.")
else                                              ; esta
condición nunca se cumple
  tc."Fecha de venta" = Valor
  tc.postRecord()
endif

endmethod
```

Vea también postRecord
setFieldValue

locate

Principiante

Método Busca un valor de campo especificado.

Tipo TCursor

Sintaxis **1. locate** (const ***nombrecampo*** const ***valorBúsqueda*** AnyType [, const ***nombreCampo*** String, const ***valorBúsqueda*** AnyType]*) Logical

2. locate (const ***númeroCampo*** SmallInt, const ***valorBúsqueda*** AnyType [, const ***númeroCampo*** SmallInt, const ***valorBúsqueda*** AnyType]*) Logical

Descripción Busca en una tabla registros cuyos valores coincidan con los criterios especificados en uno o más pares de campo/valor. Especifique en *valorBúsqueda* el valor que se va a buscar y en *nombreCampo* o *númeroCampo* el campo en que se va a buscar. Este método garantiza que se encuentra el primer valor coincidente con *valorBúsqueda*, dada la vista actual de los registros. Si el TCursor utiliza un índice secundario, **locate** busca el primer registro en el orden del índice secundario independientemente del orden de ese registro en el índice primario.

La búsqueda siempre comienza desde el principio de la tabla, pero si no se encuentra ninguna coincidencia, el TCursor vuelve al registro original. Si se encuentra una, el TCursor se desplaza a ese registro. Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *fijarDeletreo* busca un valor en el campo *Nombre* de la tabla *Clientes*. Si **locate** es satisfactorio, este método sustituye el nombre por otro valor e informa al usuario del cambio.

```
; fijarDeletreo::pushButton
method pushButton(var eventInfo Event)
var
  PedidoTC TCursor
endVar

PedidoTC.open("Clientes.db")
; si encontramos "Sumarinismo Profesional." en el campo Nombre
if PedidoTC.locate("Nombre", "Sumarinismo Profesional.") then
  PedidoTC.edit()
  ; activar el modo Edición
  PedidoTC.Name = "Submarinismo Profesional"
  ; deletreo correcto (Submarinismo)
  msgInfo("Éxito", "He corregido el error de deletreo.")
else
  msgInfo("Fallo en la búsqueda",
          "No he podido encontrar \nSumarinismo Profesional")
endIf
PedidoTC.endEdit()
endmethod
```

Vea también locateNext
locatePattern
locateNextPattern

locateNext

Principiante

Método Busca un valor de campo especificado.

Tipo TCursor

Sintaxis **1. locateNext** (const ***nombreCampo*** String, const ***valorBúsqueda*** AnyType [, const ***nombreCampo*** String, const ***valorBúsqueda*** AnyType]*) Logical

2. locateNext (const ***númeroCampo*** SmallInt, const ***valorBúsqueda*** AnyType [, const ***númeroCampo*** SmallInt, const ***valorBúsqueda*** AnyType]*) Logical

Descripción Busca en una tabla los registros cuyos valores coincidan con los criterios especificados en uno o más pares de campo/valor. Especifique en *valorBúsqueda* el valor que se va a buscar y en *nombreCampo* o *númeroCampo* el campo en que se va a buscar. Este método garantiza que se encuentra el siguiente valor coincidente con *valorBúsqueda*, dada la vista actual de los registros. Si el TCursor utiliza un índice secundario, **locateNext** busca el primer registro en el orden del índice secundario independientemente del orden de ese registro en el índice primario .

La búsqueda comienza en el registro siguiente al actual. Si se encuentra una coincidencia, el TCursor se desplaza a ese registro. Si no se encuentra, el TCursor vuelve al registro original. Para comenzar una búsqueda desde el principio de una tabla, utilice **locate**.

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Ejemplo Este ejemplo emplea **locate** y **locateNext** para contar el número de registros que tienen Madrid en el campo Provincia de la tabla *Clientes*. El código siguiente se anexa al método **pushButton** de *encontrarMadrid*.

```
; encontrarMadrid::pushButton
method pushButton(var eventInfo Event)
var
  ClienTC TCursor
  NumEncontrado LongInt
endVar
ClienTC.open("Clientes.db")

if ClienTC.locate("Provincia", "Madrid") then
  NumEncontrado = 1
  while ClienTC.locateNext("Provincia", "Madrid")
    NumEncontrado = NumEncontrado + 1
  endWhile
  msgInfo("Registros encontrados",
String("He encontrado" , NumEncontrado, " compañías en
Madrid"))
else
  msgInfo("Lo siento", "No he encontrado Madrid en ningún campo
Provincia.")
endIf

endmethod
```

Vea también locate
locateNextPattern
locatePattern

locateNextPattern

Método Busca el siguiente registro que contiene un campo con un patrón especificado de caracteres.

Tipo TCursor

Sintaxis **1. locateNextPattern** ([const ***nombreCampo*** String, const ***valorBúsqueda*** AnyType] * const ***nombreCampo*** String, const ***patrón*** AnyType) Logical

2. locateNextPattern ([const ***nombreCampo*** SmallInt, const ***valorBúsqueda*** AnyType] * const ***númeroCampo*** SmallInt, const ***patrón*** AnyType) Logical

Descripción Busca subcadenas (por ejemplo, orde en ordenador). La búsqueda comienza en el registro siguiente al actual. Si se encuentra una coincidencia, el TCursor se desplaza a ese registro. Si no se encuentra, el TCursor vuelve al registro original. Si el TCursor utiliza un índice secundario, **locateNextPattern** busca el primer registro en el orden del índice secundario independientemente del orden de ese registro en el índice primario .

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave). Para comenzar la búsqueda al comienzo de una tabla, utilice **locatePattern.**

Para buscar registros en función del valor de un solo campo, especifique el campo en *nombreCampo* o *númeroCampo*, y especifique un patrón de caracteres en *patrón*.

Es posible incluir los operadores de patrón @ y .. en el argumento *patrón*. El operador .. representa cualquier cadena de caracteres (incluso, ninguno); @ implica cualquier carácter individual. Cualquier combinación de caracteres literales y comodines puede emplearse en la construcción de una cadena de búsqueda. Si **advancedWildcardsInLocate** (del tipo Session) está activado, es posible emplear operadores de patrón de coincidencia avanzados, no los operadores de patrón estándar. Consulte la descripción de **advMatch** del tipo String para más información sobre operadores de patrón de coincidencia avanzados, y **advancedWildcardsInLocate** e **isAdvancedWildcardsInLocate** del tipo Session.

Por ejemplo, la sentencia siguiente comprueba los valores del primer campo de cada registro. Si un valor es distinto de "Borland", **locateNextPattern** devuelve True.

```
tc.locateNextPattern(1,"[^Borland]")
```

Para buscar registros en función de los valores de más de un campo, especifique coincidencias exactas en todos los campos *excepto* el último de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas con Borland en el campo Nombre , Paradox en el campo Producto y palabras que comienzan con datos (por ejemplo, datos de la base) en el campo Contraseñas .

```
tc.locateNextPattern ("Nombre", "Borland", "Producto",
"Paradox", "Contraseñas", "datos*")
```

Ejemplo En este ejemplo, supóngase que existe la tabla SOFTWARE.DB en el directorio actual. Supóngase además que dos de los campos se denominan Producto y Nombre. Este código (anexado al método **pushButton**) busca registros cuyo campo Nombre contenga "Borland" y cuyo campo Producto comience con "Par". Este código

controla las coincidencias encontradas y almacena los valores de campo en una matriz redimensionable. Cuando el método no puede encontrar más registros que cumplan los criterios, los resultados se muestran en un cuadro de diálogo.

```
; encontrarBuenosProductos::pushButton
method pushButton(var eventInfo Event)
var
  Nombres TCursor
  Buscar String
  NumEncontrado SmallInt
  NombreProductos Array[] String
endVar
Nombre.open("software.db")
Buscar = "Borland"

; buscamos registros que incluyan "Borland" en el campo Nombre
; y valores que empiecen por "Par" en el campo Producto
if Nombres.locatePattern("Nombre", Buscar, "Producto", "Par..")
then
  NumEncontrado = 1
  NombreProductos.grow(1)
  NombreProductos[NumEncontrado] = Nombres.Producto

  ; ahora continuamos buscando por los campos con el mismo
criterio
  ; y almacenamos los valores del campo Producto en la matriz
Nombres
while Nombres.locateNextPattern("Nombre", Buscar, "Producto",
"Par..")
    NumEncontrado = NumEncontrado + 1
    NombreProductos.addLast(Nombres.Producto)
  endWhile
endIf
if NombreProductos.size() > 0 then
  NombreProductos.view()
endIf
endmethod
```

Vea también locatePattern
locateNext
String::advMatch
Session::advancedWildCardsInLocate
Session::isAdvancedWildCardsInLocate

locatePattern

Método Busca un registro que contiene un campo con un patrón especificado de caracteres.

Tipo TCursor

Sintaxis **1. locatePattern** ([const ***nombreCampo*** String,

const ***valorBúsqueda*** AnyType] * const ***nombreCampo*** String,

const ***patrón*** String) Logical

2. locatePattern ([const ***númeroCampo*** SmallInt,

const ***valorBúsqueda*** AnyType] * const ***númeroCampo*** SmallInt,

const ***patrón*** String) Logical

Descripción Busca subcadenas (por ejemplo, "orde" en "ordenador"). La búsqueda comienza al principio de la tabla, pero si no encuentra ninguna coincidencia, el TCursor vuelve al registro original. Si se encuentra una coincidencia, el TCursor se desplaza a ese registro. Si el TCursor utiliza un índice secundario, **locatePattern** busca el primer registro en el orden del índice secundario independientemente del orden de ese registro en el índice primario .

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave). Para comenzar la búsqueda después del registro actual, utilice **locateNextPattern**. Para comenzarla antes del registro actual, emplee **locatePriorPattern**.

Para buscar registros en función del valor de un solo campo, especifique el campo en *nombreCampo* o *númeroCampo*, y especifique un patrón de caracteres en *patrón*.

Es posible incluir los operadores de patrón @ y .. en el argumento *patrón*. El operador .. representa cualquier cadena de caracteres (incluso, ninguno); @ implica cualquier carácter individual. Cualquier combinación de caracteres literales y comodines puede emplearse en la construcción de una cadena de búsqueda. Si **advancedWildcardsInLocate** (del tipo Session) está activado, es posible emplear operadores de patrón de coincidencia avanzados, no los operadores de patrón estándar. Consulte la descripción de **advMatch** del tipo String para más información sobre operadores de patrón de coincidencia avanzados, y **advancedWildcardsInLocate** e **isAdvancedWildcardsInLocate** del tipo Session.

Por ejemplo, la sentencia siguiente comprueba los valores del primer campo de cada registro. Si un valor es distinto de "Borland", **locatePattern** devuelve True.

```
tc.locatePattern(1,"[^Borland]")
```

Para buscar registros en función de los valores de más de un campo, especifique coincidencias exactas en todos los campos *excepto* el último de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas con Borland en el campo "Nombre", "Paradox" en el campo "Producto" y palabras que comienzan con "datos" (por ejemplo, "datos de la base") en el campo Contraseñas .

```
tc.locatePattern ("Nombre", "Borland", "Producto", "Paradox",
"Contraseñas", "datos*")
```

Ejemplo En este ejemplo, supóngase que existe la tabla SOFTWARE.DB en el directorio actual. Supóngase además que dos de los campos se denominan Producto y

Nombre. Este código (anexado al método **pushButton**) busca registros cuyo campo Nombre contenga "Borland" y cuyo campo Producto comience con "Par". Este código controla las coincidencias encontradas y almacena los valores de campo en una matriz redimensionable. Cuando el método no encuentra más registros que cumplan los criterios, los resultados se muestran en un cuadro de diálogo.

```
; encontrarBuenosProductos::pushButton
method pushButton(var eventInfo Event)
var
  Nombres TCursor
  Buscar String
  NumEncontrado SmallInt
  NombreProductos Array[] String
endVar
Nombre.open("software.db")
Buscar = "Borland"

; buscamos registros que incluyan "Borland" en el campo Nombre
; y valores que empiecen por "Par" en el campo Producto
if Nombres.locatePattern("Nombre", Buscar, "Producto", "Par..")
then
  NumEncontrado = 1
  NombreProductos.grow(1)
  NombreProductos[NumEncontrado] = Nombres.Producto

  ; ahora continuamos buscando por los campos con el mismo
criterio
  ; y almacenamos los valores del campo Producto en la matriz
Nombres
while Nombres.locateNextPattern("Nombre", Buscar, "Producto",
"Par..")
    NumEncontrado = NumEncontrado + 1
    NombreProductos.addLast(Nombres.Producto)
  endWhile
endIf
if NombreProductos.size() > 0 then
  NombreProductos.view()
endIf
endmethod
```

Vea también locateNextPattern
locate
String::advMatch
Session::advancedWildCardsInLocate
Session::isAdvancedWildCardsInLocate

locatePrior

Método Busca un valor de campo especificado.

Tipo TCursor

Sintaxis **1. locatePrior** ([const fieldName String, const searchValue AnyType] 1+) Logical

2. locatePrior (const ***númeroCampo*** SmallInt const ***valorBúsqueda*** AnyType [, const ***númeroCampo*** String, const ***valorBúsqueda*** AnyType]*) Logical

Descripción Busca en una tabla los registros cuyos valores coincidan con los criterios especificados en uno o más pares de campo/valor. Especifique en *valorBúsqueda* el valor que se va a buscar y en *nombreCampo* o *númeroCampo* el campo en que se va a buscar. Este método garantiza que se encuentra el valor anterior coincidente con *valorBúsqueda*, dada la vista actual de los registros. Si el TCursor utiliza un índice secundario, **locatePrior** busca el registro anterior en el orden del índice secundario independientemente del orden de ese registro en el índice primario .

La búsqueda comienza desde el registro actual y busca hacia atrás en la tabla la coincidencia anterior. Si se encuentra una coincidencia, el TCursor se desplaza a ese registro; en caso contrario, vuelve al registro original. Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave). Este método devuelve True si es satisfactorio; de lo contrario, devuelve False.

Ejemplo En este ejemplo, el método **pushButton** de *mostrarAnterior* busca hacia atrás en la tabla *Articulo* registros con un cierto número de pedido. La variable *líneaTC* se declara en la ventana Var de la página y se abre en la tabla *Articulo* en el método **open** para la página.

El código siguiente va en la ventana Var de *estaPágina*:

```
; estaPágina::var
Var
  LineaTC TCursor
endVar
```

El código siguiente se anexa al método **open** de *estaPágina*:

```
; estaPágina::open
method open (var eventInfo Event)
  LineaTC.open ("articulo")         ; abre un TCursor para
articulo.db
endmethod
```

El código siguiente se anexa al método **pushButton** del botón *mostrarAnterior*:

```
; mostrarAnterior::pushButton
method pushButton(var eventInfo Event)
var
  Registro Array[] AnyType
endVar

if LineaTC.locatePrior("N° de Pedido", 1005) then
  LineaTC.copyToArray(Registro)
  Registro.view("Registro n°" + String(LineaTC.recNo()))
```

```
else
  msgStop("Lo siento", "No hay más registros.")
endif
endmethod
```

Vea también locateNext
locateNextPattern
locatePattern

locatePriorPattern

Método Busca el registro anterior que contiene un campo con un patrón especificado de caracteres.

Tipo TCursor

Sintaxis **1. locatePriorPattern** ([const ***nombreCampo*** String, const ***valorBúsqueda*** AnyType] * const ***nombreCampo*** String, const ***patrón*** String) Logical

2. locatePriorPattern ([const ***númeroCampo*** SmallInt, const ***valorBúsqueda*** AnyType] * const ***númeroCampo*** SmallInt, const ***patrón*** String) Logical

Descripción Busca subcadenas (por ejemplo, "orde" en "ordenador"). La búsqueda comienza en el registro anterior al actual. Si se encuentra una coincidencia, el TCursor se desplaza a ese registro. Si no se encuentra ninguna coincidencia, vuelve al registro original. Si el TCursor utiliza un índice secundario, **locatePriorPattern** busca el registro anterior en el orden del índice secundario independientemente del orden de ese registro en el índice primario .

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave). Para comenzar la búsqueda al comienzo de una tabla, utilice **locatePattern**.

Para buscar registros en función del valor de un solo campo, especifique el campo en *nombreCampo* o *númeroCampo,* y especifique un patrón de caracteres en *patrón*.

Es posible incluir los operadores de patrón @ y .. en el argumento *patrón*. El operador .. representa cualquier cadena de caracteres (incluso, ninguno); @ implica cualquier carácter individual. Cualquier combinación de caracteres literales y comodines puede emplearse en la construcción de una cadena de búsqueda. Si **advancedWildcardsInLocate** (del tipo Session) está activado, es posible emplear operadores de patrón de coincidencia avanzados, no los operadores de patrón estándar. Consulte la descripción de **advMatch** del tipo String para más información sobre operadores de patrón de coincidencia avanzados, y **advancedWildcardsInLocate** e **isAdvancedWildcardsInLocate** del tipo Session.

Por ejemplo, la sentencia siguiente comprueba los valores del primer campo de cada registro. Si un valor es distinto de "Borland", **locatePriorPattern** devuelve True.

```
tc.locatePriorPattern(1, "[^Borland]")
```

Para buscar registros en función de los valores de más de un campo, especifique coincidencias exactas en todos los campos *excepto* el último de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas con "Borland" en el campo Nombre, "Paradox" en el campo Producto y palabras que comienzan con "datos" (por ejemplo, datos de la base) en el campo Contraseñas .

```
tc.locatePriorPattern("Nombre", "Borland", "Producto",
"Paradox", "Contraseñas", "datos*")
```

Ejemplo En este ejemplo, el método **pushButton** de *mostrarAnterior* busca hacia atrás en la tabla *Software* registros con una cierta compañía y número de producto. La variable *tc* se declara en la ventana Var de la página y se abre en la tabla *Software* en el método **open** para la página.

El código siguiente va en la ventana Var de *estaPágina*:

```
; estaPágina::var
var
  tc     TCursor
  Buscar String
endVar
```

El código siguiente se anexa al método **open** de *estaPágina*:

```
; estaPágina::open
method open(var eventInfo Event)
  tc.open("Software.db")  ; abrimos un TCursor para Software.db
  tc.end()                ; llevamos el TCursor al último
registro
  Buscar = "Borland"
endmethod
```

El código siguiente se anexa al método **pushButton** del botón *mostrarAnterior*:

```
; mostrarAnterior::pushButton
method pushButton(var eventInfo Event)
var
  Registro Array[] AnyType
endVar

; buscamos el patrón anterior
if tc.locatePriorPattern("Nombre", Buscar, "Producto", "Par..")
then
  tc.copyToArray(Registro)
  Registro.view("Registro N°"+ String(tc.recNo()))
else
  msgStop("Lo siento", "No hay más registros.")
endif
endmethod
```

Vea también locatePattern
locateNextPattern
String::advMatch
Session::advancedWildCardsInLocate
Session::isAdvancedWildCardsInLocate

lock

Principiante

Método Aplica bloqueos especificados sobre una tabla especificada.

Tipo TCursor

Sintaxis **lock** (const ***tipoBloqueo*** String) Logical

Descripción Intenta aplicar un bloqueo sobre el TCursor, donde *tipoBloqueo* es uno de los siguientes: Write (Escritura), Read (Lectura), Full (Completo) o Any (Cualquiera). Si lo logra, este método devuelve True; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente abre un TCursor para *Clientes*, aplica un bloqueo completo sobre la tabla y utiliza **reIndex** para reconstruir el índice *Teléfono*. Una vez que el índice está reconstruido, este código desbloquea *Clientes* por lo que otros usuarios de una red pueden acceder a la tabla.

```
; reindexarClientes::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Tabla String
endVar
Tabla = "Clientes.db"

if tc.open(Tabla) then
  if tc.lock("Full") then      ; intentamos obtener un bloqueo
completo
    tc.reIndex("Teléfono")     ; reconstruimos el índice
Teléfono
    tc.unLock("Full")          ; desbloqueamos la tabla
    message("Teléfono reconstruido.")
  else
    msgStop("Lo siento", "No puedo bloquear la tabla" + Tabla+
".")
  endif
endif
endmethod
```

Vea también lockStatus
unlock

lockRecord

Método Aplica un bloqueo de escritura al registro actual.

Tipo TCursor

Sintaxis **lockRecord** ( ) Logical

Descripción Paradox aplica un bloqueo de escritura sobre un registro cuando se comienza a realizar cambios (bloqueo implícito de registro). **lockRecord** intenta aplicar un bloqueo de escritura sobre el registro señalado por un TCursor (bloqueo explícito) y, si lo logra, devuelve True; en caso contrario, devuelve False.

Ejemplo En el ejemplo siguiente, el método **pushButton** de *esteBotón* busca un registro en la tabla *Clientes*. Si la búsqueda es satisfactoria, este ejemplo intenta bloquear el registro con **lockRecord**. Cuando se ha bloqueado el registro, se ejecuta un procedimiento personalizado para obtener del usuario información sobre nuevos clientes. Si **lockRecord** no es satisfactorio, se solicita al usuario que lo intente de nuevo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  ClienTC, MiClienTC TCursor
endVar
ClienTC.open("Clientes.db")

; intentamos localizar el registro en Clientes.db
if ClienTC.locatePattern("Nombre", "Jamaica..") then
  ClienTC.edit()
  if ClienTC.lockRecord() then ; intentamos bloquear el
registro
    ClienTC.initRecord()       ; iniciamos el registro con los
valores
                               ; por defecto
    ObtenInfoCliente()         ; llamamos a un procedimiento
definido por
                               ; el usuario
  else                         ; el registro no se puede
bloquear
    msgStop("Lo siento", "No puedo bloquear el registro."\n
Inténtelo otra vez luego.")
  endif
else
    msgStop("Lo siento", "No encuentro el registro.")
endif

endmethod
```

Vea también unLockRecord

lockStatus

Método Devuelve el número de veces que se ha aplicado un bloqueo sobre una tabla.

Tipo TCursor

Sintaxis **lockStatus** (***tipoBloqueo*** String) SmallInt

Descripción Devuelve el número de veces que se ha aplicado un bloqueo del tipo *tipoBloqueo* a una tabla, donde *tipoBloqueo* es uno de los siguientes valores de String: Write (Escritura), Read (Lectura), Full (Completo) o Any (Cualquiera).

Si no se ha aplicado ningún bloqueo de un tipo dado, **lockStatus** devuelve 0.

Si se especifica "Any" como *tipoBloqueo*, **lockStatus** devuelve el número total de bloqueos aplicados sobre la tabla. **lockStatus** sólo informa de los bloqueos aplicados explícitamente, no de los aplicados por Paradox o por otros usuarios o aplicaciones.

Ejemplo Este ejemplo utiliza **lockStatus** para determinar si se ha aplicado algún bloqueo de escritura sobre la tabla *CLIENTES*; si es así, se eliminan todos los bloqueos de escritura.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Tabla String
endVar

tc.open("Clientes.db")

; repetir hasta que todos los bloqueos contra escritura de
Clientes
; sean eliminados
while tc.lockStatus("Write") > 0
  tc.unlock("Write")
endWhile
message("He eliminado todos los bloqueos contra escritura de
Clientes.db")

endmethod
```

Vea también lock
unLockRecord

moveToRecNo

Método Desplaza un TCursor a un registro específico de una tabla.

Tipo TCursor

Sintaxis **moveToRecNo** (const ***númeroRegistro*** LongInt) Logical

Descripción Define el registro especificado en *númeroRegistro* como registro actual. Devuelve un error si *númeroRegistro* no está en la tabla. Utilice el método **nRecords** o examine la propiedad NRecords para averiguar cuántos registros contiene una tabla. Este método está recomendado sólo para las tablas de dBASE. Cuando se utiliza con una tabla de Paradox, **moveToRecNo** se comporta exactamente igual que el método **moveToRecord**.

Ejemplo Este ejemplo desplaza un TCursor al sexto registro de una tabla de dBASE.

```
; irAlRegistroSeis::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Pedidos.db")     ; suponemos que Pedidos.db tiene 10
registros
if not tc.moveToRecNo(6) then
  msgStop("Lo siento", "No puedo posicionarme en el registro
actual.")
endif
endmethod
```

Vea también currRecord
end
home
moveToRecord
nextRecord
nRecords
priorRecord
skip

moveToRecord

Método Desplaza un TCursor a un registro específico de una tabla.

Tipo TCursor

Sintaxis **moveToRecord** (const ***númeroRegistro*** LongInt) Logical

Descripción Define el registro especificado en *númeroRegistro* como registro actual (y memoria intermedia de registro). Devuelve un error si *númeroRegistro* es mayor que el número de registros de la tabla. Utilice el método **nRecords** o examine la propiedad NRecords para averiguar cuántos registros contiene una tabla. Estos métodos pueden ser muy lentos con las tablas de dBASE; utilice **moveToRecNo** en su lugar.

Esta operación falla si el registro actual no puede consignarse (por ejemplo, por una violación de clave).

Ejemplo En este ejemplo, método **pushButton** intenta desplazar un TCursor al sexto registro de la tabla *Pedidos*. Si no puede desplazarse el TCursor (porque el registro actual no puede almacenarse), este método muestra un mensaje de advertencia.

```
; irAlRegistroSeis::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Pedidos.db")     ; suponemos que Pedidos.db tiene 10
registros
if not tc.moveToRecord(6) then
  msgStop("Lo siento", "No puedo posicionarme en el registro
actual.")
endif
endmethod
```

Vea también nRecords
home
end
nextRecord
priorRecord
currRecord
skip

nextRecord

Principiante

Método Se desplaza al registro siguiente de una tabla.

Tipo TCursor

Sintaxis **nextRecord** ( ) Logical

Descripción Define el registro siguiente de la tabla como registro actual. Si la tabla está en modo editar, **nextRecord** consigna los cambios en el registro actual antes de realizar el desplazamiento. Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

nextRecord devuelve False si se intenta ir más allá del final de la tabla. Igualmente, el último registro de la tabla se convierte en el registro actual y **eot** devuelve True.

Ejemplo En este ejemplo, el método **pushButton** de *mostrarPróximoCliente* utiliza **nextRecord** para desplazar un TCursor por la tabla *Clientes.* Cada vez que el TCursor accede a un nuevo registro, el código emplea **copyToArray** para copiar el contenido del registro a un DynArray y, entonces, muestra los valores de campo en un cuadro de diálogo. Cuando **nextRecord** intenta desplazarse más allá del último registro de la tabla, **eot** devuelve True y el método **pushButton** termina.

```
; mostrarPróximoCliente::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Grabar DynArray[] AnyType
  NombreTabla String
endVar
NombreTabla = "Clientes.db"

if tc.open(NombreTabla) then

  while NOT tc.eot()        ; Verdadero mientras que nextRecord
no intente
                            ; desplazarse detrás del final de
la tabla
    tc.copyToArray(Grabar)  ; copiamos el registro al DynArray
Grabar
    Grabar.view("Registro"  + String(tc.recNo()))
    if msgQuestion("",
       "¿Quiere ver el siguiente registro?") = "Yes" then
      tc.nextRecord()       ; nos desplazamos al registro
siguiente
    else
      return
    endif
  endWhile

  msgStop("¡Eso es todo!", "No hay más registros.")

else
  msgStop("Lo siento", "No puedo abrir la tabla"  + NombreTabla
+ ".")
endif
```

```
endmethod
```

Vea también

home
end
priorRecord
skip
moveToRecord

nFields

Método Devuelve el número de campos de una tabla.

Tipo TCursor

Sintaxis **nFields** ( ) LongInt

Descripción Devuelve el número de campos de la tabla asociada con un TCursor.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* muestra el número de campos de la tabla *VidaMar*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
if tc.open("VidaMar.db") then
  msgInfo("Número de campos de VidaMar", tc.nFields())
else
  msgStop("Lo siento", "No puedo abrir la tabla VidaMar.db")
endif

endmethod
```

Vea también nKeyFields
nRecords

nKeyFields

Método Devuelve el número de campos existentes en el índice actual de una tabla.

Tipo TCursor

Sintaxis **nKeyFields** ( ) LongInt

Descripción Devuelve el número de campos del índice actual de la tabla asociada con un TCursor. Cuando se utiliza con una tabla de Paradox, este método trabaja con el índice primario; cuando se emplea con una de dBASE, trabaja con el índice especificado por el método **setIndex** del tipo Table.

Ejemplo Este ejemplo informa del número de campos clave de una tabla de Paradox.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  Paradox dBASE TCursor
  NCC LongInt
  ParadoxTabla, dBASETabla String
  Tabla Table
endVar
ParadoxTabla = "Pedidos.db"
dBASETabla = "Puntos.dbf"

if Paradox.open(ParadoxTabla) then
  NCC = Paradox.nKeyFields() ; recogemos el número de campos
clave en el
                              ; índice primario
  msgInfo(ParadoxTabla, ParadoxTabla +  "tiene"  + String(NCC)
+    "campos clave.")
else
  msgInfo("Lo siento", "No puedo abrir la tabla" + ParadoxTabla
+ ".")
endif

endmethod
```

Vea también nFields
nRecords

nRecords

Principiante

Método Devuelve el número de registros de una tabla.

Tipo TCursor

Sintaxis **nRecords** ( ) LongInt

Descripción **nRecords** devuelve el número de registros de la tabla asociada con un TCursor. Esta operación puede ser muy larga en las tablas grandes.

Cuando se trabaja con una tabla de dBASE, **nRecords** cuenta los registros borrados, si **showDeleted** está activado. Si **showDeleted** está desactivado, los registros borrados no se cuentan.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* ejecuta un método personalizado si hay más de 10000 registros en PEDIDOS.DB; de lo contrario, el código muestra el número actual de registros de *Pedidos*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  PedidoTC TCursor
  NPedidos LongInt
endVar
if PedidoTC.open("Pedidos.db") then
  NPedidos = PedidoTC.nRecords()
  if NPedidos > 10000 then        ; si Pedidos tiene más de
10000 registros
    ArchivarAntiguosPedidos()   ; ejecutar un método definido
por el usuario
  else
    msgInfo("Estado", "La tabla Pedidos tiene"  +
String(NPedidos) + "registros.")
  endif
else
  msgStop("Lo siento", "No puedo abrir la tabla Pedidos.")
endif
endmethod
```

Vea también nFields
nKeyFields

open

Principiante

Método Abre una tabla.

Tipo TCursor

Sintaxis **1. open** (const ***nombreTabla*** String [, const ***db*** DataBase] [, const ***nombreIndice*** String]) Logical

2. open (const ***tableVar*** Table) Logical

Descripción Asocia un TCursor con la tabla mencionada en *nombreTabla*. Si se emplea la sintaxis 1, donde *nombreTabla* es un String, es posible emplear los argumentos *db* y *nombreIndice* para especificar una base de datos y un índice. Si se emplea la sintaxis 2, donde *varTabla* es el nombre de una variable Table, puede utilizarse el método **setIndex** del tipo Table para especificar un índice y puede especificarse la base de datos mediante el método **attach** del tipo Table.

Ejemplo El ejemplo siguiente utiliza la primera sintaxis para abrir un TCursor en la tabla *Clientes* de la base de datos TablasEjemplo . Este código emplea la cláusula optativa *nombreIndice*, por lo que el TCursor emplea el índice NombreYEstado . El código siguiente se anexa al método **pushButton** de *primerBotón*:

```
; primerBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc1  TCursor
  Ejemplo DataBase
endVar

; Creamos los alias TablasEjemplo para el directorio ejemplo
por defecto
addAlias("TablasEjemplo", "Standard", "c:\\pdoxwin\\sample")

; asociamos la variable de base de datos Ejemplo con la base de
datos
; TablasEjemplo
Ejemplo.open("TablasEjemplo")

; asociamos tc1 a la tabla Clientes en la base de datos
Ejemplo, y utilizamos
; el índice NombreYEstado
tc1.open("Clientes.db", Ejemplo, "NombreYEstado")

endmethod
```

El ejemplo siguiente logra el mismo resultado que el ejemplo anterior, pero utiliza la segunda forma de la sintaxis donde se utiliza una variable Table. El código siguiente se anexa al método **pushButton** de *segundoBotón*:

```
; segundoBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc1 TCursor
  Ejemplo DataBase
  Tabla Table
```

```
endVar

; Creamos los alias TablasEjemplo para el directorio ejemplo
por defecto
addAlias("TablasEjemplo", "Standard",  c:\\pdoxwin\\ejemplo )

; asociamos la variable de base de datos Ejemplo con la base de
datos
; TablasEjemplo
Ejemplo.open("TablasEjemplo")

; conectamos el manejador de Tabla con Clientes en la base de
datos Ejemplo
Tabla.attach("Clientes.db", Ejemplo)
; ajustamos el índice de Tabla al índice NombreYEstado
Tabla.setIndex("NombreYEstado")

; ahora asociamos el TCursor tc1 a la tabla Clientes en la base
de datos
; Ejemplo
tc1.open(Tabla)

endmethod
```

Vea también close

postRecord

Principiante

Método Almacena los cambios en un registro.

Tipo TCursor

Sintaxis **postRecord** ( ) Logical

Descripción Almacena los cambios en un registro inmediatamente. El registro permanece bloqueado. Si el registro está en una tabla indexada, se desplaza a su posición adecuada y continúa como registro actual. Este método devuelve True si es satisfactorio; en caso contrario, devuelve False..

Ejemplo En este ejemplo, el método **pushButton** del botón *fijarNombre* intenta encontrar un nombre con un error mecanográfico en la tabla *Clientes*. Si se encuentra el nombre erróneo, el código lo corrige y almacena los cambios mediante **postRecord**.

```
; fijarNombre::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombreErróneo String
  NombreCorrecto String
endVar
NombreErróneo = "Usco"
NombreCorrecto = "Unisco"

tc.open("Clientes.db")
if tc.locate("Nombre", NombreErróneo) then   ; si encontramos
el nombre erróneo
  tc.edit()                                  ; ponemos el
TCursor en modo Edición
  tc.Nombre = NombreCorrecto                 ; corregir el
nombre mal escrito
  if tc.postRecord() then                    ; Verdadero si el
registro se ha
                                             ; almacenado
    message("Cambios almacenados.")
  else                                       ; el registro no
se ha almacenado
                                             ; (¿violación de
clave?)
    msgStop("Almacenamiento del registro", "No se pueden
almacenar los cambios.")
  endIf
  tc.endEdit()                               ; finalizar el
modo Edición
  ; Si se actualiza el registro, endEdit simplemente finaliza
el modo
  ; Edición de el Nombre
  ; ahora el campo almacena "Unisco". Si el registro no ha sido
actualizado,
  ; el campo retiene su valor original ("Usco").

else
```

```
                                                    ; no se ha
encontrado "Usco" en el
                                                    ; campo Nombre
  message("No encuentro"  + NombreErróneo)
endif
endmethod
```

Vea también unLockRecord

priorRecord

Principiante

Método Se desplaza al registro anterior de una tabla.

Tipo TCursor

Sintaxis **priorRecord** ( ) Logical

Descripción Define el registro anterior de una tabla como registro actual. Si la tabla está en modo editar, **priorRecord** consigna los cambios en el registro actual antes de realizar el desplazamiento. Devuelve False si el TCursor ya se encuentra en el primer registro. Igualmente, el primer registro de la tabla se convierte en registro actual, y **bot** devuelve True.

priorRecord puede no ser apropiado en todas las bases de datos, porque tal vez algunas no sean bidireccionales. Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Ejemplo En este ejemplo, el método **pushButton** *mostrarClienteAnterior* utiliza **priorRecord** para desplazar un TCursor hacia atrás en la tabla *CLIENTES*. Cada vez que el TCursor llega a un nuevo registro, este código emplea **copyToArray** para copiar el contenido del registro a un DynArray y muestra valores de campo en un cuadro de diálogo. Cuando **priorRecord** intenta desplazarse más allá del principio de la tabla, **bot** devuelve True y el método **pushButton** termina.

```
; mostrarClienteAnterior::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Grabar DynArray[] AnyType
  NombreTabla String
endVar
NombreTabla = "Clientes.db"

if tc.open(NombreTabla) then

  tc.end()                      ; saltamos al final de la tabla
  while NOT tc.bot()            ; Verdadero hasta que
priorRecord intente
                                ; desplazar el puntero más allá
del principio
                                ; de la tabla
    tc.copyToArray(Grabar)      ; copiamos el registro al
DynArray Grabar
    Grabar.view("Registro"  + String(tc.recNo()))
    if msgQuestion("",
       "¿Desea ver el siguiente registro?") = "Yes" then
      tc.priorRecord()          ; nos desplazamos al registro
anterior
    else
      return
    endif
  endWhile

  msgStop("¡Eso es todo!", "No hay más registros.")
else
```

```
  msgStop("Lo siento", "No puedo abrir la tabla"  + NombreTabla
+ ".")
endif
endmethod
```

Vea también

home
end
nextRecord
skip
moveToRecord

qLocate

Método Busca un valor de campo especificado en una tabla indexada.

Tipo TCursor

Sintaxis **qLocate** (const ***valorBúsqueda*** AnyType [, const ***valorBúsqueda*** AnyType]*) Logical

Descripción Busca en una tabla indexada registros cuyos valores coincidan exactamente con los criterios especificados en *valorBúsqueda*. **qLocate** busca valores en el índice activo; el primer valor corresponde al primer campo del índice, el segundo valor al segundo campo del índice y así sucesivamente.

La búsqueda siempre comienza al principio de la tabla, pero si no se encuentra ninguna coincidencia, el TCursor vuelve al registro original. Si se encuentra alguna, el TCursor se desplaza a ese registro. Esta operación falla si el registro actual no puede consignarse o si el número de valores de búsqueda supera el número de campos del índice actual.

Ejemplo Este código utiliza **qLocate** para averiguar un valor de clave en la tabla *Articulo*:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar

if tc.open("articulo.db") then

  ; si qLocate no encuentra 1002 en el primer campo del índice
ni 1316
  ; en el segundo   if tc.qLocate(1002, 1316) then

    ; realizamos algunos cambios en el registro
    tc.edit()
    tc.Cant = 10
    tc.Total = tc."Precio de Venta" * tc.Cant
    tc.close()
  else
    msgStop("Lo siento", "No encuentro el registro
especificado.")
    endif
else
  msgStop("Error", "No puedo abrir articulo.db")
endif

endmethod
```

Vea también locate
locateNext
locateNextPattern
locatePattern
locatePrior
locatePriorPattern

recNo

Método Devuelve el número de registro del registro actual.

Tipo TCursor

Sintaxis **recNo** ( ) LongInt

Descripción Devuelve un entero que representa la posición del registro actual en la tabla. En una tabla de dBASE, **recNo** devuelve la posición física del registro en la tabla; en una tabla de Paradox indexada, devuelve su posición ordenada.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* busca en la tabla *Clientes* clientes que residen en Madrid. Si se encuentra alguno, este código almacena sus números de registro en una matriz y muestra el contenido de la matriz en un cuadro de diálogo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  matriz Array[] SmallInt
  NombreTabla String
endVar
NombreTabla = "Clientes.db"

tc.open(NombreTabla)
if tc.locate("Provincia", "Madrid") then
  matriz.addLast(tc.recNo())                  ; añadimos el
número del registro a la matriz
  while tc.locateNext("Provincia", "Madrid")  ; buscamos el
siguiente  Madrid
  matriz.addLast(tc.recNo())                  ; añadimos más
elementos a la matriz
  endWhile
  matriz.view("Números de los registros")     ; mostramos la
matriz matriz
else
  msgInfo("¡No hay nada que hacer!", "No encuentro \"Madrid\"
en el campo \"Provincia\"")
endif
endmethod
```

Vea también nRecords

recordStatus

Método Informa acerca del estado de un registro.

Tipo TCursor

Sintaxis **recordStatus** (const ***tipoEstado*** String) Logical

Descripción Devuelve True o False a una pregunta para informar sobre el estado de un registro. Utilice el argumento *tipoEstado* para especificar el estado en cuestión, donde *tipoEstado* es uno de los siguientes valores de String: New (nuevo), Locked (bloqueado) o Modified (modificado).

"New" implica que el registro se acaba de insertar en la tabla y aún no se ha almacenado. "Locked" significa que se ha aplicado un bloqueo (implícito o explícito) sobre el registro. "Modified" implica que se ha cambiado al menos uno de los valores de campo y aún no se ha almacenado.

Ejemplo Este ejemplo comprueba si el registro actual está bloqueado. Si el registro no está bloqueado, este método emplea **lockRecord** para bloquear el registro; si lo está, este ejemplo informa al usuario de que el registro se ha bloqueado previamente.

```
; bloquearEsteRegistro::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Pedidos.db")
tc.edit()

; si el registro actual no está bloqueado
if tc.recordStatus("Locked") = False then
  ; fijar el registro actual
  tc.lockRecord()

  ; si el registro está bloqueado, esta sentencia muestra True
  MsgInfo("Estado del registro",  recordStatus(\"Locked\") = "
+
          String(tc.recordStatus("Locked")))
else
  message("El registro actual ya está bloqueado.")
endIf

endmethod
```

Vea también lockRecord
unlockRecord

reIndex

Método Reconstruye los archivos de índice especificados.

Tipo TCursor

Sintaxis **reIndex** (const ***nombreIndice*** String [, const ***nombreEtiqueta*** String]) Logical

Descripción Reconstruye un índice (o etiqueta de índice) que no se mantiene automáticamente. Cuando trabaje con una tabla de Paradox, utilice *nombreIndice* para especificar un índice (el nombre del campo, para un índice de un solo campo, o el nombre completo de un índice compuesto). Cuando trabaje con una tabla de dBASE, utilice *nombreIndice* para especificar un archivo .NDX, o *nombreIndice* y *nombreEtiqueta* para especificar una etiqueta de índice de un archivo .MDX. Este método precisa el acceso exclusivo a la tabla.

Ejemplo El ejemplo siguiente abre un TCursor para *Clientes* (una tabla de Paradox), consigue el acceso exclusivo a la tabla y utiliza **reIndex** para reconstruir el índice *Teléfono.*

```
; reindexarClientes::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  TablaParadox String
  Tabla Table
endVar
TablaParadox = "Clientes.db"

Tabla.attach(TablaParadox)
Tabla.setExclusive(Yes)

if tc.open(Tabla) then
  tc.reIndex("Teléfono")          ; reconstruir el índice
Teléfono
  message("Teléfono reindexado.")
else
  msgStop("Lo siento", "No puedo abrir la tabla"  +
TablaParadox + ".")
endif

endmethod
```

Vea también reIndexAll

reIndexAll

Método Reconstruye todos los archivos de índice de una tabla.

Tipo TCursor

Sintaxis **reIndexAll** ( ) Logical

Descripción Reconstruye todos los índices de la tabla asociada con un TCursor. Este método precisa derechos exclusivos sobre la tabla para reconstruir un índice mantenido, y precisa un bloqueo de escritura para reconstruir un índice no mantenido. **reIndexAll** sólo funciona con tablas de Paradox, puesto que cualquier índice abierto para una tabla de dBASE se mantiene automáticamente.

Ejemplo En este ejemplo, el método **pushButton** de un botón intenta aplicar un bloqueo completo sobre la tabla *Clientes*. Si **lock** es satisfactorio, este código reconstruye todos los índices de la tabla *Clientes* y desbloquea la tabla.

```
; reindexarTodosLosClientes::pushButton
method pushButton(var eventInfo Event)
var
  tc          TCursor
  TablaParadox String
  Tabla        Table
endVar
TablaParadox = "Clientes.db"

Tabla.attach(TablaParadox)
Tabla.setExclusive(Yes)

if tc.open(Tabla) then
  tc.reIndexAll()               ; reconstruir todos los índices
de Clientes
  message("Indices reconstruidos.")
else
  msgStop("Lo siento", "No puedo abrir la tabla"  +
TablaParadox + ".")
endif
endmethod
```

Vea también reIndex

seqNo

Método Devuelve el número de registro del registro actual.

Tipo TCursor

Sintaxis **seqNo** ( ) LongInt

Descripción Devuelve un entero que representa la posición del registro actual en una tabla. En las tablas de dBASE, **seqNo** devuelve la posición secuencial de un registro, según lo ve el índice actual. En las tablas de Paradox, **seqNo** y **recNo** siempre devuelven el mismo valor.

Ejemplo En este ejemplo, supóngase que PUNTOS.DBF tiene tres registros y que se ha borrado el segundo. El código anexado al método **pushButton** de *esteBotón* demuestra la diferencia entre los métodos **seqNo** y **recNo**.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar

  ; Puntos.dbf tiene 3 registros y el segundo está borrado
tc.open("Puntos.dbf")

  ; no muestra los registros borrados
tc.showDeleted(No)

  ; ahora muestra recNo() = 1
  ;               seqNo() = 1
msgInfo("Estado de tc", "recNo() = " + String(tc.recNo()) + "\
n" +
                        "seqNo() = " + String(tc.seqNo()))

  ; saltamos al último registro de la tabla
tc.end()

  ; ahora muestra recNo() = 3
  ;               seqNo() = 2   (el registro n° 2 está borrado)
msgInfo("Estado de tc", "recNo() = " + String(tc.recNo()) + "\
n"+
                        "seqNo() = " + String(tc.seqNo()))

endmethod
```

Vea también moveToRecNo
moveToRecord
recNo

setFieldValue

Método Asigna un valor a un campo especificado.

Tipo TCursor

Sintaxis **setFieldValue** (const ***nombreCampo*** String, const ***valor*** AnyType) Logical

setFieldValue (const ***númeroCampo*** SmallInt, const ***valor*** AnyType) Logical

Descripción Define el valor de un campo (*nombreCampo* o *númeroCampo*) como *valor*. Este método devuelve True si es satisfactoria; de lo contrario, devuelve False.

Es posible obtener los mismos resultados mediante la notación de punto. Por ejemplo, esta sentencia utiliza la notación de punto para cambiar el valor del campo Ultima oferta:

```
tcVar."Ultima oferta" = 32,25
```

La sentencia siguiente emplea **setFieldValue** para lograr los mismos resultados:

```
tcVar.setFieldValue("Ultima oferta", 32.25)
```

Ejemplo En este ejemplo, el método **pushButton** de *corregirNombre* busca un nombre mal tecleado en el campo Nombre y utiliza **setFieldValue** para sustituir el nombre original.

```
; corregirNombre::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  NombreErróneo, NombreCorrecto String
endVar

NombreErróneo = "Usco"
NombreCorrecto = "Unisco"
tc.open("Clientes.db")
if tc.locate("Nombre", NombreErróneo) then
  tc.edit()
  tc.setFieldValue("Nombre", NombreCorrecto)  ; corregimos el
nombre erróneo
  tc.postRecord()                             ; almacenamos el
registro en la
                                              ; tabla
  tc.endEdit()                                ; finalizamos el
modo Edición
  message("Se ha reemplazado Usco por Unisco.")
else                                          ; no encontramos
"Usco" en el
                                              ; campo Nombre
  message("No encuentro"  + NombreErróneo)
endif
endmethod
```

Vea también fieldValue

setFilter

Método Define el rango de registros que un TCursor puede señalar.

Tipo TCursor

Sintaxis **setFilter** ([const ***valCoincidenciaExacta*** AnyType,] * const ***valMin*** AnyType, const ***valMax*** AnyType) Logical

Descripción Especifica las condiciones para incluir un rango de registros. Los registros que cumplen las condiciones se incluyen; los que no se excluyen. Esta operación falla si no puede consignarse el registro actual o si el TCursor no señala a una tabla con clave.

Este método compara los criterios especificados con los valores de los campos correspondientes del índice de una tabla. Para filtrar los registros por el valor de un solo campo, especifique valores en *valMin* y *valMax*. Por ejemplo, la sentencia siguiente comprueba los valores del primer campo del índice de cada registro. Si un valor es menor que 14 o mayor que 88, ese registro se excluye.

```
tcVar.setFilter(14, 88)
```

Para especificar una coincidencia exacta en un solo campo, asigne el mismo valor a *valMin* y *valMax*. Por ejemplo, la sentencia siguiente descarta todos los valores excepto 55:

```
tcVar.setFilter(55, 55)
```

Es posible filtrar registros por los valores de más de un campo. Para ello, especifique coincidencias exactas *excepto* para el último elemento de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas de Borland y Paradox (suponiendo que son los primeros campos del índice) y valores entre 100 y 500, ambos incluidos, en el tercer campo.

```
tcVar.setFilter("Borland", "Paradox", 100, 500)
```

La ejecución de **setFilter** sin argumentos redefine el filtro para incluir toda la tabla. Si no se ejecuta este método antes de abrir un TCursor pàra una tabla de dBASE, los registros borrados no se muestran.

Ejemplo En este ejemplo, supóngase que el primer campo de la clave de *Articulo* es Número de Pedido y que se desea saber el total del pedido número 1005. Cuando el usuario pulsa el botón *obtenerDetalles*, el método **pushButton** abre un TCursor para *Articulo* y limita el número de registros incluidos en el TCursor a los que contienen 1005 en el primer campo clave. Después de la ejecución de **setFilter**, este ejemplo emplea **cSum** para mostrar la suma del campo Total. Puesto que el TCursor sólo señala al pedido número 1005, **cSum** informa de la información de resumen sólo para ese pedido.

```
; obtenerDetalles::pushButton
method pushButton(var eventInfo Event)
var
  LineaTC TCursor
  NombreTabla String
endVar
NombreTabla = "articulo.db"
if LineaTC.open(NombreTabla) then
```

```
  ; limitamos lo que muestra TCursor a registros que tienen
  ; 1005 como valor clave (Número de Pedido 1005).
  LineaTC.setFilter(1005, 1005)

  ; ahora visualizamos el total del Número de Pedido 1005
  msgInfo("Total del Número de Pedido 1005",
LineaTC.cSum("Total"))
else
  msgStop("Lo siento", "No puedo abrir la tabla"  + NombreTabla
+ ".")
endif
endmethod
```

Vea también reIndex
reIndexAll
switchIndex

setFlyAwayControl

Método Controla si el TCursor señala a un registro cuya posición ha cambiado como resultado de un **unlockRecord**.

Tipo TCursor

Sintaxis **setFlyAwayControl** ([const ***yesNo*** Logical]) Logical

Descripción Especifica en *yesNo* si el TCursor permanecerá en el registro después de una llamada satisfactoria a **unlockRecord**.

Cuando se trabaja con tablas indexadas, los métodos **didFlyAway**, **setFlyAwayControl** y **unlockRecord** están muy relacionados. Cuando se ejecuta **unlockRecord**, el registro se consigna (si no existe ninguna violación de clave) en la tabla y se desplaza a su posición ordenada. Dependiendo de si el registro se ha desplazado a otra posición, es posible que el TCursor no continúe señalando al registro almacenado. Esta conducta se conoce como *vuelo de registro*.

Es posible utilizar el método **setFlyAwayControl** para controlar la conducta de **unlockRecord** y el vuelo de registros. Si el argumento optativo *yesNo* es Yes, **setFlyAwayControl** garantiza que el TCursor señalará al registro almacenado después de una llamada a **unlockRecord**; si es No, el TCursor señala al registro que sigue a la posición original del registro almacenado. Es posible emplear el método **didFlyAway** para comprobar si el registro voló en realidad.

Cuando **setFlyAwayControl** está definido como Yes, Paradox realiza una comprobación a nivel de registro de muchas operaciones a nivel de cursor. Puesto que este trabajo suplementario puede hacer más lenta la aplicación, debería utilizarse **setFlyAwayControl** con precaución. El método **postRecord** suele ser preferible a **setFlyAwayControl** y a **unlockRecord**. Para más información, consulte la entrada de postRecord en esta sección.

Ejemplo Consulte el ejemplo del método didFlyAway.

Vea también didFlyAway
postRecord
unLockRecord

setShowDeleted

Método Especifica si se muestran los registros borrados (sólo en tablas de dBASE).

Tipo TCursor

Sintaxis **setShowDeleted** ([***yesNo***]) Logical

Descripción Especifica si se muestran los registros borrados de una tabla de dBASE. Puede utilizarse *yesNo* para especificar Yes y mostrar los registros borrados, o No si no se desea mostrarlos. Si se omite este argumento, el valor es Yes por defecto.

Note: **setShowDeleted** sólo es válido para las tablas de dBASE.

Ejemplo

```
var
   dbfTable TCursor
endVar
if dbfTable.open("pedidos.dbf") then
   dbfTable.setShowDeleted(Yes)
endIf
```

Vea también open

showDeleted

Método Especifica si se muestran los registros borrados de una tabla de dBASE.

Tipo TCursor

Sintaxis **showDeleted** ([***yesNo***]) Logical

Descripción Especifica si se muestran los registros borrados de una tabla de dBASE. Puede utilizarse *yesNo* para especificar Yes y mostrar los registros borrados, o No si no se desea mostrarlos. Si se omite este argumento, el valor es Yes por defecto. **showDeleted** sólo es válido para las tablas de dBASE porque no es posible mostrar los registros borrados de una tabla de Paradox.

Ejemplo En este ejemplo, el método **pushButton** anexado a *mostrarRegistrosEliminados* ejecuta **showDeleted** para mostrar los registros borrados en PEDIDOS.DBF.

```
; mostrarRegistrosEliminados::pushButton
method pushButton(var eventInfo Event)
var
  fbdTC TCursor
endVar
if fbdTC.open("Pedidos.dbf") then
  fbdTC.showDeleted(Yes)
else
  msgStop("Lo siento", "No puedo abrir la tabla Pedidos.dbf.")
endIf
endmethod
```

Vea también compact
isRecordDeleted
isShowDeletedOn

skip

Método Avanza o retrocede un número especificado de registros en una tabla.

Tipo TCursor

Sintaxis **skip** ([const ***nRegistros*** LongInt]) Logical

Descripción Define como registro actual (y memoria intermedia de registro) el registro situado a *nRegistros* registros del actual. Si **skip** intenta ir más allá de los límites de la tabla, se obtiene un error, y el registro actual será el primer o último registro de la tabla, según corresponda. Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Los valores positivos de *nRegistros* avanzan en la tabla (**skip**(1) equivale a **nextRecord**), mientras que los negativos retroceden (**skip**(-1) equivale a **priorRecord**); un valor de 0 no produce ningún desplazamiento (**skip**(0) equivale a **currRecord**). Si se omite, *nRegistros* es 1 por defecto.

Ejemplo Este ejemplo demuestra cómo afecta **skip** a la posición de registro de un TCursor en una tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Pedidos.db")

tc.skip(5)     ; avanzar 5 registros. tc.recNo() = 6
tc.skip(-3)    ; retroceder 3 registros. tc.recNo() = 3
tc.skip(-5)    ; cometemos un fallo al intentar movernos antes
del principio
               ; de la tabla
               ; tc.recNo() = 1
               ; tc.bot() = True

endmethod
```

Vea también home
end
nextRecord
priorRecord
currRecord
moveToRecord

sortTo

Método Ordena una tabla.

Tipo TCursor

Sintaxis **1. sortTo** (const ***tablaDestino*** String, const ***númeroCampos*** SmallInt, const ***camposOrdenar*** Array[] String, const ***criterioOrdenación*** Array[] SmallInt) Logical

2. sortTo (const ***tablaDestino*** Table, const ***númeroCampos*** SmallInt, const ***camposOrdenar*** Array[] String, const ***criterioOrdenación*** Array) Logical

Descripción Ordena una tabla por los valores de los campos y sitúa los resultados en *tablaDestino*.

camposOrdenar es una matriz de cadenas o enteros que especifica los campos por los que se va a realizar la ordenación. El tamaño de la matriz *camposOrdenar* se especifica en *númeroCampos*. *criterioOrdenación* es una matriz de enteros, donde 0 especifica una ordenación ascendente y 1, una descendente. Las dos matrices deben tener el mismo tamaño, especificado en *númeroCampos*. El elemento 1 de *criterioOrdenación* especifica cómo ordenar el campo mencionado en el elemento 1 de *camposOrdenar*, y así sucesivamente.

Este método precisa al menos un bloqueo de sólo lectura sobre la tabla origen, y uno completo sobre la de destino. Si *tablaDestino* existe, se sustituye sin pedir confirmación. Si *tablaDestino* está abierta, este método falla. No es posible utilizar **sortTo** para ordenar una tabla y almacenar el resultado en ella misma.

Ejemplo Este ejemplo ordena la tabla *Clientes* en la tabla ORDCLIEN.DB y, a continuación, abre la tabla ordenada. Si no es posible aplicar un bloqueo de escritura sobre la tabla *Clientes*, este ejemplo informa al usuario del error y cancela la operación. Si la tabla destino *OrdClien* existe, se da al usuario una oportunidad de continuar o cancelar.

El código siguiente va en la ventana Var del botón *botónDeOrdenarClientes*:

```
; botónDeOrdenarClientes::var
var
  OrdenarCampos Array[2] String
  ModoOrdenar  Array[2]  SmallInt
  tc                     TCursor
  TablaOrig, TablaDest   String
  NoOrdenar              Logical
  TablaOrdenada          TableView
endVar
```

El código siguiente se anexa al método **open** del botón. Este código asigna un TCursor para la tabla *Clientes*, lo abre e inicializa los elementos de la matriz. Estas asignaciones determinan el criterio de ordenación para **sortTo**.

```
; botónDeOrdenarClientes::pushButton
method open(var eventInfo Event)
TablaOrig = "Clientes.db"
TablaDest = "OrdClien.db"
if tc.open(TablaOrig) then
  NoOrdenar = False                    ; bandera para el
método pushButton
```

```
  OrdenarCampos[1] = "Primer Contacto"  ; ordenamos por Primer
Contacto
  ModoOrdenar[1] = 0                    ; en orden ascendente
  OrdenarCampos[2] = "País"             ; después por país
  ModoOrdenar[2] = 0                    ; en orden descendente
else
  NoOrdenar = True
endif

endmethod
```

El código siguiente se anexa al método **pushButton** del botón *botónDeOrdenarClientes*. Cuando se pulsa el botón, el código intenta aplicar un bloqueo de escritura sobre la tabla origen (CLIENTES.DB), pregunta al usuario si existe la tabla destino (ORDCLIEN.DB) y ordena *Clientes* en *OrdClien* en función de los valores de las matrices *OrdenarCampos* y *ModoOrdenar*. Después de que se cree ORDCLIEN.DB (o se sustituya), este ejemplo la abre como un TableView.

```
; botónDeOrdenarClientes::pushButton
method pushButton(var eventInfo Event)
if NoOrdenar = False then
  if tc.lock("Write") then
    if isTable(TablaDest) then
      if msgQuestion("¿Sobreescribir?",
         "¿Desea reemplazar la tabla " + TablaDest + "?") =
"Yes" then
        msgInfo("Cancelado", "Operación cancelada.")
        return
      endif
    endif
    tc.sortTo(TablaDest, 2, OrdenarCampos, ModoOrdenar)
    TablaOrdenada.open(TablaDest)
  else
    msgStop("¡Atención!", "No puedo bloquear para escritura la
tabla"  +    TablaOrig + ".")
  endif
else
  msgStop("Lo siento", "No puedo abrir la tabla"  + TablaOrig +
".")
endif
endmethod
```

Vea también add
copy
subtract

subtract

Método Extrae los registros en una tabla de otra tabla.

Tipo TCursor

Sintaxis

1. subtract (const ***tablaDestino*** String) Logical

2. subtract (const ***tablaDestino*** Table) Logical

3. subtract (const ***tablaDestino*** TCursor) Logical

Descripción Comprueba si hay registros en la tabla origen que también se hallen en *tablaDestino*. Si es así, **subtract** los borra de *tablaDestino* sin pedir confirmación.

Si *tablaDestino* está indexada, **subtract** borra todos los registros con índices que coincidan exactamente con los valores de los campos de índice correspondientes en la tabla origen. Si *tablaDestino* no está indexada, **subtract** borra todos los registros que coincidan exactamente con cualquier registro de la tabla origen. Tanto si las tablas están indexadas como si no, este método sólo considera los campos que *podrían* tener clave. Por ejemplo, se consideran los campos numéricos, pero no los campos memo con formato. Este método precisa acceso de lectura/escritura a ambas tablas.

Nota: Si la tabla destino no está indexada, esta operación puede ser prolongada.

Ejemplo En este ejemplo, el método **pushButton** de *eliminarClientes* borra registros de la tabla *Clientes* que coincidan exactamente con los de la tabla *Respuest*.

```
; eliminarClientes::pushButton
method pushButton(var eventInfo Event)
var
  RespTC, ClienTC TCursor
endVar

if RespTC.open("Respuest.db") and
   ClienTC.open("Clientes.db") then

  RespTC.subtract(ClienTC) ; Eliminamos los registros de
Respuest que
                           ; coincidan con los de Clientes

else
  msgStop("¡Atención!", "No puedo abrir las tablas.")
endif

endmethod
```

Vea también add
copy

switchIndex

Principiante

Método Especifica otro índice que se utilizará para mostrar los registros de una tabla.

Tipo TCursor

Sintaxis **1. switchIndex** ([const ***nombreIndice*** String] [, const ***permanecerEnRegistro*** Logical]) Logical

2. switchIndex ([const ***nombreArchivoIndice*** String [, const ***nombreEtiqueta*** String]] [, const ***permanecerEnRegistro*** Logical]) Logical

Descripción Especifica en *nombreIndice* un archivo de índice para su uso con una tabla. En la sintaxis 1, *nombreIndice* indica un índice para su uso con una tabla de Paradox. La sintaxis 2 es para las tablas de dBASE, donde *nombreArchivoIndice* puede indicar un archivo .NDX o .MDX y el argumento optativo *nombreEtiqueta* especifica una etiqueta de índice de un archivo de índice de producción (.MDX).

En ambas sintaxis, si el argumento optativo permanecer *EnRegistro* es Yes, este método mantiene el registro actual después del cambio de índice; si es No, el registro actual cambia al primer registro de la tabla. El valor por defecto de este argumento es No.

Ejemplo En este ejemplo, supóngase que *Clientes* es una tabla de Paradox con clave que tiene un índice secundario llamado "NombreYEstado" . Este ejemplo abre un TCursor para *Clientes*, ejecuta **switchIndex** para cambiar del índice primario al índice "NombreYEstado" y muestra el primer valor del campo Nombre. Puesto que el TCursor se ordena por los campos Nombre y País en sentido ascendente, el primer valor mostrado es el primer nombre en orden ascendente.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar

tc.open("Clientes.db")                ; abrimos un TCursor para
Clientes
tc.switchindex("NombreYEstado")       ; cambiamos el índice
NombreYEstado
tc.home()                             ; nos aseguramos de que
estamos en el
                                      ; primer registro
msgInfo("Primer Registro", tc.Nombre) ; mostrar el valor del
campo Nombre

endmethod
```

Vea también reIndex
reIndexAll
Table::setIndex

tableName

Método Devuelve el nombre de la tabla asociada con un TCursor.

Tipo TCursor

Sintaxis **tableName** ( ) String

Descripción Devuelve el nombre de la tabla asociada con un TCursor. Este método es útil cuando se pasan variables al método **open** del TCursor.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* utiliza los métodos **findFirst** y **findNext** del tipo FileSystem para buscar en tablas de Paradox del directorio de trabajo actual. Este ejemplo busca en cada tabla un valor en el campo *Nombre* de la tabla actual. Este ejemplo abre todas las tablas del directorio actual que tienen "Unisco" en el campo "Nombre".

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  tc TCursor
  Tabla TableView
endVar
if fs.findFirst("*.db") then
  while fs.findNext()
    tc.open(fs.Name())                    ; abrimos un TCursor
para un fichero .db
    if tc.locate("Nombre", "Unisco") then ; si encontramos
Unisco en el campo
                                          ; Nombre
      tb.open(tc.tableName())             ; abrimos la tabla
asociada al TCursor
    endIf
    tc.close()
  endWhile
endIf

endmethod
```

Vea también tableRights

tableRights

Método Informa acerca de las operaciones que pueden realizarse en una tabla.

Tipo TCursor

Sintaxis **tableRights** (const ***derechos*** String) Logical

Descripción Informa de los derechos de un usuario sobre una tabla, donde *derechos* es uno de los siguientes:
ReadOnly (leer en la tabla, pero sin modificarla)

Modify (introducir o modificar datos)

Insert (añadir nuevos registros)

InsDel (añadir y borrar registros)

All (realizar todas las operaciones)

Ejemplo Este ejemplo informa de si el usuario tiene derechos InsDel sobre la tabla *Pedidos*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  MisDerechos Logical
  PedidosTC TCursor
endVar
PedidosTC.open("Pedidos.db")
PedidosTC.edit()
MisDerechos = PedidosTC.tableRights("InsDel")

  ; esto muestra True si usted tiene derechos de Inserción y
Borrado
  ; sobre Pedidos.db
msgInfo("¿Derechos para entrar?", MisDerechos)

endmethod
```

Vea también fieldRights

type

Método Devuelve el tipo de una tabla.

Tipo TCursor

Sintaxis **type** ( ) String

Descripción Devuelve una cadena que describe el tipo de una tabla: Paradox o dBASE.

Ejemplo Este ejemplo compacta (elimina registros borrados) de la tabla *Pedidos* si **type** devuelve dBASE; en caso contario, un mensaje indica que *Pedidos* es una tabla de Paradox.

```
; compactar::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar

tc.open("Pedidos.db")

; si Pedidos.db es una tabla de dBASE
if tc.type() = "dBASE" then
  ; eliminamos los registros borrados
  tc.compact()
else
  ; en otro caso, mostramos el tipo de tabla
  msgStop("¡Atención!", "Pedidos.db es una tabla de " +
tc.type() + ".")
endif

endmethod
```

Vea también isAssigned
tableName

unDeleteRecord

Principiante

Método Anula el borrado del registro actual de una tabla de dBASE.

Tipo TCursor

Sintaxis **unDeleteRecord** ( ) Logical

Descripción Recupera el registro borrado de una tabla de dBASE. Esta operación sólo puede ser satisfactoria si **showDeleted** se ha definido como True, el registro actual es un registro borrado y el TCursor está en modo editar.

Ejemplo Este ejemplo abre un TCursor para PUNTOS.DBF (una tabla de dBASE) y utiliza **showDeleted** para mostrar todos los registros borrados. Entonces, el código intenta encontrar un registro específico de la tabla. Este ejemplo usa **isRecordDeleted** para determinar si se ha borrado el registro; si es así, se recupera mediante **undeleteRecord**. El código siguiente se anexa al método **pushButton** de *esteBotón*:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
tc.open("Puntos.dbf")                  ; abrimos un TCursor
para una tabla de dBASE
tc.showDeleted()                       ; mostramos los
registros borrados
if tc.locate("Nombre", "Carmen") then  ; si locate encuentra
Carmen en el campo
                                       ; Nombre
  if tc.isRecordDeleted() then         ; si el registro está
borrado
    tc.edit()                          ; activamos el modo
Edición
    tc.undeleteRecord()                ; recuperamos el
registro
    message("He recuperado el registro Carmen")
  endif
else
  msgStop("Error", "No encuentro a Carmen.")
endif
endmethod
```

Vea también deleteRecord
isRecordDeleted
isShowDeletedOn
showDeleted

unlock

Principiante

Método Elimina los bloqueos especificados de un TCursor.

Tipo TCursor

Sintaxis **unlock** (const ***tipoBloqueo*** String) Logical

Descripción Intenta eliminar bloqueos aplicados explícitamente sobre la tabla señalada por un TCursor. *tipoBloqueo* debe ser una expresión que se evalúe en uno de los siguientes valores de String: Write (escritura), Read (lectura), Full (Completo) o Any (Cualquiera). Si lo consigue, este procedimiento devuelve True; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente abre un TCursor para *Clientes* (una tabla de Paradox), aplica un bloqueo completo sobre la tabla y utiliza **reIndex** para reconstruir el índice *Teléfono*. Una vez que se ha reconstruido el índice, este código desbloquea *Clientes* para que otros usuarios de una red puedan acceder a la tabla.

```
; reindexarClientes::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Tabla String
endVar
Tabla = "Clientes.db"

if tc.open(Tabla) then
  if tc.lock("Full") then   ; intentamos obtener el acceso
exclusivo
    tc.reIndex("Teléfono")  ; Reconstruimos el índice Teléfono
    tc.unLock("Full")       ; desbloqueamos la tabla
  else
    msgStop("Lo siento", "No puedo bloquear la tabla"  + Tabla
+ ".")
  endif
else
  msgStop("Lo siento", "No puedo abrir la tabla"  + Tabla +
".")
endif
endmethod
```

Vea también lockStatus
lock

unLockRecord

Principiante

Método Desbloquea el registro actual.

Tipo TCursor

Sintaxis **unLockRecord** ( ) Logical

Descripción Desbloquea el registro actual si está bloqueado. Si se intenta desbloquear un registro que no está bloqueado, se obtiene un error. Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Si la tabla está indexada, el registro se almacena en la tabla y se desplaza a su posición ordenada. Dependiendo de si el registro se ha desplazado a otra posición, es posible que el TCursor no continúe señalando al registro almacenado. Esta conducta se conoce como *vuelo de registro*.

Es posible utilizar el método **setFlyAwayControl** para controlar la conducta de **unlockRecord** y el vuelo de registros. Si **setFlyAwayControl** está definido como True, el TCursor continuará señalando al registro almacenado después de una llamada a **unlockRecord**. También es posible emplear el método **didFlyAway** para comprobar si el registro voló en realidad. Para más información sobre el vuelo de registros, consulte **didFlyAway** y **postRecord**.

Ejemplo En el ejemplo siguiente, el método **pushButton** de *esteBotón* intenta encontrar un valor mal tecleado en el campo *Nombre* de la tabla *Clientes*. Si se encuentra el valor, este código bloquea el registro, corrige el valor del campo y, entonces, desbloquea el registro mediante **unLockRecord**.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar
if tc.open("Clientes.db") then
  if tc.locate("Nombre", "Usco") then
    tc.edit()
    tc.lockRecord()       ; bloqueamos el registro actual
    tc.Nombre = "Unisco"  ; cambiamos el valor del campo
    tc.unlockRecord()     ; desbloqueamos el registro actual
    message("He cambiado el nombre por \"Unisco\"")
  else
    msgStop("Lo siento", "No encuentro \"Usco\" en el
campo \"Nombre\".")
  endif
else
  msgStop("Lo siento", "No puedo abrir la tabla Clientes.db.")
endif

endmethod
```

Vea también didFlyAway
lockRecord
postRecord
setFlyAwayControl

updateRecord

Principiante

Método Actualiza el registro existente con los datos del nuevo registro cuando existe una violación de tecla.

Tipo TCursor

Sintaxis **updateRecord** ([const ***irA*** Logical]) Logical

Descripción Sustituye el registro existente por los valores del nuevo registro no almacenado cuando existe una violación de clave. El registro se almacena en la tabla y no permanece bloqueado. Si el argumento optativo *irA* es True, el TCursor señalará al registro después de que se haya almacenado en la tabla; si es False, el TCursor señala al registro que sigue a la posición del registro original.

Ejemplo Consulte el ejemplo de attachToKeyViol

Vea también attachToKeyViol
lockRecord
postRecord

close

Método Cierra un vínculo DDE.

Tipo DDE

Sintaxis **close** ( )

Descripción Termina una conversación DDE. Cierra los documentos en la otra aplicación, pero ésta queda abierta.

Ejemplo El código siguiente abre la ficha LISTIN.OVD de ObjectVision, obtiene los valores de los campos Nombre y Empresa y ejecuta **close** para cerrar el vínculo DDE:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   ObtenNombres DDE
   Nombre, Empresa AnyType
endVar

; enlace con una Ficha de ObjectVision
ObtenNombres.open("Vision", "C:\\vision\\ejemplos\\Listin.ovd")

ObtenNombres.setItem("Nombre")   ; el ítem es el campo "Nombre"
Nombre = ObtenNombres            ; asignamos a Nombre el campo
"Nombre"
ObtenNombres.setItem("Empresa")  ; el ítem es el campo
"Empresa"
Empresa = ObtenNombres           ; asignamos a Empresa el campo
"Empresa"

msgInfo("De Listin.ovd",       ; mostramos información de
ObjectVision
         "Nombre  : " + Nombre + "\n" +
         "Empresa : " + Empresa )

ObtenNombres.close()           ; cerramos el enlace

endmethod
```

Vea también open
setItem

execute

Método Envía un comando a través de un vínculo DDE.

Tipo DDE

Sintaxis **execute** (const ***comando*** String)

Descripción Envía la cadena comando a una aplicación a través de un vínculo DDE. La naturaleza de comando variará de una aplicación a otra. Por ejemplo, una cadena que tiene sentido para un procesador de textos puede carecer de él para una hoja de cálculo, y las hojas de cálculo de diferentes fabricantes pueden utilizar comandos diferentes para realizar la misma operación.

Ejemplo El código siguiente utiliza la función @TITULO de ObjectVision para especificar el texto que se visualizará en la barra de título de ObjectVision.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   d1 DDE
endVar

; abrimos un enlace a una Ficha de ObjectVision llamado Listin
d1.open("Vision", "C:\\vision\\fichas\\Listin.ovd")

; ejecutamos el comando @TITULO de ObjectVision
d1.execute("[@TITULO(\"¡Estoy funcionando!\")]")

endmethod
```

Vea también setItem
open
close

open

Método Abre un vínculo DDE con otra aplicación.

Tipo DDE

Sintaxis
1. open (const ***servidor*** String) Logical
2. open (const ***servidor*** String, const ***tema*** String) Logical
3. open (const ***servidor*** String, const ***tema*** String, const ***elemento*** String) Logical

Descripción Crea un vínculo DDE con la aplicación *servidor* e indica a *servidor* que abra el documento *tema* (optativo) en una posición especificada en *elemento* (optativo).

Este método devuelve True si se logra abrir la a*plicación servidor*; en caso negativo, devuelve False. Si la aplicación servidora no puede abrir *tema*, o si no se encuentra *elemento* en *tema*, este método falla.

La naturaleza de *elemento* varía de una aplicación a otra. Por ejemplo, una cadena que tiene sentido para un procesador de textos puede carecer de él para una hoja de cálculo, y las hojas de cálculo de diferentes fabricantes pueden utilizar comandos distintos para realizar la misma operación.

Ejemplo El código siguiente abre dos vínculos DDE (representados por las variables DDE *d1* y *d2*) con la ficha LISTIN.OVD de ObjectVision.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   d1, d2 DDE
   Nombre, Empresa AnyType
endVar

d1.open("Vision", "C:\\vision\\fichas\\Listin.ovd", "Nombre")
d2.open("Vision", "C:\\vision\\fichas\\Listin.ovd", "Empresa")

Nombre = d1     ; el nombre toma el valor de "Nombre"
Empresa = d2    ; la Empresa toma el valor de "Empresa"

    ; mostramos información de los campos debjectVision
msgInfo("Listin.ovd Info",
       "Nombre : " + Nombre + "\n" +
       "Empresa : " + Empresa)

d1.close()                          ; cerramos los enlaces DDE
d2.close()

endmethod
```

Vea también close
setItem
execute

setItem

Método Especifica un elemento en una conversación DDE.

Tipo DDE

Sintaxis **setItem** (const ***servidor*** String)

Descripción Se utiliza en un vínculo DDE con una aplicación y tema establecidos. *servidor* especifica un nuevo elemento. La naturaleza de *servidor* varía de una aplicación a otra. Por ejemplo, una cadena que tiene sentido para un procesador de textos puede carecer de él para una hoja de cálculo, y las hojas de cálculo de diferentes fabricantes pueden utilizar comandos distintos para realizar la misma operación.

Ejemplo El código siguiente abre un vínculo DDE con la ficha LISTIN.OVD de ObjectVision, ejecuta **setItem** para realizar el vínculo primero con el campo Nombre y luego con el campo Empresa de la ficha ObjectVision.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   ObtenNombres DDE
   Nombre, Empresa AnyType
endVar

; enlace con una Ficha de ObjectVision
ObtenNombres.open("Vision", "C:\\vision\\ejemplos\\Listin.ovd")

ObtenNombres.setItem("Nombre")      ; el ítem es el campo
"Nombre"
Nombre = ObtenNombres               ; asignamos a Nombre el
campo "Nombre"

ObtenNombres.setItem("Empresa")     ; el ítem es el campo
"Empresa"
Empresa = ObtenNombres              ; asignamos a Empresa el
campo "Empresa"

msgInfo("De Listin.ovd",           ; mostramos información de
ObjectVision
          "Nombre : " + Nombre + "\n" +
          "Empresa : " + Empresa)

ObtenNombres.close()                ; cerramos el enlace

endmethod
```

Vea también execute
close

accessRights

Método Informa acerca de los derechos de acceso (también llamados atributos de archivo) de un archivo.

Tipo FileSystem

Sintaxis **accessRights** ( ) String

Descripción Devuelve una cadena que describe los derechos de acceso. Los valores devueltos pueden ser uno o más de los siguientes: A, D, H, R, S, V (para archivo, directorio, oculto, sólo lectura, sistema y volumen, respectivamente). Si el valor devuelto es una cadena vacía, el archivo no tiene atributos definidos. Es necesario utilizar **findFirst** antes de emplear **accessRights**.

Ejemplo Este ejemplo comprueba los atributos del archivo MEMO14.TXT. Ejecuta **findFirst** para comprobar que el archivo existe y, a continuación, ejecuta **accessRights**. Si el archivo no está marcado como de sólo lectura, este código ejecuta el Bloc de Notas de Windows para editar el archivo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   NombreArch String
   fs         FileSystem
endvar
NombreArch = "c:\\pdoxwin\\Archivos\\memo14.txt"

if fs.findFirst(NombreArch) then

   ; si los atributos del archivo incluyen R (sólo lectura)
   if search(fs.accessRights(), "R") > 0 then
     msgStop(NombreArch, "Este archivo está marcado como Sólo
para lectura.")
   else
    ; ejecutar el editor Bloc de Notas para el archivo
     execute("NotePad.exe " + NombreArch)
   endIf
else
   msgStop("Error", "No encuentro el archivo " + NombreArch)
endIf
endmethod
```

Vea también getFileAccessRights
setFileAccessRights

copy

Método Copia un archivo.

Tipo FileSystem

Sintaxis **copy** (const ***nombreOrigen***, String ***nombreDestino*** String) Logical

Descripción Devuelve True si **copy** ha logrado copiar el archivo origen *nombreOrigen* al archivo destino *nombreDestino*; si no lo logra, devuelve False. Si *nombreDestino* existe, **copy** lo sustituye sin pedir confirmación. **copy** sólo puede copiar un archivo cada vez. Los caracteres comodín del DOS no pueden utilizarse con **copy**.

Ejemplo El código de este ejemplo busca en el directorio actual el archivo especificado en la variable *Origen*. Si existe el archivo, este código lo copia y da al nuevo archivo el nombre especificado en *Destino*.

```
; BotónDeCopia::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   Origen, Destino String
endvar

Origen = "memo14.txt"
Destino = "memo14.bak"

if fs.findFirst(Origen) then
   if fs.copy(Origen, Destino) then
      message("He copiado el archivo " + Origen +  "en"  +
Destino)
   else
      message("Fallo en la copia...")
   endif
else
   msgInfo(Origen, "No encuentro el archivo.")
endIf
endmethod
```

Vea también rename
delete

delete

Método Borra un archivo.

Tipo FileSystem

Sintaxis **delete** (const ***nombre*** String) Logical

Descripción Devuelve True si **delete** logra borrar el archivo especificado en *nombre*; en caso contrario, devuelve False. **delete** sólo puede borrar un archivo cada vez. Pueden utilizarse los caracteres comodín del DOS, pero **delete** sólo borrará el primer archivo que encuentre que corresponda con la especificación de archivos.

Ejemplo El primer ejemplo muestra un cuadro de diálogo que pregunta al usuario si desea borrar el archivo especificado en la variable *NombreArch*. Si el usuario elige Yes, la ejecución de **delete** borra el archivo:

```
; EliminarUno::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   NombreArch String
endvar

NombreArch = "Texto.ant"

if fs.findFirst(NombreArch) then
   if msgYesNoCancel("¿Eliminar?", Nombrefic) = "Yes" then
     fs.delete(NombreArch)
   endIf
else
   msgInfo(NombreArch, "No encuentro el archivo.")
endIf
endmethod
```

El ejemplo siguiente utiliza un bucle **while** para borrar todos los archivos del directorio actual que tengan la extensión .OLD.

```
; EliminarTodos::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endVar

if fs.findFirst("*.ant") then
   fs.delete(fs.name())
   while fs.findNext()
     fs.delete(fs.name())
   endWhile
else
   msgInfo("*.OLD", "No encuentro ningún archivo.")
endIf
endmethod
```

Vea también deleteDir

deleteDir

Método Borra un directorio.

Tipo FileSystem

Sintaxis **deleteDir** (const ***nombre*** String) Logical

Descripción Devuelve True si **deleteDir** logra borrar el directorio especificado en *nombre*; en caso contrario, devuelve False. **deleteDir** no solicita confirmación.

Ejemplo El primer ejemplo intenta borrar el directorio C:\DOS. Si la operación falla (por ejemplo, porque el directorio no está vacío), un cuadro de diálogo muestra un mensaje de error.

```
; EliminarDOS::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endvar

if fs.findFirst("c:\\dos") then
   if not fs.deleteDir("c:\\dos") then
      msgStop("Error", "No puedo eliminar un directorio.")
   endIf
endIf
endmethod
```

Un directorio sólo puede borrarse si está vacío (no contiene archivos). Los ejemplos siguientes utilizan **enumFileList** para averiguar si un directorio está vacío. Como muestra el ejemplo, la forma de utilizar **enumFileList** depende de la vía de acceso al directorio y la especificación de archivos. En el primer ejemplo que sigue, **enumFileList** crea una matriz que contiene un elemento (el nombre de directorio) cuando el directorio está vacío:

```
; EliminarDir1::pushButton
method pushButton(var eventInfo event)
var
   fs FileSystem
   NombresArch Array[] String
endvar

fs.enumFileList("c:\\scan\\subscan", Nombresarch)

; comparamos el tamaño con 1 porque no se especifica nada
detrás del directorio     if NombresArch.size() = 1 then
   fs.deleteDir("c:\\buscar\\busqueda")
else
   msgStop("Alto", "El directorio no está vacío.")
endIf
endMethod
```

En el segundo ejemplo, **enumFileList** crea una matriz que contiene dos elementos (uno para el directorio actual y otro para su directorio superior), por la especificación de archivos ***.*** situada al final de la vía de acceso.

```
; EliminarDir2::pushButton
```

```
method pushButton(var eventInfo event)
var
   fs FileSystem
   NombresArch Array[] String
endvar
fs.enumFileList("c:\\buscar\\busqueda\\*.*", NombresArch)

; comparamos el tamaño con 2 porque se ha especificado *.*
detrás del directorio
if NombresArch.size() = 2 then ; tamaño = 2 porque se
especificó *.*
   fs.deleteDir("c:\\buscar\\busqueda")
else
   msgStop("Alto", "El directorio no está vacío.")
endIf
endmethod
```

Vea también delete
makeDir

drives

Método Devuelve las letras de las unidades disponibles en el sistema y conocidas por Windows.

Tipo FileSystem

Sintaxis **drives** ( ) String

Descripción Devuelve una cadena que contiene las letras (sin el signo de dos puntos) de las unidades disponibles en el sistema y conocidas por Windows.

Ejemplo El ejemplo siguiente muestra un cuadro de diálogo que enumera las letras de identificación de las unidades disponibles en el sistema:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endvar
; esto muestra una lista de las unidades de disco asociadas
; por ejemplo:    ABCHJKXY
msgInfo("Unidades de disco", fs.drives())
endmethod
```

Vea también existDrive

enumFileList

Método Escribe información sobre archivos en una tabla o una matriz.

Tipo FileSystem

Sintaxis **1. enumFileList** (const ***especArchivos*** String, var ***nombreMatriz*** Array[] String)

2. enumFileList (const ***especArchivos*** String, const ***nombreTabla*** String)

Descripción Escribe información sobre los archivos que cumplen los criterios de *especArchivos* en la matriz nombrada en *nombreMatriz* (sintaxis 1) o en la tabla nombrada en *nombreTabla* (sintaxis 2).

Si *especArchivos* es *.*, la matriz o tabla incluye registros para el directorio actual (.) y el directorio superior (..).

La matriz nombrada en *nombreMatriz* debe declararse antes de ejecutar este método. La matriz resultante sólo contiene nombres y extensiones de archivo.

Si no existe *nombreTabla*, se crea automáticamente. Si *nombreTabla* se crea en el directorio que se está listando, la tabla no aparece en la lista. Si existe *nombreTabla*, la información se añade en ella y su nombre de archivo aparece en la lista.

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
Name	A	20
Size	N	
Attributes	A	10
Date	A	10
Time	A	10

Los nombres de archivo se enumeran en el orden en que se hallan en el directorio no necesariamente en orden alfabético .

Ejemplo El ejemplo siguiente muestra el uso de ambas sintaxis de **enumFileList**. En primer lugar, **enumFileList** busca fichas en el directorio especificado y utiliza la sintaxis 1 para crear una matriz de nombres de archivo, que se muestra en un menú emergente. A continuación, **enumFileList** emplea la sintaxis 2 para crear una tabla de información sobre los archivos y la muestra en una ventana de tabla.

```
; BotónDeDemostración::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   Directorio, Ficha String
   NombresFicha Array[] String
   tv tableView
   p PopUpMenu
endvar

Directorio = "C:\\pdoxwin\\ejemplo\\*.fsl"

if fs.findFirst(Directorio) then  ; si se encuentra un *.f?l
```

```
   fs.enumFileList(Directorio, NombresFicha)  ; creamos una
matriz de archivos *.f?l
   p.addarray(NombresFicha)       ; mostramos la matriz en un
menú desplegable
     Ficha = p.show()       ; mostramos el menú desplegable de
archivos
endIf

if fs.findFirst(Directorio) then  ; si se encuentra un *.f?l
   fs.enumFileList(Directorio, "Fichas.db") ; creamos Fichas.DB
con una lista
    ; de archivos  *.f?l
   tv.open("Fichas.db")     ; mostramos la tabla Fichas.DB
endIf

endmethod
```

Vea también getValidFileExtensions
isFile

existDrive

Método Informa de si una unidad está disponible en el sistema.

Tipo FileSystem

Sintaxis **existDrive** (const ***letraUnidad*** String) Logical

Descripción Devuelve True si *letraUnidad* está disponible en el sistema; de lo contrario, devuelve False. *letraUnidad* puede especificarse utilizando una letra (C) o una letra y dos puntos (C:).

Ejemplo Este ejemplo ejecuta **existDrive** para comprobar la existencia de la unidad P. Si existe, la ejecución de **setDrive** la convierte en la unidad por defecto.

```
; ComprobarUnidad::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   NombreUnidad String
endvar

NombreUnidad = "P"

if fs.existDrive(NombreUnidad) then
   fs.setDrive(NombreUnidad)
else
   msgStop("Alto", "La Unidad"  + NombreUnidad +  "no está
asociada.")
endIfend
method
```

Vea también drives
setDrive
getDrive
isRemote
isRemovable

findFirst

Método Busca un nombre de archivo en un sistema de archivos.

Tipo FileSystem

Sintaxis **findFirst** (const ***patrón*** String) Logical

Descripción Devuelve True si se encuentra un archivo cuyo nombre coincida con **patrón**; en caso contrario, devuelve False. *patrón* puede contener los caracteres comodín del DOS * y ?, como se utilizan en el comando DIR del DOS. Ejemplos de **patrón** incluyen:

"C:*.*"

"..\\miDir*.*"

"*.txt"

"fr*.db?"

Utilice **findFirst** para averiguar si existe un archivo o directorio y para inicializar una variable FileSystem antes de ejecutar otro método o procedimiento de FileSystem.

Nota: findFirst encuentra los nombres de archivo y directorio en el orden en que se hallan en el directorio, que no es necesariamente alfabético. El primer valor devuelto por **findFirst** depende de la especificación de vía de acceso y archivo.

Ejemplo Este ejemplo demuestra cómo se comporta **findFirst** dependiendo de la especificación de archivos de *patrón*:

```
; botónUno::pushButton
method pushButton(var eventInfo Event)

var
   fs FileSystem
endVar

; buscamos un archivo en el directorio raíz
; o el directorio PDOXWIN
fs.findFirst("c:\\pdoxwin")

; esto muestra PDOXWIN (findFirst encuentra el directorio)
msgInfo("Comprobación de findFirst", fs.name())

; buscamos un archivo en el directorio raíz
; o el directorio PDOXWIN
fs.findFirst("c:\\pdoxwin\\")

; esto muestra PDOXWIN (findFirst encuentra el directorio)
msgInfo("Comprobación de findFirst", fs.name())

; buscamos un archivo en el directorio raíz
; o el directorio PDOXWIN
fs.findFirst("c:\\pdoxwin\\*.*")

; esto muestra un punto (.) porque el
; primer archivo de un directorio es un sólo punto (.)
msgInfo("Comprobación de findFirst", fs.name())
endmethod
```

Vea también findNext
name

findNext

Método Busca varias apariciones de un nombre de archivo en un sistema de archivos.

Tipo FileSystem

Sintaxis **findNext** ([const ***especArchivos*** String]) Logical

Descripción Después de que **findFirst** sea satisfactorio, **findNext** busca el archivo siguiente cuyo nombre coincida con el patrón. **findNext** devuelve True si es satisfactorio; en caso contrario, devuelve False.

Como forma abreviada, es posible utilizar el argumento optativo *especArchivos* para indicar una especificación de vía de acceso y archivos. Si lo emplea, la llamada a **findFirst** es innecesaria.

Ejemplo El primer ejemplo ejecuta **findFirst** y **findNext** para rellenar una lista con los nombres de las tablas existentes en el directorio actual. El ejemplo supone que ya se ha situado un objeto de lista desplegable en la ficha:

```
; ListarRelleno::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endvar

if fs.findFirst("c:\\pdoxwin\\ejemplo\\*.db") then
   ; iniciar el listado en el cuadro de edición
   fileListField.fileList.list.count = 0

   ; este bucle while rellena la lista del cuadro de edición
   ; con archivos *.db en el directorio ejemplo por defecto
   while fs.findNext()
     fileListField.fileList.list.selection =
     fileListField.fileList.list.selection + 1
     fileListField.fileList.list.value = fs.name()
   endWhile
else
   msgStop( *.db , "No encuentro ningún archivo.")
endIf
endmethod
```

El ejemplo siguiente utiliza **findNext** con una especificación de archivos como argumento, y muestra un menú emergente que lista los archivos existentes en el directorio C:\PDOXWIN:

```
; editarTexto::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   p  PopUpMenu
   Selección String
endvar

; buscamos archivos *.txt en el directorio PDOXWIN
; después, añadimos sus nombres a un menú desplegable
while fs.findNext("c:\\pdoxwin\\*.txt")
```

```
   p.addText(fs.name())
endWhile

Selección = p.show()  ; mostramos el menú desplegable
if not Selección.isBlank() then   ; si el usuario selecciona un
archivo
   execute("Notepad.exe " + Selección) ; editamos el archivo en
el Bloc de Notas
endif
endmethod
```

Vea también findFirst

freeDiskSpace

Método Devuelve la cantidad de espacio libre en una unidad.

Tipo FileSystem

Sintaxis **freeDiskSpace** (const ***letraUnidad*** String) LongInt

Descripción Devuelve el número de bytes disponibles en la unidad *letraUnidad*, que puede especificarse utilizando una letra (C) o una letra y dos puntos (C:).

Ejemplo El primer ejemplo muestra un cuadro de diálogo con el número de bytes disponibles en la unidad C:

```
; mostrarEspacioDeC::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endvar

msgInfo("Bytes libres en la unidad C:", fs.freeDiskSpace("C"))
endmethod
```

El ejemplo siguiente compara el tamaño del archivo MEMO14.TXT y la cantidad de espacio disponible en la unidad actual. Si hay espacio suficiente, el código ejecuta **copy** para copiar el archivo.

```
; copiarArchivo::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endvar

; si existe MEMO14.TXT en el directorio actual
if fs.findFirst("memo14.txt") then

   ; si hay suficiente espacio en el disco para contener otra
copia de MEMO14.TXT
   if fs.size()  fs.freeDiskSpace(fs.getDrive()) then

    ; copiamos el archivo
    fs.copy("memo14.txt", "memo14.bak")

   else
     msgStop("Copia", "No hay espacio suficiente en el disco
para copiar el archivo.")
   endIf
else
   msgStop("MEMO14.TXT", "No encuentro el archivo.")
endIf
endmethod
```

Vea también totalDiskSpace
findFirst
getDrive

fullName

Método Devuelve la vía de acceso completa de un archivo.

Tipo FileSystem

Sintaxis **fullName** ( ) String

Descripción Después de un **findFirst** o findNext satisfactorio, **fullName** devuelve la vía de acceso completa del archivo encontrado. Utilice este método con **splistFullFileName** para analizar los componentes de un nombre de archivo.

Ejemplo Este ejemplo ejecuta **fullName** para obtener el nombre completo de la primera ficha enumerada en el directorio actual. A continuación, ejecuta **splitFullFileName** para dividir el nombre en sus componentes y almacenarlos en un DynArray. Entonces, ejecuta **view** para mostrar el DynArray.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   Nombres DynArray[] String
   NombreCompleto String
endVar

; si el archivo clientes.db está en el directorio ejemplos
if fs.findFirst("c:\\pdoxwin\\ejemplos\\clientes.db") then

   ; almacenamos el nombre del archivo completo en una variable
     NombreCompleto = fs.fullName()

    ; dividimos el nombre del archivo en partes y almacenarlas
en un DynArray
    splitFullFileName(NombreCompleto, Nombres)

   ; mostramos las partes que componen el archivo
   splitName.view("Nombre dividido")
endIf
endmethod
```

Vea también name
findFirst
findNext
splitFullFileName

getDir

Método Devuelve la vía de acceso al directorio que señala la variable FileSystem.

Tipo FileSystem

Sintaxis **getDir** ( ) String

Descripción Devuelve una cadena que representa la vía de acceso al directorio indicado por la variable FileSystem. La cadena no incluye la letra de unidad utilice **getDrive** para ello .

Ejemplo Este ejemplo copia la ficha DIVEPLAN.FSL del directorio especificado en *DirFuente* al indicado en *DirDestino* y utiliza **getDir** para extraer el directorio de *DirDestino*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs1, fs2 FileSystem
   FichOrigen, DirDestino String
endvar

FichOrigen = "c:\\pdoxwin\\diveplan\\diveplan.fsl"
DirDestino = "c:\\pdoxwin\\Fichas\\*.*"
if fs1.findFirst(FichOrigen) then      ; si el archivo origen
existe
   if fs2.findFirst(DirDestino) then   ; si el directorio
destino existe
     fs1.copy(FichOrigen, fs2.getDir()) ; copiamos el archivo en
el
                ; directorio destino
   else
     msgStop(DirDestino, "No encuentro el directorio.")
   endIf
else
   msgStop(FichOrigen, "No encuentro el archivo.")
endIf
endmethod
```

Vea también setDir
getDrive

getDrive

Método Devuelve la letra de la unidad señalada por la variable FileSystem.

Tipo FileSystem

Sintaxis **getDrive** ( ) String

Descripción Devuelve una cadena que representa la letra de la unidad señalada por la variable FileSystem.

Ejemplo Este ejemplo ejecuta **getDrive** para devolver el alias del directorio de trabajo. Entonces, define H: como unidad por defecto y ejecuta **getDrive** de nuevo para confirmar el cambio.

```
; asignarH::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   NuevaUnidad String
endvar
msgInfo("Unidad por defecto", fs.getDrive())  ; Muestra :TRAB:
NuevaUnidad = "H"
if fs.existDrive(NuevaUnidad) then
   if fs.setDrive(NuevaUnidad) then
      msgInfo("Unidad por defecto", fs.getDrive())  ; Muestra H:
   else
     msgStop(NuevaUnidad, "No puedo asignar la unidad.")
   endIf
else
   msgStop(NuevaUnidad, "La unidad no está conectada.")
endIf
endmethod
```

Vea también existDrive
getDir
setDrive

getFileAccessRights

Procedimiento Informa de los derechos de acceso (también llamados atributos de archivo) de un archivo.

Tipo FileSystem

Sintaxis **getFileAccessRights** (const ***nombreArchivo*** String) String

Descripción Devuelve una cadena que describe los derechos de acceso de un archivo. Los valores devueltos pueden ser uno o más de los siguientes: A, D, H, R, S, V (para archivo, directorio, oculto, sólo lectura, sistema y volumen, respectivamente). Si el valor devuelto es una cadena vacía, el archivo no tiene atributos definidos. Este procedimiento es similar al método **accessRights**. Sin embargo, **getFileAccessRights** no precisa que se ejecute el método **findFirst** con anterioridad.

Ejemplo Este ejemplo muestra en un cuadro de diálogo los atributos de archivo de C:\ CONFIG.SYS.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   NombreArchivo String
endvar

NombreArchivo = "C:\\CONFIG.SYS"

msgInfo(NombreArchivo, getFileAccessRights(NombreArchivo))

endmethod
```

Vea también accessRights
setFileAccessRights

getValidFileExtensions

Procedimiento Devuelve las extensiones de archivo válidas para un objeto especificado.

Tipo FileSystem

Sintaxis **getValidFileExtensions** (const ***tipoObjeto*** String) String

Descripción Devuelve una cadena que contiene las extensiones de archivo válidas para el objeto especificado en tipoObjeto, donde tipoObjeto es uno de los siguientes: Form, Library, MailMerge, Report o Script.

Ejemplo Este ejemplo muestra un cuadro de diálogo con las extensiones de archivo válidas para las fichas:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Cadena String
endVar

Cadena = getValidFileExtensions("Form")
msgInfo("Extensiones de archivos de Ficha:", Cadena)     ;
muestra fsl fdl
endmethod
```

Vea también getDir
getDrive
splitFullFileName

isDir

Procedimiento Informa de si una cadena especificada representa el nombre de un directorio.

Tipo FileSystem

Sintaxis **isDir** (const ***nombreDir*** String) Logical

Descripción Devuelve True si *nombreDir* es un nombre de directorio válido; en caso contrario, devuelve False.

Ejemplo Este ejemplo ejecuta **isDir** para cerciorarse de que el directorio especificado por la variable *Directorio* es válido. En caso afirmativo, la ejecución de **setDir** lo convierte en directorio por defecto. En este ejemplo, el valor de *Directorio* se proporciona entre comillas, pero en la práctica podría suministrarlo el usuario, leerse de una tabla, etc.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
    fs FileSystem
    Directorio String
endvar

Directorio = "C:\\pdoxwin\\vacación"
if isDir(Directorio) then
   fs.setDir(Directorio)
   msgInfo("Directorio actual", fs.getDir())
else
   msgStop(Directorio, "Ese directorio no existe.")
endIf
endmethod
```

Vea también deleteDir
getDir
makeDir
setDir

isFile

Procedimiento Informa de si una cadena especificada es el nombre un archivo del sistema de archivos actual.

Tipo FileSystem

Sintaxis **isFile** (const ***nombreArchivo***) Logical

Descripción Devuelve True si **nombreArchivo** es un archivo existente en el sistema de archivos actual; de lo contrario, devuelve False.

Ejemplo El primer ejemplo ejecuta **isFile** y muestra mensajes que informan de si las especificaciones de archivo representan a archivos reales.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endvar

message(isFile("c:\\dos\\chkdsk.exe" )) ; muestra True
sleep(1500)
message(isFile("c:\\dos\\xxxx.xxx"))   ; muestra False
sleep(1500)

endmethod
```

El segundo ejemplo solicita al usuario que introduzca la vía de acceso y nombre completos de un archivo que desea borrar. Una llamada a **isFile** comprueba si el archivo existe y, si es así, una llamada a **delete** lo borra.

```
; botónUno::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   NombreArch String
endvar

NombreArch = "Escriba aquí la vía de acceso y el archivo."
NombreArch.view("Eliminar archivo")

if isFile(NombreArch) then  ; si el archivo especificado existe
   fs.delete(NombreArch)    ; eliminamos el archivo
   message("He eliminado el archivo.")
else
   msgStop(NombreArch, "No encuentro el archivo.")
endIf

endmethod
```

Vea también isDir

isFixed

Método Informa de si una unidad es fija.

Tipo FileSystem

Sintaxis **isFixed** (const ***letraUnidad*** String) Logical

Descripción Devuelve True si *letraUnidad* representa a una unidad fija (no extraíble ni de red); en caso contrario, devuelve False. *letraUnidad* puede especificarse utilizando una letra (C) o una letra y dos puntos (C:).

Ejemplo En el ejemplo siguiente, la unidad C es el disco duro local del usuario y la unidad H es una unidad de red:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
endVar

msgInfo("¿Es fija la unidad C?", fs.isFixed("C"))  ; muestra
True     msgInfo("¿Es fija la unidad H?", fs.isFixed("H"))  ;
muestra False
endmethod
```

Vea también isRemote
isRemovable

isRemote

Método Informa de si una unidad es remota (una unidad de red).

Tipo FileSystem

Sintaxis **isRemote** (const ***letraUnidad*** String) Logical

Descripción Devuelve True si *letraUnidad* representa a unidad remota (de red); en caso contrario, devuelve False. *letraUnidad* puede especificarse utilizando una letra (C) o una letra y dos puntos (C:).

Ejemplo En este ejemplo, H es una unidad de red (remota). Este código ejecuta **existDrive** para cerciorarse de que la unidad H está disponible; a continuación, ejecuta **isRemote** para averiguar si es una unidad de red.

```
var
  h FileSystem
endVar
if h.existDrive("h") then ; si la unidad H está conectada
   if h.isRemote() then
      msgInfo("Unidad H:", "Unidad remota")
   else
      msgInfo("Unidad H:", "No es una Unidad remota.")
   endIf
else
   msgStop("Unidad H:", "La unidad no está conectada.")
endIf
```

Vea también isFixed
isRemovable

isRemovable

Método Informa de si una unidad es extraíble.

Tipo FileSystem

Sintaxis **isRemovable** (const ***unidadLetra*** String) Logical

Descripción Devuelve True si *unidadLetra* representa a una unidad extraíble; en caso contrario, devuelve False. *unidadLetra* puede especificarse con una letra (C) o con una letra y dos puntos (C:).

Ejemplo En este ejemplo, D es una unidad extraíble. Este código ejecuta **existDrive** para cerciorarse de que la unidad D está disponible y, entonces, ejecuta **isRemovable** para averiguar si es una unidad extraíble.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   c  String
endVar

if fs.existDrive("D:") then ; si la unidad D está conectada
   if fs.isRemovable("D") then
      msgInfo("Unidad D:", "Unidad extraíble")
   else
      msgInfo("Unidad D:", "No es una unidad extraíble.")
   endIf
endIf

endmethod
```

Vea también isFixed
isRemote

makeDir

Método Crea un nuevo directorio.

Tipo FileSystem

Sintaxis **makeDir** (const ***nombre*** String) Logical

Descripción Crea todos los directorios y subdirectorios especificados en *nombre*. Este método devuelve True si lograr crear *nombre*; en caso contrario, devuelve False. Este método también devuelve True si el directorio ya existe.

Ejemplo Este código intenta crear un nuevo directorio en la unidad M. El ejemplo muestra un cuadro de diálogo que informa de si pudo hacerlo o no.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
endVar

     ; esto crea \Nuevo, \Nuevo\Director etc...
l = fs.makeDir("c:\\Nuevo\\Director\\Arbol")
msgInfo("Estado", iif(l, "He creado el nuevo directorio",
"Fallo en makeDir"))
endmethod
```

Vea también deleteDir
isDir

name

Método Devuelve el nombre de un archivo.

Tipo FileSystem

Sintaxis **name** ( ) String

Descripción Después de un **findFirst** o **findNext** satisfactorio, name devuelve el nombre del archivo cuyo nombre coincide con el patrón.

Ejemplo Este ejemplo ejecuta **findFirst** y **findNext** para buscar las tablas del directorio actual y, a continuación, ejecuta **name** para crear un menú emergente con sus nombres de archivo..

```
; MostrarNombre::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   p PopUpMenu
   tv TableView
   Selección String
endvar

if fs.findFirst("*.db") then      ; si existe un archivo *.db
   p.addStaticText("Tablas")      ; creamos un menú desplegable
   p.addSeparator()
   p.addText(fs.name())     ; utilizamos los nombres de los
archivos
    ; en el menú desplegable
   while fs.findNext()
       p.addText(fs.name())
   endWhile
   Selección = p.show()     ; mostramos el menú
   if not Selección.isBlank() then      ; si el usuario
selecciona una tabla
     tv.open(Selección)     ; mostramos la tabla seleccionada
   endif
endIf

endmethod
```

Vea también fullName
findFirst
findNext

privDir

Procedimiento Devuelve el nombre del directorio personal del usuario.

Tipo FileSystem

Sintaxis **privDir** () String

Descripción Devuelve una cadena que contiene la vía de acceso completa del DOS (incluyendo la letra de la unidad) al directorio personal del usuario.

Cada usuario debe tener un directorio personal en que se almacenan las tablas temporales. Puede estar en una unidad de red o en una local. Puede utilizar **setAliasPath** definido por Session para especificar la vía de acceso a su directorio personal.

Ejemplo Este ejemplo ejecuta **privDir** para mostrar la vía de acceso de :PERSONAL: en la barra de estado.

```
method pushButton(var eventInfo Event)
   message("Su directorio personal es: ", privDir())
endmethod
```

Vea también getDir
startUpDir
workingDir

rename

Método Renombra un archivo.

Tipo FileSystem

Sintaxis **rename** (const ***nombreAntiguo***, String ***nombreNuevo*** String) Logical

Descripción Cambia el nombre del archivo *nombreAntiguo* a *nombreNuevo*. Si otro archivo utiliza *nombreNuevo*, el método falla; no sustituye el archivo existente. **rename** devuelve True si es satisfactorio; en caso contrario, devuelve False. Este método es independiente de **findFirst** y de **findLast**.

Ejemplo El ejemplo siguiente busca en el directorio actual el archivo especificado en la variable *AntNombre*. Si existe, la llamada a **rename** intenta cambiar su nombre. Aparece un cuadro de diálogo para informar de los errores, si los hay.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   AntNombre, NuevoNombre String
endvar

AntNombre   = "memo14.txt"
NuevoNombre = "memo14.bak"

if fs.findFirst(AntNombre) then
   if not fs.rename(AntNombre, NuevoNombre) then
     msgStop("No puedo cambiar el nombre al archivo, porque ",
    ; NuevoNombre + " ya existe.")
   endIf
else
   msgStop(AntNombre, "No encuentro el archivo.")
endIf
endmethod
```

Vea también copy
delete
findFirst
findNext

setDir

Método Define la vía de acceso a un directorio para una variable FileSystem.

Tipo FileSystem

Sintaxis **setDir** (const ***nombre*** String) Logical

Descripción Define la vía de acceso a un directorio para una variable FileSystem como *nombre*. Compare este método con **setDrive**, que define la unidad por defecto.

Ejemplo Este ejemplo ejecuta **isDir** para averiguar si el directorio especificado en la variable *Directorio* es válido. Si lo es, el código ejecuta **setDir** para convertirlo en directorio por defecto.

```
method pushButton(var eventInfo Event)
var
   fs FileSystem
   Directorio String
endvar

Directorio = "c:\\pdoxwin\\mine\\zap"

if isDir(Directorio) then
   fs.setDir(Directorio)
else
   msgStop(Directorio, "No es un directorio válido.")
endIf
message(fs.getDir()) ; muestra \pdoxwin\mine\zap
endmethod
```

Vea también getDir
setDrive

setDrive

Método Convierte una unidad especificada en unidad por defecto.

Tipo FileSystem

Sintaxis **setDrive** (const ***nombre*** String) Logical

Descripción Devuelve True si logra definir *nombre* como unidad por defecto; en caso contrario, devuelve False. *nombre* puede especificarse con una letra (C), con una letra y dos puntos (C:) o con un alias (:THAM:).

Ejemplo El primer ejemplo ejecuta **view**, definido para el tipo String, para mostrar un cuadro de diálogo y solicitar al usuario que introduzca una unidad. Si el usuario introduce la letra de una unidad válida, la llamada a **setDrive** la convierte en unidad por defecto.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   fs FileSystem
   Unidad String
endvar

Unidad = "Escriba aquí el nombre de la unidad o su alias."
Unidad.view("Cambio de la unidad por defecto.") ; indicativo
para el usuario

if fs.existDrive(Unidad) then
  fs.setDrive(Unidad)
else
  msgStop(Unidad, "La unidad no está disponible.")
endIf
endmethod
```

El ejemplo siguiente muestra cómo utilizar un alias con **setDrive**. Supone que el alias :THAM: ya se ha definido.

```
; AsignarUnidad::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
endVar

fs.setDrive(":THAM:")
endmethod
```

Vea también getDrive
setDir

setFileAccessRights

Procedimiento Define los derechos de acceso (también llamados atributos de archivo) de un archivo.

Tipo FileSystem

Sintaxis **setFileAccessRights** (const ***nombreArchivo*** String , const ***derechos*** String) Logical

Descripción Define los atributos (derechos de acceso) de *nombreArchivo* a los atributos especificados en *derechos*. *derechos* es una cadena que se evalúa en uno o más de los siguientes: A, D, H, R, S, V (para archivo, directorio, oculto, sólo lectura, sistema y volumen, respectivamente). Si se especifica una cadena vacía ( ) para *derechos*, se eliminan todos los atributos de *nombreArchivo*. No es necesario declarar una variable FileSystem (ni utilizar el método **findFirst**) antes de ejecutar **setFileAccessRights**.

Ejemplo Este ejemplo define los atributos del archivo C:\CONFIG.SYS como sólo lectura (R) y oculto (H).

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  NombreArch String
endvar

  NombreArch = "C:\\CONFIG.SYS"

; asignamos a CONFIG.SYS los atributos de Sólo Lectura y Oculto
if setFileAccessRights(NombreArch, "RH") then
  ; si tenemos éxito mostramos un mensaje con los atributos
actuales
  message ("He asignado a " + NombreArch +  "los atributos " +
                getFileAccessRights(NombreArch))
else
  ; de otro modo, el procedimiento ha fallado
  message("No puedo asignar los atributos a " + NombreArch)
endif
endmethod
```

Vea también accessRights
getFileAccessRights

size

Método Devuelve el tamaño de un archivo.

Tipo FileSystem

Sintaxis **size** ( ) LongInt

Descripción Después de un **findFirst** o **findNext** satisfactorio, **size** devuelve el número de bytes del archivo encontrado.

Ejemplo Este ejemplo crea un DynArray que contiene los nombres y tamaños de los archivos de tabla de Paradox existentes en el directorio actual. La llamada a **view**, definido para el tipo DynArray, muestra la información en un cuadro de diálogo.

```
; BotónDeDemostración::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  da DynArray[] LongInt
endvar

if fs.findFirst("*.db") then
  da[fs.name()] = fs.size()
  while fs.findNext()
    da[fs.name()] = fs.size()
  endWhile
  da.view("Nombres y tamaños")
else
  msgStop("*.db", "No encuentro ningún archivo.")
endIf
endmethod
```

Vea también freeDiskSpace
totalDiskSpace

splitFullFileName

Procedimiento Divide una vía de acceso completa en sus componentes.

Tipo FileSystem

Sintaxis **1. splitFullFileName** (const ***nombreArchivoCompleto*** String, var ***componentes*** DynArray[] String)

2. splitFullFileName (const ***nombreArchivoCompleto*** String, var ***nombreUnidad*** String, var ***víaAcceso*** String, var ***nombreArchivo*** String, var ***extensión*** String)

Descripción Divide una vía de acceso completa (obtenida mediante **fullName**) en sus componentes. Es posible utilizar la sintaxis 1 para almacenar el resultado en un DynArray o la sintaxis 2 para almacenarlo en variables distintas.

Si utiliza la sintaxis 1, debe declarar el DynArray antes de ejecutar **splitFullFileName**. El DynArray tiene las claves siguientes: DRIVE, EXT, NAME y PATH.

Ejemplo El primer ejemplo ejecuta **fullName** para obtener el nombre completo de la primera ficha listada en el directorio actual. A continuación, ejecuta **splitFullFileName** para dividir el nombre en sus componentes y almacenarlos en un DynArray. Entonces, ejecuta el DynArray.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  NombreDividido DynArray[] anytype
  NombreArch String
endVar

; si el archivo clientes.db está en el directorio ejemplos
if fs.findFirst("c:\\pdoxwin\\ejemplos\\clientes.db") then

    ; almacenar el nombre completo del archivo en una variable
    NombreArch = fs.fullName()

   ; dividir el nombre del archivo en partes y almacenarlas en
el DynArray
   splitFullFileName(NombreArch, NombreDividido)

   ; mostrar las partes
   splitName.view("Nombre dividido")
endIf

endmethod
```

El ejemplo siguiente ejecuta **splitFullFileName** para dividir el nombre completo de una ficha en sus componentes y, entonces, muestra la vía de acceso y nombre de archivo (sin extensión) en cuadros de diálogo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  Unidad, Ruta, Nombre, Extensión String
```

```
endVar

if fs.findFirst("*.fsl") then
  splitFullFileName(fs.fullName, Unidad, Ruta, Nombre,
Extensión)
  Ruta.view("Vía de acceso")      ; muestra la ruta
  Nombre.view("Nombre del archivo") ; muestra el nombre del
    ; archivo (sin extensión)
endIf
endmethod
```

Vea también fullName

startUpDir

Procedimiento Devuelve una cadena que contiene la vía de acceso al directorio de inicio del usuario.

Tipo FileSystem

Sintaxis **startUpDir** () String

Descripción Devuelve una cadena que contiene la vía de acceso completa (incluida la letra de unidad) del directorio de inicio del usuario; éste es el directorio desde el que se arrancó Paradox.

Ejemplo Este ejemplo muestra un cuadro de diálogo que enumera la vía de acceso al directorio desde el que el usuario arrancó Paradox:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)

  msgInfo("Directorio de inicio", startUpDir())
endmethod
```

Vea también privDir
workingDir

time

Método Devuelve la fecha y hora en que se modificó un archivo por última vez.

Tipo FileSystem

Sintaxis **time** ( ) DateTime

Descripción Devuelve un valor de DateTime que representa la hora y fecha de la última modificación de un archivo.

Ejemplo Este ejemplo ejecuta **time** para obtener la hora y fecha de los cambios más recientes realizados en la tabla *Clientes*. A continuación, compara la fecha de modificación con la de hoy, e informa del resultado.

```
method pushButton(var eventInfo Event)
var
  fs FileSystem
endvar

if fs.findFirst("clientes.db") then
  if fs.time() < DateTime(today()) then
    message("Antigua versión")
  else
    message("Nueva versión")
  endif
endIf
endmethod
```

totalDiskSpace

Método Devuelve la capacidad de una unidad.

Tipo FileSystem

Sintaxis **totalDiskSpace** (const ***letraUnidad*** String) LongInt

Descripción Devuelve el número total de bytes que la unidad *letraUnidad* puede contener. *letraUnidad* puede especificarse con una letra (C) o con una letra y dos puntos (C:).

Ejemplo Este ejemplo ejecuta **totalDiskSpace** y **freeDiskSpace** para calcular el espacio utilizado. Almacena la información en un DynArray, y ejecuta el método **view**, definido para el tipo DynArray, para mostrar la información en un cuadro de diálogo.

```
; EspacioUsado::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  da DynArray[] LongInt
endvar

da["Espacio total"] = fs.totalDiskSpace("C")
da["Espacio libre"] = fs.freeDiskSpace("C")
da["Espacio en uso"] = da["Espacio total"] - da["Espacio
Libre"]     da.view("Drive C")

endmethod
```

Vea también freeDiskSpace

windowsDir

Procedimiento Devuelve la vía de acceso al directorio de Windows.

Tipo FileSystem

Sintaxis **windowsDir** ( ) String

Descripción Devuelve la vía de acceso al directorio de Windows.

Ejemplo Este ejemplo lee el archivo WIN.INI en la unidad B y lo copia al directorio de Windows de la unidad por defecto:

```
; CopiarWinIni::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  NombreArch, Destino String
endvar
NombreArch = "\\win.ini"

fs.setDrive( B )
if fs.findFirst(NombreArch) then
  Destino = windowsDir() + NombreArch
  fs.copy(NombreArch, Destino)
endIf

endmethod
```

Vea también windowsSystemDir
workingDir
privDir
startUpDir

windowsSystemDir

Procedimiento Devuelve la vía de acceso al directorio de sistema de Windows.

Tipo FileSystem

Sintaxis **windowsSystemDir** ( ) String

Descripción **windowsSystemDir** devuelve la vía de acceso al directorio de sistema de Windows.

Ejemplo Este ejemplo lee el archivo ESPECIAL.DRV de la unidad B y lo copia al directorio de sistema de Windows de la unidad actual:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  fs FileSystem
  NombreArch, Destino String
endvar

NombreArch = "\\especial.drv"

fs.setDrive("B")
if fs.findFirst(NombreArch) then
  Destino = windowsSystemDir() + NombreArch
  fs.copy(NombreArch, Destino)
endIf

endmethod
```

Vea también windowsDir

workingDir

Procedimiento Devuelve el nombre del directorio de trabajo actual.

Tipo FileSystem

Sintaxis **workingDir** ( ) String

Descripción Devuelve el nombre del directorio de trabajo actual.

Ejemplo Este ejemplo muestra un cuadro de diálogo que contiene la vía de acceso al directorio de trabajo actual:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)

message("El directorio de trabajo es: " + workingDir())

endmethod
```

Vea también privDir
windowsDir
windowsSystemDir

close

Método Cierra una biblioteca.

Tipo Library

Sintaxis **close** ( )

Descripción Cierra una biblioteca y termina la asociación entre una variable Library y el archivo de biblioteca subyacente.

Ejemplo Este ejemplo declara una variable Library llamada lib y ejecuta **open** para asociarla con la biblioteca TOOLS.LSL. Ejecuta un método de esa biblioteca y, a continuación, activa **close** para terminar la asociación entre la variable y la biblioteca. Otra llamada a **open** asocia *lib* con la biblioteca KIT.LSL, de modo que los métodos incluidos en ella estén disponibles.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  lib Library               ; declaramos una variable Library
endVar
lib.open("TOOLS.LSL")       ; asociamos lib con la librería
TOOLS.LSL
lib.HacerEsto()             ; ejecutamos un método desde la
librería
lib.close()                 ; finalizamos la asociación entre
lib y la librería
lib.open("KIT.LSL")         ; asociamos lib con otra librería
lib.HacerAquello()          ; ejecutamos un método desde la
librería
endmethod
```

Vea también open

enumSource

Método Escribe el código de una biblioteca en una tabla de Paradox.

Tipo Library

Sintaxis **enumSource** (const ***nombreTabla*** String [, const ***recursivo*** Logical])

Descripción Enumera, en la tabla de Paradox especificada en *nombreTabla*, todo el código personalizado (métodos, procedimientos, variables, etc.) guardado en una biblioteca. Si la tabla no existe, Paradox la crea en el directorio de trabajo actual; si existe, la información se añade en la tabla

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
Object	A	128
MethodName	A	128
Source	M	64

El campo Object contiene el nombre UIObject de la biblioteca, el campo MethodName almacena el nombre del método, procedimiento o ventana (Var, Const, Proc, Type o Uses), y el campo Source alberga el código fuente correspondiente.

Este método también se aplica al tipo Form. En las fichas, el argumento optativo *recursivo* especifica si se incluyen los métodos anulados de todos los objetos contenidos en la ficha. Puesto que un Library no contiene objetos, el argumento *recursivo* no es significativo en el contexto de un Library.

Es necesario ejecutar **open** para abrir la biblioteca antes de ejecutar este método.

Ejemplo Este ejemplo declara una variable Library llamada *lib* y ejecuta **open** para asociar *lib* con la biblioteca TOOLS.LSL. A continuación, ejecuta **enumSource** para enumerar el código de la biblioteca en una tabla de Paradox llamada ORIGLIB.DB:

```
; OrigenATabla::pushButton
method pushButton(var eventInfo Event)
var
  lib Library
endVar
if lib.open("TOOLS.LSL", PrivateToForm) then
  ; escribimos el contenido de TOOLS.LSL en ORIGLIB.DB
  ; vamos a :TRABAJO: por defecto
  lib.enumSource("ORIGLIB.DB")
else
  msgStop("TOOLS.LSL", "No puedo abrir la librería.")
endIf
endmethod
```

Vea también enumSourceToFile
open

enumSourceToFile

Método Escribe el código de una biblioteca en un archivo de texto.

Tipo Library

Sintaxis **enumSourceToFile** (const ***nombreArchivo*** String [, const ***recursivo*** Logical])

Descripción Enumera, en el archivo de texto especificado en *nombreArchivo*, todo el código personalizado (métodos, procedimientos, variables, etc.) contenido en una biblioteca. Si el archivo no existe, Paradox lo crea en el directorio de trabajo actual; si existe, Paradox lo sustituye sin pedir confirmación.

En el archivo de texto, se utilizan líneas de comentario para identificar y marcar el comienzo y final de cada método, procedimiento, etc. El ejemplo siguiente muestra el código de la ventana Var y método estándar **open** de una biblioteca:

```
Var
  PunteroTabla TCursor
endVar

method open(var eventInfo Event)

if not PunteroTabla.open("MensAyud.db")
    then msgStop("Error", "No puedo abrir MensAyud.db")
       fail()
endif

endmethod
```

Este método también se aplica al tipo Form. En las fichas, el argumento optativo *recursivo* especifica si se incluyen los métodos anulados de todos los objetos contenidos en la ficha. Puesto que un Library no contiene objetos, el argumento *recursivo* no es significativo en el contexto de un Library.

Antes de ejecutar este método, es necesario activar **open** para abrir la biblioteca.

Ejemplo Este ejemplo declara una variable Library llamada *lib* y ejecuta **open** para asociar *lib* con la biblioteca TOOLS.LSL. A continuación, ejecuta **enumSourceToFile** para listar el código de la biblioteca en un archivo de texto llamado ORIGLIB.TXT.

```
; obtenOrigen::pushButton
method pushButton(var eventInfo Event)
var
  lib Library
endVar

if lib.open("TOOLS.LSL", PrivateToForm) then
  ; escribimos el contenido de TOOLS.LSL en ORIGLIB.TXT
  ; vamos a :personal: por defecto
  lib.enumSourceToFile("ORIGLIB.TXT")

else
  msgStop("TOOLS.LSL", "No puedo abrir la librería.")
endIf
endmethod
```

Vea también enumSource
open

execMethod

Método Ejecuta un método personalizado que no toma argumentos.

Tipo Library

Sintaxis **execMethod** (const ***nombreMétodo*** String)

Descripción Ejecuta el método personalizado indicado por la cadena *nombreMétodo*. El método mencionado en *nombreMétodo* no toma argumentos. **execMethod** permite ejecutar un método de una biblioteca en función del contenido de una variable, lo que implica que el compilador no conoce el método que se activará hasta el momento de la ejecución. Sin embargo, el método debe declararse en la ventana Uses adecuada.

Ejemplo Este ejemplo crea una matriz de tres elementos, siendo cada elemento el nombre de un método personalizado de una biblioteca. El código abre la biblioteca y ejecuta **execMethod** para cada elemento de la matriz:

```
var
  lib              Library
  Métodos Array[3] String
  i                SmallInt
endVar

Métodos[1] = "HacerEsto"
Métodos[2] = "HacerAquello"
Métodos[3] = "HacerLoOtro"

if lib.open("tools.lsl", GlobalToDeskTop) then
  for i from 1 to Métodos.size()
    lib.execMethod(Métodos[i])
  endFor
else
  msgStop("TOOLS.LSL", "No puedo abrir la librería.")
endIf
```

Vea también open

open

Método Asocia una variable Library con una biblioteca y hace que el código de la biblioteca esté disponible.

Tipo Library

Sintaxis **open** (const ***nombreBiblioteca*** String [, const ***ámbitoBibl*** SmallInt]) Logical

Descripción Asocia una variable Library con una biblioteca y hace que el código de la misma esté disponible para una o más fichas, dependiendo del valor de *ámbitoBibl*. ObjectPAL define dos constantes que especifican el ámbito de una biblioteca: PrivateToForm y GlobalToDesktop (enumeradas en el cuadro de diálogo Constantes bajo LibraryScope).

PrivateToForm: Sólo la ficha que ha abierto la biblioteca tiene aceso a su código.

GlobalToDesktop: Todas las fichas del Escritorio (sesión de Paradox) tienen acceso a la biblioteca.

Para abrir una biblioteca y ponerla a la disposición de todas las fichas de la sesión actual de Paradox, utilice el argumento GlobalToDesktop. Por ejemplo, la sentencia siguiente abre la biblioteca LIBRERIA.LSL:

```
lib.open ("Libreria.lsl", GlobalToDesktop)
```

Para que dos o más fichas compartan la misma biblioteca, cada ficha debe abrir la biblioteca de forma global al Escritorio y cada una debe tener una ventana Uses que declare qué rutinas de la biblioteca va a utilizar. Este nivel de ámbito es muy útil en aplicaciones multificha, porque permite que varias fichas accedan a los mismos métodos personalizados y compartan las mismas variables globales.

Una biblioteca puede abrirse de forma privada a la ficha en una ficha y de forma global al Escritorio en otra. Paradox cargará la biblioteca de nuevo, si es necesario.

Por defecto, una biblioteca se abre de forma global al Escritorio. Las sentencias siguientes son equivalentes:

```
lib.open ("Libreria.lsl")   ; estas sentencias son equivalentes
lib.open ("Libreria.lsl", GlobalToDesktop)
```

Ejemplo Este ejemplo muestra cómo dos fichas pueden abrir una biblioteca de forma global al Escritorio y compartir la biblioteca. En el código siguiente, anexado al método estándar **open** de una ficha, *libUno* se abre de forma privada a la ficha, por lo que no puede compartirse, pero *libDos* se abre de forma global al Escritorio y puede compartirse.

```
; formaUno::open method open(var eventInfo Event)
var
  libUno, libDos Library
endVar

if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
```

```
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    libUno.open("TOOLS.LSL", PrivateToForm)   ; no se comparte
con otras fichas
    libDos.open("KIT.LSL", GlobalToDesktop)         ; se puede
compartir con otras
                              ; fichas
endIf

endmethod
```

El código siguiente, anexado al método estándar **open** de otra ficha, ejecuta **open** para abrir la biblioteca KIT.LSL de forma global al Escritorio. Ahora, esta ficha y la anterior pueden compartir KIT.LSL.

```
; formaDos::open method open(var eventInfo Event)
var
  Lib Library
endVar

if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    Lib.open("KIT.LSL", GlobalToDesktop)      ; se puede
compartir con otras Fichas
endIf

endmethod
```

Vea también close

addAlias

Método/ Procedimiento Añade un alias de base de datos en una sesión.

Tipo Session

Sintaxis **addAlias** (const ***nombreAlias*** String, const ***tipo*** String, const ***víaAcceso*** String) Logical

Descripción Añade un alias de base de datos en una sesión. Especifique el nombre del alias en *nombreAlias*, el tipo de alias (Standard) en *tipo* y la vía de acceso completa en *víaAcceso*.

Un alias añadido mediante este método sólo es conocido para la sesión para la que se define, y sólo existe hasta que se cierra la sesión.

Ejemplo El ejemplo siguiente añade un alias en la sesión actual y, a continuación, suministra el nuevo alias al método **open** definido por el tipo Database. Este código se anexa al método estándar **open** de la página *páginaUno*:

```
; páginaUno::open
method open(var eventInfo Event)
var
  InfoClientes Database
endVar

; añadimos el alias InfoClientes a la sesión actual
addAlias("InfoClientes", "Standard", "D:\\pdoxwin\\tablas\\
DatClien")

; ahora utilizamos el alias que especifica la base de datos a
abrir
InfoClientes.open("InfoClientes") ; abre la base de datos
InfoClientes

endmethod
```

Vea también getAliasPath

addPassword

Método/ Procedimiento Presenta una contraseña que permite el acceso a una tabla protegida.

Tipo Session

Sintaxis **addPassword** (const ***contraseña*** String)

Descripción Presenta a una sesión de Paradox la contraseña especificada en *contraseña*. Los sucesivos intentos de acceso a una tabla protegida mediante esa contraseña son satisfactorios. El argumento *contraseña* puede representar una contraseña de propietario o auxiliar. Las contraseñas auxiliares suelen conceder derechos menos amplios que las de propietario. En *contraseña* se diferencia el uso de mayúsculas y minúsculas; una tabla protegida con Sésamo no se abrirá con SESAMO .

Las contraseñas añadidas mediante este método sólo son válidas para la sesión en que se añadieron y sólo tienen efecto hasta que se cierra la sesión. La presentación de una contraseña no afecta al estado de las tablas; por ejemplo, una tabla abierta permanece abierta.

El acceso a las tablas abiertas antes de que se presente la contraseña se controla mediante las contraseñas presentadas previamente. Por ejemplo, si se abrió una tabla utilizando una contraseña auxiliar, los derechos de acceso a esa tabla no cambian con la presentación de la contraseña de propietario. Para conceder derechos de propietario a una tabla abierta previamente, debería cerrar la tabla primero, a continuación presentar la contraseña de propietario y reabrir la tabla.

Utilice **removePassword** para restaurar la protección.

Ejemplo El ejemplo siguiente toma una contraseña del usuario y la presenta a la sesión actual.

```
; obtenAddPassword::pushButton
method pushButton(var eventInfo Event)
var
  Contraseña String
endvar
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  Contraseña.view("Introduzca la contraseña que vamos a
añadir")
  ses.addPassword(Contraseña)
else
   msgStop("Alto","¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también removePassword
removeAllPasswords

advancedWildcardsInLocate

Procedimiento Especifica si esta sesión puede utilizar caracteres comodín avanzados en las operaciones de búsqueda.

Tipo Session

Sintaxis **advancedWildcardsInLocate** ([const ***yesNo*** Logical])

Descripción Especifica si la sesión actual debería utilizar caracteres comodín avanzados encontrados en cadenas de patrón durante las operaciones de búsqueda. Si *yesNo* es Yes, las cadenas de patrón empleadas en las operaciones de búsqueda pueden contener caracteres comodín avanzados; si es No, no pueden contenerlos. Si se omite, yes*No* es Yes por defecto.

Ejemplo Este ejemplo ejecuta **advancedWildcardsInLocate**, si es necesario, para especificar que los comodines avanzados pueden emplearse en una operación de búsqueda. Entonces, el código continúa con una llamada a **locatePattern** que utiliza un patrón con un comodín avanzado.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Sesión Session
endvar

if tc.open("Pedidos.db") then
  ; si los meta caracteres avanzados no se pueden utilizar como
patrones
  if NOT isAdvancedWildcardsInLocate() then
    ; especificamos que esta sesión puede utilizar patrones con
caracteres
    ; avanzados en las siguientes operaciones con locate
    advancedWildcardsInLocate(Yes)
  endif

  if tc.locatePattern("Forma de Envío", "[^SEUR]") then
     msgInfo("Número de Pedido", tc."N° de Pedido")
  else
     msgStop("Error", "No encuentro ningún registro")
  endif
else
   msgStop("Error", "No puedo abrir la tabla Pedidos.")
endif

endmethod
```

Vea también isAdvancedWildcardsInLocate

blankAsZero

Método/ Procedimiento Especifica si los valores vacíos se tratan como ceros en los cálculos.

Tipo Session

Sintaxis **blankAsZero** (const ***yesNo*** Logical)

Descripción Especifica si a los campos numéricos vacíos se les asigna un valor de 0 en los cálculos. Si *yesNo* es Yes, los valores vacíos se tratan como ceros. Si *yesNo* es No, no se tratan así.

Entre los cálculos afectados por **blankAsZero**, se incluyen:
Campos calculados en fichas e informes
Cálculos en consultas
Cálculos de columna que afectan al número de campos o al número de campos no vacíos, por ejemplo, los realizados con **cCount**, **cAverage** y otros

El programador probablemente desee realizar estos cálculos de forma diferente dependiendo del estado de **blankAsZero**. Puede utilizar **isBlankZero** para comprobar el estado y **setBlankZero** para definirlo.

Ejemplo Este ejemplo define **blankAsZero**, si es necesario, como True para que una llamada al método **cAverage** trata los valores de campo vacíos como ceros.

```
; obtenPagoMedio::pushButton
method pushButton(var eventInfo Event)

var
  tc TCursor
endvar

if tc.open("Pedidos.db") then
  if isBlankZero() then
    blankAsZero(True)
endif
  msgInfo("Cantidad Media Pagada", tc.cAverage("Pagado"))
else
   msgStop("Error", "No puedo abrir la tabla Pedidos.")
endif

endmethod
```

Vea también isBlankZero

close

Método Cierra una sesión.

Tipo Session

Sintaxis **close** ( ) Logical

Descripción Termina una sesión cerrando el canal con el Engine de base de datos. **close** libera un contador de usuarios, y deja sin asignación la variable Session.

Ejemplo En el ejemplo siguiente, supóngase que la variable *ses* está asignada a una sesión abierta. Este ejemplo cierra la sesión *ses*.

```
; cerrarSesión::pushButton
method pushButton(var eventInfo Event)
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  if ses.close() then
    msgInfo("Hemos terminado","He cerrado la Sesión con
éxito.")
  else
    msgStop("Atención","No he podido cerrar la Sesión con
éxito.")
  endif
else
  msgStop("Alto","¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también open

enumAliasNames

Método/ Procedimiento Crea una tabla que enumera los nombres de los alias de base de datos disponibles en una sesión.

Tipo Session

Sintaxis **enumAliasNames** (const ***nombreTabla*** String) Logical

Descripción Crea una tabla de Paradox, *nombreTabla*, con una lista de los alias de bases de datos disponibles en una sesión. Si *nombreTabla* ya existe, este método lo sustituye sin pedir confirmación. Si *nombreTabla* está abierto, este método falla.

Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de *nombreTabla* es

Nombre campo	Tipo	Tamaño
DBName	A	32*
DBType	A	32
DBPath	A	82

Ejemplo En este ejemplo, el método **pushButton** de *obtenBotónDeAlias* escribe los nombres de alias activos para la sesión *ses* en la tabla *NomAlias*; a continuación, muestra la tabla.

```
; obtenBotónDeAlias::pushButton
method pushButton(var eventInfo Event)
var
  tv       TableView
endvar
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  ses.enumAliasNames("NomAlias.db")   ; creamos la tabla
  tv.open("NomAlias.db")             ; vemos la tabla
else
   msgStop("Alto","¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también addAlias

enumDataBaseTables

Método/ Procedimiento Crea una tabla que enumera las tablas de una base de datos.

Tipo Session

Sintaxis **enumDataBaseTables** (const ***nombreTabla*** String, const ***nombreBasedatos*** String, const ***especArchivos*** String)

Descripción Crea una tabla de Paradox, *nombreTabla*, en una base de datos, *nombreBasedatos*. La tabla enumera las tablas de una base de datos. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de la tabla

Nombre campo	Tipo	Tamaño
DBName	A	32*
TableName	A	32*

Ejemplo Este ejemplo enumera el conjunto de tablas u otros archivos asociados con un alias.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  Nombre  String
  Archivo  String
  tv       TableView
endvar

; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  Nombre.view("Escriba el nombre (Alias) de la base de
datos") ; indicativo para el alias
  Archivo.view("Escriba la especificación de Archivo")  ;
indicativo para el archivo
  ses.enumDataBaseTables("ListTabl", Nombre, Archivo)
  tv.open("ListTabl")                                ; abrimos
la tabla creada
else
   msgStop("Alto", "¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también enumDriverCapabilities

enumDriverCapabilities

Procedimiento Crea tres tablas de Paradox que enumeran las funciones de la tabla de configuración actual.

Tipo Session

Sintaxis **enumDriverCapabilities** (const ***nombreFuncTablaConfig*** String, const ***nombreFuncTabla*** String, const ***nombreFuncCampo*** String

Descripción Crea tres tablas de Paradox que enumeran las funciones de la tabla de configuración actual. Las tablas se sustituyen (si existen) sin pedir confirmación. Es posible incluir un alias o vía de acceso en los nombres de tabla especificados; si no se especifica un alias ni una vía de acceso, Paradox crea las tablas en el directorio de trabajo.

Las funciones de tabla de configuración se escriben en la tabla *nombreFuncTablaConfig* (cada tipo de tabla permitido se describe mediante un registro), que tiene la estructura siguiente:

Nombre de campo	Tipo	Tamaño
DriverType	A	32*
Description	A	32
Category	A	32
DB	A	4
DBType	A	32
MultiUser	A	4
ReadWrite	A	4
Transactions	A	4
PassThruSQL	A	4
Login	A	4
CreateDb	A	4
DeleteDb	A	4
CreateTable	A	4
DeleteTable	A	4
MultiPasswords	A	4

Las funciones de tabla se escriben en la tabla *nombreFuncTabla* (cada tipo de tabla permitido se describe mediante un registro), que tiene la estructura siguiente:

Nombre de campo	Tipo	Tamaño
DataType	A	32*
TableType	A	32*
Format	A	32*
ReadWrite	A	4
Create	A	4
Restructure	A	4
ValChecks	A	4
Security	A	4
RefInt	A	4
PrimaryKey	A	4
Indexing	A	4
NoFieldType	A	6

MaxRecSize	A	6
MaxFlds	A	6

Las funciones de campo se escriben en la tabla *nombreFuncCampo*, que tiene la estructura siguiente:

Nombre de campo	Tipo	Tamaño
DriverType	A	32*
TableType	A	32*
Format	A	32*
FieldType	A	32*
Description	A	32
NativeType	A	6
XType	A	6
XSubType	A	6
MaxUnits1	A	6
MaxUnits2	A	6
Size	A	6
Required	A	4
Default	A	4
Min	A	4
Max	A	4
RefInt	A	4
Other	A	4
Key	A	4
Multi	A	4

Ejemplo En este ejemplo, el botón *describirControlador* crea y muestra tres tablas que describen la tabla de configuración del Engine.

```
; describirControlador::pushButton
method pushButton(var eventInfo Event)
var
  tv1, tv2, tv3  TableView
endVar
enumDriverCapabilities("CapBd", "CapTabla", "CapCampo")
tv1.open("CapBd")
tv2.open("CapTabla")
tv3.open("CapCampo")
endmethod
```

Vea también enumDriverInfo
enumDriverNames
enumDriverTopics

enumDriverInfo

Procedimiento Enumera las tablas de configuración disponibles.

Tipo Session

Sintaxis **enumDriverInfo** (const ***nombreTabla*** String)

Descripción Lista, en la tabla *nombreTabla*, los tipos de tabla de configuración disponibles actualmente. Si *nombreTabla* existe, se sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de *nombreTabla* es

Nombre de campo	Tipo	Tamaño
DriverType	A	32*
Topic	A	32*
Property	A	32*
PropertyValue	A	68

Ejemplo Este ejemplo enumera información de tablas de configuración en una tabla llamada *InfContr* y, entonces, muestra la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tv1 TableView
endVar
; creamos y vemos la tabla InfContr
enumDriverInfo("InfContr")
tv1.open("InfContr")
endmethod
```

Vea también enumDriverCapabilities
enumDriverNames
enumDriverTopics

enumDriverNames

Método/ Procedimiento Crea una tabla de Paradox que enumera los nombres de las tablas de configuración disponibles en la sesión actual.

Tipo Session

Sintaxis **enumDriverNames** (const ***nombreTabla*** String)

Descripción Escribe, en la tabla *nombreTabla*, los nombres de las tablas de configuración disponibles actualmente. Si *nombreTabla* ya existe, se sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de la tabla es DriverType, A32*.

Ejemplo Este ejemplo enumera, en una tabla llamada *NomContr*, los nombres de las tablas de configuración y muestra la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tv1 TableView
endVar
; creamos y vemos la tabla NomContr
enumDriverNames("NomContr")
tv1.open("NomContr")
endmethod
```

Vea también enumDriverCapabilities
enumDriverInfo
enumDriverTopics

enumDriverTopics

Procedimiento Crea una tabla de Paradox que enumera los temas disponibles actualmente para cada tipo de tabla de configuración.

Tipo Session

Sintaxis **enumDriverTopics** (const ***nombreTabla*** String)

Descripción Escribe los temas de tabla de configuración disponibles para cada tipo de tabla de configuración en la tabla *nombreTabla*. Si *nombreTabla* ya existe, se sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de *nombreTabla* es:

Nombre de campo	Tipo	Tamaño
DriverType	A	32*
Topic	A	32*

Ejemplo Este ejemplo enumera, en una tabla llamada *TopContr*, los temas de tabla de configuración disponibles y muestra la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tv1 TableView
endVar
; creamos y vemos la tabla TopContr
enumDriverTopics("TopContr")
tv1.open("TopContr")
endmethod
```

Vea también enumDriverCapabilities
enumDriverInfo
enumDriverNames

enumEngineInfo

Procedimiento Crea una tabla de Paradox que enumera las propiedades del Engine ODAPI actual.

Tipo Session

Sintaxis **enumEngineInfo** (const ***nombreTabla*** String)

Descripción Crea una tabla de Paradox que describe el contenido del cuadro de diálogo Información del sistema en ODAPI. El nombre y valor de cada parámetro se escribe en un registro de la tabla *nombreTabla*. Si *nombreTabla* ya existe, se sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de *nombreTabla* es:

Nombre de campo	Tipo	Tamaño
Property	A	32*
PropertyValue	A	68

Ejemplo Este ejemplo enumera la información del Engine en una tabla llamada *InfEngin* y muestra la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tv1 TableView
endvar
enumEngineInfo("InfEngin")
tv1.open("InfEngin")
endmethod
```

Vea también enumDriverInfo

enumFolder

Procedimiento Crea una tabla o matriz de Paradox que enumera los archivos existentes en una carpeta folder.

Tipo Session

Sintaxis **1. enumFolder** (const ***nombreTabla*** String [, const ***especArchivos*** String]) Logical

2. enumFolder (var ***resultado*** Array[] String [, const **especArchivos** String]) Logical

Descripción Crea la tabla de Paradox *nombreTabla* o la matriz *resultado* que enumera los archivos de una carpeta de una sesión. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo.

Optativamente, puede especificarse una especificación de archivos en *especArchivos* para listar los archivos que tengan una extensión en particular. Por ejemplo, para listar todas las fichas de un archivo, incluya ".FSL" en *especArchivos*.

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
Name	A	128
LocalName	A	68
IsReference	A	4
IsPrivate	A	4
IsTemp	A	4
Position	A	10

Ejemplo En este ejemplo, el método pide al usuario que introduzca una especificación de archivos (como "*.FSL"). La especificación de archivos introducida se utiliza en **enumFolder** para crear una tabla que enumere los archivos que correspondan con la especificación.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  Archivo String
  tv      TableView
endvar
Archivo.view("Escriba la especificación del nombre de archivo")
enumFolder("Catálogo", Archivo)
message("La tabla tiene la lista de Archivos que coinciden con
la
    ; especificación indicada.");
tv.open("Catálogo")
endmethod
```

Vea también enumDataBaseTables

enumOpenDatabases

Método/ Procedimiento Crea una tabla de Paradox que enumera las bases de datos abiertas.

Tipo Session

Sintaxis **enumOpenDatabases** (const ***nombreTabla*** String) Logical

Descripción Crea la tabla de Paradox *nombreTabla* enumerando las bases de datos abiertas en la sesión actual. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso,Paradox crea *nombreTabla* en el directorio de trabajo.

La estructura de la tabla es:

Nombre campo		Tipo	Tamaño
DBName	A	32*	
DBType	A	32	
ShareMode	A	32	
OpenMode	A	32	

Vea también enumDataBaseTables

enumUsers

Procedimiento Crea una tabla de Paradox que enumera todos los usuarios conocidos con un canal abierto con el Engine ODAPI.

Tipo Session

Sintaxis **enumUsers** (const ***nombreTabla*** String) LongInt

Descripción Crea la tabla *nombreTabla* que enumera todos los usuarios con una vía de acceso abierta con ODAPI. Si *nombreTabla* existe, se sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea **nombreTabla** en el directorio de trabajo.

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
UserName	A	15
NetSession	N	
ProductClass	N	
SerialNumber	A	22

Ejemplo Este ejemplo escribe información sobre los usuarios actuales en la tabla *Usuarios* y, a continuación, muestra la tabla.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tv TableView
endVar
  enumUsers("Usuarios")
  tv.open("Usuarios")
endmethod
```

Vea también getNetUserName

getAliasPath

Método/ Procedimiento Devuelve la vía de acceso de un alias especificado.

Tipo Session

Sintaxis **getAliasPath** (const ***nombreAlias*** String) String

Descripción Devuelve la vía de acceso del alias *nombreAlias*.

Ejemplo Este ejemplo pide al usuario un nombre de alias y, a continuación, muestra las vía de acceso asociada actualmente con ese alias.

```
; obtenViaDeAcceso::pushButton
method pushButton(var eventInfo Event)
var
  Nombre string
  Vía    string
endvar
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  Nombre.view("Escriba el Alias") ; indicativo para el alias
  Vía = ses.getAliasPath(Nombre)  ; obtenemos la vía de acceso
  Vía.view("La respuesta es")     ; mostramos la vía de acceso
else
  msgStop("Alto","¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también addAlias
setAliasPath

getNetUserName

Método/ Procedimiento Devuelve el nombre del usuario de red de una sesión.

Tipo Session

Sintaxis **getNetUserName** ( ) String

Descripción Devuelve el nombre del usuario de red actual.

Ejemplo Este ejemplo muestra el nombre en la red del usuario en un cuadro de diálogo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
msgInfo("¿Quién soy?", getNetUserName())
endmethod
```

Vea también enumUsers

ignoreCaseInLocate

Procedimiento Especifica si se diferencia el uso de mayúsculas y minúsculas en las operaciones de búsqueda.

Tipo Session

Sintaxis **ignoreCaseInLocate** ([const ***yesNo*** Logical])

Descripción Especifica si la sesión actual debería diferenciar el uso de mayúsculas y minúsculas durante las operaciones de búsqueda. Si el argumento optativo *yesNo* es Yes, las operaciones de búsqueda siguientes no diferenciarán el uso de mayúsculas y minúsculas en la comparación de cadenas; si *yesNo* es No, las operaciones de búsqueda sí lo diferenciarán. Si se omite, *yesNo* es Yes por defecto.

Ejemplo Este ejemplo ejecuta **ignoreCaseInLocate**, si es necesario, para preparar una llamada al método **locate**.

```
; buscarNombre::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endvar

if tc.open("Clientes.db") then
  ; si el control de mayúsculas está desactivado
  if isIgnoreCaseInLocate() then

    ; buscamos valores basados en el dato tal y como se
proporcionó
    ; (no se ignoran las mayúsculas ni las minúsculas en la
comparación de las cadenas)
    ignoreCaseInLocate()
  endif

  ; buscamos Buceadores tal y como se escribe en el campo
Nombre
  if tc.locate("Nombre", "Buceadores") then
    tc.edit()
    tc.Nombre = "Buceadores"
    tc.endEdit()
  else
    message("No encuentro Buceadores...")
  endif
else
  msgStop("Error", "No puedo abrir la tabla Clientes.")
endif

endmethod
```

Vea también isIgnoreCaseInLocate

isAdvancedWildcardsInLocate

Procedimiento Informa de si esta sesión está empleando caracteres de comodín en las operaciones de búsqueda.

Tipo Session

Sintaxis **isAdvancedWildcardsInLocate** ( ) Logical

Descripción Informa de si la sesión actual utiliza comodines avanzados durante las operaciones de búsqueda que incluyen cadenas de patrón.

Ejemplo Este ejemplo ejecuta **advancedWildcardsInLocate**, si es necesario, para especificar que pueden emplearse comodines avanzados en una operación de búsqueda. Entonces, el código continúa con una llamada a **locatePattern** que emplea un patrón con un comodín avanzado.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
  Sesión Session
endvar

if tc.open("Pedidos.db") then

  ; si los caracteres de comodín avanzados no se pueden
utilizar como patrones
  if NOT isAdvancedWildcardsInLocate() then
    ; especificamos que esta sesión puede utilizar patrones de
caracteres
    ; avanzados en las siguientes operaciones con locate
    advancedWildcardsInLocate(Yes)
  endif

  if tc.locatePattern("Forma de Envío", "[^SEUR]") then
    msgInfo("Número de Pedido", tc."N° de Pedido")
  else
    msgStop("Error", "No encuentro ningún registro")
  endif
else
  msgStop("Error", "No puedo abrir la tabla Pedidos.")
endif

endmethod
```

Vea también advancedWildcardsInLocate

isAssigned

Método	Informa de si se ha asignado un valor a una variable Session.
Tipo	Session
Sintaxis	**isAssigned** ( ) Logical
Descripción	Informa de si se ha asignado un valor a una variable Session.
Ejemplo	Consulte el ejemplo close.
Vea también	open close

isBlankZero

Método/ Procedimiento Informa de si los valores vacíos se están tratando como cero en los cálculos.

Tipo Session

Sintaxis **isBlankZero** ( ) Logical

Devuelve True si los campos vacíos se tratan como campos con un valor de cero en los cálculos o se cuentan como campos rellenos en los cálculos de recuento (como **cCount**). Si los campos vacíos se tratan como vacíos o no se tienen en cuenta en los cálculos y recuentos, **isBlankZero** devuelve False. Utilice **blankAsZero** para cambiar este valor.

Ejemplo Consulte el ejemplo de blankAsZero.

Vea también blankAsZero

isIgnoreCaseInLocate

Procedimiento Informa de si la sesión actual no diferencia el uso de mayúsculas y minúsculas en las operaciones de búsqueda.

Tipo Session

Sintaxis **isIgnoreCaseInLocate** ( ) Logical

Descripción Informa de si la sesión actual no diferencia el uso de mayúsculas y minúsculas en las operaciones de búsqueda.

Ejemplo Consulte el ejemplo de ignoreCaseInLocate.

Vea también ignoreCaseInLocate

lock

Procedimiento Bloquea una o más tablas.

Tipo Session

Sintaxis **lock** (const ***tabla*** { Table | TCursor| String}, const ***tipoBloqueo*** String [, const ***tabla*** { Table | TCursor| String }, const ***tipoBloqueo*** String]*) Logical

Descripción Bloquea una o más tablas especificadas en una lista de pares tabla-tipo de bloqueo separados por comas. Es posible utilizar un TCursor o un Table para especificar una tabla o mezclar variables TCursor y Table en la lista.

tipoBloqueo debe ser una expresión de cadena que se evalúe como uno de los valores siguientes: Write (Escritura), Read (Lectura) y Full (Completo) (Read y Full sólo se aplican a las tablas de Paradox).

Si este método bloquea todas las tablas de la lista, devuelve True; en caso contrario, devuelve False. Si no puede bloquear todas las tablas, no bloquea ninguna.

Ejemplo Este ejemplo intenta aplicar un bloqueo de escritura sobre la tabla *Pedidos* y uno completo sobre la tabla *Clientes*. Si **lock** logra bloquear ambas tablas, el código muestra datos de ambas tablas en un cuadro de diálogo. Entonces, el código ejecuta **unlock** para eliminar los bloqueos explícitos de *Clientes* y *Pedidos*.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  Tabla    Table
  ClienTC   TCursor
  BaseDatos Database
  Sesión    Session
endvar

Sesión.open()
Sesión.addAlias("Ejemplos", "Standard", "c:\\pdoxwin\\ejemplo")
BaseDatos.open("Ejemplos")

ClienTC.open("Clientes.db", "Ejemplos")
Tabla.attach("Pedidos.db", "Ejemplos")

if lock(ClienTC, "Read", Tabla, "Write") then
  if custTC.locate("Nombre", "Unisco") then
     NumClien = ClienTC."N° de Cliente"
     Tabla.setFilter(NumClien, NumClien)
    msgInfo(String("Total del Pedido ", NumClien),
Tabla.cSum("Total Factura"))
    unlock(ClienTC, "Read", Tabla, "Write")
  else
    msgStop("Error", "No encuentro Unisco.")
  endif
else
  msgStop("Error", "No puedo bloquear una o más tablas.")
endif

endmethod
```

Vea también unLock

open

Método Abre una sesión.

Tipo Session

Sintaxis **open** ([const ***nombreSesión*** String]) Logical

Descripción Abre un canal con el Engine de base de datos (una sesión) y consume un contador de usuario. Es posible abrir más de una sesión desde la misma estación de trabajo, y Paradox considerará cada sesión como un usuario distinto; por ejemplo, los bloqueos definidos en un bloque de sesión acceden desde la otra.

Ejemplo El ejemplo siguiente abre una nueva sesión en la variable *ses*.

```
; abrirSesión::pushButton
method pushButton(var eventInfo Event)
var
  ses Session
endVar
if ses.open() then
  msgInfo("Hemos comenzado","He abierto la Sesión con éxito.")
else
  msgStop("Atención","No he podido abrir la Sesión con éxito.")
endif
endmethod
```

Vea también close

removeAlias

Método/ Procedimiento Elimina un alias de una sesión.

Tipo Session

Sintaxis **removeAlias** (const ***nombreAlias*** String) Logical

Descripción Elimina el alias *nombreAlias* de una sesión.

Ejemplo El ejemplo siguiente añade un alias en la sesión actual y, a continuación, suministra el nuevo alias al método **open** definido para el tipo Database. Cuando el alias ya no es necesario, este código ejecuta **removeAlias** para eliminar el nombre de alias de la sesión actual.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  InfoClien Database
endVar

; añadimos el alias InfoCliente a la sesión actual
addAlias("InfoCliente", "Standard", "D:\\pdoxwin\\tablas\\
datclien")

; ahora utilizamos el alias que especifica la base de datos a
abrir
InfoClien.open("InfoCliente")     ; abre la base de datos
InfoCliente

: hacemos algo con la base de datos abierta,
; luego, cuando no se necesite el alias para nada más,
; lo eliminamos de la sesión actual

removeAlias("InfoCliente")

endmethod
```

Vea también addAlias

removeAllPasswords

Método/ Procedimiento Elimina todas las contraseñas presentadas en una sesión.

Tipo Session

Sintaxis **removeAllPasswords** ( )

Descripción Invierte los efectos de las sentencias **password** emitidas para una sesión. No elimina la seguridad de las tablas; retira las contraseñas presentadas para acceder a las tablas protegidas. Las tablas abiertas no se ven afectadas por **removeAllPasswords**. Para eliminar el acceso a las tablas abiertas, es necesario cerrar las tablas y retirar las contraseñas que permiten el acceso a las tablas.

Ejemplo Este ejemplo elimina todas las contraseñas de la sesión *ses*.

```
; eliminarConstraseñas::pushButton
method pushButton(var eventInfo Event)
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  ses.removeAllPasswords()
else
  msgStop("Alto","La variable de tipo Session no está
asignada")
endif
endmethod
```

Vea también addPassword
removePassword

removePassword

Método/ Procedimiento Elimina una contraseña presentada en una sesión.

Tipo Session

Sintaxis **removePassword** (const ***contraseña*** String)

Descripción Invierte el efecto de una sentencia **password** emitida para una sesión. No desprotege la tabla; simplemente retira la contraseña especificada en el argumento *contraseña* que se presentó para acceder a ella. Tenga en cuenta que se diferencia el uso de mayúsculas y minúsculas en *contraseña*.

Ejemplo En este ejemplo, el botón *obtenRemovePassword* pide al usuario una contraseña para su eliminación y, a continuación, la borra de la sesión actual. Los intentos siguientes de abrir tablas protegidas mediante esa contraseña fallarán.

```
; obtenRemovePassword::pushButton
method pushButton(var eventInfo Event)
var
  Contraseña string
endvar
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
   Contraseña.view("Escriba la contraseña que desea eliminar")
   ses.removePassword(Contraseña)
else
   msgStop("Alto", "¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también addPassword
removeAllPasswords

retryPeriod

Método/ Procedimiento Devuelve el número de segundos en que se reintentará una operación sobre un registro o tabla bloqueada.

Tipo Session

Sintaxis **retryPeriod** ( ) SmallInt

Descripción Devuelve el número de segundos en que se reintentará una operación sobre un registro o tabla bloqueada. El valor por defecto es 0, que significa que las operaciones no se reintentan.

Ejemplo El ejemplo siguiente muestra el periodo de reintento actual al usuario.

```
; obtenReintentar::pushButton
method pushButton(var eventInfo Event)
var
  rp smallint
endvar
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  rp = ses.RetryPeriod()                  ; obtenemos el actual
periodo de
    ; reintento
  rp.view("El periodo de reintento es...") ; muestra el valor
else
   msgStop("Alto","¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también setRetryPeriod

saveCFG

Método/ Procedimiento Guarda la información de alias de la sesión actual en un archivo.

Tipo Session

Sintaxis **saveCfg** (const ***nombreArchivo*** String) Logical

Descripción Guarda la configuración de ODAPI para la sesión actual en *nombreArchivo*. El archivo de configuración especificado en *nombreArchivo* puede cargarse (con la opción de línea de comando **-o**) en lugar de ODAPI.CFG para definir la información de la sesión cuando se arranca Paradox para Windows.

Ejemplo Consulte el ejemplo de System::dlgNetSystem

Vea también System::dlgNetSystem

setAliasPath

Método/ Procedimiento Define la vía de acceso para un alias.

Tipo Session

Sintaxis **setAliasPath** (const ***nombreAlias*** String, const ***víaAccesoAlias*** String) Logical

Descripción Define la vía de acceso *víaAccesoAlias* para el alias *nombreAlias*.

Vea también getAliasPath

setRetryPeriod

Método/ Procedimiento Define el número de segundos en que se reintentará una operación sobre un registro o tabla bloqueada.

Tipo Session

Sintaxis **setRetryPeriod** (const ***periodo*** SmallInt) Logical

Descripción Especifica en *periodo* el número de segundos en que se reintentará una acción sobre una tabla o registro bloqueado. Un valor de 0 significa que las acciones no se reintentan.

Ejemplo Este ejemplo pide al usuario que introduzca un periodo de reintento y, entonces, define con ese valor el reintento para la sesión.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  rp Smallint
endvar
; asumimos que la variable ses es global, y que ha sido
; abierta por otro método
if ses.isAssigned() then
  rp = ses.retryPeriod()
  rp.view("Escriba el periodo de reintento")  ; obtenemos del
usuario
    ; el periodo de reintento
  ses.setRetryPeriod(rp)                      ; asignamos el
periodo
    ; de reintento a la sesión
else
  msgStop("Alto","¡La variable de tipo Session no está
asignada!")
endif
endmethod
```

Vea también retryPeriod

unLock

Procedimiento Desbloquea una o más tablas.

Tipo Session

Sintaxis **unLock** (const ***tabla*** { Table | TCursor| String}, const ***tipoBloqueo*** String [, const ***tabla*** { Table | TCursor| String }, const ***tipoBloqueo*** String]*) Logical

Descripción Desbloquea una o más tablas especificadas en una lista de tablas y tipos de bloqueo separados por comas.

unlock elimina los bloqueos aplicados explícitamente por un usuario o aplicación específica mediante **lock**; no tiene efecto sobre bloqueos aplicados automáticamente por Paradox. *tipoBloqueo* debe ser una expresión de cadena que se evalúe en uno de los siguientes valores: Write (Escritura), Read (Lectura) y Full (Completa). Read y Full sólo se aplican a las tablas de Paradox.

Si falla un **unlock** de la lista, los bloqueos anteriores no se restauran; las tablas permanecen sin bloquear. No es necesario especificar una sesión para utilizar este método, ya que los datos de la sesión se definen al abrir (**open**) un TCursor o anexarla (**attach**) a un Table.

Cada vez que bloquee una tabla explícitamente, asegúrese de que la desbloquea tan pronto como deje de ser necesario el bloqueo explícito. De esta forma, se garantiza la máxima disponibilidad simultánea de las tablas. Además, cuando se bloquea una tabla dos veces, es necesario desbloquearla dos veces. Es posible utilizar el método **lockStatus** (definido para los tipos TCursor y UIObject) para determinar cuántos bloqueos explícitos se han aplicado sobre una tabla. **unlock** devuelve False si se intenta desbloquear una tabla que no está bloqueada o no puede desbloquearse.

Ejemplo Consulte el ejemplo de lock

Vea también lock

beep

Principiante

Procedimiento Emite la señal sonora por defecto de Windows.

Tipo System

Sintaxis **beep ( )**

Descripción Activa el sonido por defecto de Windows. El sonido será audible sólo si la opción Activar sonidos del sistema está verificada en el cuadro de diálogo Sonido del Panel de control.

Para emitir un sonido de un tono y duración especificados a través del altavoz del ordenador, utilice **sound**.

Ejemplo El código siguiente se anexa al método **pushButton** de un botón. Solicita al usuario que introduzca un número y emite una señal sonora si el número está fuera de rango.

```
; obtenUnNúmero::pushButton
method pushButton(var eventInfo Event)
var
  Número SmallInt
endVar
Número = 1
Número.view("Elija un número entre 1 y 10")
while Número  1 OR Número  10
  beep()          ; piii
  sleep(100)      ; pequeña pausa, de otro modo los
                  ; pitidos se ejecutarían juntos como uno solo
beep()
  msgStop("Huy", "Número demasiado grande o pequeño. Inténtelo
de nuevo.")
  Número.view("Elija un número entre 1 y 10")
endwhile
endmethod
```

Vea también sound

close

Principiante

Procedimiento Cierra la ficha actual.

Tipo System

Sintaxis **close ([** const ***valorDevuelto*** AnyType **])**

Descripción Consigna una petición para cerrar la ficha actual. Si se especifica valorDevuelto, se devolverá un valor a la ficha llamadora (si la hay). La especificación de *valorDevuelto* cuando no hay una ficha llamadora no produce un error. close inicia el proceso de cerrar la ficha, lo cual implica la eliminación del foco y la salida.

Ejemplo El código siguiente cierra la ficha actual después de pedir confirmación al usuario.

```
; botónDeCerrar::pushButton
method pushButton(var eventInfo Event)
var
  Respuesta String
endVar
Respuesta = msgYesNoCancel("Cerrando la aplicación", "¿Desea
cerrar " +
                   "esta ficha?")
if Respuesta = "Yes" then
  close()                 ; cerramos la ficha actual
else
  message("No he cerrado la aplicación.")
endif
endmethod
```

Vea también exit

constantNameToValue

Procedimiento Devuelve el valor numérico de una constante.

Tipo System

Sintaxis **constantNameToValue (** const ***nombreConstante*** String **)** AnyType

Descripción Devuelve el valor representado por la constante de ObjectPAL especificada en *nombreConstante*. **constantNameToValue** sólo devuelve valores para las constantes de ObjectPAL predefinidas; no devolverá ningún valor para una constante definida por el programador.

Nota: Por razones de legibilidad y portabilidad (entre otras ventajas), es recomendable que utilice nombres de constantes en lugar de valores numéricos.

Ejemplo El código siguiente devuelve el valor numérico de la constante de acción DataBeginEdit.

```
; MostrarElValorDeUnaConstante::pushButton
method pushButton(var eventInfo Event)
var
  Valor AnyType
  Cadena String
  tf Logical
endvar
Valor = constantNameToValue("DataBeginEdit")  ; la constante se
pasa como
                                              ; una cadena
msgInfo("El valor de DataBeginEdit es", Valor)
tf = constantValueToName("ActionDataCommands", Valor, Cadena)
if tf then   ; si la conversión funciona adecuadamente,
mostramos la cadena
  msgInfo("El nombre de " + String(Valor) + " es", Cadena)
else
  msgInfo("Estado", "Algo fue mal en la conversión.")
endif
endmethod
```

Vea también constantValueToName
enumRTLConstants

constantValueToName

Procedimiento Informa del nombre de una constante.

Tipo System

Sintaxis **constantValueToName (** const ***nombreGrupo*** String, const ***valor*** AnyType, var ***nombreConstante*** String **)** Logical

Descripción Escribe en *nombreConstante* el nombre de una constante con el *valor* especificado que pertenezca al grupo *nombreGrupo*. El nombre de constante se escribe en la variable *nombreConstante*.
constantValueToName sólo funciona con los nombres de constantes de ObjectPAL predefinidas; no funciona con una constante definida por el programador.

Ejemplo Consulte el ejemplo de constantNameToValue.

Vea también constantNameToValue
enumRTLConstants

cpuClockTime

Principiante

Tipo System

Procedimiento Devuelve el número de segundos transcurridos desde que se arrancó el ordenador.

Sintaxis **cpuClockTime ( )** LongInt

Descripción Devuelve el número de milisegundos transcurridos desde que se arrancó el ordenador. El incremento mínimo del reloj es 55 milisegundos. Este procedimiento es útil para medir el intervalo entre dos sucesos.

Ejemplo Este ejemplo emplea **cpuClockTime** para comparar los tiempos de ejecución de dos bucles **for**: uno con una variable sin declarar y otro con una variable declarada. Aunque los tiempos de ejecución varían de un sistema a otro, el bucle se ejecuta significativamente más rápido cuando la variable está declarada.

```
; variablesDeReloj::pushButton
method pushButton(var eventInfo Event)
var
  Variable   SmallInt
  Delta      String
  Comenzar,
  Parar     LongInt
endvar
Comenzar = cpuClockTime()           ; hora del reloj antes del
inicio
for OtraVariable from 1 to 10000    ; OtraVariable no está
declarada
  OtraVariable = OtraVariable + 1
endFor
Parar = cpuClockTime()              ; hora del reloj después de
10000 iteraciones
Delta = String(Para  Comenzar)     ; obtenemos el tiempo
transcurrido utilizando
Delta.view("Tiempo de la variable no declarada") ; una variable
no declarada
                                    ; los tiempos varían
dependiendo del sistema
Comenzar = cpuClockTime()
for Variable from 1 to 10000        ; Variable si está
declarada
  Variable = Variable + 1
endFor
Parar = cpuClockTime()
Delta = String(Parar  Comenzar)    ; obtenemos el tiempo
transcurrido utilizando
Delta.view("Tiempo de la variable declarada")   ; una variable
declarada
msgInfo("Y el resultado es:", "Para un mejor funcionamiento, "
+
             "¡Declare las variables!")
endmethod
```

Vea también Time::time

debug

Procedimiento Detiene la ejecución de un método y activa el Depurador.

Tipo System

Sintaxis **debug ( )**

Descripción La inclusión de **debug** en un método tiene el mismo efecto que el establecimiento de un punto de ruptura. Cuando Paradox encuentra **debug** en un método, el método detiene su ejecución y se abre la ventana Depurador con el cursor situado en la línea que contiene **debug**. A diferencia de los puntos de ruptura, las sentencias **debug** se guardan con el código fuente del método. Además, las sentencias **debug** sólo tienen efecto cuando se elige Depurar | Activar modo DEPURADOR; en caso contrario, no se tienen en cuenta.

Nota: debug sólo funciona en métodos y procedimientos escritos por el programador, no en los incluidos en la biblioteca runtime de ObjectPAL.

debug es útil para establecer puntos de ruptura persistentes en métodos mientras se desarrolla una aplicación. Es posible comprobar la aplicación estando activado Depurar | Activar modo DEPURADOR y enviar la aplicación con dicha opción desactivada.

Ejemplo En este ejemplo, supóngase que está seleccionado el comando de menú Depurar | Activar modo DEPURADOR del Editor de ObjectPAL. El código siguiente ejecuta un bucle **for**. En medio del bucle, la ejecución de **debug** suspende su ejecución y abre una ventana del Editor con el código. Elija Depurar | Ejecutar para reanudar la ejecución o utilice las otras funciones del Depurador Depurador features.

```
; iniciarDepuradorEn50::pushButton
method pushButton(var eventInfo Event)
var
  i SmallInt
endVar
for i from 1 to 100
  message(i)
  if i = 50 then
    debug() ; sólo funcionará si el item del menú Depurar|
Activar modo Depurador
            ; del Editor de ObjectPAL está activado
  endIf
endFor
endmethod
```

Vea también execute

dlgAdd

Procedimiento Activa el cuadro de diálogo Añadir tabla.

Tipo System

Sintaxis **dlgAdd (** const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Añadir tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Añadir. La variable *nombreTabla* especifica la tabla origen.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Añadir tabla y rellena el nombre de tabla *Clientes* como tabla origen. El usuario debe rellenar la tabla destino y cerrar el cuadro de diálogo.

```
; mostrarDiálogoAñadir::pushButton
method pushButton(var eventInfo Event)
;llamamos al cuadro de diálogo Añadir Tabla con Clientes como
tabla
dlgAdd("clientes.db") ; es lo mismo que Archivo|Utilidades|
Añadir
endmethod
```

Vea también dlgCopy
dlgEmpty
dlgSubtract

dlgCopy

Procedimiento Activa el cuadro de diálogo Copiar tabla.

Tipo System

Sintaxis **dlgCopy (** const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Copiar tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Copiar. La variable *nombreTabla* especifica la tabla origen.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Copiar tabla y rellena el nombre de tabla *Clientes* como tabla origen. El usuario debe rellenar la tabla destino y cerrar el cuadro de diálogo.

```
; mostrarDiálogoCopiar::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Copiar Tabla con Clientes como
tabla
DlgCopy("clientes.db") ; es lo mismo que Archivo|Utilidades|
Copiar
endmethod
```

Vea también dlgAdd
dlgEmpty
dlgSubtract

dlgCreate

Procedimiento Activa el cuadro de diálogo Crear tabla.

Tipo System

Sintaxis **dlgCreate (** const nombreTabla String **)**

Descripción Muestra el cuadro de diálogo Crear tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Nuevo | Tabla. La variable *nombreTabla* especifica el nombre de la tabla que se creará.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente abre el cuadro de diálogo Crear tabla. El usuario debe elegir el tipo de tabla, rellenar la descripción de campos y guardar la tabla creada.

```
; mostrarDiálogoCrear::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Crear Tabla, el nombre
; de la tabla no se utiliza
dlgCreate("Tabla.db") ; es lo mismo que Archivo|Nuevo|Tabla
endmethod
```

Vea también dlgCopy
dlgDelete

dlgDelete

Procedimiento Activa el cuadro de diálogo Eliminar tabla.

Tipo System

Sintaxis **dlgDelete (** const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Eliminar tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Eliminar. La variable *nombreTabla* especifica el nombre de la tabla que se borrará.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Eliminar tabla y rellena el nombre de tabla *Clientes* como tabla que se va a borrar. El usuario deber cerrar el cuadro de diálogo y confirmar el borrado.

```
; mostrarDiálogoEliminar::pushButton
method pushButton(var eventInfo Event)
     ; llamamos al cuadro de diálogo Eliminar Tabla con
Clientes como tabla
     DlgDelete("Clientes.db")  ; es lo mismo que Archivo|
Utilidades|Eliminar
endmethod
```

Vea también dlgCreate
dlgEmpty

dlgEmpty

Procedimiento Activa el cuadro de diálogo Vaciar tabla.

Tipo System

Sintaxis dlgEmpty (const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Vaciar tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Vaciar. La variable *nombreTabla* especifica el nombre de la tabla que se vaciará.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Vaciar tabla y rellena el nombre de tabla *Clientes* como tabla que se va a vaciar. El usuario deber cerrar el cuadro de diálogo y confirmar la pérdida de datos.

```
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Vaciar Tabla con Clientes
como tabla
DlgEmpty("Clientes.db") ; es lo mismo que Archivo|Utilidades|
Vaciar
endmethod
```

Vea también dlgDelete
dlgSubtract

dlgNetDrivers

Procedimiento Activa el cuadro de diálogo Tablas de configuración.

Tipo System

Sintaxis **dlgNetDrivers ( )**

Descripción Muestra el cuadro de diálogo Tablas de configuración (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Sistema | Tablas configuración.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El código siguiente abre el cuadro de diálogo Tablas de configuración.

```
; mostrarUsuariosDeRed::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Controladores
DlgNetDrivers()  ; es lo mismo que Archivo|Sistema|Tablas
configuración
endmethod
```

Vea también Session**::**enumDriverCapabilities
Session**::**enumDriverInfo
Session**::**enumDriverNames

dlgNetLocks

Procedimiento Crea y muestra una tabla de información de bloqueos.

Tipo System

Sintaxis **dlgNetLocks ( )**

Descripción Activa el cuadro de diálogo Seleccionar archivo y solicita al usuario que elija una tabla, como si hubiera elegido Archivo | Multiusuario | Ver bloqueos. Al elegir una tabla y hacer clic sobre Aceptar, Paradox crea una tabla de Paradox llamada BLOQUEOS.DB en el directorio personal. Si la tabla existe, Paradox la sustituye sin pedir confirmación. Este método falla si la tabla ya está abierta.

Esta es la estructura de BLOQUEOS.DB:

Nombre de campo	Tipo	Tamaño
Type	S	
UserName	A	14
NetSession	S	
OurSession	S	
RecordNum	N	
Count	S	

El código de ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Después de crear la tabla BLOQUEOS, Paradox la muestra en una ventana Table.

Ejemplo El ejemplo siguiente abre el cuadro de diálogo Seleccionar archivo. Una vez que el usuario elige un archivo, se crea y se muestra una tabla *Bloqueos*.

```
; mostrarBloqueosDeRed::pushButton
method pushButton(var eventInfo Event)
; creamos una tabla de información de bloqueos
:PERSONAL:BLOQUEOS.DB, luego la mostramos
DlgNetLocks()  ; es lo mismo que Archivo|Multiusuario|Ver
bloqueos
endmethod
```

Vea también dlgNetSetLocks
dlgNetRetry
TCursor::enumLocks

dlgNetRefresh

Procedimiento Activa el cuadro de diálogo Frecuencia de actualización en red.

Tipo System

Sintaxis **dlgNetRefresh ( )**

Descripción Muestra el cuadro de diálogo Frecuencia de actualización en red (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Sistema | Autoactualización.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo Este ejemplo abre el cuadro de diálogo Frecuencia de actualización en red.

```
; mostrarRefrescoDeRed::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Frecuencia de actualización en
red
DlgNetRefresh()  ; es lo mismo que Archivo|Sistema|
Autoactualización
endmethod
```

Vea también dlgNetRetry
dlgNetWho

dlgNetRetry

Procedimiento Activa el cuadro de diálogo Periodo de reintento en red.

Tipo System

Sintaxis **dlgNetRetry ( )**

Descripción Muestra el cuadro de diálogo Periodo de reintento en red (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Multiusuario | Establecer reintentos.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo Este ejemplo abre el cuadro de diálogo Periodo de reintento en red.

```
; mostrarDiálogoReintentar::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Periodo de Reintento en Red
DlgNetRetry() ; es lo mismo que Archivo|Multiusuario|Establecer
reintentos
endmethod
```

Vea también dlgNetLocks

dlgNetSetLocks

Procedimiento Activa el cuadro de diálogo Bloqueos de tabla.

Tipo System

Sintaxis **dlgNetSetLocks ( )**

Descripción Muestra el cuadro de diálogo Bloqueos de tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Multiusuario | Establecer bloqueos.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo Este ejemplo abre el cuadro de diálogo Bloqueos de tabla.

```
; mostrarConjuntoDeBloqueos::pushButton
method pushButton(var eventInfo Event)
DlgNetSetLocks()  ; llamamos al cuadro de diálogo Bloqueos de
Tabla
                  ; es lo mismo que elegir Archivo|Multiusuario|
Establecer bloqueos
endmethod
```

Vea también dlgNetLocks

dlgNetSystem

Procedimiento Activa el cuadro de diálogo Información del sistema en ODAPI.

Tipo System

Sintaxis **dlgNetSystem ( )**

Descripción Muestra el cuadro de diálogo Información del sistema en ODAPI (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Sistema | ODAPI.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo Este código abre el cuadro de diálogo Información del sistema en ODAPI.

```
; mostrarSistemaDeRed::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Informacion del Sistema en
ODAPI
DlgNetSystem()    ; es lo mismo que Archivo|Sistema|ODAPI
endmethod
```

Vea también dlgNetDrivers
Session::enumDriverCapabilities
Session::enumDriverInfo
Session::enumDriverNames

dlgNetUserName

Procedimiento Activa el cuadro de diálogo Nombre de usuario de red.

Tipo System

Sintaxis **dlgNetUserName ( )**

Descripción Muestra el cuadro de diálogo Nombre de usuario de red (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Multiusuario | Nombre de usuario.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El código siguiente abre el cuadro de diálogo Nombre de usuario de red, que muestra el nombre del usuario actual en la red.

```
; mostrarNombreUsuario::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Nombre de Usuario de Red
DlgNetUserName()   ; es lo mismo que Archivo|Multiusuario|
Nombre de usuario
endmethod
```

Vea también dlgNetWho

dlgNetWho

Procedimiento Activa el cuadro de diálogo Usuarios actuales.

Tipo System

Sintaxis **dlgNetWho ( )**

Descripción Muestra el cuadro de diálogo Usuarios actuales (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Multiusuario | Usuarios.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El código siguiente abre el cuadro de diálogo Usuarios actuales.

```
; mostrarListaDeUsuarios::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo de Usuarios Actuales
DlgNetWho()  ; es lo mismo que Archivo|Multiusuario|Usuarios
endmethod
```

Vea también dlgNetUserName

dlgRename

Procedimiento Activa el cuadro de diálogo Renombrar tabla.

Tipo System

Sintaxis **dlgRename (** const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Renombrar tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Renombrar. La variable *nombreTabla* especifica el nombre de la tabla que se cambiará.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Renombrar tabla y rellena el nombre de tabla *Clientes* como tabla cuyo nombre se va a cambiar. El usuario debe introducir un nuevo nombre y cerrar el cuadro de diálogo.

```
; mostrarDiálogoRenombrar::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Renombrar Tabla
dlgRename("clientes.db") ; es lo mismo que Archivo|Utilidades|
Renombrar
endmethod
```

Vea también dlgCopy
dlgDelete
dlgSort

dlgRestructure

Procedimiento Activa el cuadro de diálogo Restructurar tabla.

Tipo System

Sintaxis **dlgRestructure (** const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Restructurar tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Restructurar. La variable *nombreTabla* especifica el nombre de la tabla que se restructurará.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Restructurar tabla y rellena el nombre de tabla *Clientes* como tabla que se va a restructurar. El usuario debe modificar la estructura y cerrar el cuadro de diálogo.

```
; mostrarDiálogoReestructurar::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Reestructurar Tabla con
Clientes como tabla
DlgRestructure("clientes.db")   ; es lo mismo que
                                ; Archivo|Utilidades|
Restructurar
endmethod
```

Vea también dlgCreate

dlgSort

Procedimiento Activa el cuadro de diálogo Ordenar tabla.

Tipo System

Sintaxis **dlgSort (** const***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Ordenar tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Ordenar. La variable *nombreTabla* especifica el nombre de la tabla que se ordenará.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente muestra el cuadro de diálogo Ordenar tabla y elige el nombre de tabla *Clientes* como tabla que se va a ordenar. El usuario debe crear una especificación de ordenación y cerrar el cuadro de diálogo.

```
; mostrarDiálogoOrdenar::pushButton
method pushButton(var eventInfo Event)
     ; llamar al cuadro de diálogo Ordenar Tabla
     DlgSort("clientes.db")      ; es lo mismo que Archivo|
Tabla
Utilities|Sort
endmethod
```

Vea también dlgRename

dlgSubtract

Procedimiento Activa el cuadro de diálogo Extraer tabla.

Tipo System

Sintaxis **dlgSubtract (** const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Extraer tabla (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Extraer. La variable *nombreTabla* especifica el nombre de la tabla de la que se van a extraer registros.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Extraer tabla y rellena el nombre de tabla *Clientes* como tabla origen –de la que se van a extraer registros–. El usuario debe rellenar el nombre de la tabla en que se van a extraer registro y cerrar el cuadro de diálogo.

```
; mostrarDiálogoExtraer::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Extraer Tabla
DlgSubtract("clientes.db")  ; Archivo|Utilidades|Extraer
endmethod
```

Vea también dlgAdd
dlgDelete

dlgTableInfo

Procedimiento Activa el cuadro de diálogo Información de estructura.

Tipo System

Sintaxis **dlgTableInfo (** const ***nombreTabla*** String **)**

Descripción Muestra el cuadro de diálogo Información de estructura (descrito en la *Guía del Usuario*), como si se hubiera elegido Archivo | Utilidades | Estructura. La variable *nombreTabla* especifica el nombre de la tabla de cuya estructura se va a informar.

El código ObjectPAL suspende su ejecución hasta que el usuario cierra este cuadro de diálogo. ObjectPAL no tiene control sobre él una vez que se muestra; depende del usuario rellenar la información necesaria y cerrar el cuadro de diálogo.

Ejemplo El ejemplo siguiente activa el cuadro de diálogo Información de estructura para la tabla *Clientes*.

```
; mostrarInformaciónDeTabla::pushButton
method pushButton(var eventInfo Event)
; llamamos al cuadro de diálogo Información de Estructura con
Clientes como tabla
dlgTableInfo("clientes.db") ; es lo mismo que Archivo|
Utilidades|Estructura
endmethod
```

Vea también dlgCreate
dlgRestructure

enumDesktopWindowNames

Procedimiento Crea una tabla que enumera las ventanas de Paradox que están abiertas.

Tipo System

Sintaxis **1. enumDesktopWindowNames (** const nombreTabla String **)**

2. enumDesktopWindowNames (const ***nombreVentanas*** Array[] String **)**

Descripción Enumera los nombres de las ventanas de Paradox abiertas. La sintaxis 1 crea la tabla de Paradox mencionada en *nombreTabla* con el nombre, clase, posición y tamaño de cada ventana abierta en el ordenador del usuario. La tabla enumera todas las aplicaciones abiertas por Paradox. Por defecto, la tabla se crea en el directorio de trabajo. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, este método falla.

Esta es la estructura de la tabla:

Nombre de campo	Tipo	Tamaño
WindowName	A	32
ClassName	A	32
Position	A	12
Size	A	12
Handle	S	

La sintaxis 2 rellena la matriz mencionada en *nombreVentanas* con los nombres de las aplicaciones. Es necesario declarar la matriz antes de ejecutar este método. Las aplicaciones se enumeran en el orden z de Windows; es decir, la aplicación mostrada al principio se lista primera en la matriz, la aplicación de la segunda capa se lista segunda, y así sucesivamente.

Compare este método con **enumWindowNames**, que enumera todas las aplicaciones Windows abiertas que se están ejecutando en el ordenador del usuario.

Ejemplo Este ejemplo escribe en una matriz los títulos de las ventanas abiertas en el escritorio y muestra la matriz. A continuación, el método crea y muestra una tabla que enumera los nombres de las ventanas abiertas.

```
; ObtenNombresEntornoWindows::pushButton
method pushButton(var eventInfo Event)
var
  Nombres Array[] String
  TV              TableView
endvar
TV.open("Clientes")                    ; abrimos una tabla
enumDesktopWindowNames(Nombres)        ; listamos los nombres de
entorno de
                                       ; windows en una matriz
Nombres.view()    ; lista todas las ventanas abiertas en el
escritorio de
```

```
                    ; Paradox, si el método editor de ventanas
está abierto,
                    ; lista los primeros 32 caracteres
enumDesktopWindowNames("NombTabl.db") ; volcamos en una tabla
TV.open("NombTabl")                   ; mostramos la tabla
endmethod
```

Vea también enumFormNames
enumReportNames
enumWindowNames

enumFonts

Procedimiento Crea una tabla que enumera las fuentes disponibles en el sistema del usuario.

Tipo System

Sintaxis **enumFonts (** const **nombreTabla** String **)**

Descripción Crea la tabla de Paradox *nombreTabla* enumerando las fuentes disponibles en el sistema del usuario. Por defecto, este método crea la tabla en el directorio de trabajo del usuario. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, el método falla.

Esta es la estructura de la tabla:

Nombre de campo	Tipo	Tamaño
FaceName	A	64
FontSize	A	8
Attribute	A	64

Ejemplo Este ejemplo ejecuta **enumFonts** para enumera las fuentes en una tabla llamada FONTS.DB. A continuación, busca en un TCursor una fuente llamada Modern. Si Modern está en la tabla, define la propiedad Font.TipoFace de un objeto de campo llamado *Campo* como Modern.

```
; obtenTiposDeLetra::pushButton
method pushButton(var eventInfo Event)
var
  TC TCursor
  TV TableView
endVar
enumFonts("Tipos.db")   ; escribe los nombres de los tipos de
letra en una tabla
TV.open("Tipos.db")     ; mostramos la tabla
dlgTableInfo("Tipos.db") ; mostramos la estructura de la tabla
TC.open("Tipos.db")
if TC.locate("NombreTipo", "Modern") then
  Campo.Font.TypeFace = "Modern"
endIf
TC.close()
endmethod
```

Vea también enumRTLConstants

enumFormNames

Procedimiento Crea una matriz que enumera las fichas abiertas.

Tipo System

Sintaxis **enumFormNames (** var ***nombreFichas*** Array[] String **)**

Descripción Rellena la matriz mencionada en *nombreFichas* con los nombres de las fichas abiertas en el escritorio del usuario. Es necesario declarar *nombreFichas* como matriz redimensionable antes de activar este método. Las fichas se listan en el orden z de Windows; es decir, la ficha mostrada al principio se lista primera en la matriz, la ficha de la segunda capa se lista segunda, y así sucesivamente.

Ejemplo El código de este ejemplo escribe las fichas, informes y bibliotecas abiertas en una matriz llamada *Fichas* y entonces muestra la matriz *Fichas*.

```
;obtenNombresDeTipos::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Array[] String
endVar
enumFormNames(Fichas)
Fichas.view()        ; lista las Fichas
endmethod
```

Vea también enumWindowNames
enumReportNames
enumDesktopWindowNames

enumReportNames

Procedimiento Crea una matriz que enumera los informes abiertos.

Tipo System

Sintaxis **enumReportNames (** var ***nombreInformes*** Array[] String **)**

Descripción Rellena la matriz mencionada en *nombreInformes* con los nombres de los informes abiertos en el escritorio del usuario. Es necesario declarar *nombreInformes* como matriz redimensionable antes de activar este método. Los informes se listan en el orden z de Windows; es decir, el informe mostrado al principio se lista primero en la matriz, el informe de la segunda capa se lista segundo, y así sucesivamente.

Ejemplo El código de este ejemplo escribe las fichas, informes y bibliotecas abiertas en una matriz llamada *Informes* y entonces muestra la matriz *Informes*.

```
; obtenNombresDeInformes::pushButton
method pushButton(var eventInfo Event)
var
  Informes Array[] String
endVar
enumReportNames(Informes)
Informes.view()           ; lista los Informes
endmethod
```

Vea también enumFormNames
enumDesktopWindowNames
enumWindowNames

enumRTLClassNames

Procedimiento Crea una tabla que enumera los tipos de objeto de ObjectPAL.

Tipo System

Sintaxis **enumRTLClassNames (** const ***nombreTabla*** String **)** Logical

Descripción Crea la tabla de Paradox *nombreTabla* listando los nombres de todos los tipos de objetos existentes en la biblioteca runtime de ObjectPAL. Por defecto, esta tabla se crea en el directorio de trabajo (:TRABAJO:). Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, el método falla.

Esta es la estructura de la tabla:

Nombre de campo	Tipo	Tamaño
ClassName	A	32*

Ejemplo Este ejemplo escribe los nombres de tipo de la biblioteca runtime en una tabla llamada *Claselrt* y muestra la tabla.

```
; obtenClasesLRT::pushButton
method pushButton(var eventInfo Event)
var
  TV TableView
endVar
enumRTLCLassNames("ClaseLRT.db") ; escribimos los tipos de
nombres en la tabla
TV.open("claselrt")              ; mostramos la tabla
endmethod
```

Vea también enumRTLConstants
enumRTLMethods

enumRTLConstants

Procedimiento Crea una tabla que enumera las constantes definidas por ObjectPAL.

Tipo System

Sintaxis **enumRTLConstants (** const ***nombreTabla*** String **)** Logical

Descripción Crea la tabla de Paradox *nombreTabla* listando todas las constantes definidas en la biblioteca runtime de ObjectPAL. Por defecto, esta tabla se crea en el directorio de trabajo (:TRABAJO:). Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, el método falla.

Esta es la estructura de la tabla:

Nombre de campo	Tipo	Tamaño
GroupName	A	32*
ConstantName	A	48*
Type	A	48
Value	A	64

Nota: Aunque Paradox proporciona los valores de las constantes, debería utilizarse el valor de la constante en el código; llame a las constantes por su nombre. Utilice los métodos **constantValueToName** y **constantNameToValue** para convertir los valores en constantes, si es necesario.

Ejemplo Este ejemplo escribe las descripciones de las constantes de la biblioteca runtime en una tabla llamada *Constlrt* y muestra la tabla.

```
; obtenConstantesLRT::pushButton
method pushButton(var eventInfo Event)
var
  TV TableView
endVar
enumRTLConstants("Constlrt.db") ; escribimos las
constantes en la tabla
TV.open("constlrt")             ; mostramos la tabla
endmethod
```

Vea también constantValueToName
constantNameToValue
enumRTLClassNames
enumRTLMethods

enumRTLMethods

Procedimiento Crea una tabla que enumera los métodos de ObjectPAL.

Tipo System

Sintaxis numRTLMethods (const ***nombreTabla*** String **)** Logical

Descripción Crea la tabla de Paradox *nombreTabla* listando todos los métodos definidos en la biblioteca runtime de ObjectPAL. Por defecto, esta tabla se crea en el directorio de trabajo (:TRABAJO:). Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, el método falla.

Esta es la estructura de la tabla:

Nombre de campo	Tipo	Tamaño
ClassName	A	32*
MethodType	A	8*
MethodName	A	64*
MethodArgs	A	255*
ReturnType	A	32*

Ejemplo Este ejemplo escribe las descripciones de los métodos de la biblioteca runtime en una tabla llamada *metodlr* y muestra la tabla.

```
; obtenMétodosLRT::pushButton
method pushButton(var eventInfo Event)
var
  TV TableView
endVar
enumRTLMethods("metodlrt.db") ; escribimos los nombres de los
métodos en la tabla
TV.open("métodlrt")           ; mostramos la tabla
endmethod
```

Vea también enumRTLClassNames
enumRTLConstants

enumWindowNames

Procedimiento Crea una tabla o una matriz que enumera las ventanas abiertas.

Tipo System

Sintaxis **1. enumWindowNames (** const ***nombreTabla*** String **)** Logical

2. enumWindowNames (var ***nombreVentanas*** Array[] String **)**

Descripción Crea una lista de las aplicaciones que se están ejecutando bajo Windows en el ordenador del usuario.

La sintaxis 1 crea la tabla de Paradox mencionada en *nombreTabla* listando el nombre, clase, posición y tamaño de cada ventana abierta en el sistema del usuario. La tabla lista las ventanas abiertas por cualquier aplicación, no sólo por Paradox. Por defecto, la tabla se crea en el directorio de trabajo (:TRABAJO:). Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, el método falla.

Esta es la estructura de la tabla:

Nombre de campo		Tipo	Tamaño
WindowName	A	32	
ClassName	A	32	
Position	A	12	
Size	A	12	
Handle	S		

La sintaxis 2 rellena la matriz mencionada en *nombreVentanas* con los nombres de las aplicaciones. Es necesario declarar la matriz antes de ejecutar este método. Las aplicaciones se listan en el orden z de Windows; es decir, la aplicación mostrada al principio se lista primera en la matriz, la aplicación de la segunda capa se lista segunda, y así sucesivamente.

Compare este método con **enumDesktopWindowNames**, que sólo lista las aplicaciones de Paradox.

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *obtenNombresWindows* escribe y muestra información sobre las ventanas abiertas de dos formas. En primer lugar, el método rellena una matriz con los títulos de las ventanas abiertas y muestra la matriz. Entonces, el método rellena una tabla con las descripciones de las ventanas abiertas y muestra la tabla, así como su estructura.

```
; obtenNombresWindows::pushButton
method pushButton(var eventInfo Event)
var
  Nombres Array[] String
  TV      TableView
endvar
enumWindowNames(Nombres)       ; escribimos los nombres en una
matriz
Nombres.view()                 ; lista todas las ventanas
abiertas
```

```
                                    ; si el método editor de
ventanas está abierto,
                                    ; lista los primeros 32
caracteres
enumWindowNames("NombTabl.db") ; escribimos las descripciones
de las ventanas
                                    ; en la tabla
TV.open("NombTabl")             ; mostramos la tabla
dlgTableInfo("NombTabl.db")     ; mostramos la estructura de la
tabla
endmethod
```

Vea también enumDesktopWindowNames

errorClear

Procedimiento Borra la pila de errores.

Tipo System

Sintaxis **errorClear ( )**

Descripción Borra (vacía) la pila de errores de todos los códigos y mensajes de error. Para más información sobre la pila de errores, consulte la *Guía del Programador ObjectPAL*.

Ejemplo Este código borra la pila de errores.

```
; vaciarErrores::pushButton
method pushButton(var eventInfo Event)
errorClear()          ; vacía la pila de errores
endmethod
```

Vea también errorCode
errorMessage
errorPop

errorCode

Principiante

Procedimiento Devuelve un número que describe la condición de error o el error runtime más reciente.

Tipo System

Sintaxis **errorCode ( )** SmallInt

Descripción Devuelve un entero que describe la condición de error o el error runtime más reciente. ObjectPAL proporciona constantes para estos enteros (por ejemplo, peObjectNotFound); consulte Errors en el cuadro de diálogo Constantes.

Ejemplo El método de este ejemplo utiliza una cláusula **try** para intentar anexarse a un objeto llamado *CajaUno* en la ficha actual. Si el objeto no existe, se produce un error grave y el control pasa a la cláusula **onFail**. La cláusula **onfail** emplea **errorCode** para descubrir el error y entonces toma la acción oportuna.

```
; manejarCódigosDeError::pushButton
method pushButton(var eventInfo Event)
var
  obj UIObject
endVar
try
  obj.attach("CajaUno")
  obj.color = Red
onFail
  msgInfo("Estado", "Fallo en el primer intento. Probamos de
otra forma.")
  if errorCode() = peObjectNotFound then
    obj.create(BoxTool, 180, 180, 360, 360)
    obj.name = "CajaUno"
    obj.visible = Yes
    reTry
  else
    fail()
  endIf
endTry
endmethod
```

Vea también errorCode
errorLog
errorMessage

errorLog

Procedimiento Añade información a la pila de errores.

Tipo System

Sintaxis **errorLog (** const ***códigoError*** SmallInt, const ***mensajeError*** String **)**

Descripción Añade la información de error especificada en *códigoError* y *mensajeError* a la pila de errores.

Para más información sobre la pila de errores, consulte la *Guía del Programador ObjectPAL*.

Ejemplo El método de este ejemplo emplea una cláusula **try** para intentar anexarse a un objeto llamado *CajaUno* en la ficha actual. Si el objeto no exite, se produce un error grave y el control pasa a la cláusula **onFail**. Si el código de error no es peObjectNotFound, el método crea y registra un error personalizado.

```
; enviarMensaje::pushButton
method pushButton(var eventInfo Event)
var
  obj    UIObject
  Código  LongInt
  Mensaje String
endVar
try
  obj.attach("CajaUno")
  obj.color = "RedBlue"   ; constante de color no válida
causará un error
                          ; si no es peObjectNotFound
onFail
  if errorCode() = peObjectNotFound then
    msgInfo("Y el error es", errorMessage())
    obj.create(BoxTool, 180, 180, 360, 360)
    obj.name = "CajaUno"
    obj.visible = Yes
    reTry
  else
    ; extraemos el error original
    Código = errorCode()
    Mensaje = errorMessage()
    errorPop()
    ; llevamos el error original de nuevo a la pila, pero
    ; modificando el mensaje de error
    errorLog(Código, self.Name + "::pushButton falló a las " +
                 String(time()) + ". " + Mensaje)
    msgInfo("Y el nuevo error es", errorMessage())
    fail()
  endIf
endTry
endmethod
```

Vea también errorCode
errorMessage

errorPop

errorMessage

Principiante

Procedimiento Devuelve el texto del mensaje de error más reciente.

Tipo System

Sintaxis **errorMessage ( )** String

Descripción Devuelve una cadena que contiene el mensaje mostrado por la condición de error o error runtime más reciente. Si no se ha producido ningún error, **errorMessage** devuelve la cadena vacía (""). Este método es útil para registrar mensajes de error durante una sesión.

Ejemplo Consulte el ejemplo de errorLog.

Vea también errorLog
errorCode
errorPop

errorPop

Procedimiento Elimina la capa superior de información de la pila de errores.

Tipo System

Sintaxis **errorPop ( )** Logical

Descripción Borra la capa superior (el código y mensaje de error añadido más recientemente) de la pila de errores, permitiendo el acceso a la capa inferior.

Para más información sobre la pila de errores, consulte la *Guía del Programador ObjectPAL*.

Ejemplo Consulte el ejemplo de errorLog.

Vea también errorLog
errorCode
errorMessage
errorShow

errorShow

Procedimiento Muestra el cuadro de diálogo de error.

Tipo System

Sintaxis **errorShow ([**const ***ayudaSuperior*** String **[,** const ***ayudaInferior*** String **]])** Logical

Descripción Abre el cuadro de diálogo Error y muestra la información de errores actual. La cadena suministrada en *ayudaSuperior* se utiliza para titular la parte superior del cuadro de diálogo; *ayudaInferior* rotula la parte inferior del mismo.

Para información sobre la pila de errores, consulte la *Guía del Programador ObjectPAL*.

Ejemplo En este ejemplo, el botón *probarUnError* registra varios errores en la pila de errores y los muestra con **errorShow**.

```
; probarUnError::pushButton
method pushButton(var eventInfo Event)
; añadimos dos errores a la pila de errores
errorLog(1, "Primer error")
errorLog(2, "Segundo error")
; mostramos el cuadro de diálogo de errores (el segundo error
está el primero)
errorShow("Título para la parte de arriba", "Título para la
parte de abajo")
endmethod
```

Vea también errorCode
errorLog
errorMessage

errorTrapOnWarnings

Principiante

Tipo System

Procedimiento Especifica si los errores de advertencia se gestionan como errores graves.

Sintaxis **errorTrapOnWarnings (** const ***yesNo*** Logical **)**

Descripción Especifica si los errores de advertencia se gestionan como errores graves (críticos). Por defecto, los errores de advertencia no pueden detectarse en un bloque **try...onFail**. La especificación de **errorTrapOnWarnings(Yes)** indica a Paradox que detecte los errores de advertencia así como los graves. Para más información sobre los errores de advertencia y graves, consulte la *Guía del Programador ObjectPAL*.

Ejemplo El código siguiente se anexa para abrir una ficha simulada: cuando **errorTrapOnWarnings** está definido como No (valor por defecto), no se produce ningún error en este intento. Una vez que **errorTrapOnWarnings** se define como Yes, el mismo código genera un mensaje de error.

```
; warningAError::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.open("Archivo.fsl") ; intentamos asociar un ficha que no
existe
                          ; normalmente, esto no causa un error
errorTrapOnWarnings(Yes)  ; preparar la trampa
Ficha.open("Archivo.fsl") ; esta vez, se obtiene un
mensaje de error
errorTrapOnWarnings(No)   ; restauramos el modo normal
endmethod
```

Vea también errorLog
errorCode
errorMessage
errorShow

execute

Principiante

Procedimiento Ejecuta un comando del DOS.

Tipo System

Sintaxis **execute (** const ***nombrePrograma*** String **[,** const ***espera*** Logical **[,** const ***modoVisualización*** Smallint **]])** Logical

Descripción Ejecuta el comando del DOS especificado en *nombrePrograma*. El argumento optativo *modoVisualización* especifica el modo de visualización que se utilizará al ejecutar el comando.

Si el comando no se encuentra en la vía de acceso indicada en la sentencia PATH del usuario, es necesario especificar la vía de acceso en *nombrePrograma*. Utilice barras invertidas dobles en la vía de acceso.

Ejemplo El ejemplo siguiente arranca la aplicación Reloj de Windows con el estilo de ventana por defecto y espera a que vuelva.

```
; mostrarReloj::pushButton
method pushButton(var eventInfo Event)
execute("Clock.exe", Yes, WinStyleDefault)  ; activamos el
reloj de Windows
endmethod
```

Vea también play

exit

Principiante

Procedimiento Cierra Paradox y sale a Windows.

Tipo System

Sintaxis **exit ( )**

Descripción Cierra Paradox y sale a Windows. Si una aplicación de Paradox ha cambiado, se solicita al usuario que la guarde.

Ejemplo En este ejemplo, el método **menuAction** de la ficha detecta una acción MenuFileExit o MenuControlClose. Si el usuario intenta salir de Paradox, el método pide confirmación; si el usuario lo confirma, el método emplea **exit** para cerrar Paradox.

```
; estaFicha::menuAction
method menuAction(var eventInfo MenuEvent)
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    ; llama a File|Exit y pide confirmación antes de salir
  if eventInfo.id() = MenuFileExit OR
    eventInfo.id() = MenuControlClose then
    if msgYesNoCancel("Salir", "¿Quiere salir?") = "Yes" then
      exit()          ; salimos
    else
      disableDefault  ; si el usuario pulsa [Esc], o eligió
Cancel o No
                      ; detenemos la acción
    endIf
  endIf
else
    ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif
endmethod
```

Vea también close

fail

Procedimiento Provoca que un método falle.

Tipo System

Sintaxis **fail ([** const ***númeroError*** SmallInt, const ***mensajeError*** String **])**

Descripción Provoca que un método falle. La ejecución de este método en las sección **onFail** de un bloque **try...onFail** produce un salto al bloque superior siguiente, si existe, o al bloque **try...onFail** explícito. El argumento optativo *númeroError* especifica un código de error para el fallo. El argumento optativo *mensajeError* especifica un mensaje para su visualización.

ObjectPAL proporciona constantes para los códigos de error; consulte Errors en el cuadro de diálogo Constantes.

Para más información sobre el uso de los bloques **try...onFail**, consulte la *Guía del Programador ObjectPAL*.

Ejemplo En este ejemplo, supóngase que se desea variar la posición de un cuadro llamado *Caja.* Los valores de un objeto de campo no asociado llamados *Campo* van de 1 a 10; la posición de *Caja* se determina por el valor de *Campo*. El código siguiente se anexa al método **changeValue** de *Campo*. Este método emplea un bloque **try...fail** para detectar y gestionar un tipo de datos inadecuado introducido en un campo campo no asociado.

```
; Campo::changeValue
method changeValue(var eventInfo ValueEvent)
Const
  PosX = LongInt(3000)
  PosY = LongInt(1000)
endConst
Var
   X  LongInt
endVar
try
   ; esta estructura if fallará si el contenido del campo no se
puede
   ; comparar con los números enteros del 0 al 10  por ejemplo,
si
   ; el usuario escribe una cadena
   if eventInfo.newValue() = 0 AND eventInfo.newValue() <> 10
then
     X = (eventInfo.newValue() * 400) + PosX
     Caja.Position = Point(X, PosY)
   else
     fail() ; si el valor es un número que está fuera de rango
            ; llamamos al bloque de fallo (fail)
   endif
onFail
  disableDefault
  eventInfo.setErrorCode(CanNotDepart)
  msgStop("Alto", "El dato debe ser un número entre 0 y 10.")
endTry
endmethod
```

Vea también errorCode
errorMessage
errorShow

fileBrowser

Procedimiento Muestra el cuadro de diálogo Examinar de Paradox y devuelve los nombres de uno o más archivos seleccionados por el usuario.

Tipo System

Sintaxis

1. **fileBrowser (** var ***archivoSeleccionado*** String **[,** var ***infoExaminar*** FileBrowserInfo **])** Logical
2. **fileBrowser (** var ***archivosSeleccionados*** Array[] String **[,** var |***infoExaminar*** FileBrowserInfo **])** Logical

Descripción Muestra el cuadro de diálogo Examinar de Paradox y devuelve los nombres de uno o más archivos seleccionados por el usuario. La ejecución de ObjectPAL se suspende hasta que el usuario cierra el cuadro de diálogo. Utilice la sintaxis 1 para devolver un solo nombre de archivo en *archivoSeleccionado* y la sintaxis 2 para devolver una matriz de nombres de archivo en *archivosSeleccionados,* donde *archivosSeleccionados* se declara como una matriz redimensionable. Con cualquiera de las sintaxis, es posible proporcionar un registro ExaminarParam que contenga información que se pasa al cuadro de diálogo Examinar (consulte el segundo ejemplo a continuación).

Ejemplo El primer ejemplo ejecuta **fileBrowser** dos veces: la primera vez, devuelve un solo nombre de archivo y, si es el nombre de una tabla, abre una ventana de tabla. La segunda vez, devuelve una matriz de nombres de archivo (seleccionados pulsando *Mayús* y haciendo clic simultáneamente) y muestra la matriz en un cuadro de diálogo.

```
; BotónDeVerArchivos::pushButton
method pushButton(var eventInfo Event)
var
   Archivo          String
   Archivos Array[] String
   TV               TableView
endVar
fileBrowser(Archivo)    ; muestra el visor de archivos, y espera
                        ; que el usuario eliga un archivo
                        ; la variable Archivo almacena el nombre
del archivo
                        ; elegido
if isTable(Archivo) then
  TV.open(Archivo)     ; abre una ventana de Tabla para el
fichero elegido
endIf

fileBrowser(Archivos)  ; permite al usuario elegir varios
archivos y almacenar
                        ; los nombres de los archivos en una
matriz
Archivos.view()         ; muestra las elecciones del usuario
endmethod
```

También es posible pasar un registro como argumento a **fileBrowser** para especifica qué datos muestra el cuadro de diálogo Examinar. Por ejemplo,

es posible hacer que sólo muestre tablas de Paradox, o sólo fichas, o fichas e informes.

ObjectPAL proporciona un tipo de datos especial, llamado FileBrowserInfo, que puede emplearse *sólo* con el procedimiento **fileBrowser**. FileBrowserInfo es un registro declarado previamente con la estructura siguiente:

```
x, y, w, h        SmallInt    ; tamaño de la ventana del visor
en twips
EstiloVentana     LongInt     ; estilo de la ventana (véase
constantes WindowStyle)
TiposDisponibles LongInt      ; tipo de archivo (véase la tabla
siguiente)
TipoElegido       LongInt     ; Uno de los TiposDisponibles
FiltrosDeArchivo String       ; el filtro de ficheros del cuadro
de edición
Alias             String      ; alias o nombre de la unidad
Ruta              String      ; vía de acceso relativa al Alias
```

Esta estructura está predefinida y es estándar en ObjectPAL, por lo que sólo es necesario declarar una variable del tipo FileBrowserInfo y asignar valores a sus campos, como se muestra en el ejemplo –no es necesario declarar el tipo cada vez que se desea emplear–.

Después de la llamada a **fileBrowser**, los campos Alias, Path y FileFilter se rellenan con los valores que había en el cuadro de diálogo Examinar. En otras palabras, es posible averiguar qué introdujo el usuario en esas áreas del cuadro de diálogo.

El campo AllowableTypes especifica lo que aparece en la lista de edición desplegable para el panel Tipos del cuadro de diálogo Examinar. EL campo SelectedType indica cuál de los AllowableTypes está seleccionado actualmente. La tabla siguiente enumera las constantes de ObjectPAL válidas para su uso en los campos SelectedType y AllowableTypes. Estas constantes también se enumeran bajo FileBrowserFileTypes en el cuadro de diálogo Constantes.

Constante	**Extensión**	**Descripción**
fbFiles	*.*	Todos los archivos
fbTable	*.db	Tablas de Paradox
fbQuery	*.qbe	Archivos QBE
fbForm	*.fsl,*.fdl	Fichas de Paradox
fbReport	*.rsl,*.rdl	Informes de Paradox
fbScript	*.ssl,*.sdl	Macros de Paradox
fbGraphic	*.bmp	Imagen de mapa de bits
fbText	*.txt	Archivos de texto
fbSQL	*.sql	Archivos SQL
fbAllTables	*.db,*.dbf	Tablas de usuario y del sistema
fbTableView	*.tv	Archivos de vista de tabla
fbParadox	*.db	Tablas de Paradox
fbDBase	*.dbf	Tablas de dBASE
fbASCII	*.txt	Archivos de texto
fbQuattroProWindows	*.wt1	Hojas de cálculo de Quattro Pro para Windows
fbQuattroPro	*.wq1	Hojas de cálculo de Quattro Pro para DOS
fbQuattro	*.wkq	Hojas de cálculo de Quattro
fbLotus2	*.wk1	Hojas de cálculo de Lotus (versión 2)

fbLotus1	*.wks	Hojas de cálculo de Lotus (versión 1)
fbExcel	*.xls	Hojas de cálculo de Excel
fbConfig	*.cfg	Archivos de configuración
fbLibrary	*.lsl	Bibliotecas de Paradox

El procedimiento **fileBrowser** sólo mira los nombres dados en la estructura. Es posible pasarle una estructura de registro distinta y el procedimiento halla los campos con los nombres apropiados y los usa. En otras palabras, es posible definir una estructura de registro más simple sólo con los elementos en que se está interesado.

Ejemplo El código siguiente se anexa al método estándar **pushButton** de un botón. Cuando se ejecuta, activa el cuadro de diálogo Examinar y espera a que el usuario elija un archivo. Entonces, muestra información sobre la elección del usuario en el área de estado.

```
method pushButton(var eventInfo Event)
var
  iva FileBrowserInfo ; declaramos una variable que utiliza la
estructura
                      ; de registro predefinida FileBrowserInfo
  ArchivoElegido String
endVar

; las sentencias siguientes asignan valores a los campos en el
; registro de información del visor de archivos
iva.Alias = "TRABAJO" ; busca el directorio de trabajo actual
iva.TiposDisponibles = fbTable + fbForm ; busca Tablas y Fichas
; muestra el visor y procesa la elección del usuario
if fileBrowser(ArchivoElegido, iva) then
  message("Se eligió ", ArchivoElegido," en la vía de acceso ",
iva.path)
else
  message("Se eligió cancelar")
endif

endmethod
```

Vea también FileSystem

formatAdd

Procedimiento Añade un formato.

Tipo System

Sintaxis **formatAdd (** const ***nombreFormato*** String, const ***especFormato*** String **)** Logical

Descripción Crea el formato descrito por *especFormato* y llamado *nombreFormato*. El formato creado está disponible en la sesión actual.

Nota: Los especificadores de anchura de campo (W*n*), alineación (AR, AL, AC) y mayúsculas/minúsculas (CU, CL, CC) no se guardan con una nueva definición de formato; pero sí se guarda la precisión decimal (W.*n*). Consulte **format** en el tipo String para una descripción completa de los especificadores de formato.

Ejemplo El ejemplo siguiente añade una nueva especificación de formato en la sesión y definen como formato de Currency por defecto el nuevo formato.

```
; añadirUnFormato::pushButton
method pushButton(var eventInfo Event)
var
  número Currency
endVar
; primero, añadimos un número con un formato de 4 cifras
decimales y
; un símbolo dólar (símbolo dólar de Windows)
formatAdd("CuatroDecimales", "W15.4, E$W")
; luego, asignamos como formato por defecto el número con el
nuevo formato
formatSetCurrencyDefault("CuatroDecimales")
número = 41324,09876
número.view()                  ; muestra $41.324,0988
endmethod
```

Vea también formatDelete
formatExist

formatDelete

Procedimiento Elimina un formato.

Tipo System

Sintaxis **formatDelete (** const ***nombreFormato*** String **)** Logical

Descripción Suprime el formato llamado *nombreFormato* de la sesión actual.

Ejemplo Este ejemplo borra el formato personalizado llamado *CuatroDecimales*, si existe.

```
; eliminarUnFormato::pushButton
method pushButton(var eventInfo Event)
if formatExist("CuatroDecimales") then
  formatDelete("CuatroDecimales")
else
  msgInfo("", "No he encontrado el formato.")
endif
endmethod
```

Vea también formatAdd
formatExist

formatExist

Procedimiento Informa de si existe un formato.

Tipo System

Sintaxis **formatExist (** const ***nombreFormato*** String **)** Logical

Descripción Comprueba si el formato llamado *nombreFormato* está disponible en la sesión actual. El método devuelve True si lo está; de lo contrario, devuelve False.

Ejemplo En este ejemplo, el método comprueba si existe un formato personalizado llamado *CuatroDecimales*; si no existe, el método añade el nuevo formato y muestra un número con el formato *CuatroDecimales*.

```
; añadirActualFormato::pushButton
method pushButton(var eventInfo Event)
var
  número Currency
endVar
; comprobar si ya existe el formato personalizado
if NOT formatExist("CuatroDecimales") then
  ; si no, añadimos el formato con 4 cifras decimales y
  ; un símbolo dólar (símbolo dólar de Windows)
  msgInfo("", "El formano no existe. Añadiéndolo.")
  formatAdd("CuatroDecimales", "W15.4, E$W")
else
  msgInfo("", "El formato ya existía.")
endif
; asignamos como formato por defecto el número con el
nuevo formato

formatSetCurrencyDefault("CuatroDecimales")
número = 41324,09876
número.view()       ; muestra $41324,0988, porque
                    ; número es una variable de tipo Currency
endmethod
```

Vea también formatAdd
formatDelete

formatSetCurrencyDefault

Procedimiento Define el formato de visualización por defecto para los valores de Currency.

Tipo System

Sintaxis **formatSetCurrencyDefault (** const ***nombreFormato*** String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de Currency. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo Consulte el ejemplo de formatExist.

Vea también formatSetNumberDefault

formatSetDateDefault

Procedimiento Define el formato de visualización por defecto para los valores de Date.

Tipo System

Sintaxis **formatSetDateDefault (** const ***nombreFormato*** String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de Date. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *asignarFormatoDeFecha* define el formato de visualización por defecto para los valores de Date con el formato de fecha larga de Windows. Entonces, el método emplea **view** para mostrar una fecha; la fecha se muestra en el nuevo formato por defecto.

```
; asignarFormatoDeFecha::pushButton
method pushButton(var eventInfo Event)
var
  Fecha Date
endVar
if formatExist("Windows Long") then
  formatSetDateDefault("Windows Long")
  Fecha = date("9/15/92")
  Fecha.view()             ; muestra "Setiembre/15/1992"
else
  msgStop("Alto", "El formato requerido no existe.")
endif
endmethod
```

Vea también formatSetTimeDefault
formatSetDateTimeDefault

formatSetDateTimeDefault

Procedimiento Define el formato de visualización por defecto para los valores de DateTime.

Tipo System

Sintaxis **formatSetDateTimeDefault (** const ***nombreFormato*** String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de DateTime. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *asignarFormatoDeFechaYHora* define el formato de visualización por defecto para los valores de DateTime. Entonces, el método utiliza **view** para mostrar un valor de DateTime; el valor se muestra en el nuevo formato por defecto.

```
; asignarFormatoDeFechaYHora::pushButton
method pushButton(var eventInfo Event)
var
  FechaHora DateTime
endVar
if formatExist("h:m:s am m/d/y") then
  formatSetDateTimeDefault("h:m:s am m/d/y")
  FechaHora = DateTime("11:45:25 am 11/24/61")
  FechaHora.view()                ; muestra 11:45:25 AM,
11/24/61
else
  msgInfo("Estado", "El formato requerido no existe.")
endif
endmethod
```

Vea también formatSetDateDefault
formatSetTimeDefault

formatSetLogicalDefault

Procedimiento Define el formato de visualización por defecto para los valores de Logical.

Tipo System

Sintaxis **formatSetLogicalDefault (** Mconst nombreFormato String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de Logical. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo En la tabla siguiente, el método **pushButton** del botón *asignarFormatoLógico* define el formato de visualización por defecto para los valores de Logical al formato Masculino/Femenino. Entonces, el método utiliza **view** para mostrar un valor lógico; el valor se muestra en el nuevo formato por defecto.

```
; asignarFormatoLógico::pushButton
method pushButton(var eventInfo Event)
var
  Lógico Logical
endVar
if formatExist("Masculino/Femenino") then
  formatSetLogicalDefault("Masculino/Femenino")
  Lógico = True
  Lógico.view()              ; muestra Male
else
  msgStop("Alto", "El formato requerido no existe.")
endif
endmethod
```

Vea también formatAdd

formatSetLongIntDefault

Procedimiento Define el formato de visualización por defecto para los valores de LongInt.

Tipo System

Sintaxis **formatSetLongIntDefault (** const **nombreFormato** String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de LongInt. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *asignarFormatoInteger* define el formato de visualización por defecto para los valores de LongInt al formato Entero. Entonces, el método emplea **view** para mostrar un entero grande; el valor se muestra en el nuevo formato por defecto.

```
; asignarFormatoInteger::pushButton
method pushButton(var eventInfo Event)
var
  Número LongInt
endVar
if formatExist("Entero") then
  formatSetLongIntDefault("Entero")
  Número = 238756
  Número.view()              ; muestra 238756
else
  msgStop("Alto", "El formato requerido no existe.")
endif
endmethod
```

Vea también formatSetSmallIntDefault

formatSetNumberDefault

Procedimiento Define el formato de visualización por defecto para los valores de Number.

Tipo System

Sintaxis **formatSetNumberDefault (** const **nombreFormato** String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de Number. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *asignarFormatoNumérico* define el formato de visualización por defecto para los valores de Number al formato Scientific. Entonces, el método emplea **view** para mostrar un número; el valor se muestra en el nuevo formato por defecto.

```
; asignarFormatoNumérico::pushButton
method pushButton(var eventInfo Event)
var
  número Number
endVar
if formatExist("Científico") then
  formatSetNumberDefault("Científico")
  Número = 3489,283
  número.view()                    ; muestra 3.4893+e3
else
  msgStop("Alto", "El formato requerido no existe.")
endif
endmethod
```

Vea también formatSetCurrencyDefault

formatSetSmallIntDefault

Procedimiento Define el formato de visualización por defecto para los valores de SmallInt.

Tipo System

Sintaxis **formatSetSmallIntDefault (** const **nombreFormato** String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de SmallInt. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *asignarFormatoEnterosCortos* define el formato de visualización por defecto para los valores de SmallInt al formato Entero. Entonces, el método utiliza **view** para mostrar un entero pequeño; el valor se muestra en el nuevo formato por defecto.

```
; asignarFormatoEnterosCortos::pushButton
method pushButton(var eventInfo Event)
var
  Entero SmallInt
endVar
if formatExist("Entero") then
  formatSetSmallIntDefault("Entero")
  Entero = 324
  Entero.view()                    ; muestra 324
else
  msgStop("Alto", "El formato requerido no existe.")
endif
endmethod
```

Vea también formatSetLongIntDefault

formatSetTimeDefault

Procedimiento Define el formato de visualización por defecto para los valores de Time.

Tipo System

Sintaxis **formatSetTimeDefault (** const **nombreFormato** String **)** Logical

Descripción Define el formato por defecto para la visualización de los valores de Time. Esta definición permanece efectiva mientras dure la sesión.

Ejemplo En el ejemplos siguiente, el método **pushButton** del botón *asignarFormatoHora* define el formato de visualización por defecto para los valores de Time al formato *hh:mm:ss am*. Entonces el método emplea **view** para mostrar una hora; el valor se muestra en el nuevo formato por defecto.

```
; asignarFormatoHora::pushButton
method pushButton(var eventInfo Event)
var
  Hora   Time
endVar
if formatExist("hh:mm:ss am") then
  formatSetTimeDefault("hh:mm:ss am")
  Hora = time("12:22:45 pm")
  Hora.view()                   ; muestra 12:22:45 pm
else
  msgInfo("Estado", "El formato requerido no existe.")
endif
endmethod
```

Vea también formatSetDateDefault
formatSetDateTimeDefault

getMouseScreenPosition

Procedimiento Devuelve la posición del ratón como un Point.

Tipo System

Sintaxis **getMouseScreenPosition ( )** Point

Descripción Devuelve las coordenadas (en twips) de la posición del ratón respecto a la pantalla, no al Escritorio. Utilice los métodos del tipo Point (como **getX** y **getY**) para obtener más información.

Este método obtiene la posición del ratón en el momento del suceso; su posición actual puede ser diferente.

Ejemplo En este ejemplo, cuando el usuario hace clic sobre el botón *ratónNervioso*, el ratón salta una pulgada hacia abajo y una pulgada hacia la izquierda.

```
; ratónNervioso::pushButton
method pushButton(var eventInfo Event)
var
  PosRatón,
  NuePosRatón Point
endVar
PosRatón = getMouseScreenPosition()
NuePosRatón = PosRatón + Point(1440, 1440)
setMouseScreenPosition(NuePosRatón) ; movemos el puntero del
ratón una pulgada
                                    ; hacia abajo y una pulgada
a la derecha
endmethod
```

Vea también setMouseScreenPosition

helpOnHelp

Procedimiento Muestra información sobre cómo utilizar el sistema de Ayuda.

Tipo System

Sintaxis **helpOnHelp ( )** Logical

Descripción Muestra información que explica cómo utilizar la aplicación Ayuda de Windows, abriéndola si es necesario.

Ejemplo Consulte la aplicación THAM en el directorio VACACION para ejemplos del uso de una aplicación de ayuda de Windows con ObjectPAL.

Vea también helpShowContext
helpShowIndex

helpQuit

Procedimiento Indica que la aplicación Ayuda ya no es necesaria.

Tipo System

Sintaxis **helpQuit (** const ***nombreArchivoAyuda*** String **)** Logical

Descripción Indica a la aplicación Ayuda que ya no es necesaria. Si ninguna otra aplicación ha solicitado Ayuda, Windows cierra la aplicación Ayuda.

Ejemplo Consulte la aplicación THAM en el directorio VACACION para ejemplos del uso de una aplicación de ayuda de Windows con ObjectPAL.

Vea también helpOnHelp

helpSetIndex

Procedimiento Define el índice de ayuda.

Tipo System

Sintaxis **helpSetIndex (** const ***nombreArchivoAyuda*** String, const ***idIndice*** LongInt **)** Logical

Descripción Indica a la aplicación Ayuda de Windows que defina el índice actual como *idIndice* en *nombreArchivoAyuda*.

Ejemplo Consulte la aplicación THAM en el directorio VACACION para ejemplos del uso de una aplicación de ayuda de Windows con ObjectPAL.

Vea también helpShowIndex

helpShowContext

Procedimiento Muestra ayuda para un contexto en particular.

Tipo System

Sintaxis **helpShowContext (** const ***nombreArchivoAyuda*** String,

const ***idAyuda*** LongInt **)** Logical

Descripción Indica a la aplicación Ayuda de Windows que busque *idAyuda* en *nombreArchivoAyuda* y muestre la ayuda relacionada.

Ejemplo Consulte la aplicación THAM en el directorio VACACION para ejemplos del uso de una aplicación de ayuda de Windows con ObjectPAL.

Vea también helpShowIndex
helpShowTopic

helpShowIndex

Procedimiento Muestra el índice de un archivo de ayuda especificado.

Tipo System

Sintaxis **helpShowIndex (** const ***nombreArchivoAyuda*** String **)** Logical

Descripción Muestra el índice del archivo *nombreArchivoAyuda*.

Ejemplo Consulte la aplicación THAM en el directorio VACACION para ejemplos del uso de una aplicación de ayuda de Windows con ObjectPAL.

Vea también helpShowContext
helpShowTopic

helpShowTopic

Procedimiento Muestra ayuda para un tema especificado.

Tipo System

Sintaxis **helpShowTopic (** const ***nombreArchivoAyuda*** String,

const ***claveTema*** String **)** Logical

Descripción Indica a la aplicación Ayuda de Windows que busque *claveTema* en *nombreArchivoAyuda* y muestre la ayuda asociada.

Ejemplo Consulte la aplicación THAM en el directorio VACACION para ejemplos del uso de una aplicación de ayuda de Windows con ObjectPAL.

Vea también helpShowTopicInKeywordTable
helpShowContext
helpShowIndex

helpShowTopicInKeywordTable

Procedimiento Muestra ayuda para un tema identificado por una palabra clave en una tabla de palabras clave alternativas.

Tipo System

Sintaxis **helpShowTopicInKeywordTable (** const ***nombreArchivoAyuda*** String, const ***letraTablaClaves*** String, const ***claveTema*** String) Logical

Descripción Indica a la aplicación Ayuda de Windows que busque *letraTablaClaves* y *claveTema* en el archivo *nombreArchivoAyuda* y muestre la ayuda asociada.

Ejemplo Consulte la aplicación THAM en el directorio VACACION para ejemplos del uso de una aplicación de ayuda de Windows con ObjectPAL.

Vea también helpShowTopic
helpShowContext
helpShowIndex

message

Principiante

Procedimiento Muestra un mensaje en la línea de estado.

Tipo System

Sintaxis **message (** const ***mensaje*** String **[,** const ***mensaje*** String]*)

Descripción Muestra un mensaje compuesto por un máximo de seis cadenas en la línea de estado.

Ejemplo El método siguiente escribe un mensaje en la línea de estado:

```
; mostrarMensaje::pushButton
method pushButton(var eventInfo Event)
var
  Nombre, Apellidos String
endVar
Apellidos = "Borland"
Nombre = "Paco"
message("Hola, mi nombre es ", Nombre, " ", Apellidos, ".")
endmethod
```

Vea también msgInfo
msgQuestion
msgStop

msgAbortRetryIgnore

Procedimiento Muestra un cuadro de diálogo que contiene un mensaje y tres botones: Sí, No y Omitir.

Tipo System

Sintaxis **msgAbortRetryIgnore (** const ***título*** String, const ***texto*** String **)** String

Descripción Muestra un cuadro de diálogo con tres botones donde *título* especifica el texto para la barra de título y *texto* especifica el mensaje mostrado al usuario. El valor devuelto depende del botón sobre el que se hace clic: "Cancelar", "Reintentar" u "Omitir". Si el usuario pulsa *Esc* o hace clic sobre Cerrar, el valor devuelto es "Cancelar".

Ejemplo En este ejemplo, el botón *mostrarAbortarReintentarIgnorar* advierte al usuario de que una operación puede ser muy prolongada y le pide que elija un botón.

```
; mostrarAbortarReintentarIgnorar::pushButton
method pushButton(var eventInfo Event)
var
  Cadena String
endVar
Cadena = msgAbortRetryIgnore("Nota", "Esto puede llevar un buen
rato.
¿Desea parar?")  ; esto ocupa dos líneas
Cadena.view()
; ejecutamos un método personalizado basado en la elección del
usuario
switch
  case Cadena = "Abort"  : message("Abortando la operación.")
  case Cadena = "Retry"  : message("Reintentando la
operación.")
  case Cadena = "Ignore" : message("Ignorando el problema.")
  case Cadena = "Cancel" : message("Cancelando la operación.")
endSwitch
endmethod
```

Vea también msgRetryCancel
msgYesNoCancel

msgInfo

Principiante

Procedimiento Muestra un cuadro de diálogo que contiene un icono de información, un mensaje y un botón Aceptar.

Tipo System

Sintaxis **msgInfo (** const **título** String, const **texto** String **)**

Descripción Muestra un cuadro de diálogo con un solo botón. El texto de *título* se muestra en la barra de título y *texto* en el cuadro en sí. El usuario debe hacer clic sobre Aceptar o pulsar *Esc* para cerrar el cuadro. Este método no devuelve ningún valor.

Ejemplo Este ejemplo emplea el método **msgInfo** para mostrar un hecho al usuario.

```
; mostrarMsgInfo::pushButton
method pushButton(var eventInfo Event)
msgInfo("Trivial", "La capital de España es Madrid.")
endmethod
```

Vea también msgQuestion
msgStop

msgQuestion

Principiante

Procedimiento Muestra un cuadro de diálogo que contiene un mensaje, un icono de pregunta y dos botones: Sí y No.

Tipo System

Sintaxis **msgQuestion (** const **título** String, const **texto** String **)** String

Descripción Muestra un cuadro de diálogo con dos botones. El texto de *título* se muestra en la barra de título y *texto* se muestra el cuadro en sí. El valor devuelto depende del botón sobre el que el usuario hace clic para cerrar el cuadro de diálogo: "Sí" o "No". Si el usuario pulsa *Esc* o hace clic sobre el cuadro Cerrar, el valor devuelto es "Cancelar".

Ejemplo El código siguiente pregunta al usuario si desea cambiar el título del Escritorio. Si el usuario pulsa el botón Sí, se cambia el título del Escritorio y, a continuación, se restaura.

```
; mostrarMsgQuestion::pushButton
method pushButton(var eventInfo Event)
var
  Elección   String
  Aplicación Application
endVar
Elección = msgQuestion("Confirme", "Está seguro de que quiere
cambiar
el titulo a 'Aplicación personalizada'?")
switch
  case Elección = "Yes" :
    Aplicación.setTitle("Aplicación personalizada") ; cambiamos
el título del
                                                    ;
escritorio
    sleep(2000)                                     ; pausa
    Aplicación.setTitle("Paradox para Windows")     ; lo
dejamos como estaba
  case Elección = "No"  :

    message("No he cambiado el título de la Aplicación.")
endSwitch
endmethod
```

Vea también msgInfo
msgStop

msgRetryCancel

Procedimiento Muestra un cuadro de diálogo que contiene un mensaje y dos botones: Reintentar y Cancelar.

Tipo System

Sintaxis **msgRetryCancel (** const **título** String, const **texto** String **)** String

Descripción Muestra un cuadro de diálogo con dos botones, donde *título* especifica el texto para la barra de título y *texto* especifica un mensaje que se muestra al usuario. El valor devuelto depende del botón sobre el que el usuario haga clic: "Reintentar" o "Cancelar". Si el usuario pulsa *Esc* o hace clic sobre el cuadro Cerrar, el valor devuelto es "Cancelar".

Ejemplo El ejemplo siguiente realiza una pregunta al usuario en respuesta a un problema y muestra un mensaje en la línea de estado dependiendo de cómo responde el usuario.

```
; mostrarReintentarCancelar::pushButton
method pushButton(var eventInfo Event)
var
  Cadena String
endVar
Cadena = msgRetryCancel("Dilema", "¿Qué hará usted?")
switch
  case Cadena = "Retry"  : message("Reintentando.")
  case Cadena = "Cancel" : message("Cancelando.")
endSwitch
endmethod
```

Vea también msgYesNoCancel
msgAbortRetryIgnore

msgStop

Procedimiento Muestra un cuadro de diálogo que contiene un icono de signo Stop, un mensaje y un botón Aceptar.

Tipo System

Sintaxis **msgStop (** const **título String, const *texto*** String **)**

Descripción Muestra un cuadro de diálogo con un solo botón. El texto de *título* se muestra en la barra de título y *texto* se muestra el cuadro en sí junto con un icono Stop. El usuario debe hacer clic sobre Aceptar o pulsar *Esc* para cerrar el cuadro de diálogo. Este método no devuelve ningún valor.

Ejemplo En este ejemplo, el método **pushButton** de *mostrarmsgStop* alerta al usuario de una acción potencialmente peligrosa.

```
; mostrarmsgStop::pushButton
method pushButton(var eventInfo Event)
msgStop("Alto", "Si hace eso, no se almacenaran los cambios.")
endmethod
```

Vea también msgInfo
msgQuestion

msgYesNoCancel

Procedimiento Muestra un cuadro de diálogo que contiene un mensaje y tres botones: Sí, No y Cancelar.

Tipo System

Sintaxis **msgYesNoCancel (** const **título** String, const **texto** String **)** String

Descripción Muestra un cuadro de diálogo con dos botones, donde *título* especifica el texto para la barra de título y *texto* especifica el mensaje que se muestra al usuario. El valor devuelto depende del botón sobre el que el usuario haga clic: "Sí", "No" o "Cancelar". Si el usuario pulsa *Esc* o hace clic sobre el cuadro Cerrar, el valor devuelto es "Cancelar".

Ejemplo En este ejemplo, se utiliza **msgYesNoCancel** para averiguar si el usuario desea guardar los datos antes de salir, desecharlos o cancelar la operación de salida y permanecer en la aplicación.

```
; mostrarSiNoCancelar::pushButton
method pushButton(var eventInfo Event)
var
  Elección String
endVar
Elección = msgYesNoCancel("Salir", "¿Almaceno los datos antes
de salir?")
switch
  case Elección = "Yes"    : message("Almaceno los datos.")
  case Elección = "No"     : message("Datos olvidados.")
  case Elección = "Cancel" : message("Cancelo y vuelvo a la
aplicación.")
endSwitch
endmethod
```

Vea también msgRetryCancel
msgAbortRetryIgnore

pixelsToTwips

Procedimiento Convierte las coordenadas de pantalla de pixels a twips.

Tipo System

Sintaxis **pixelsToTwips (** const **pixels** Point **)** Point

Descripción Convierte las coordenadas de pantalla especificadas en *pixels* de pixels a twips. Un pixel es un punto en la pantalla, mientras que twip es una unidad independiente del dispositivo que es igual a 1/1440 de pulgada (1/20 de un punto de la impresora).

Ejemplo El ejemplo siguiente muestra la posición del botón (*self*) primero en twips y, a continuación, en pixels. El método continúa averiguando la resolución de visualización del sistema (dada en pixels) y utiliza esa información para abrir una ventana en el centro de la pantalla.

```
; transformarTwipsEnPixels::pushButton
method pushButton(var eventInfo Event)
var
  Puntos,
  Twips   Point
  Info    DynArray[] AnyType
  x, y   SmallInt
  Ficha   Form
endVar
Puntos = self.Position
Puntos.view("Posición de este botón en twips")
Puntos = twipsToPixels(Puntos)
Puntos.view("Posición de este botón en pixels")
; abrimos algo de 2" por 2" exactamente en el centro de la
pantalla
sysInfo(Info)                ; rellenamos la matriz con
información del sistema

Twips = Point(Info["FullWidth"], Info["FullHeight"])
Twips = pixelsToTwips(Twips)
x = int(Twips.x()/2)  1440  ; calculamos la coordenada x una
pulgada a la
                             ; izquierda del centro
y = int(Twips.y()/2)  1440  ; calculamos la coordenada y una
pulgada sobre el
                             ; centro
Ficha.open("Clientes.fsl", WinStyleDefault, x, y, 2880, 2880)

endmethod
```

Vea también twipsToPixels

play

Principiante

Procedimiento Ejecuta una macro independiente.

Tipo System

Sintaxis **play (** const ***nombreMacro*** String **)** AnyType

Descripción Ejecuta las sentencias de la macro especificada en *nombreMacro*. Una macro independiente consta de una o más sentencias de ObjectPAL anexadas a un objeto que nunca se muestra. Equivale a un tipo especial de método personalizado. Para más información, consulte la *Guía del Programador ObjectPAL*.

Ejemplo El código siguiente ejecuta una macro llamada MACRO.SSL, que se supone que está en el directorio de trabajo.

```
; hacerSonarUnaMacro::pushButton
method pushButton(var eventInfo Event)
play("Macro.ssl")
endmethod
```

Vea también execute

readEnvironmentString

Procedimiento Lee un elemento del entorno DOS.

Tipo System

Sintaxis **readEnvironmentString (** const ***clave*** String **)** String

Descripción Devuelve una cadena que contiene información sobre la variable de entorno del DOS especificada en *clave*. Los valores se asignan a las variables de entorno mediante el comando SET del DOS. Controlan el aspecto y funcionamiento del DOS y de algunos programas y archivos de proceso por lotes (.BAT). Entre las variables de entorno más utilizadas, se incluyen PATH, PROMPT y COMSPEC. Para más información, consulte el manual del DOS, en especial el comando SET.

Ejemplo Este ejemplo utiliza **readEnvironmentString** y **writeEnvironmentString** para obtener el valor de la variable de entorno PATH.

```
; cambiarCadenaDelEntorno::pushButton
method pushButton(var eventInfo Event)
var
  fs              FileSystem
  Vía, Directorio String
  Rutas   Array[] String
endVar
; fs.getDir() proporciona algunos caracteres ANSIaltosno una
cadena significativa
Directorio = fs.getDir()                    ; obtenemos el
directoro actual
Directorio = "c:\\pdoxwin\\reftrabj\\pc0syall"
Directorio.view("Directorio actual")
Vía = readEnvironmentString("PATH")        ; lee la variable
PATH del entorno
Vía.breakApart(";", Rutas)                 ; terminar en un
punto y coma

Rutas.view("Una matriz de vías de acceso") ; vemos los
resultados
if NOT Rutas.contains(Directorio) then     ; si el directorio
actual no está en
                                           ; la vía de acceso
  msgInfo("", "Añadiendo el directorio actual a la vía de
acceso.")
  writeEnvironmentString("PATH", Vía + ";" + Directorio)
; lo añadimos
endif
Vía = readEnvironmentString("PATH")        ; leemos la variable
del entorno ya
                                           ; cambiada
Vía.view()
Vía.breakApart(";", Rutas)                 ; terminar
Rutas.view("Una matriz de vías de acceso") ; vemos los
resultados
endmethod
```

Vea también readProfileString
writeEnvironmentString

readProfileString

Procedimiento Lee un elemento de un archivo de configuración inicial.

Tipo System

Sintaxis **readProfileString (** const **nombreArchivo** String, const **sección** String, const **clave** String **)** String

Descripción Devuelve un valor de una sección especificada de un archivo especificado en el sistema del usuario. Este método busca por defecto en el directorio de Windows del usuario. Normalmente, se utilizará este método para leer el archivo WIN.INI del usuario, por lo que *nombreArchivo* sería WIN.INI.

Una sección se marca en WIN.INI mediante una cadena incluida entre corchetes en una línea por sí sola (por ejemplo, [windows]). Sin embargo, al especificar *sección*, omita los corchetes; es decir, para especificar la sección, utilice "windows".

Dentro de una sección, la marca de un valor es una cadena seguida de un signo igual (por ejemplo, Beep =), pero no incluya el signo = al especificar *clave*.

Ejemplo Este ejemplo usa **readProfileString** y **writeProfileString** para obtener y cambiar el valor de la señal sonora de Windows.

```
; cambiarCadenaDeCaracterísticas::pushButton
method pushButton(var eventInfo Event)
var
  Pitido     String
  DirWindows String
endVar
DirWindows = windowsDir()
Pitido = readProfileString(DirWindows + "\\win.ini", "windows",
"Beep")
msgInfo("¿Pita?", Pitido) ; muestra si o no, dependiendo de las
caracteristicas
                          ; asignadas por el usuario
if Pitido  "SI" then
  msgInfo("Atención", "Cambiando la cadena de características
del Pitido a SI.")
  writeProfileString(DirWindows + "\\win.ini", "windows",
"Beep", "SI")
  beep()
else
  msgInfo("Atención", "Cambiando la cadena de características
del Pitido a NO.")
  writeProfileString(DirWindows + "\\win.ini", "windows",
"Beep", "SI")
  beep()
endIf
endmethod
```

Vea también writeProfileString
readEnvironmentString

setMouseScreenPosition

Procedimiento Muestra el puntero en una posición especificada.

Tipo System

Sintaxis **1. setMouseScreenPosition (** const ***posiciónRatón*** Point)

2. setMouseScreenPosition (const **x** LongInt, const **y** LongInt **)**

Descripción Muestra el puntero en la posición especificada en *posiciónRatón* (sintaxis 1) o *x* e *y* (sintaxis 2). Las coordenadas deberían especificarse en twips.

Ejemplo Consulte el ejemplo de getMouseScreenPosition.

Vea también getMouseScreenPosition

setMouseShape

Procedimiento Especifica la forma del puntero.

Tipo System

Sintaxis **setMouseShape (** const ***idFormaRatón*** LongInt **)** LongInt

Descripción Especifica en *idFormaRatón* la forma del puntero. ObjectPAL proporciona constantes para *idFormaRatón*; consulte MouseShapes en el cuadro de diálogo Constantes.

Ejemplo Este ejemplo muestra una a una todas las formas de puntero disponibles cuando el usuario hace clic sobre un campo.

```
; cambiarCampoDelRatón::mouseUp
method mouseUp(var eventInfo MouseEvent)
beep()
message("Tenga cuidadocambiando las características del
ratón.")
setMouseShape(MouseIBeam)
sleep(1000)
setMouseShape(MouseCross)
sleep(1000)
setMouseShape(MouseWait)
sleep(1000)
setMouseShape(MouseUpArrow)
sleep(1000)
setMouseShape(MouseArrow)
endmethod
```

Vea también setMouseScreenPosition
getMouseScreenPosition

sleep

Principiante

Procedimiento Produce una demora de una duración especificada.

Tipo System

Sintaxis **sleep ([** const **númeroMilisegundos** LongInt **])**

Descripción Suspende la ejecución de un método durante el intervalo especificado en *númeroMilisegundos*. Los demás programas que no son Paradox, continúan ejecutándose. La impresión también continúa. El Escritorio de Paradox, incluyendo los TimerEvent, se desactiva.

Ejemplo El código siguiente muestra un mensaje en la línea de estado y espera cinco segundos antes de mostrar un segundo mensaje.

```
; irAPausa::pushButton
method pushButton(var eventInfo Event)
var
  SuTurno SmallInt
endVar
SiTurno = 5000
beep()
message("El próximo mensaje aparecerá en 5 segundos.")
sleep(SuTurno)                      ; esperamos 5 segundos
message("Han transcurrido 5 segundos.")
endmethod
```

Vea también Form::wait

sound

Principiante

Procedimiento Crea un sonido de la frecuencia y duración especificadas.

Tipo System

Sintaxis **sound (** const **frecHercios** LongInt, const **duraciónMiliseg** LongInt **)**

Descripción Crea un sonido con la frecuencia (en hercios) *frecHercios* durante un tiempo (en milisegundos) especificado en *duraciónMiliseg*. Los valores de frecuencia pueden hallarse entre 1 y 50.000 hercios (el límite audible es aproximadamente 20.000 hercios).

Ejemplo En este ejemplo, el método **pushButton** de *hacerMúsica* declara primero un número de constantes para los valores de frecuencia en una escala. Estas notas se utilizan para especificar el argumento *frecuenciaHercios* en las ejecuciones del método **sound**. Después de reproducir algunas notas de una canción, el método demuestra el cálculo de las notas de una escala crómatica (avanza por medias notas).

```
; hacerMúsica::pushButton
method pushButton(var eventInfo Event)
var
  CuartoDeNota, octava, nota LongInt
endVar
; valores de frecuencia para las notas de una escala musical
const
  notaDO1  = 110
  notaDO#1 = 116
  notaRE1  = 123
  notaMI1  = 130
  notaMI#1 = 138
  notaFA1  = 146
  notaFA#1 = 155
  notaSOL1 = 164
  notaLA1  = 174
  notaLA#1 = 184
  notaSI1  = 195
  notaSI#1 = 207
  notaDO2  = 220
  notaDO#2 = 234
  notaRE2  = 249
  notaMI2  = 265
  notaMI#2 = 282
  notaFA2  = 300
endConst
; algunas notas de Pedro y el Lobo
sound(notaDO1, 200)
sound(notaFA1, 150)
sound(notaLA#1, 50)
sound(notaDO2, 100)
sound(notaRE2, 100)
sound(notaDO2, 150)

sound(notaLA#1, 50)
sound(notaDO2, 100)
```

```
sound(notaRE2, 100)
sound(notaMI#2, 150)
sound(notaFA2, 50)
sound(notaDO2, 100)
sound(notaLA#1, 100)
sound(notaFA1, 100)
sleep(1000)

; hacemos sonar unas pocas escalas cromáticas
CuartoDeNota = 120
for octava from 0 to 1
  for nota from 0 to 11
     sound(int(pow(2, octava + nota / 12.0) * 110),
CuartoDeNota)

   endFor
endFor
sound(int(pow(2, 2) * 110), CuartoDeNota) ; terminamos la
escala
endmethod
```

Vea también beep

sysInfo

Procedimiento Crea una matriz dinámica con información acerca del sistema en que se ejecuta Paradox.

Tipo System

Sintaxis **sysInfo (** var **info** DynArray[] AnyType **)**

Descripción Crea la matriz dinámica *info* con información sobre el sistema en que se ejecuta Paradox. Es necesario declarar el DynArray antes de activar este método. El DynArray contiene índices para los atributos del sistema y sus valores. Los índices se describen en la tabla siguiente.

Indice	Definición
AreMouseButtonsSwapped	Informa de si se han intercambiado las funciones de los botones izquierdo y derecho del ratón
CodePage	Informa de qué página de códigos está utilizando Windows
CPU	Tipo de procesador
FullHeight	Zona de trabajo vertical (en pixels) en una ventana maximizada
FullWidth	Zona de trabajo horizontal (en pixels) en una ventana maximizada
IconHeight	Altura de iconos (en pixels)
IconWidth	Anchura de iconos (en pixels)
KeyboardFNKeys	Número de teclas de función
KeyboardSubType	Subtipo de teclado,que es un valor dependiente de OEM
KeyboardType	Tipo y fabricante del teclado
MathCoprocessor	Informa de si hay presente un coprocesador matemático
Memory	Memoria disponible,incluyendo archivo de intercambio (si lo hay) en bytes
Mouse	Número de ratones conectados al sistema
NumTasks	Número de tareas activas (programas)
Protected	Informa de si el sistema se está ejecutando en modo protegido
ScreenHeight	Altura total de la pantalla (en pixels)
ScreenWidth	Anchura total de la pantalla (en pixels)
WindowsDir	Vía de acceso al directorio de Windows
WindowsSystemDir	Vía de acceso al directorio SYSTEM de Windows
WindowsVersion	Número de versión de Windows

Ejemplo El código siguiente escribe la información del sistema en una matriz dinámica llamada *Sistema* y, a continuación, muestra *Sistema* en un cuadro de diálogo de **view** (consulte también el ejemplo de pixelsToTwips.).

```
; mostrarInfoDeSistema::pushButton
method pushButton(var eventInfo Event)
var
  Sistema DynArray[] AnyType
endVar
sysInfo(Sistema)   ; rellenamos la matriz con información del
sistema
Sistema.view()     ; mostramos la matriz
```

```
endmethod
```

Vea también readProfileString

tracerClear

Procedimiento Borra la ventana de seguimiento.

Tipo System

Sintaxis **tracerClear ( )**

Descripción Vacía la ventana de seguimiento. Dicha ventana puede abrirse por el procedimiento **tracerOn** en el momento de la ejecución o por la selección del comando de menú Depurar | Seguimiento de la ejecución en el Editor de ObjectPAL.

Ejemplo El código siguiente vacía la ventana de seguimiento. Para este ejemplo, supóngase que la ventana de seguimiento está abierta y contiene información.

```
; vaciarSeguimiento::pushButton
method pushButton(var eventInfo Event)
tracerClear()                    ; vacía la ventana de
Seguimiento
endmethod
```

Vea también tracerHide
tracerOn

tracerHide

Procedimiento Oculta la ventana de seguimiento.

Tipo System

Sintaxis **tracerHide ( )**

Descripción Hace que la ventana de seguimiento quede invisible, pero no borra ni cierra la ventana. Para que esté visible de nuevo, utilice **tracerShow**.

Ejemplo El código siguiente oculta la ventana de seguimiento, realiza una pausa y muestra la ventana de nuevo. Para este ejemplo, supóngase que la ventana de seguimiento ya está abierta.

```
; cambiarSeguimientoWindows::pushButton
method pushButton(var eventInfo Event)
tracerHide()                    ; hacemos invisible la ventana
de seguimiento
message("Ocultando la ventana de seguimiento. En pausa...")
sleep(2000)
message("Mostrando la ventana de seguimiento.")
tracerShow()                    ; la hacemos visible de nuevo
tracerToTop()                   ; hacemos de ella la más
destacada
endmethod
```

Vea también tracerOn
tracerShow

tracerOff

Procedimiento Cierra la ventana de seguimiento.

Tipo System

Sintaxis **tracerOff ( )**

Descripción Deja de escribir en la ventana de seguimiento. Es posible reanudar el seguimiento del código con el procedimiento **tracerOn**. El seguimiento está activado por defecto cuando se abre la ventana de seguimiento por primera vez.

Ejemplo Este código desactiva el seguimiento del código.

```
; cerrarSeguimiento::pushButton
method pushButton(var eventInfo Event)
tracerOff()                    ; cerramos la ventana de
seguimiento
endmethod
```

Vea también tracerOn
tracerSave

tracerOn

Procedimiento Activa el seguimiento del código.

Tipo System

Sintaxis **tracerOn ( )**

Descripción Reanuda la escritura en la ventana de seguimiento.

Ejemplo Este ejemplo reactiva el seguimiento de código.

```
; iniciarSeguimiento::pushButton
method pushButton(var eventInfo Event)
tracerOn()                      ; reaactivamos la ventana de
seguimiento
endmethod
```

Vea también tracerOff
tracerShow

tracerSave

Procedimiento Guarda el contenido de la ventana de seguimiento en un archivo.

Tipo System

Sintaxis **tracerSave (** const ***nombreArchivo*** String **)**

Descripción Guarda el contenido de la ventana de seguimiento en *nombreArchivo*.

Ejemplo Este ejemplo guarda el contenido de la ventana de seguimiento en un archivo llamado SEGUIMIE.TXT.

```
; almacenarSeguimientoEnUnArchivo::pushButton
method pushButton(var eventInfo Event)
tracerSave("Seguimie.txt")      ; almacenamos la ventana de
seguimiento en un
                                ; fichero
endmethod
```

Vea también tracerWrite

tracerShow

Procedimiento Convierte la ventana de seguimiento en visible.

Tipo System

Sintaxis **tracerShow ( )**

Descripción Muestra la ventana de seguimiento. Es posible ocultarla mediante el procedimiento **tracerHide**.

Ejemplo Consulte el ejemplo de tracerHide..

Vea también tracerHide
tracerToTop

tracerToTop

Procedimiento Sitúa la ventana de seguimiento como ventana superior.

Tipo System

Sintaxis **tracerToTop ( )**

Descripción Sitúa la ventana de seguimiento como ventana superior del Escritorio.

Ejemplo Consulte el ejemplo de tracerWrite.

Vea también tracerHide
tracerShow

tracerWrite

Procedimiento Escribe un mensaje en la ventana de seguimiento.

Tipo System

Sintaxis **tracerWrite (**const ***mensaje*** String **[,** const ***mensaje*** String **]*)**

Descripción Escribe un mensaje en la ventana de seguimiento.

Ejemplo El código siguiente registra un mensaje en la ventana de seguimiento y, a continuación, la sitúa como capa superior del Escritorio.

```
; anotarMensajeDeSeguimiento::pushButton
method pushButton(var eventInfo Event)
tracerWrite("Dato de Seguimiento por " +
String(self.Name) +
             " a las " + String(time()))              ;
anotar un mensaje
tracerToTop()            ; hacer de Seguimiento la ventana
visible más destacada
endmethod
```

Vea también tracerSave

twipsToPixels

Procedimiento Convierte las coordenadas de pantalla de twips a pixels.

Tipo System

Sintaxis **twipsToPixels (** const **twips** Point **)** Point

Descripción Convierte las coordenadas de pantalla especificadas en *twips* de twips a pixels. Un pixel es un punto en la pantalla, mientras que twip es una unidad independiente del dispositivo que es igual a 1/1440 de pulgada (1/20 de un punto de la impresora).

Ejemplo Consulte el ejemplo de pixelsToTwips.

Vea también pixelsToTwips

version

Procedimiento Devuelve el número de versión de Paradox.

Tipo System

Sintaxis **version ( )** String

Descripción Devuelve una cadena que contiene el número de la versión de Paradox que se está utilizando.

Ejemplo En este ejemplo, el método **pushButton** de *mostrarVersión* indica al usuario qué versión de Paradox se está empleando.

```
; mostrarVersión::pushButton
method pushButton(var eventInfo Event)
msgInfo("", "Está usted ejecutando la versión " +
String(version()) + ".")
endmethod
```

Vea también sysInfo

winGetMessageID

Procedimiento Devuelve el ID de un mensaje de Windows.

Tipo System

Sintaxis **winGetMessageID (** const ***nombreMensaje*** SmallInt **)**

Descripción Obtiene el ID del mensaje **nombreMensaje**. Este método sólo deberían utilizarlo los programadores de Windows. Para más información, consulte la documentación de programación de Windows.

Vea también
winPostMessage
winSendMessage
MenuEvent**::**data
MenuEvent**::**id
Form**::**windowClientHandle
Form**::**windowHandle

winPostMessage

Procedimiento Consigna un mensaje a Windows.

Tipo System

Sintaxis **winPostMessage (** const ***hWnd*** SmallInt, const ***mensaje*** SmallInt, const ***wParam*** SmallInt, const ***lParam*** LongInt **)** Logical

Descripción Consigna un mensaje a Windows. Este método sólo deberían utilizarlo los programadores de Windows. Para más información, consulte la documentación de programación de Windows.

Vea también
winGetMessageID
winSendMessage
MenuEvent**::**data
MenuEvent**::**id
Form**::**windowClientHandle
Form**::**windowHandle

winSendMessage

Procedimiento Envía un mensaje a Windows.

Tipo System

Sintaxis **winSendMessage (** const ***hWnd*** SmallInt, const ***mensaje*** SmallInt, const ***wParam*** SmallInt, const ***lParam*** LongInt **)** Logical

Descripción Envía un mensaje a Windows. Este método sólo deberían utilizarlo los programadores de Windows. Para más información, consulte la documentación de programación de Windows.

Vea también
winGetMessageID
winPostMessage
MenuEvent**::**data
MenuEvent**::**id
Form**::**windowClientHandle
Form**::**windowHandle

writeEnvironmentString

Procedimiento Escribe en un archivo información sobre el de sistema del usuario.

Tipo System

Sintaxis **writeEnvironmentString (** const **clave** String, const **valor** String **)** Logical

Descripción Define una variable de entorno del DOS. Los valores se asignan a las variables de entorno mediante el comando SET del DOS. Controlan el aspecto y funcionamiento del DOS y de algunos programas y archivos de proceso por lotes (.BAT). Entre las variables de entorno más utilizadas, se incluyen PATH, PROMPT y COMSPEC. Para más información, consulte el manual del DOS, en especial el comando SET.

Ejemplo Consulte el ejemplo de readEnvironmentString.

Vea también readEnvironmentString
writeProfileString

writeProfileString

Procedimiento Escribe en un archivo información sobre el sistema del usuario.

Tipo System

Sintaxis **writeProfileString (** const **nombreArchivo** String, const **sección** String, const ***clave*** String, const **valor** String **)** Logical

Descripción Escribe un valor en una sección especificada de un archivo sobre el sistema del usuario. Normalmente, este método se empleará para modificar el archivo WIN.INI del usuario, por lo que *nombreArchivo* sería WIN.INI.

Una sección se marca en WIN.INI mediante una cadena incluida entre corchetes en una línea por sí sola (por ejemplo, [windows]). Sin embargo, al especificar *sección*, omita los corchetes; es decir, para especificar la sección [windows], utilice "windows".

Dentro de una sección, una cadena seguida de un signo igual especifica *clave* (por ejemplo, "Beep = "), pero no incluya el signo = al especificar *valor*. Especifíquelo escribiendo una cadena después del signo = en la clave.

Ejemplo Consulte el ejemplo de readProfileString

Vea también readProfileString
writeEnvironmentString

advMatch

Método Busca un patrón de caracteres en un archivo de texto.

Tipo TextStream

Sintaxis advMatch (var ***inicioIndice*** LongInt, var ***finIndice*** LongInt,
const ***patrón*** String **)** Logical

Descripción Busca en un archivo de texto un patrón de caracteres representado por la variable *patrón*. Si se asigna un valor a *inicioIndice*, la búsqueda comienza en la posición de inicioIndice; en caso contrario, la búsqueda se inicia al principio del archivo. La posición de *finIndice no* indica el final del rango en que se realiza la búsqueda. Si se encuentra el patrón, la posición del primer carácter coincidente se almacena en *inicioIndice* y la posición del último carácter coincidente se almacena en *finIndice*.

advMatch devuelve True si se encuentra *patrón* en el archivo; en caso contrario, devuelve False. Este método tiene en cuenta por defecto el uso de mayúsculas o minúsculas, pero es posible utilizar el procedimiento **ignoreCaseInStringCompares** de System para cambiar este comportamiento.

Si suministra patrón desde dentro de un método, debe emplear dos barras invertidas cuando desee indicar a advMatch que trate a un carácter especial como carácter literal; por ejemplo, \\ indica a advMatch que trate el paréntesis como carácter literal. Son necesarias dos barras invertidas en esta situación por la ambigüedad entre la interpretación de una barra invertida por parte del compilador (se emplea en secuencias de escape, como "\t" para un tabulador) y la interpretación de la barra invertida por parte de advMatch. Cuando el compilador ve una cadena con una secuencia de escape incluida, como "\tinicio", interpreta el "\t" como un tabulador, seguido de la palabra "inicio". El carácter de barra invertida tiene un significado especial para el compilador, pero también tiene un significado especial para advMatch (consulte la entrada de advMatch en el tipo String)

Si suministra *patrón* desde un campo o un TextStream, los símbolos especiales de **advMatch** se reconocen sin la barra invertida precedente, y una barra invertida y el símbolo más (\+) produce un carácter literal.

Para especificar *patrón*, utilice una cadena con los símbolos optativos enumerados en la tabla siguiente.

Símbolo	Corresponde con
\	Utilice la barra invertida para incluir cualquiera de los anteriores como caracteres normales (recuerde utilizar dos barras invertidas en las cadenas entrecomilladas).
[ ]	Corresponde con el grupo de caracteres incluidos entre los corchetes. Por ejemplo,[aeiou09] corresponde con a, e, i, o, u, y 0 a 9.
[^]	*No* corresponde con el grupo de caracteres incluidos entre los corchetes. Por ejemplo, [^aeiou09] corresponde con todo excepto a, e, i, o, u, y 0 a 9.
()	Agrupamiento.

^	Comienzo de línea (no lo confunda con [^], donde el ^ actúa como un NOT lógico).
$	Fin de cadena.
..	Corresponde con todo.
@	Corresponde con cualquier carácter individual.
*	Cero o más de la expresión o carácter anterior.
+	Una o más de la expresión o carácter anterior.
?	Ninguna o una de la expresión o carácter anterior.
|	Operación OR.

Ejemplo

Este ejemplo supone que un archivo llamado CITAPDX.TXT existe en el directorio de trabajo actual. El archivo contiene el texto siguiente:

Qué maravilla que hayamos encontrado Paradox.
Ahora tenemos alguna esperanza de progreso.

La ejecución de **advMatch** especifica "@o@e" como patrón que se busca. Este patrón corresponde a cualquier carácter seguido por una **o** seguida por cualquier carácter que, a su vez, está seguido por una **e**. Cuando se encuentra este patrón, las variables *PrimerCarácter* y *UltimoCarácter* contienen las posiciones del primer y último caracteres coincidentes. Las llamadas a **setPosition** y **readChars** leen los caracteres coincidentes y los almacenan en la variable *Equivalencia*.

```
; encontrarAlgunos::pushButton
method pushButton(var eventInfo Event)
var
  pdc                                TextStream
   PrimerCarácter, UltimoCarácter   LongInt
  Equivalencia                       String
endvar
if pdc.open("CitaPdx.txt", "R") then
  if pdc.advMatch(PrimerCarácter, UltimoCarácter, "@o@e") then
     msgInfo("La posición encontrada", PrimerCarácter)
     pdc.setPosition(PrimerCarácter)
      pdc.readChars(Equivalencia, UltimoCarácter   PrimerCarácter)
     message(Equivalencia)                          ; muestra "come"
  else
      msgInfo("Lo siento", "No he encontrado la equivalencia.")
  endIf
  pdc.close()
else
  msgInfo("Lo siento", "No puedo abrir el fichero de texto solicitado.")
endif
endmethod
```

Vea también

home
end
setPosition
position
String**::**advMatch

close

Método Cierra un archivo de texto.

Tipo TextStream

Sintaxis **close ( )** Logical

Descripción Cierra un archivo de texto y escribe el contenido de todas las memorias intermedias de texto en el disco. También termina la asociación entre una variable TextStream y el archivo de texto subyacente.

Ejemplo Este ejemplo declara una variable TextStream, *ts*, ejecuta **open** para asociar *ts* con el archivo de texto CITAPDX.TXT y, después, ejecuta **close** para terminar la asociación.

```
; citarUnaLinea::pushButton
method pushButton(var eventInfo Event)
var
  ts              TextStream
   PrimeraLinea String
endvar
ts.open("CitaPdx.txt", "R")
ts.readLine(PrimeraLinea)
PrimeraLinea.view("Línea 1 de CITAPDX.TXT")
ts.close()
endmethod
```

Vea también open
commit

commit

Método Escribe el contenido de la memoria intermedia de texto en el disco.

Tipo TextStream

Sintaxis **commit ( )**

Descripción Vacía la memoria intermedia de texto y escribe el contenido en el disco. El archivo permanece abierto y la posición del puntero en el archivo no cambia.

Ejemplo En este ejemplo, el botón *crearTexto* crea un nuevo archivo llamado MITEXTO.TXT, escribe una línea en él, almacena la versión actual del TextStream y cierra el archivo.

```
; crearTexto::pushButton
method pushButton(var eventInfo Event)
var
  ts TextStream
endVar

ts.create("MiTexto.txt")
msgInfo("Ahora la posición de TextStream es", ts.position()) ; muestra 1

ts.writeLine("Este es un texto.")
msgInfo("Ahora la posición de TextStream es", ts.position()) ; muestra 20

ts.commit()
msgInfo("Ahora la posición de TextStream es", ts.position()) ; aún es 20

endmethod
```

Vea también writeString

create

Método Crea un archivo de texto.

Tipo TextStream

Sintaxis **create (** const ***nombreArchivo*** String **)** Logical

Descripción Crea el archivo de texto *nombreArchivo* y lo abre para lectura y escritura. Si *nombreArchivo* existe, **create** lo sustituye sin pedir confirmación. Es posible especificar un directorio en que se cree el archivo utilizando una vía de acceso del DOS completa o un alias. Si no se especifica una vía de acceso ni un alias, Paradox crea el archivo en el directorio de trabajo (:TRABAJO:).

Este método devuelve True si es satisfactorio; en caso contrario, devuelve False. Si se logra crear el archivo, se abre para lectura y escritura.

Nota: Las sentencias siguientes son equivalentes:

```
ts.create("NueTexto.txt")
ts.open("NueTexto.txt, "NW")
```

Ejemplo El código siguiente se anexa al método **pushButton** de un botón. Consta de un bloque de declaración de variable, una declaración de procedimiento y el cuerpo del método. En el cuerpo del método, la llamada al método **findFirst** de FileSystem comprueba la existencia de un archivo llamado JUAN.TXT. Si no existe, el procedimiento personalizado **addLine** lo crea y añade una línea en él. Si el archivo existe, un cuadro de diálogo confirma la decisión de sustituir el archivo.

```
; crearFichero::pushButton
var
  ts                  TextStream
   PrimeraLinea       String
   TodasLasLíneas     Array[] String
  fs                  FileSystem
endvar

proc addLine()
  ts.create(":PERSONAL:Juan.txt")          ; crea el fichero, lo abre para lectura y
                    ; escritura
  ts.writeLine("Aquí hay algo para tí, muchacho.")
  ts.home()
  ts.readLine(TodasLasLíneas)
  TodasLasLíneas.view("Juan dice:")
endProc

method pushButton(var eventInfo Event)
if not fs.findFirst(":PERSONAL:Juan.txt") then
  addLine()
else
    if msgYesNoCancel(":PERSONAL:Juan.TXT", "¿Sobreescribir este fichero?") =
                          "Yes" then
       addLine()
     endif
endif
```

```
endmethod
```

Vea también open
close

end

Método Sitúa el puntero al final de un archivo de texto.

Tipo TextStream

Sintaxis **end ( )**

Descripción Sitúa el puntero de archivo en el último carácter de un archivo de texto.

Ejemplo Este ejemplo supone que existe un archivo llamado CITAPDX.TXT en el directorio de trabajo actual. El archivo contiene el texto siguiente:

```
Qué maravilla que hayamos encontrado Paradox.
Ahora tenemos alguna esperanza de progreso.
```

El código de este ejemplo se anexa al método estándar **newValue** de un objeto de campo mostrado como dos botones de radio. Los valores de los botones de radio son "Overwrite" y "Append". Se elige uno para especificar si se inserta texto al principio del archivo (lo cual sustituye al texto existente) o se añade al final del mismo. Si se elige "Overwrite", la llamada a **home** desplaza el puntero a la posición 1. Si se elige "Append", la llamada a **end** desplaza el puntero a la posición 105 (final de este archivo en concreto).

```
; insertarAñadirCampo::changeValue
method newValue(var eventInfo Event)
var
  ts TextStream
  TodasLasLíneas Array[] String
endVar
if eventInfo.reason() = EditValue then
  ts.open(":PERSONAL:Citapdx.txt", "W")
  switch
     case self.value = "Overwrite" :
          ts.home()
           ts.writeLine(DateTime())              ; Escribimos la fecha y la hora en el
           ; principio del fichero
           ; el fichero leerá:
           ; Fecha y hora
           ; Qué maravilla que hayamos encontrado Paradox
           ; Ahora tenemos alguna esperanza de progreso
     case self.value = "Append" :
          ts.end()
           ts.writeLine(DateTime())    ; Escribimos la fecha y la hora en el final
           ; del fichero
           ; el fichero leerá:
           ; Qué maravilla que hayamos encontrado Paradox
           ; Ahora tenemos alguna esperanza de progreso
           ; Fecha y hora
    endSwitch
    ts.home()
     ts.readLine(TodasLasLíneas)
    TodasLasLíneas.view()
    ts.close()
endif
```

```
endmethod
```

Vea también home
setPosition
eof

eof

Método Comprueba si se intenta un desplazamiento más allá del final de un archivo de texto.

Tipo TextStream

Sintaxis **eof ( )** Logical

Descripción Devuelve True si una operación intenta desplazar el puntero de archivo más allá del final de un archivo de texto; en caso contrario, devuelve False.

Ejemplo Este ejemplo supone que existe un archivo llamado CITAPDX.TXT en el directorio de trabajo actual. El archivo contiene el texto siguiente:

Qué maravilla que hayamos encontrado Paradox.
Ahora tenemos alguna esperanza de progreso.

El bucle **while** lee cada una de las dos líneas del archivo y lo muestra en un cuadro de diálogo. Entonces, **eof** devuelve True, y un cuadro de diálogo indica al usuario que no hay más texto en el archivo.

```
; linea::pushButton
method pushButton(var eventInfo Event)
var
  pdc           TextStream
  LineaTexto String
endVar

pdc.open(":PERSONAL:CitaPdx.txt", "r")
while not pdc.eof()              ; salimos del bucle al alcanzar el final del fichero
   pdc.readLine(LineaTexto)    ; leemos la siguiente línea
  msgInfo("Posición " + String(pdc.position()), LineaTexto)
endWhile
msgInfo("Finalizado", "No hay más texto")
endmethod
```

Vea también end
position
setPosition

home

Método	Sitúa el puntero al principio de un archivo de texto.
Tipo	TextStream
Sintaxis	**home( )**
Descripción	Sitúa el puntero de archivo en el primer carácter de un archivo de texto.
Ejemplo	Consulte el ejemplo de end.
Vea también	end eof setPosition

open

Método Abre un archivo de texto en un modo especificado.

Tipo TextStream

Sintaxis **open (** const ***nombreArchivo*** String, const ***modo*** String **)** Logical

Descripción Abre *nombreArchivo* en el modo especificado en *modo*, y asocia una variable FileSystem con el archivo subyacente.

Modo	Derechos
a	Adición/lectura
r	Sólo lectura
w	Escritura/lectura
nw	Nuevo/escritura/lectura

Si el archivo existe, el modo Nuevo/escritura/lectura lo sustituye sin pedir confirmación.

Nota: Las sentencias siguientes son equivalentes:

```
ts.open("Nuevo.txt", "NW")
ts.create("Nuevo.txt")
```

La apertura de un archivo en cualquier modo *excepto* Adición/lectura sitúa el puntero al principio del archivo.

Es posible especificar un directorio desde el que se abra el archivo mediante una vía de acceso del DOS completa o un alias. Si no se especifica una vía de acceso ni un alias, Paradox busca el archivo en el directorio de trabajo.

Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Ejemplo Este ejemplo utiliza un alias con **open** para crear un archivo de texto en el directorio personal y escribir una línea de texto en él:

```
var
   ts TextStream
endVar
if ts.open(":PERSONAL:memo14.txt", "NW") then
   ts.writeLine("¡Esto es privado!")
endIf
```

Es posible asociar más de una variable TextStream con el mismo archivo. Ambas variables tienen iguales derechos sobre el archivo, y Paradox mantiene punteros separados para cada variable. El ejemplo siguiente declara dos variables TextStream, *ts1* y *ts2*, y ejecuta **open** para asociar cada una de ellas con el archivo de texto NUETEXTO.TXT. Conforme se añaden sentencias en el archivo, aparecen mensajes que muestran la posición del puntero para cada variable.

```
; abrirFicheros::pushButton
```

```
method pushButton(var eventInfo Event)
var
   ts1, ts2    TextStream
  PrimeraLinea String
  TodasLasLíneas Array[] String
endvar
ts1.open("NueTexto.txt", "nw")              ; abrimos un nuevo fichero para
                    ; lectura/escritura
ts1.writeLine("Escrito por ts1.")
ts1.writeLine("Esta es la línea 2.")
msgInfo("Fichero de texto uno", ts1.position()) ; muestra 40
ts1.commit()                                ; lo almacenamos en el disco, de ese modo
                    ; ts2 obtendrá la versión más actualizada

ts2.open("NueTexto.txt", "w")               ; abrimos el fichero existente para
                    ; lectura/escritura
msgInfo("Fichero de texto uno", ts1.position())     ; muestra 40
msgInfo("Fichero de texto dos", ts2.position())     ; muestra 1

ts2.writeLine("Escrito por ts2.")
msgInfo("Fichero de texto uno", ts1.position())     ; muestra 40        msgInfo("Fichero de texto
dos", ts2.position())      ; muestra 19

ts1.home()
ts1.readLine(TodasLasLíneas)                ; Lee todas las líneas
TodasLasLíneas.view("ts1")                  ; muestra:
                    ; Escrito por ts2.
                    ; Esta es la línea 2.

endmethod
```

Vea también create
close

position

Método Devuelve la posición del puntero en un archivo de texto.

Tipo TextStream

Sintaxis **position ( )** LongInt

Descripción Devuelve un entero que representa la posición del puntero en un archivo de texto. **position** cuenta tanto los caracteres imprimibles como los no imprimibles. La numeración comienza en 1 (no en 0).

Ejemplo Puede ser útil considerar que **position** devuelve el número del siguiente carácter del archivo. Como muestra este ejemplo, cuando se crea un nuevo archivo de texto y se ejecuta **position**, éste devuelve 1. La llamada a **writeLine** añade 14 caracteres en el archivo: 12 caracteres imprimibles y el par CR/LF (retorno de carro y avance de línea). El carácter siguiente será 15, por lo que **position** devuelve 15.

```
var
NuevoFichero TextStream
endVar
NuevoFichero.open("NueMemo.txt", "nw")
message(NuevoFichero.position())          ; muestra 1
sleep(1000)
NuevoFichero.writeLine("No te asustes.")
message(NuevoFichero.position())          ; muestra 17
                                ; 14 caracteres imprimibles más CR/LF = 16
                                ; el próximo carácter será el 17
sleep(1000)
```

Vea también setPosition
size

readChars

Método Lee un número especificado de caracteres de un archivo de texto.

Tipo TextStream

Sintaxis **readChars (** var ***cadena*** String, const ***nCaracteres*** SmallInt **)** Logical

Descripción Lee el número de caracteres especificado en *nCaracteres* y los almacena en *cadena*. **readChars** comienza a leer desde la posición actual del puntero. Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Ejemplo Este ejemplo supone que existe un archivo llamado CITAPDX.TXT en el directorio de trabajo actual. El archivo contiene el texto siguiente:

Qué maravilla que hayamos encontrado Paradox.
Ahora tenemos alguna esperanza de progreso.

La llamada a **readChars** lee los 100 primeros caracteres del archivo:

```
; obtenerLetras::pushButton
method pushButton(var eventInfo Event)
var
  Fichero TextStream
  MisCaracteres String
endVar
Fichero.open("CitaPdx.txt", "r")
if Fichero.readChars(MisCaracteres, 100) then
  msgInfo("Los primeros 100 caracteres son:", MisCaracteres)
endIf
endmethod
```

Vea también readLine
setPosition
El ejemplo de advMatch.

readLine

Método Lee una línea o varias líneas de un archivo de texto.

Tipo TextStream

Sintaxis **readLine (** var ***valor*** String **)** Logical **readLine (** var ***matrizCadena*** Array[] String **)** Logical

Descripción Lee caracteres de una línea de un archivo de texto hasta que se encuentra un par CR/LF (retorno de carro y avance de línea) y desplaza el puntero de archivo a la primera posición después del par CR/LF. **readLine** comienza a leer desde la posición actual del puntero.

Es posible almacenar una sola línea en *valor* o almacenar todo el archivo en *matrizCadena*, donde es una matriz redimensionable de cadenas y cada elemento de la matriz almacena una línea del archivo. En cualquier caso, el par CR/LF no se almacena.

Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Ejemplo El primer ejemplo crea un archivo de texto con dos líneas y, a continuación, ejecuta **readLine** para leer la primera línea en una variable String. **readLine** lee los cuatro caracteres situados antes del CR/LF de la primera línea, se salta los caracteres CR/LF y sitúa el puntero.

```
method pushButton(var eventInfo Event)
var
  ts      TextStream
  Linea String
endvar
ts.create("NueTexto.txt")
ts.writeLine("1234")
ts.writeline("5678")
ts.home()

ts.readLine(Linea)
message(Linea.size())              ; muestra un 4 (no incluye CR/LF)
sleep(1234)
message(ts.position())             ; muestra un 7 (pasa sobre CR/LF)
sleep(1234)
endmethod
```

El ejemplo siguiente crea un archivo de texto de tres líneas, ejecuta **readLine** para leer todo el archivo en una matriz y la muestra en un cuadro de diálogo.

```
var
  Fichero                    TextStream
  TodasLasLineas Array[]   String
endVar

Fichero.open("Fichero.txt", "nw")
Fichero.writeLine("Estimado Cliente,")
Fichero.writeLine("Gracias por su interÉs en nuestro producto.")
Fichero.writeLine("Un representante le llamará la próxima semana.")
```

```
Fichero.home()                    ; desplaza el puntero al principio del fichero

Fichero.readLine(TodasLasLineas)
TodasLasLineas.view("Fichero completo")          ; muestra el fichero entero
Fichero.close()
```

Vea también readChars
writeLine

setPosition

Método Sitúa el puntero en un archivo de texto.

Tipo TextStream

Sintaxis **setPosition (** const ***desplazamiento*** LongInt **)**

Descripción Sitúa el puntero de archivo en el número de caracteres indicado en *desplazamiento* desde el principio de un archivo de texto. Los caracteres de retorno de carro y avance de línea (CR/LF) se consideran parte del archivo, y es posible sobrescribirlos. La especificación de una posición antes del comienzo o después del final del archivo sitúa el puntero en el extremo correspondiente.

Ejemplo En este ejemplo, el botón *mostrarPosiciones* escribe una línea en un nuevo archivo de texto, MEMO.TXT. El método retrocede entonces al cuarto carácter, sustituye ese carácter por "4", lee la línea de nuevo y la muestra.

```
; mostrarPosiciones::pushButton
method pushButton(var eventInfo Event)
var
  Fichero   TextStream
  LineaUno String
endVar
Fichero.open(":PERSONAL:memo.txt", "nw")      ; abrimos el fichero nuevo para
                                    ; lectura/escritura
Fichero.writeLine("1235")                     ; 4 caracteres más CR/LF
msgInfo("¿Dónde estoy?", Fichero.position())  ; muestra un 7
Fichero.setPosition(4)                                ; nos desplazamos al carácter 4
Fichero.writeString("4")                              ; ahora la línea es "1234"
Fichero.home()                                ; idéntico a la función
                                          ; setPosition(1)
Fichero.readLine(LineaUno)
msgInfo("Esta es la primera línea", LineaUno)      ; muestra "1234"
endmethod
```

Este ejemplo muestra lo que sucede cuando se intenta desplazar el puntero más allá del final de un archivo o antes del principio del mismo.

```
; mostrarPosiciones::pushButton
method pushButton(var eventInfo Event)
var
  Fichero   TextStream

endVar
Fichero.open(":PERSONAL:memo.txt", "r")       ; abrimos el fichero existente para
                          ; lectura
Fichero.setPosition(100)                       ; más allá del final del fichero
msgInfo("Fin", Fichero.position())            ; muestra un 7 (el final real)
Fichero.setPosition(100)                      ; antes del principio del fichero
msgInfo("Inicio", Fichero.position())      ; muestra un 1 (el principio)
endmethod
```

Vea también position
size
eof

size

Método Devuelve el número de caracteres existentes en un archivo de texto.

Tipo TextStream

Sintaxis **size ( )** LongInt

Descripción Devuelve el número de caracteres de un archivo de texto, incluidos los caracteres no imprimibles como retorno de carro y avance de línea (CR/LF).

Ejemplo Este ejemplo crea un TextStream, escribe una línea en él y muestra el tamaño del archivo.

```
; mostrarTamaño::pushButton
method pushButton(var eventInfo Event)


var
  Texto TextStream
endVar
Texto.create("Corto.txt")
Texto.writeLine("1234")
msgInfo("¿Qué tamaño tengo?", Texto.size())        ; muestra un 6
; 4 caracteres imprimibles "1234", y dos caracteres no imprimibles CR y LF
Texto.close()
endmethod
```

Vea también position
setPosition
eof

writeLine

Método	Escribe una cadena en un archivo de texto.
Tipo	TextStream
Sintaxis	**writeLine (** const ***valor*** AnyType **[,** const ***valor*** AnyType **] *)** Logical
Descripción	Escribe una lista de *valores* separada por comas en un archivo de texto y añade un par CR/LF (retorno de carro y avance de línea). Compare este método con **writeString**, que no añade un par CR/LF.
Ejemplo	Consulte el ejemplo de create.

Vea también readLine
writeString

writeString

Método Escribe una cadena de caracteres en un archivo de texto.

Tipo TextStream

Sintaxis **writeString (** const ***valor*** AnyType **[,** const ***valor*** AnyType **] *)** Logical

Descripción Escribe una lista de *valores* separada por comas en un archivo de texto, pero no añade un par CR/LF (retorno de carro y avance de línea). Compare este método con **writeLine**, que sí añade un par CR/LF.

Ejemplo El ejemplo siguiente asigna cadenas a las variables *Bajo* y *Alto*, y utiliza **writeString** para escribirlas en un TextStream abierto.

```
; buenAviso::pushButton
method pushButton(var eventInfo Event)
var
   Texto          TextStream
  Bajo, Alto    String
endVar
Bajo = "Comprar bajo."
Alto = "Vender alto."
Texto.open(":PERSONAL:Aviso.txt", "nw")                    ; abrimos un nuevo fichero
Texto.writeString(Bajo, Alto)
msgInfo("Tamaño del fichero:", string(Texto.size()))       ; muestra un 25
; Comprar bajo. = 13, Vender alto. = 12
Texto.close()
endmethod
```

Vea también writeLine

blank

Principiante

Método/ procedimiento Devuelve un valor vacío.

Tipo AnyType

Sintaxis **1.** (Método) **blank ( )**

2. (Procedimiento) **blank ( )** AnyType

Descripción Genera un valor vacío para su asignación a una variable o campo. Un valor vacío no es lo mismo que un valor numérico de cero, pero es posible utilizar el método **blankAsZero** del tipo Session para tratar los valores vacíos como ceros en ciertos cálculos. El método **isBlankZero** del tipo Session puede utilizarse para averiguar si Blanco=Cero está activado o desactivado.

Ejemplo Este ejemplo supone que una ficha tiene un marco de tabla asociado con la tabla Articulo y un botón llamado esteBotón. Cuando un usuario hace clic sobre esteBotón, el código examina el campo Cantidad de Articulo y sustituye los valores no vacíos por valores vacíos. Este código se anexa al método estándar **pushButton** de esteBotón.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   tc TCursor
endVar

if tc.attach(ARTICULO) then; asocia tc a la tabla
  tc.edit(); edita la tabla
  scan tc for tc.cantidad.isBlank() = False : ; busca valores
distintos : de blanco en el campo cantidad
  tc.Cantidad.blank()   ; y rellena el campo con valor blanco
  endScan
  tc.endEdit() ; finaliza el modo edición
endif

endmethod
```

Vea también isBlank

dataType

Método Devuelve una cadena que representa el tipo de datos de una variable.

Tipo AnyType

Sintaxis **dataType** ( ) String

Descripción Devuelve una cadena que representa el tipo de datos de una variable o expresión: Binary, Currency, Date, DateTime, Graphic, Logical, LongInt, OLE, Memo, Number, Point, SmallInt, String o Time. En las sentencias de comparación, es necesario emplear uno de los valores de cadena mostrados aquí. Por ejemplo, lo siguiente está codificado incorrectamente porque compara String con string.

```
var s AnyType endVar
s = Este es un dato de tipo String."
msgInfo("Comprobación", s.dataType() = "string")  ; muestra
False- debería utilizarse "String"
```

Nota: Este método funciona para todos los tipos de ObjectPAL, no sólo para AnyType.

Ejemplo Este ejemplo supone que una ficha tiene un botón y un campo gráfico llamado *campoBMP*. El código siguiente carga un DynArray con varios tipos de datos diferentes y, a continuación, utiliza **dataType** para mostrar el tipo de datos de cada valor del DynArray. Este código se anexa al método estándar **pushButton** del botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  mezclaTipos DynArray[] AnyType
endVar

mezclaTipos["Marca"]  = "Ford"; cadena
mezclaTipos["Modelo"] = "Cobra"; cadena
mezclaTipos["Año"]    = 1969; SmallInt (no se trata como fecha)
mezclaTipos["Color"]  = Black; LongInt - aquí se usa como
constante
mezclaTipos["Foto"]   = campoBMP.value ; Gráfico

forEach elemento in mezclaTipos; visualiza un mensaje para cada
elemento

    msgInfo("Tipo de dato(" + elemento + ")",
dataType(mezclaTipos[elemento]))

endForEach

endmethod
```

Vea también isAssigned

isAssigned

Método Informa de si se ha asignado un valor a una variable.

Tipo AnyType

Sintaxis **isAssigned** ( ) Logical

Descripción Devuelve True si se ha asignado un valor a la variable; en caso contrario, devuelve False.

Nota: Este método funciona para todos los tipos de ObjectPAL, no sólo para AnyType.

Ejemplo Este ejemplo utiliza **isAssigned** para comprobar el valor de i antes de asignarle un valor. Si i ya tiene un valor asignado, este código incrementa i en uno. El código siguiente va en la ventana Var de un botón:

```
;esteBotón::var
var
  i smallInt
endVar
```

Este código se anexa al método estándar **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)

if i.isAssigned() then ; si i tiene un valor
   i = i + 1           ; incrementa i
else
   i = 1               ; en otro caso, inicializa i a 1
endif
                       ; y ahora muestra el valor de i
message("El valor de i es : " + String(i))

endmethod
```

Vea también dataType
unAssign

isBlank

Principiante

Método Informa de si una expresión tiene un valor vacío.

Tipo AnyType

Sintaxis **isBlank** ( ) Logical

Descripción Devuelve True si la expresión tiene un valor vacío; de lo contrario, devuelve False. Los valores de cadena vacía se denotan mediante "". Otros valores vacíos pueden generarse utilizando **blank**. Observe que los valores vacíos no equivalen a 0, a espacios (" ") ni a valores sin asignar.

Ejemplo El código siguiente (anexado al método **pushButton** de un botón) utiliza **isBlank** para comprobar varios valores y muestra los resultados en un cuadro de diálogo:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)

msgInfo("¿Está la cadena vacía en blanco?", isBlank(""))  ;
Verdadero
msgInfo("¿Está la cadena de espacios en blanco?", isBlank("
")); Falso
msgInfo("¿Tiene 5 blancos?", isBlank(5))  ; Falso
msgInfo("¿Es blanco blanco?", isBlank(blank()))° ; Verdadero

endmethod
```

Vea también blank

isFixedType

Método Informa de si se ha declarado explícitamente el tipo de datos de una variable.

Tipo AnyType

Sintaxis **isFixedType** ( ) Logical

Descripción Devuelve True si se ha declarado la variable utilizando un bloque **var**...**endvar**; en caso contario, devuelve False.

Ejemplo El código siguiente demuestra cuándo **isFixedType** devuelve True. Este código se anexa al método estándar **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   x SmallInt                  ; declara x
endVar

message(x.isFixedType())       ; Visualiza True
sleep(2000)

pruébame = 4                   ; pruébame no ha sido declarada
message(pruébame.isFixedType()); Visualiza False

endmethod
```

Vea también dataType
isAssigned

unAssign

Método Define el estado de una variable como no asignada.
Tipo AnyType

Sintaxis **unAssign** ( ) Logical

Descripción Define el estado de una variable como no asignada. Dicho estado no equivale a un valor de cero ni a Blank.

Ejemplo El ejemplo siguiente muestra el uso de **unAssign**. Este código se anexa al método **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  x AnyType
endVar

msgInfo("¿Tiene la variable x valor asignado?", x.isAssigned())
; Visualiza False
x = 5
msgInfo("¿Tiene la variable x valor asignado?", x.isAssigned())
; Visualiza True
x.unAssign()
msgInfo("¿Tiene la variable x valor asignado?", x.isAssigned())
; Visualiza False
endmethod
```

Vea también blank
isAssigned

view

Principiante

Método Muestra el valor de una variable en un cuadro de diálogo.

Tipo AnyType

Sintaxis **view** ([const ***título*** String])

Descripción Muestra el valor de una variable en un cuadro de diálogo modal. La ejecución de ObjectPAL se suspende hasta que el usuario cierra este cuadro de diálogo. Existe la opción de especificar, en título, un título para el cuadro de diálogo. Si no se especifica un título, aparece el tipo de datos de la variable.

El usuario puede cambiar el valor mostrado en un cuadro de diálogo de **view**, siempre que el tipo de datos no sea Array, DynArray o Record. **view** no puede mostrar los AnyType Binary, Graphic, Memo ni OLE. La tabla siguiente muestra los AnyType que pueden mostrarse o modificarse.

Tipo	Puede mostrarse	Puede modificarse
Array	sí	no
Binary	no	no
Currency	sí	sí
Date	sí	sí
DateTime	sí	sí
DynArray	sí	no
Graphic	no	no
Logical	sí	sí
LongInt	sí	sí
Memo	no	no
Number	sí	sí
OLE	no	no
Point	sí	sí
Record	sí	no
SmallInt	sí	sí
String	sí	sí
Time	sí	sí

Ejemplo Este ejemplo usa **view** para mostrar en un cuadro de diálogo el valor de varias variables. Este código se anexa al método estándar **pushButton** de un botón:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  dy DynArray[] AnyType
  x AnyType
endVar

MatrizDin["Nombre"]   = "Paco Pascual"
MatrizDin["Contrato"] = Date("5/2/84")
MatrizDin["C.Postal"] = 28044
MatrizDin["Titulación"] = "Ingeniero"
```

```
MatrizDin.view(); Visualiza índices y valores del DynArray

x = 5
x.view() ; Visualiza 5
x = "Hola"
x.view("Saludo") ; Visualiza Hola, el título es saludo
endmethod
```

El ejemplo siguiente emplea un cuadro de diálogo de **view** para solicitar una fecha al usuario. Si el usuario introduce una fecha válida, el código muestra el día de la semana de esa fecha; en caso contrario, se muestra un mensaje de error.

```
;verfecha::pushButton
method pushButton(var eventInfo Event)
var
   laFecha AnyType
   MatrizDías Array[7] String
endVar

MatrizDías[1] = "Domingo"
MatrizDías[2] = "Lunes"
MatrizDías[3] = "Martes"
MatrizDías[4] = "Miércoles"
MatrizDías[5] = "Jueves"
MatrizDías[6] = "Viernes"
MatrizDías[7] = "Sábado"

laFecha = today()
 ; inicializa la variable laFecha
 laFecha.view("Introduzca fecha nueva")
 ; muestra la fecha de hoy en una caja de diálogo ; y solicita
del usuario una nueva fecha

 ; es posible que el usuario introduzca una fecha inválida
(como "Sábado")
 ; por lo que el bloque try..fail intentará convertir laFecha a
Fecha usando la función
 ; dateVal() y, si es correcta, visualizará el correspondiente
día de la semana
 ; debe tenerse en cuenta que la función dowOrd devuelve un 1
para el domingo
try
   msgInfo("Día de la semana", "El " + String(laFecha) + " cae
en " +
           MatrizDías[dowOrd(dateVal(laFecha))])
onfail
   msgStop("¡Error!", laFecha + " no es correcta.")
endtry
endmethod
```

Vea también isAssigned
unAssign

addLast

Método Inserta un elemento al final de una matriz redimensionable.

Tipo Array

Sintaxis **addLast** (const ***valor*** AnyType)

Descripción Inserta valor después del último elemento de una matriz redimensionable. La matriz crece, si es necesario, para hacer sitio para el nuevo elemento. Si es necesario añadir más de un elemento en una matriz, suele ser preferible utilizar **grow** o **setSize** para asignar más espacio en la matriz que varias sentencias **addLast**. Por ejemplo, el código siguiente emplea **addLast** en un bucle **for** para añadir 10 nuevos elementos en la matriz *matriz*. Obsérvese que este uso de **addLast** obliga a ObjectPAL a reasignar el espacio en la matriz 10 veces: una vez por cada proceso del bucle.

```
for i from 11 to 20
  matriz.addLast(i * 10)
endfor
```

El código siguiente logra el mismo resultado que el anterior, pero se ejecuta con mayor rapidez porque ObjectPAL asigna el espacio una sola vez:

```
ar.grow(10)    ; incrementa en 10 elementos la longitud de la
matriz
for i from 11 to 20
  matriz[i] = (i * 10)
endfor
```

Ejemplo Este ejemplo añade un elemento en una matriz redimensionable cada vez que se pulsa *esteBotón*. El método **pushButton** de *esteBotón* aumenta el valor del elemento más reciente en diez y muestra el contenido de la matriz en un cuadro de diálogo de **view**. El código que sigue a continuación va en la ventana Var de *esteBotón*:

```
; esteBotón::Var
var
  Matriz Array[] SmallInt  ; declaramos Matriz como una matriz
redimensionable
  i  SmallInt              ; variable que se incrementa
endVar
```

El código siguiente se anexa al método estándar **pushButton** de *esteBotón*:

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)

                           ; inicializa o incrementa la
variable i
i = iif(isAssigned(i), i + 10, 0)

if Matriz.size() = 0 then  ; verdadero si es la primera vez que
se pulsa el botón
  Matriz.setSize(0)        ; inicializa  Matriz con 0 elementos
```

```
endif

Matriz.addLast(i)          ; añade otro elemento a Matriz y le
asigna
     el valor de i

  ; visualiza la longitud de la Matriz en el título
  ; y el valor de cada elemento dentro de la caja de diálogo
Matriz.view("El tamaño de la matriz es " +
strVal(Matriz.size()))

endmethod
```

Vea también append
insert
insertAfter
insertBefore
insertFirst

append

Método Añade el contenido de una matriz en otra.

Tipo Array

Sintaxis **append** (const ***matrizNueva*** Array[] String)

Descripción Añade los elementos de matrizNueva en una matriz redimensionable. La matriz crece, si es necesario, para albergar los elementos añadidos.

Ejemplo El código siguiente crea dos matrices redimensionables, *MatrizAñadir* y *MatrizBase*, y las carga con valores numéricos. Este ejemplo demuestra el uso de **append** mediante la adición de la matriz *MatrizAñadir* en *MatrizBase*, y muestra, a continuación, el resultado en un cuadro de diálogo de **view**. Este código se anexa al método estándar **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   MatrizBase, MatrizAñadir Array[] SmallInt
   i SmallInt
endVar

MatrizBase.setSize(3)
MatrizAñadir.setSize(3)            ; ahora ambas Matrices pueden
almacenar 3 valores
for i from 1 to 3
   MatrizBase[i] = i  ; MatrizBase[1] = 1, [2] = 2, [3] = 3
   MatrizAñadir[i] = (i + 3)      ; MatrizAñadir[1] = 4, [2] =
5, [3] = 6
endFor

MatrizBase.append(MatrizAñadir)   ; añade MatrizAñadir a
MatrizBase
                                  ; incrementando a 6 los
elementos de MatrizBase

   ; muestra la longitud de MatrizBase en el título
   ; y los elementos en el interior de la caja de diálogo
MatrizBase.view("Longitud de MatrizBase : " +
strVal(MatrizBase.size()))
endmethod
```

Vea también addLast
insert
insertAfter
insertFirst

contains

Método Busca un patrón de caracteres en los elementos de una matriz.

Tipo Array

Sintaxis **contains** (const ***valor*** AnyType) Logical

Descripción Devuelve True si algún elemento de una matriz corresponde exactamente a valor; de lo contrario, devuelve False.

Ejemplo Este ejemplo define y carga una matriz redimensionable llamada *perros* cuando se abre una ficha. Una vez que el método **open** de la ficha carga la matriz con nombres de perros, el código muestra el contenido de la matriz en un cuadro de diálogo. Un botón de la ficha contiene código que utiliza el método **contains** para buscar en la matriz un nombre específico. Si **contains** no encuentra el nombre, el método estándar **pushButton** anexado al botón utiliza **insertFirst** para añadir el nombre al principio de la matriz.

El código siguiente se anexa a la ventana Var de la ficha:

```
;estaFicha::Var
var
   perros Array[] String    ; matriz redimensionable
endVar
```

El código siguiente se anexa al método estándar **open** de la ficha:

```
;estaFicha::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
   then
      ;este código fuente lo ejecutan todos los objetos de la
Ficha
   else
      ;este código fuente lo ejecuta sólo la propia Ficha

      perros.setSize(4)     ; ahora perros puede almacenar
cuatro valores
      perros[1] = "Canelo"  ; añadimos algunos nombres de perro
      perros[2] = "Colín"
      perros[3] = "Turco"
      perros[4] = "Sultán"

         ; muestra el contenido de la matriz perros en una caja
de diálogo
      perros.view("perros se ha inicializado con estos
valores")
endif
endmethod
```

Este código se anexa al método **pushButton** del botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)

if perros.contains("Pirata") = False then
```

```
   perros.insertFirst("Pirata")   ; añade un nombre nuevo al
comienzo de la Matriz
                                  ; muestra el contenido de la
matriz perros
                                  ; en una caja de diálogo
   perros.view("número de perros : " + strVal(perros.size()))
else                              ; "Pirata" ya existe
   msgInfo("Con uno es suficiente", "La Matriz de perros ya
incluye a Pirata.")
endif

endmethod
```

Vea también countOf

countOf

Método Cuenta las veces que un valor aparece en una matriz.

Tipo Array

Sintaxis **countOf** (const ***valor*** AnyType) LongInt

Descripción Compara valor con cada elemento de una matriz y devuelve el número de correspondencias exactas, o 0 si no se ha encontrado ninguna.

Ejemplo Este código (anexado al método **pushButton** de un botón) crea y carga una matriz fija, y utiliza **countOf** para mostrar el número de valores iguales de la matriz:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   zoo Array[4] String
   i   SmallInt
endVar
for i from 1 to 3
   zoo[i] = "gato"    ; añade tres veces el valor "gato"endFor
zoo[4] = "perro"      ; añade una vez el valor "perro"

msgInfo("¿Cuántos gatos?", zoo.countOf("gato"))   ; muestra un
3
msgInfo("¿Cuántos perros?", zoo.countOf("perro")) ; muestra un
1
msgInfo("¿Cuántos monos?", zoo.countOf("mono"))   ; muestra un
0

endmethod
```

Vea también contains

empty

Método Borra todos los elementos de una matriz.

Tipo Array

Sintaxis **empty** ( )

Descripción Borra todos los elementos de una matriz. Una matriz de tamaño fijo permanece con el mismo tamaño, y todos sus elementos quedan sin asignar. Una matriz redimensionable se redefine con un tamaño de 0.

Ejemplo Este ejemplo muestra cómo funciona **empty** en una matriz fija. El código que sigue a continuación declara una matriz fija en la ventana Var de una ficha. Esta matriz es global a todos los objetos de la ficha.

```
;estaFicha::Var
Var
   Matriz Array[5] AnyType  ; declara una matriz de longitud
fija
endVar
```

El código siguiente se anexa al método **pushButton** de un botón. Cuando se pulsa este botón (*BotónDeLlenado*), el código asigna valores numéricos a cada elemento de la matriz Matriz:

```
;BotónDeLlenado::pushButton
method pushButton(var eventInfo Event)
Matriz[1] = 234 ; almacena números en la Matriz
Matriz[2] = 356
Matriz[3] = 98
Matriz[4] = 989
Matriz[5] = 2341
Matriz.view("Contenido de la matriz Matriz")
    ; y muestra su contenido
endmethod
```

El código siguiente se anexa al método **pushButton** de un botón. Cuando se pulsa este botón (BotónDeVaciado), el código vacía la matriz Matriz y muestra el contenido de la matriz. Puesto que Matriz es una matriz fija, el número de elementos no cambia; continúa habiendo cinco elementos, pero quedan sin asignación todos los valores.

```
;BotónDeVaciado::pushButton
method pushButton(var eventInfo Event)
Matriz.empty()  ; vacía la matriz Matriz
Matriz.view("Contenido de la matriz Matriz")
                ; y muestra su contenido
endmethod
```

Vea también remove
removeAllItems

exchange

Método	Intercambia el contenido de dos celdas de una matriz.
Tipo	Array
Sintaxis	**exchange** (const ***índice1*** LongInt, const ***índice2*** LongInt)
Descripción	Intercambia el contenido de las celdas de *índice1* e *índice2* de una matriz.
Ejemplo	Consulte el ejemplo de indexOf.
Vea también	contains countOf indexOf

fill

Método Rellena una matriz con un valor.

Tipo Array

Sintaxis **fill** (const ***valor*** AnyType)

Descripción Asigna valor a cada elemento de una matriz.

Ejemplo Este código crea una matriz fija y la rellena con valores de cadena. Este código se anexa al método **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Matriz Array[4] String
endVar

Matriz.fill("Hola")   ; rellena Matriz con el valor "Hola"
Matriz.view()         ; muestra cuatro Holas en la caja de
diálogo

endmethod
```

Vea también append
insert

grow

Método Aumenta el tamaño de una matriz redimensionable.

Tipo Array

Sintaxis **grow** (const ***incremento*** LongInt)

Descripción Añade el número de celdas indicado en incremento a una matriz redimensionable, o borra celdas si el valor de incremento es negativo. Si se intenta borrar más celdas de las que hay en la matriz, se obtiene un error.

Ejemplo El ejemplo siguiente utiliza **grow** para aumentar y reducir el tamaño de una matriz redimensionable. Este código se anexa al método **pushButton** de un botón.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  Matriz Array[] SmallInt
endVar

Matriz.setSize(2)
Matriz[1] = 6
Matriz[2] = 123
message(Matriz.size()) ; muestra un 2
sleep(1000)
Matriz.grow(3)
message(Matriz.size()) ; muestra un 5
sleep(1000)
Matriz.grow(-3)
message(Matriz.size()) ; muestra un 2
sleep(1000)

endmethod
```

Vea también addLast
insert
insertFirst
isResizeable

indexOf

Método Devuelve la posición de un elemento en una matriz.

Tipo Array

Sintaxis **indexOf** (const ***valor*** AnyType) LongInt

Descripción Devuelve el índice de la primera aparición de valor en una matriz, o 0 si no se encuentra una correspondencia exacta.

Ejemplo El ejemplo siguiente supone que una ficha tiene un objeto de campo no definido llamado *esteCampo*. Cuando el usuario hace clic con el botón del ratón sobre el campo, aparece un menú emergente que ofrece una lista de tipos de pago. El elemento seleccionado se inserta en el campo. Cuando el usuario hace clic con el botón derecho sobre el campo la vez siguiente, la última opción de menú seleccionada es la primera que aparece en la lista de opciones del menú. El código que sigue a continuación se anexa a la ventana Var de *esteCampo*:

```
;esteCampo::Var
Var
   FormaPago  Array[5] String
   MenuPago   PopUpMenu
endVar
```

El código siguiente se anexa al método **open** de *esteCampo*. Cuando el campo se abre por primera vez, el código asigna valores a la matriz que se utilizó en el menú emergente.

```
;esteCampo::open
method open(var eventInfo Event)
FormaPago[1] = "Talón"      ; inicializa los elementos de la
matriz
FormaPago[2] = "Caja"
FormaPago[3] = "Visa"
FormaPago[4] = "MasterCard"
FormaPago[5] = "AmEx"
endmethod
```

El código siguiente se anexa al método **mouseRightUp** de *esteCampo*. Este código muestra el menú emergente e inserta la selección en *esteCampo*. El método **indexOf** se utiliza aquí para obtener el valor ordinal de la opción de menú seleccionada; entonces, la selección se desplaza al comienzo de la matriz mediante el método **exchange**.

```
;esteCampo::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
   IndiceOpción SmallInt
   opción       String
endVar

disableDefault                    ; no muestra el menú normal
menuPago.addArray(FormaPago)      ; añade la matriz al menú
desplegable
```

```
opción = menuPago.show()          ; muestra el menú y permite
seleccionar
self.value = opción               ; introduce la selección en el
campo

   ; prepara el menú desplegable para el siguiente clic
menuPago.empty()                  ; vacía el menú
IndiceOpción = FormaPago.indexOf(opción) ; recoge el índice de
la selección     FormaPago.exchange(IndiceOpción, 1)      ; y la
pone al principio
endmethod
```

Vea también contains
countOf

insert

Método Inserta una o más celdas vacías en una matriz.

Tipo Array

Sintaxis **insert** (const ***índice*** LongInt [, const ***númeroElementos*** LongInt])

Descripción Inserta el número *númeroElementos* de celdas vacías en una matriz redimensionable, que crece si es necesario. Si no se especifica *númeroElementos*, se inserta una celda. Los índices de los elementos siguientes se incrementan en el número de celdas insertadas.

Ejemplo El ejemplo siguiente inserta elementos vacíos en dos posiciones de una matriz redimensionable y muestra el resultado. Este código se anexa al método **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   MiArray Array[] SmallInt
endVar
MiArray.setSize(20)   ; reserva espacio para 20 elementos
MiArray.fill(1)       ; rellena la matriz con unos
MiArray.insert(5)     ; inserta un elemento vacío en la posición
5
MiArray.insert(12, 4) ; inserta cuatro elementos vacíos en la
posición 12
MiArray.view()
endmethod
```

Vea también insertAfter
insertBefore

insertAfter

Método Inserta un elemento en una matriz después de un elemento especificado.

Tipo Array

Sintaxis **insertAfter** (const ***elementoClave*** AnyType, const ***elementoInsertado*** AnyType)

Descripción Inserta *elementoInsertado* en una matriz redimensionable en una posición mayor en uno que la primera aparición de *elementoClave*. Si no se encuentra *elementoClave*, no se inserta *elementoInsertado*, y los índices no cambian. Si se inserta *elementoInsertado*, los índices de los elementos siguientes aumentan en 1.

Ejemplo Este ejemplo carga una matriz redimensionable y utiliza **insertAfter** para insertar un nuevo elemento después de un elemento existente en la matriz. Este código se anexa al método **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   zoo Array[] String
endVar
zoo.setSize(0)
zoo.addLast("mono")   ; [1] = "mono"
zoo.addLast("vaca")   ; [2] = "vaca"
zoo.addLast("perro")  ; [3] = "perro"

zoo.insertAfter("mono", "oso")
   ; muestra el tamaño 4 en el título; zoo [mono, oso, vaca,
perro]     zoo.view("Tamaño del zoo: " + strVal(zoo.size()))

endmethod
```

Vea también insert
insertBefore

insertBefore

Método Inserta un elemento en una matriz antes de un elemento especificado.

Tipo Array

Sintaxis **insertBefore** (const ***elementoClave*** AnyType, const ***elementoInsertado*** AnyType)

Descripción Busca *elementoClave* en una matriz redimensionable e inserta *elementoInsertado* en la posición de *elementoClave*. Los índices de *elementoClave* y de los elementos siguientes se incrementan en 1. Si no se encuentra *elementoClave*, no se inserta *elementoInsertado*, y los índices no cambian.

Ejemplo Este ejemplo añade un elemento a una matriz redimensionable mediante **insertBefore**. Este código se anexa al método **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  Comida Array[] String
endVar

Comida.grow(3)  ; inicia la matriz con 3 elementos
Comida[1] = "Halcón"
Comida[2] = "Serpiente"
Comida[3] = "Mosca"

  ; insertamos un elemento - esto incrementa la matriz a 4
elementos
Comida.insertBefore("Mosca", "Rana")
  ; muestra el tamaño 4 en el título; [Halcón, Serpiente, Rana,
Mosca]
Comida.view("Tamaño de Comida: " + strVal(Comida.size()))

endmethod
```

Vea también insert
insertAfter

insertFirst

Método Inserta un elemento al principio de una matriz.

Tipo Array

Sintaxis **insertFirst** (const ***valor*** AnyType)

Descripción Inserta valor al principio de una matriz redimensionable. Los índices de los elementos siguientes se incrementan en 1.

Ejemplo Este ejemplo crea una matriz redimensionable y añade un nuevo elemento al principio de la matriz. Este código se anexa al método estándar **pushButton** de un botón:

```
method pushButton(var eventInfo Event)
var
   Zoo Array[] String
endVar
Zoo.setSize(2)  ; inicia la matriz con dos elementos
Zoo[1] = "león"
Zoo[2] = "tigre"

                    ; inserta un elemento en el principio de la
matriz-
                    ; lo cual incrementa dicha matriz a tres
elementos
Zoo.insertFirst("oso")
                    ; muestra el tamaño 3 en el título; [oso,
león, tigre]
Zoo.view("Tamaño de Zoo: " + strVal(Zoo.size()))

endmethod
```

Vea también addLast
append
insert
insertAfter
insertBefore

isResizeable

Método Informa de si una matriz puede redimensionarse.

Tipo Array

Sintaxis **isResizeable** ( ) Logical

Descripción Devuelve True si es posible cambiar el tamaño de una matriz; en caso contrario, devuelve False.

Ejemplo Este código comprueba si una matriz en particular puede redimensionarse antes de intentar aumentar su tamaño. Este código se anexa al método **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Matriz Array[] String
endVar
if Matriz.isResizeable() = True then  ; si la matriz puede
cambiar su tamaño
    Matriz.grow(5)                    ; le añadimos cinco
elementos
else
   msgStop("Problema", "No se puede cambiar el tamaño de la
matriz.")
   endif
endmethod
```

Vea también grow
size

remove

Método Borra uno o más elementos de una matriz.

Tipo Array

Sintaxis **remove** (const ***índice*** SmallInt[const ***númeroElementos*** SmallInt])

Descripción Suprime el número *númeroElementos* de elementos (o un elemento, si no se especifica *númeroElementos*) en el *índice* de una matriz. Los índices de los elementos siguientes se reducen en *númeroElementos* (o en 1, si no se especifica *númeroElementos*).

Ejemplo Este ejemplo borra un solo elemento de una matriz redimensionable. Obsérvese que es común utilizar el método **indexOf** para determinar qué elemento se desea eliminar. Este código se anexa al método estándar **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Zoo Array[] String
endVar

Zoo.setSize(3)              ; inicia Zoo con tres elementos
Zoo[1] = "león"
Zoo[2] = "tigre"
Zoo[3] = "oso"

Zoo.remove(Zoo.indexOf("tigre")) ; esto es lo mismo que
Zoo.remove(2)

                            ; el título muestra el tamaño 2
                            ; la ventana de diálogo muestra
Zoo[león, oso]
Zoo.view("Tamaño de Zoo: " + strVal(Zoo.size()))

endmethod
```

El ejemplo siguiente muestra cómo utilizar **remove** para eliminar más de un elemento de una matriz redimensionable. Este código se anexa al método **pushButton** de un botón:

```
;eseBotón::pushButton
method pushButton(var eventInfo Event)
var
Números Array[] SmallInt
       i      SmallInt
endVar

Números.grow(9)       ; inicia Números con nueve elementos
for i from 1 to 9     ; asigna nueve elementos
   Números[i] = i
endFor

                      ; muestra Números[1, 2, 3, 4, 5, 6, 7, 8,
9]
```

```
Números.view("Antes de eliminar los elementos")
                        ; eliminar cuatro elementos, comenzando en
el tercer elemento
Números.remove(3, 4)  ; Números = [1, 2, 7, 8, 9]
                        ; muestra Números[1, 2, 7, 8, 9]
Números.view("Después de eliminar los elementos")
endmethod
```

Vea también insert
removeItem
removeAllItems

removeAllItems

Método Borra todas las apariciones de un elemento de una matriz.

Tipo Array

Sintaxis **removeAllItems** (const ***valor*** AnyType)

Descripción Suprime todas las apariciones de valor en una matriz. Los índices de los elementos siguientes disminuyen en 1.

Ejemplo Este ejemplo muestra cómo funciona **removeAllItems** con una matriz redimensionable. El código siguiente se anexa al método estándar **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Zoo Array[] String
endVar
Zoo.setSize(5)
Zoo[1] = "mono"
Zoo[2] = "vaca"
Zoo[3] = "cerdo"
Zoo[4] = "vaca"
Zoo[5] = "león"

  ; mostrar el actual contenido de la matriz en un cuadro de
diálogo
Zoo.view("Antes de eliminar los elementos")

  ; eliminar todas las "vacas"
Zoo.removeAllItems("vaca")

  ; ahora,
  ; Zoo[1] = "mono"
  ; Zoo[2] = "cerdo"
  ; Zoo[3] = "león"

  ; mostramos el nuevo contenido de la matriz en un cuadro de
diálogo
Zoo.view("Después de eliminar los elementos")

endmethod
```

Vea también remove
removeItem

removeItem

Método Borra un elemento especificado de una matriz.

Tipo Array

Sintaxis **removeItem** (const ***valor*** AnyType)

Descripción Elimina la primera aparición de valor en una matriz. Los índices de los elementos siguientes disminuyen en 1.

Ejemplo Este ejemplo utiliza **removeItem** para suprimir un elemento de una matriz redimensionable. Este código se anexa al método estándar **pushButton** de un botón.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Zoo Array[] String
endVar

Zoo.setSize(4)
Zoo[1] = "mono"
Zoo[2] = "león"
Zoo[3] = "tigre"
Zoo[4] = "león"

  ; esto muestra [mono, león, tigre, león]
Zoo.view("Antes de eliminar un león")

  ; eliminamos el primer "león" de la tabla
Zoo.removeItem("león")

  ; esto muestra [mono, tigre, león] en un cuadro de diálogo
Zoo.view("Después de eliminar un león")

endmethod
```

Vea también remove
removeAllItems
replaceItem

replaceItem

Método Sustituye un elemento de una matriz por otro elemento.

Tipo Array

Sintaxis **replaceItem** (const ***elementoClave*** AnyType, const ***nuevoElemento*** AnyType)

Descripción Busca *elementoClave* en una matriz y sustituye la primera aparición de *elementoClave* por *nuevoElemento*.

Ejemplo Este ejemplo sustituye un elemento de una matriz redimensionable y muestra el valor inicial y el resultado en un cuadro de diálogo. Este código se anexa al método estándar **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Comida Array[] String
endVar

Comida.setSize(3)
Comida[1] = "Tiburón"
Comida[2] = "Elefante"
Comida[3] = "Pececillo"

  ; muestra el contenido de la matriz en un cuadro de diálogo
Comida.view("Antes de replaceItem...")

Comida.replaceItem("Elefante", "Atún")
  ; muestra el contenido de la matriz en un cuadro de diálogo
([Tiburon, Atún, Pececillo])
Comida.view("Después de replaceItem...")

endmethod
```

Vea también removeItem

setSize

Método Especifica el tamaño de una matriz.

Tipo Array

Sintaxis **setSize** (const ***tamaño*** LongInt)

Descripción Reserva espacio para el número tamaño de elementos en una matriz redimensionable. Si **setSize** reduce el tamaño de la matriz, ésta se trunca.

Ejemplo Este ejemplo declara una matriz redimensionable en la sección de declaración de variables y utiliza **setSize** para inicializar el tamaño de la matriz con tres elementos. El código rellena cada elemento de la matriz y ejecuta **setSize** de nuevo, esta vez para redimensionar la matriz a dos elementos. El resultado de reducir la matriz (mostrada en un cuadro de diálogo) es la eliminación del tercer y último elemento. Este código se anexa al método estándar **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Matriz Array[] SmallInt
endVar

Matriz.setSize(3)          ; el tamaño es 3

Matriz[1] = 123
Matriz[2] = 2353
Matriz[3] = 18

  ; muestra el tamaño 3 en el título ([123, 2353, 18]) de un
cuadro de diálogo
Matriz.view("Tamaño de Matriz: " + strVal(Matriz.size()))

Matriz.setSize(2)          ; truncamos el tamaño de la matriz a
2

  ; muestra el tamaño 2 en el título ([123, 2353]) de un cuadro
de diálogo
Matriz.view("Ahora el tamaño de Matriz es: " +
strVal(Matriz.size()))

endmethod
```

Vea también grow
isResizeable

size

Método	Devuelve el número de elementos de una matriz.
Tipo	Array
Sintaxis	**size** ( ) LongInt
Descripción	Devuelve el número total de elementos de una matriz, aunque haya uno o más elementos vacíos.
Ejemplo	Consulte el ejemplo de setSize.
Vea también	grow setSize

view

Método Muestra el contenido de una matriz en un cuadro de diálogo.

Tipo Array

Sintaxis **view** ([const ***título*** String])

Descripción Muestra el contenido de una matriz en un cuadro de diálogo modal. La ejecución de ObjectPAL se suspende hasta que el usuario cierra este cuadro de diálogo. Existe la opción de especificar, en *título*, un título para el cuadro de diálogo. Si se omite el título, éste será "Array".

A diferencia de muchos otros tipos de datos, los valores de Array mostrados en un cuadro de diálogo de **view** no pueden modificarse interactivamente. Consulte "AnyType" en este capítulo para más información sobre otros tipos de datos y el método **view**.

Ejemplo Este ejemplo muestra el contenido de una matriz en un cuadro de diálogo sin un título personalizado y, posteriormente, con uno. Observe que título puede ser cualquier expresión que se evalúe en una cadena. Este código se anexa al método **pushButton** de un botón:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Matriz Array[]  SmallInt
   i               SmallInt
endVar

Matriz.setSize(10)
for i from 1 to 10
   Matriz[i] = i * 10
endfor

Matriz.view()   ; muestra 10, 20, 30, etc (sin título)
                ; esto muestra "tamaño de Matriz: 10" en el
título
Matriz.view("Tamaño de Matriz: " + strVal(Matriz.size()))

endmethod
```

Vea también contains

readFromFile

Método Lee datos de un archivo y los almacena en una variable Binary.

Tipo Binary

Sintaxis **readFromFile** (const ***nombreArchivo*** String) Logical

Descripción Lee datos binarios del archivo en disco mencionado en nombreArchivo. Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Ejemplo Las sentencias siguientes declaran una variable Binary *ElSonido*, leen datos binarios de un archivo en *ElSonido* y asignan el valor de la variable a un campo Binary de una tabla (supóngase que SONIDOS.DB es una tabla de Paradox con la estructura siguiente: NombreSonido, A32; DatosSonido, B).

```
;obtenArchivo::pushButton
method pushButton(var eventInfo Event)
var
   SonidosTC TCursor
   ElSonido Binary
endVar
if ElSonido.readFromFile("ruido.bin") then ; True si
readFromFile tiene éxito
   if SonidosTC.open("sonidos.db") then
      SonidosTC.edit()
      SonidosTC.insertRecord()
      SonidosTC.NombreSonido = "Ruido"
      SonidosTC.DatosSonido = ElSonido ; lleva el contenido del
archivo
    ; a un campo binario
      SonidosTC.endEdit()
      SonidosTC.close()
   endIf
endIf
endmethod
```

Vea también size
writeToFile
Methods and procedures defined for FileSystem.

size

Método Devuelve el número de bytes de una variable Binary.

Tipo Binary

Sintaxis **size** ( ) LongInt

Descripción Devuelve un valor que representa el número de bytes contenidos en una variable Binary.

Ejemplo El ejemplo siguiente pasa por los registros de una tabla que contienen campos Binary, comprueba el tamaño de cada campo Binary y, si hay suficiente espacio libre en el disco, escribe los datos en un archivo en disco (supóngase que SONIDOS.DB es una tabla de Paradox con la estructura siguiente: NombreSonido, A32; DatosSonido, B). Este código se anexa a un método personalizado llamado *EscribirFichBinarios*:

```
method EscribirFichBinarios()
var
   VarBin       Binary
   sf          FileSystem
   SonidosTC   TCursor
   EspacioLibre LongInt
endVar

if SonidosTC.open("Sonidos.db") then
   scan SonidosTC for not isBlank(SonidosTC.DatosSonido) :
      VarBin = SonidosTC.DatosSonido    ; VarBin = valor del
campo DatosSonido
      EspacioLibre = sf.freeDiskSpace("B")
      if EspacioLibre > VarBin.size() then    ; si hay sitio en
B:
      VarBin.writeToFile(SonidosTC.NombreSonido) ; escribimos
VarBin en el archivo
else                                      ; si no, el archivo no
cabe en B:
       msgStop("Alto", "El disco de la unidad B: está lleno.")
          return
      endIf
   endScan
endif

endmethod
```

Vea también readFromFile
writeToFile
Métodos y procedimientos definidos para el tipo FileSystem.

writeToFile

Método Escribe en un archivo en disco los datos contenidos en una variable Binary.

Tipo Binary

Sintaxis **writeToFile** (const ***nombreArchivo*** String) Logical

Descripción Escribe los datos contenidos en una variable Binary en el archivo en disco especificado en *nombreArchivo*. Este método devuelve True si es satisfactorio; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente pasa por los registros de una tabla que contiene campos Binary, comprueba el tamaño de cada campo Binary y, si hay suficiente espacio libre en el disco, escribe los datos en un archivo en disco (supóngase que SONIDOS.DB es una tabla de Paradox con la estructura siguiente: NombreSonido, A32; DatosSonido, B). Este código se anexa a un método personalizado llamado **EscribirFichBinarios**:

```
method EscribirFichBinarios()
var
   VarBin        Binary
   sf            FileSystem
   SonidosTC     TCursor
   EspacioLibre   LongInt
endVar

if SonidosTC.open("Sonidos.db") then
   scan SonidosTC for not isBlank(SonidosTC.DatosSonido) :
     VarBin = SonidosTC.DatosSonido   :VarBin = valor del campo
DatosSonido
     EspacioLibre = sf.freeDiskSpace("B")
     if EspacioLibre > VarBin.size() then ; si hay sitio en B:
     VarBin.writeToFile(SonidosTC.NombreSonido) ; escribimos
Varbin en el archivo
      else      ; si no, el archivo no cabe en B:
         msgStop("Alto", "El disco de la unidad B: está
lleno.")
         return
      endIf
   endScan
endif

endmethod
```

Vea también readFromFile
size
Métodos y procedimientos definidos para el tipo FileSystem.

currency

Procedure Convierte un valor a Currency.

Tipo Currency

Sintaxis **currency** (const ***valor*** AnyType) Currency

Descripción Convierte el tipo de datos de valor a Currency.

Ejemplo En este ejemplo, un número se almacena en una variable String y se convierte al tipo Currency para su uso en un cálculo. El método **pushButton** de *mostrarDoble* muestra el tipo de la variable y, a continuación, calcula y muestra el resultado de la cadena convertida a Currency y multiplicada por 2:

```
;mostrarDoble::pushButton
method pushButton(var eventInfo Event)

var
   StrNum   String
endVar
StrNum = "12,34"
msgInfo("El tipo de datos de StrNum es:", dataType(StrNum))
; antes de multiplicar StrNum por dos, debemos amoldarlo
; al tipo numérico
msgInfo("El doble de " + StrNum + " es", currency(StrNum) * 2)
endmethod
```

En el ejemplo siguiente, el método **pushButton** del botón *Precisión* calcula un número utilizando variables del tipo Number y, a continuación, realiza los mismos cálculos con los valores convertidos a Currency. El resultado de los dos cálculos varía ligeramente.

```
;Precisión::pushButton
method pushButton(var eventInfo Event)

var
   x, y, z Number
endVar

x = 1,2 / 3,323          ; almacena la mayor precisión
y = 4,9 / 7,3
z = 2,0 * x * y          ; realiza el cálculo para valores
completos
msgInfo("Resultado del cálculo de los números",
             format("W14.6", z))     ; muestra 0,484790
x = Currency(1,2 / 3,323) ; almacena una precisión de 6
decimales
y = Currency(4,9 / 7,3)
z = 2,0 * x * y          ; calcula los valores con seis
decimales
msgInfo("Resultado del cálculo con Currency",
             format("W14.6", z))     ; muestra 0,484791

endmethod
```

Vea también Currency

date

Principiante

Procedimiento Convierte un valor a Date.

Tipo date

Sintaxis **1. date** (const ***valor*** AnyType) Date

2. date ( ) Date

Descripción Convierte *valor* a fecha. Si la fecha suministrada en *valor* no es válida, el método falla. Si no se suministra *valor*, **date** devuelve la fecha actual como un tipo de datos Date.

Ejemplo El código siguiente convierte un valor a fecha, utiliza el valor de fecha en un cálculo y muestra el resultado en un cuadro de diálogo.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   c String
   d Date
endVar

c = "11/11/99"   ; c es un dato de tipo String
d = date(c) + 7  ; convertimos el tipo String en tipo Date

d.view()         ; mostramos el valor de d en un cuadro de
diálogo (11/11/99)
                 ; el título del cuadro de diálogo muestra
"Date"
endmethod
```

Vea también dateVal

dateVal

Procedimiento Devuelve un valor como fecha.

Tipo Date

Sintaxis **dateVal** (const ***valor*** AnyType) Date

Descripción Devuelve un valor como fecha.

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón utiliza **dateVal** para obtener la fecha equivalente de un valor String y muestra el valor en un cuadro de diálogo:

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  c String
  d Date
endVar
c = "11/11/99"   ; c es un dato de tipo String
d = dateVal(c)   ; d contiene la fecha equivalente de c

d.view()         ; mostramos el valor de d en un cuadro de
diálogo (11/11/99)
                 ; el título del cuadro de diálogo muestra
"Date"
endmethod
```

Vea también date

today

Procedimiento Devuelve la fecha actual.

Tipo Date

Sintaxis **today** ( ) Date

Descripción Devuelve la fecha actual, según el reloj/calendario del ordenador.

Ejemplo Este ejemplo muestra la fecha actual en un cuadro de diálogo:

```
;FechaActual::pushButton
method pushButton(var eventInfo Event)
msgInfo("La fecha de hoy es", today())       ; muestra la fecha
actual
endmethod
```

Vea también date
DateTime::day
DateTime::month
DateTime::year

dateTime

Principiante

Método Convierte un valor al tipo de datos DateTime.

Tipo dateTime

Sintaxis **1. dateTime** (const ***valor*** AnyType) DateTime

2. dateTime ( ) DateTime

Descripción Convierte valor al tipo de datos DateTime. Si no se suministra *valor*, **dateTime** devuelve la hora y fecha actuales como un tipo de datos DateTime.

Ejemplo Las sentencias siguientes asignan a la variable *DateTime* un valor horario de 11 horas, 10 minutos y 40 segundos y una fecha de 21 de diciembre de 1997. Este código supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
var FH DateTime endVar
FH = DateTime ("11:10:40 am 12/21/97")
```

Las comillas alrededor del valor son necesarias.

Es posible utilizar los caracteres siguientes como separadores: blanco, tabulador, espacio, coma (,), guión (-), barra inclinada (/), punto (.), dos puntos (:) y punto y coma (;). El formato de los valores de DateTime depende de la especificación definida por el método **formatSetDateTimeDefault** (tipo System) o por las sentencias de formateo de ObjectPAL.

Un valor de DateTime debe especificarse completamente; no es posible omitir ninguno de los campos, pero sí especificar un valor de cero para cualquier campo.

Vea también day
hour
month
year

day

Principiante

Método Extrae el día del mes de un DateTime.

Tipo DateTime

Sintaxis **day** ( ) SmallInt

Descripción Extrae el día del mes de un valor de DateTime y devuelve un valor entre 1 y 31. Si el DateTime no es válido, el método falla.

Ejemplo En este ejemplo, el método **pushButton** de un botón muestra el día actual del mes en un cuadro de diálogo. Este código supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Día DateTime
endVar
Día = DateTime("12:00:00 am 12/12/92")

   ; muestra 22 en un cuadro de diálogo
msgInfo("Día del mes", Día.day())

endmethod
```

Vea también dow
dowOrd
doy
month
moy
year

daysInMonth

Principiante

Método Devuelve el número de días de un mes.

Tipo DateTime

Sintaxis **daysInMonth** ( ) SmallInt

Descripción Dado un valor de DateTime válido, **daysInMonth** devuelve el número de días de ese mes. Si el DateTime no es válido, el método falla.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *díasDeFebrero* muestra el número de días de Febrero de 1992. Este código supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;díasDeFebrero::pushButton
method pushButton(var eventInfo Event)
var
   DíasFebrero SmallInt
endVar
DíasFebrero = daysInMonth(DateTime("5:15:35 AM 2/1/92"))
msgInfo("Número de días", "Hay " + String(DíasFebrero) +
         " días en Febrero de 1992")

       ; muestra "Hay 29 días en Febrero de 1992" en un cuadro
de diálogo
       ; (1992 es un año bisiesto)
endmethod
```

Vea también day
dow
dowOrd
moy

dow

Principiante

Método Devuelve el día de la semana de un DateTime.

Tipo DateTime

Sintaxis **dow** ( ) String

Descripción Dado un valor de DateTime válido, **dow** devuelve las tres primeras letras del día de la semana de ese DateTime. Si el DateTime no es válido, el método falla.

Ejemplo Este ejemplo muestra, en un cuadro de diálogo, el día de la semana de un DateTime dado. Este código supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;mostrarDía::pushButton
method pushButton(var eventInfo Event)
var
   Fecha DateTime
endVar

Fecha = DateTime("11:20:15 pm 3/9/93")

     ; muestra "Mar" en un cuadro de diálogo
msgInfo("Días de la semana", strVal(Fecha) + " cae en " +
dow(Fecha))

endmethod
```

Vea también dateTime
day
dowOrd
doy
moy

dowOrd

Principiante

Método Devuelve el número de un día de la semana.

Tipo DateTime

Sintaxis **dowOrd** ( ) SmallInt

Descripción Dado un valor de DateTime válido, **dowOrd** devuelve un número entero de 1 a 7 que representa la posición del día en la semana. El domingo es el día 1, el lunes es el día 2, y así sucesivamente. Si el DateTime no es válido, el método falla.

Ejemplo El ejemplo siguiente muestra el día actual de la semana como una palabra completa (por ejemplo, "Lunes") y no como abreviatura ni como número. Este código utiliza **dowOrd** para recuperar el subíndice adecuado de una matriz fija y muestra el valor del elemento de la matriz en un cuadro de diálogo. Este código se anexa al método **pushButton** del botón *nombresCompletos*. Este ejemplo supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;nombresCompletos::pushButton
method pushButton(var eventInfo Event)
var
   Nombre Array[7] String
   Fecha           DateTime
endVar

Nombre[1] = "Domingo"
Nombre[2] = "Lunes"
Nombre[3] = "Martes"
Nombre[4] = "Miércoles"
Nombre[5] = "Jueves"
Nombre[6] = "Viernes"
Nombre[7] = "Sábado"

Fecha = DateTime("5:35:20 AM 1/8/93")
   ; esto muestra "Sábado" en un cuadro de diálogo
msgInfo("Día de la semana", Nombre[dowOrd(Fecha)])

endmethod
```

Vea también dateTime
day
dow
doy
moy

doy

Principiante

Método Devuelve el número de un día del año.

Tipo DateTime

Sintaxis **doy** ( ) SmallInt

Descripción Dado un DateTime válido, **doy** devuelve un número entero de 1 a 366 que representa la posición del día en el año. El 1 de enero es el día 1, el 1 de febrero es el día 32, y así sucesivamente. Si el DateTime no es válido, el método falla.

Ejemplo Este ejemplo muestra la posición de un día en un año especificado. Este código se anexa al método **pushButton** de un botón. Este ejemplo supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Fecha DateTime
endVar

Fecha = DateTime("5:35:20 AM 6/1/92")

   ; esto muestra "5:35:20, 6/1/92 han
   ; pasado 153 días desde el primero del año"
msgInfo("Fecha", String(Fecha) + " han pasado " +
String(Fecha.doy()) +
                " días desde el primero del año.")

endmethod
```

Vea también dow
moy

hour

Principiante

Método Extrae la hora de un DateTime como un número.

Tipo DateTime

Sintaxis **hour** ( ) SmallInt

Descripción Dado un DateTime válido, **hour** devuelve un número entero que representa la hora del día en formato de 24 horas. Este método falla si el DateTime no es válido.

Ejemplo El código siguiente extrae la hora de un DateTime dado y la muestra en un cuadro de diálogo. Observe que, aunque el DateTime dado está en forma de 12 horas, **hour** devuelve el equivalente en 24 horas.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
FH       DateTime
endVar

FH = DateTime("8:15:18 pm 12/29/92")

msgInfo("Hora", FH.hour())   ; muestra 20 en un cuadro de
diálogo

endmethod
```

Vea también day
month
minute
milliSec
year

isLeapYear

Principiante

Método Informa de si un año tiene 366 días.

Tipo DateTime

Sintaxis **isLeapYear** ( ) Logical

Descripción Dado un DateTime válido, **isLeapYear** devuelve True si el año dado en DateTime tiene 366 días; en caso contrario, devuelve False. Este método falla si el DateTime no es válido.

Ejemplo Para este ejemplo, el método **pushButton** del botón *comprobAñoBisiesto* muestra True si el DateTime dado es un año bisiesto; si no lo es, muestra False. Este ejemplo supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Día        DateTime
   Bisiesto   Logical
endVar

Día = DateTime("5:35:20 AM 6/1/92")

Bisiesto = Día.isLeapYear()
Bisiesto.view("Día")          ; muestra True

endmethod
```

Vea también year

milliSec

Principiante

Método Extrae los milisegundos de un DateTime como un número.

Tipo DateTime

Sintaxis **milliSec** ( ) SmallInt

Descripción Dado un DateTime válido, **milliSec** devuelve un número entero que representa los milisegundos. Este método falla si el DateTime no es válido.

Ejemplo Este ejemplo crea un valor de DateTime a partir de cálculos con números enteros y muestra la parte de milisegundos del DateTime en un cuadro de diálogo.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   FH DateTime
   Segundo, Minuto, Hora LongInt
endVar
Segundo = 1000            ; milisegundos
Minuto  = Segundo * 60
Hora    = Minuto * 60

   ; la sentencia siguiente asigna a FH el dato de tipo
DateTime
   ; "1:20:30.4 pm 00/00/00" (la sentencia no
   ; asigna una fecha, de modo que DateTime ajusta una parte de
la fecha a 0)
FH = DateTime(13 * Hora   +
              20 * Minuto +   ; especifica 1:20 pm
              30 * Segundo +  ; + 30 segundos
              400)           ; + 400 milisegundos

msgInfo("Milisegundos", FH.milliSec())   ; muestra 400

endmethod
```

Vea también hour
minute
second

minute

Principiante

Método Extrae los minutos de un DateTime como un número.

Tipo DateTime

Sintaxis **minute** ( ) SmallInt

Descripción Dado un DateTime válido, **minute** devuelve un número entero que representa los minutos. Este método falla si el DateTime no es válido.

Ejemplo En este ejemplo, el método **pushButton** de *esteBotón* muestra la parte de minutos de un DateTime dado. Este código supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   FH   DateTime
endVar

FH = DateTime("9:20:15 am 8/2/93")

msgInfo("Minutos", FH.minute())    ; muestra 20

endmethod
```

Vea también hour
milliSec
second

month

Principiante

Método Extrae el mes de un DateTime como un número.

Tipo DateTime

Sintaxis **month** ( ) SmallInt

Descripción Dado un DateTime válido, **month** devuelve un número entero que representa devuelve la posición del mes de esa fecha en el año. El mes 1 es enero, el mes 2 es febrero, y así sucesivamente. Este método falla si el DateTime no es válido.

Ejemplo El ejemplo siguiente muestra el mes del año como una palabra completa (por ejemplo, Agosto), y no como abreviatura ni número. Este código utiliza **month** para recuperar el subíndice adecuado de una matriz fija y muestra el valor del elemento de la matriz en un cuadro de diálogo. Este código se anexa al método **pushButton** del botón *meses*. Este ejemplo supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;meses::pushButton
method pushButton(var eventInfo Event)
var
   Meses Array[12] String
   Fecha DateTime
endVar

Meses[1] =  "Enero"
Meses[2] =  "Febrero"
Meses[3] =  "Marzo"
Meses[4] =  "Abril"
Meses[5] =  "Mayo"
Meses[6] =  "Junio"
Meses[7] =  "Julio"
Meses[8] =  "Agosto"
Meses[9] =  "Septiembre"
Meses[10] = "Octubre"
Meses[11] = "Noviembre"
Meses[12] = "Diciembre"

Fecha = DateTime("5:35:20 AM 9/18/93")
  ; esto muestra "Septiembre" en un cuadro de diálogo
msgInfo("Mes", Meses[month(Fecha)])

endmethod
```

Vea también moy

moy

Principiante

Método Extrae el mes de un DateTime como una cadena.

Tipo DateTime

Sintaxis **moy** ( ) String

Descripción Dado un DateTime válido, **moy** devuelve las tres primeras letras del nombre del mes de esa fecha. Este método falla si el DateTime no es válido.

Ejemplo Para este método, el método **pushButton** de *esteBotón* muestra el nombre abreviado del mes de un DateTime especificado. Este código supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Fecha DateTime
endVar

Fecha = DateTime("2:09:00 AM 3/3/97")
msgInfo("Fecha", Fecha.moy())    ; muestra Mar

endmethod
```

Vea también month

second

Principiante

Método Extrae los segundos de un DateTime como un número.

Tipo DateTime

Sintaxis **second** ( ) SmallInt

Descripción Dado un DateTime válido, **second** devuelve un número entero que representa los segundos. Este método falla si el DateTime no es válido.

Ejemplo Este ejemplo crea un valor de DateTime a partir de cálculos con números enteros y muestra la parte de segundos del DateTime en un cuadro de diálogo.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   FH   DateTime
   Segundo, Minuto, Hora LongInt
endVar
Segundo = 1000                    ; milisegundos
Minuto  = Segundo * 60
Hora    = Minuto *  60

   ; la siguiente sentencia asigna a FH el valor de tipo
DateTime
   ; "1:20:30.4 pm 00/00/00" (la sentencia no
   ; asigna una fecha, de modo que DateTime ajusta una parte de
la fecha a 0)
FH = DateTime(13 * Hora  +
              20 * Minuto  +     ; especifica 1:20 pm
              30 * Segundo +     ; + 30 segundos
              400)               ; + 400 milisegundos

msgInfo("Segundos", FH.second()) ; muestra 30

endmethod
```

Vea también hour
milliSec
minute

year

Principiante

Método Extrae el año de un DateTime como un número.

Tipo DateTime

Sintaxis **year** ( ) SmallInt

Descripción Dado un DateTime válido, **year** devuelve un número entero que representa el año contenido en el DateTime. Este método falla si el DateTime no es válido.

Ejemplo Para este ejemplo, el método **pushButton** del botón *botónDeAño* muestra el año con cuatro dígitos de un DateTime especificado. Este código supone que el formato actual de hora y fecha es la forma hh:mm:ss am/pm mm/dd/aa.

```
;botónDeAño::pushButton
method pushButton(var eventInfo Event)
var
   Fecha DateTime
endVar

Fecha = DateTime("2:15:24 pm 3/3/97")
msgInfo("Fecha", Fecha.year())   ; muestra 1997

endmethod
```

Vea también day
isLeapYear
month
moy

contains

Método Busca un valor en los índices de un DynArray.

Tipo DynArray

Sintaxis **contains** (const ***valor*** AnyType) Logical

Descripción Devuelve True si el índice de cualquier elemento de un DynArray coincide con valor carácter por carácter; en caso contrario, devuelve False. **contains** tiene en cuenta el uso de mayúsculas o minúsculas.

Ejemplo El ejemplo siguiente utiliza **contains** para comprobar si el índice de una matriz dinámica corresponde a una opción de menú. En este ejemplo, el método **open** de la ficha crea un menú y asigna varios valores a una matriz dinámica. Cuando el usuario selecciona una opción en el menú, el método **menuAction** de la ficha compara la selección de menú con los índices del DynArray. Si un índice del DynArray está definido para la opción de menú seleccionada, el método **menuAction** muestra el valor asociado con ese elemento del DynArray; en caso contrario, muestra el valor de otro elemento.

Este código va en la ventana Var de la ficha:

```
;estaFicha::Var
var
   Mensaje DynArray[] AnyType  ; almacena mensajes
   m1                  Menu     ; barra de menú
   p1                 PopUpMenu ; Menú desplegable asociado al
ítem de tipo Menu
   selección           String   ; seleción del usuario del menú
endVar
```

El código que sigue a continuación se anexa al método **open** de una ficha:

```
;estaFicha::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
   then
     ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha       else
     ; el código fuente que hay aquí se ejecuta para la propia
Ficha

     p1.addText("Hora")              ; añadimos items al menú
desplegable
     p1.addText("Fecha")
     p1.addText("Colores")

     m1.addPopUp("&Utilidades", p1) ; asociamos el desplegable
al ítem
     ; de la barra de menú
     m1.show()                     ; mostramos la barra de menú

    ; ahora se inicializa la matriz dinámica de mensajes. Los
índices de
```

```
     ; Mensaje corresponden a los items del menú desplegable
generado arriba.
       Los valores de Mensaje son datos que aparecen en un
cuadro de diálogo
       cuando el usuario selecciona un menú. Nótese que
Mensaje NO contiene el
        índice "Colores".
    msg["Hora"] = time()    ; muestra la fecha actual para la
selección "Hora"
    msg["Fecha"] = date()   ; muestra la fecha actual para la
selección "Fecha"
    msg["Error"] = "Lo siento, este ítem no está diseñado."

endif
endmethod
```

Este código se anexa al método **menuAction** de una ficha.

```
;estaFicha::menuAction
method menuAction(var eventInfo MenuEvent)
if eventInfo.isPreFilter()
   then
    ; el código fuente que hay aquí se ejecuta sólo para cada
objeto de la Ficha

    Selección = eventInfo.menuChoice()

      if isBlank(Selección) = False then  ; si el usuario
seleciona una opción
        if msg.contains(Selección) then   ; si Selección se
corresponde  con un
                  ; índice de  la matriz dinámica Mensaje
        msgInfo(Selección, msg[Selección]); mostramos el valor
de ese elemento
        else                              ; si no, Selección no
se corresponde
                  ; con ningún elemento
          msgStop("¡Alto!", msg["Error"]) ; mostramos el valor
de otro elemento
        endif
      endif

      else
         ; el código fuente que haya aquí se ejecuta sólo para
la propia Ficha
endif
endmethod
```

Vea también getKeys
AnyType::view

empty

Método Borra todos los elementos de una matriz dinámica.

Tipo DynArray

Sintaxis **empty** ( )

Descripción Borra todos los elementos de una matriz dinámica, cuyo tamaño pasa a ser 0.

Ejemplo Este ejemplo muestra cómo funciona **empty** en una matriz dinámica. El código que sigue a continuación declara una matriz dinámica en la ventana Var de una ficha. Esta matriz dinámica es global a todos los objetos de la ficha.

```
;estaFicha::Var
Var
   Coche DynArray[] AnyType  ; se declara una matriz dinámica
endVar
```

El código siguiente se anexa al método **pushButton** de *botónDeRelleno*. Cuando se pulsa este botón, el código asigna varios elementos del DynArray *Coche*.

```
;botónDeRelleno::pushButton
method pushButton(var eventInfo Event)

Coche["Fabricante"]  = "Porshe"  ; cargamos el DynArray
Coche["Modelo"] = "911 sc"
Coche["Color"]  = "Azul Oscuro"
Coche["Año"]    = 1986
   ; Muestra el DynArray Coche e indica su tamaño en el título
(4)
Coche.view("El tamaño de Coche es: " + String(Coche.size()))
endmethod
```

El código siguiente se anexa al método **pushButton** del botón *botónDeVaciado*. Cuando se pulsa este botón, el código vacía la matriz *Coche* y muestra su contenido.

```
;botónDeVaciado::pushButton
method pushButton(var eventInfo Event)
Coche.empty()     ; Vaciamos el DynArray Coche

   ; Muestra el DynArray Coche e indica su tamaño en el título
(0)

Coche.view("El Tamaño de Coche es: " + String(Coche.size()))
endmethod
```

Vea también contains

getKeys

Método Carga una matriz redimensionable con índices de un DynArray existente.

Tipo DynArray

Sintaxis **getKeys** (var ***nombresClaves*** Array[] String)

Descripción Crea la matriz redimensionable especificada en ***nombresClaves*** y asigna el índice del DynArray a los valores de cada elemento. En otras palabras, este método almacena todos los valores de índice de un DynArray en una matriz redimensionable. Si *nombresClaves* existe, se sustituye sin pedir confirmación. Los valores de índice se ordenan en la nueva matriz de forma tal que el valor de índice menor se convierte en *nombresClaves*[1], y así sucesivamente.

Ejemplo Este ejemplo asigna varios elementos al DynArray *coche* y utiliza **getKeys** para crear una matriz que contenga los índices de *coche*. El resultado se muestra en un cuadro de diálogo de **view**.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   Coche DynArray[] AnyType
   matriz    Array[] String
endVar

; añadir algunos elementos al DynArray
Coche["Fabricante"]  = "Porshe"  ; cargamos el DynArray
Coche["Modelo"] = "911 sc"
Coche["Color"]  = "Azul Oscuro"
Coche["Año"]    = 1986

; ahora iniciamos matriz con 4 items y vemos la
; nueva matriz en un cuadro de diálogo
Coche.getKeys(matriz)
matriz.view()

; muestra
;  Color      (matriz[1])
;  Fabricante (matriz[2])
;  Modelo     (matriz[3])
;  Año        (matriz[4])

endmethod
```

Vea también contains

removeItem

Método Borra un elemento especificado de un DynArray.

Tipo DynArray

Sintaxis **removeItem** (const ***valor*** AnyType)

Descripción Borra el elemento (especificado por su índice) de *valor* de un DynArray. **removeItem** no tiene en cuenta el uso de mayúsculas o minúsculas.

Ejemplo El ejemplo siguiente concatena dos valores en una matriz dinámica y utiliza **removeItem** para borrar el elemento obsoleto.

El código que sigue a continuación se anexa a la ventana Var de una ficha:

```
;estaFicha::Var
var
   InfoCliente DynArray[] AnyType
endVar
```

Este código se anexa al método **pushButton** del botón *obtenInfoDeCliente*. Este código carga la matriz dinámica con información de direcciones. La aplicación podría tener un método personalizado que cargase la matriz dinámica desde una tabla o desde información introducida por el usuario.

```
;ObtenInfoDeCliente::pushButton
method pushButton(var eventInfo Event)
   ; cargamos el DynArray
InfoCliente["Compañía"]  = "Ordenadores Ultra-Rápidos S.A."
InfoCliente["Dirección"] = "Calle del Suspiro verde, 13"
InfoCliente["Ciudad"]    = "Cualquiera"
InfoCliente["Provincia"] = "Madrid"
InfoCliente["Prefijo"]   = "91"
InfoCliente["Teléfono"]  = "444-44-44"

   ; muestra el contenido del DynArray InfoCliente
InfoCliente.view("Contenido de InfoCliente")
endmethod
```

En el código que sigue, el valor del elemento Prefijo (si existe) se concatena con el valor del elemento Zip. Puesto que el elemento Prefijo ya no es necesario, este código lo borra de la matriz dinámica. El código siguiente se anexa al método **pushButton** del botón *concatenarTeléfonos*.

```
;concatenarTeléfonos::pushButton
method pushButton(var eventInfo Event)
if InfoCliente.contains("Teléfono") then
   InfoCliente["Prefijo"] = "(" + InfoCliente["Prefijo"] + ")"
+
           ; InfoCliente["Teléfono"]
   InfoCliente.removeItem("Teléfono") ; eliminamos el elemento
innecesario
else
   msgInfo("Con esto es bastante", "Los números se han
concatenado")
endif
```

```
   ; mostrar el resultado
InfoClient.view("Contenido de InfoCliente")
endmethod
```

Vea también empty
contains

size

Método Devuelve el número de elementos de un DynArray.

Tipo DynArray

Sintaxis **size** ( ) LongInt

Descripción Devuelve el número de elementos de un DynArray.

Ejemplo Para este ejemplo, el método **pushButton** de *esteBotón* crea una matriz dinámica y, a continuación, muestra su tamaño en un cuadro de diálogo.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   dy DynArray[] String
endVar

dy["Nombre"]    = "THAM"       ; cargamos el DynArray
dy["Ocupación"] = "Submarinismo"
dy["Contacto"]  = "José López"

   ; esto muestra "dy tiene 3 elementos"
msgInfo("dy", "dy tiene " + string(dy.size()) + " elementos.")
endmethod
```

Vea también contains

view

Método Muestra el contenido de un DynArray en un cuadro de diálogo.

Tipo DynArray

Sintaxis **view** ([const ***título*** String])

Descripción Enumera los índices y elementos de un DynArray en un cuadro de diálogo modal. La ejecución de ObjectPAL se suspende hasta que el usuario cierra este cuadro de diálogo. Puede especificarse un título para el cuadro de diálogo en *título* u omitirse para mostrar DynArray en su lugar. **view** ordena el DynArray por su índice antes de que se muestre el cuadro de diálogo.

A diferencia de muchos otros tipos de datos, los valores de DynArray mostrados en un cuadro de diálogo de **view** no pueden modificarse interactivamente. Consulte AnyType en este capítulo para más información sobre otros tipos de datos y el método **view**.

Ejemplo Para este ejemplo, el método **pushButton** del botón *esteBotón* crea una matriz dinámica y, a continuación, muestra su contenido ordenado en un cuadro de diálogo.

```
;esteBotón::pushButton
method pushButton(var eventInfo Event)
var
   dy DynArray[] String
endVar

dy["uno"]  = "primero"
dy["dos"]  = "segundo"
dy["tres"] = "tercero"
dy.view("Este DynArray contiene:")
   ; muestra lo siguiente:
   ; Este DynArray contiene:
   ; uno     primero
   ; dos     segundo
   ; tres    tercero
endmethod
```

Vea también contains

readFromClipboard

Método Lee un mapa de bits del Portapapeles.

Tipo Graphic

Sintaxis **readFromClipboard** ( ) Logical

Descripción Lee una imagen del Portapapeles en una variable del tipo Graphic. Si el Portapapeles contiene una imagen que puede copiarse en la variable Graphic, **readFromClipboard** devuelve True. Si el Portapapeles está vacío o no contiene una imagen válida, **readFromClipboard** devuelve False. **readFromClipboard** puede leer los formatos de mapa de bits (BMP) y de mapa de bits independiente del dispositivo (DIB).

Ejemplo En este ejemplo, supóngase que una ficha contiene un objeto multirregistro llamado VIDAMAR asociado con la tabla *Vidamar* y un botón llamado *ObtenGráfico*. El método **pushButton** de *ObtenGráfico* busca el registro con el valor Pez Fuego en el campo Nombre Común, y escribe el contenido del Portapapeles en el campo Imagen de ese registro. Si el Portapapeles está vacío o no contiene una imagen, el método **readFromClipboard** devuelve False y el valor del campo Imagen no cambia.

```
; ObtenGráfico::pushButton
method pushButton(var eventInfo Event)

var
  Gráfico Graphic
endVar

if VIDAMAR.locate( Nombre común ,  Pez Fuego ) then

  if Gráfico.readFromClipboard() then
    ; obtener el actual contenido del portapapeles en Gráfico
    VIDAMAR.edit()                ; activamos el modo de Edición
en la tabla
    VIDAMAR.Graphic = Gráfico     ; escribimos el mapa de bits
en el campo
    VIDAMAR.endEdit()             ; salimos del modo de Edición
  endIf
endIf
endmethod
```

Vea también readFromFile
writeToClipboard

readFromFile

Método Lee una imagen de un archivo.

Tipo Graphic

Sintaxis **readFromFile** (const ***nombreArchivo*** String) Logical

Descripción Lee una imagen de un archivo en disco especificado en *nombreArchivo*. **readFromFile** devuelve True si el nombre *nombreArchivo* existe y contiene un formato gráfico que pueda importarse; en caso contrario, devuelve False. Paradox puede importar los formatos gráficos siguientes: mapa de bits (BMP), Postscript encapsulado (EPS), formato de intercambio gráfico (GIF), Paintbrush (PCX) y formato de archivo con información etiquetada (TIF)

Ejemplo El ejemplo siguiente supone que una ficha contiene un botón llamado *recogerAjedrez* y un campo gráfico no asociado llamado *CampoMapaBits*. El método **pushButton** de *recogerAjedez* intenta leer el archivo de mapa de bits CHESS.BMP del directorio C:\WINDOWS y lo almacena en la variable *Ajedrez*. Si **readFromFile** es satisfactorio, *Ajedrez* se escribe en el objeto *CampoMapaBits*.

```
; recogerAjedrez::pushButton
method pushButton(var eventInfo Event)
var
  Ajedrez Graphic
endVar
; obtenemos el mapa de bits chess.bmp del directorio C:\
Windows,
; y lo escribimos en el gráfico CampoMapaBits
if Ajedrez.readFromFile( c:\\windows\\chess.bmp ) then
  CampoMapaBits = Ajedrez
endIf
endmethod
```

Vea también readFromClipboard
writeToFile

writeToClipboard

Método Escribe un mapa de bits en el Portapapeles.

Tipo Graphic

Sintaxis **writeToClipboard** ( ) Logical

Descripción Escribe un mapa de bits en el Portapapeles. **writeToClipboard** devuelve True si lo logra y False si falla. Los formatos copiados al Portapapeles pueden ser mapa de bits (BMP) o mapa de bits independiente del dispositivo (DIB).

Ejemplo El ejemplo siguiente supone que una ficha contiene un botón llamado *pasarAjedrezAlPortapap* y un archivo de mapa de bits llamado *CampoMapaBits*. El método **pushButton** de *pasarAjedrezAlPortapap* almacena el valor de *CampoMapaBits* en *Ajedrez* y escribe *Ajedrez* en el Portapapeles:

```
; pasarAjedrezAlPortapap::pushButton
method pushButton(var eventInfo Event)
var
  Ajedrez Graphic
endVar
; obtenemos el mapa de bits de CampoMapaBits,
; y lo guardamos en el Portapapeles
if NOT CampoMapaBits.isblank() then
  Ajedrez = CampoMapaBits
  Ajedrez.writeToClipboard()
endif
endmethod
```

Vea también writeToFile
readFromClipboard

writeToFile

Método Escribe un mapa de bits en un archivo.

Tipo Graphic

Sintaxis **writeToFile** (const ***nombreArchivo*** String) Logical

Descripción Escribe un mapa de bits en un archivo en disco especificado en *nombreArchivo*. **writeToFile** devuelve True si puede crearse el archivo especificado; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente supone que una ficha contiene un botón llamado *escribirAjedrezEnFichero* y un mapa de bits llamado *CampoMapaBits*. El método **pushButton** de *escribirAjedrezEnFichero* almacena el valor de *CampoMapaBits* en *Ajedrez* y escribe *Ajedrez* en un archivo llamado AJEDREZ1.BMP en el directorio actual:

```
; escribirAjedrezEnFichero::pushButton
method pushButton(var eventInfo Event)
var
  ajedrez Graphic
endVar
; obtenemos el mapa de bits de CampoMapaBits
; y lo escribimos en el Portapapeles
if NOT CampoMapaBits.isblank() then
  ajedrez=CampoMapaBits
  ajedrez.writeToFile( ajedrez1.bmp )
endIf
endMethod
```

Vea también writeToClipboard

logical

Principiante

Procedimiento Convierte un valor al tipo Logical.

Tipo Logical

Sintaxis **logical** (const ***valor*** AnyType) Logical

Descripción Convierte el tipo de datos de *valor* a Logical. Si *valor* es de un tipo de datos numérico, los valores distintos de cero se evalúan como True, mientras que cero se evalúa como False. Si *valor* es una cadena, debe evaluarse como True o False (sin embargo, es posible emplear True o False sin las comillas). ObjectPAL también proporciona constantes Logical: On y Yes para True, y Off y No para False; consulte General en el cuadro de diálogo Constantes.

Ejemplo Para este ejemplo, el método **pushButton** de un botón llamado *mostrarLogico* crea una cadena, la convierte al tipo Logical y muestra el resultado:

```
; mostrarLogico::pushButton
method pushButton(var eventInfo Event)
var
  Dato      String
  Resultado Logical
endVar
Dato =  True                ; asigna a una cadena el dato True
Resultado = logical(Dato)   ; y lo amolda al tipo Logical
Resultado.view()            ; muestra el resultado El título
muestra Logical
endmethod
```

Vea también Logical

bitAND

Método Realiza una operación binaria AND sobre dos valores.

Tipo LongInt

Sintaxis **bitAND** (const ***valor*** LongInt) LongInt

Descripción Devuelve el resultado de una operación binaria AND sobre valor. **bitAND** opera sobre las representaciones binarias de dos enteros, comparándolas bit a bit. La tabla real de **bitAND** es:

a	**b**	**a bitAND b**
0	0	0
1	0	0
0	1	0
1	1	1

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *andDeDosNúmeros* toma dos enteros y realiza un cálculo binario AND sobre ellos. El resultado del cálculo se muestra en un cuadro de diálogo.

```
; andDeDosNúmeros::pushButton
method pushButton(var eventInfo Event)
var
  a, b LongInt
endVar
a = 33333         ; en binario: 00000000 00000000 10000010
00110101
b = -77777        ; en binario: 11111111 11111110 11010000
00101111
a.bitAND(b)       ; en binario: 00000000 00000000 10000000
00100101
msgInfo( El resultado de a bitAND b es: , a.bitAND(b))      ;
muestra 32805
endmethod
```

Vea también bitOR
bitXOR

bitIsSet

Método Informa de si un bit es 1 ó 0.

Tipo LongInt

Sintaxis **bitIsSet** (const ***valor*** LongInt) Logical

Descripción Examina la representación binaria de un número entero, informando de si el bit **valor** es 0 ó 1. **bitIsSet** devuelve True si el bit especificado es 1, y False si es 0.

valor es un número especificado mediante 2n, siendo *n* un entero entre 0 y 30. El exponente *n* corresponde a una unidad menos que la posición del bit que se comprueba, contando desde la derecha. Por ejemplo para especificar el tercer bit desde la derecha, utilice 4 (2(3-1), que es 22).

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *esUnConjuntoDeBits*, examina los valores existentes en dos objetos de campo no asociado: *QuéBit*y *QuéNumero*. *QuéBit* contiene la posición del bit (desde la derecha) que se desea comprobar. *QuéNumero* contiene el número entero grande que se va a comprobar.

El método **pushButton** utiliza *QuéBit* para calcular el valor de la posición y, a continuación, asigna el resultado a *NúmeroBit*. Entonces, el método comprueba en *Num* si el bit *NúmeroBit* está definido y muestra el resultado Logical en un cuadro de diálogo **msgInfo**.

```
; esUnConjuntoDeBits::pushButton
method pushButton(var eventInfo Event)
var
  NúmeroBit,
  Num        LongInt
endVar
; obtenemos el número de posición del bit del campo
; QuéBit y lo transforma en múltiplo de 2
NúmeroBit = LongInt(pow(2, QuéBit - 1))
; obtenemos el número para comprobar desde el campo QuéNumero
Num = QuéNumero
; ¿está en Num el bit con valor NumeroBit 1?
msgInfo("¿Está asignado Bit?", Num.bitIsSet(NumeroBit))
endmethod
```

El ejemplo siguiente ilustra cómo puede utilizarse **bitIsSet** para mostrar un número entero grande como número binario. El método **pushButton** de *mostrarBinario* construye una cadena de ceros y unos comprobando cada bit de un entero grande de cuatro bytes. Por razones de legibilidad, se añade un blanco a la cadena cada ocho dígitos.

```
; mostrarBinario::pushButton
method pushButton(var eventInfo Event)
var
  CadenaBin String          ; para construir la cadena en
binario
  Num       LongInt
  i         SmallInt  ; para el bucle
endVar
```

```
if NOT QuéNumero.isBlank() then
  Num = QuéNumero                        ; obtenemos el valor de
QuéNumero
  CadenaBin = ""                         ; iniciamos la cadena
  for i from 0 to 30
    if Num.bitIsSet(LongInt(pow(2, i))) then
      CadenaBin = "1" + CadenaBin        ; añade cero al
principio de la cadena
    else
      CadenaBin = "0" + CadenaBin        ; añade cero al
principio de la cadena
    endif
    if i = 7 OR i = 15 OR i = 23 then
      CadenaBin = " " + CadenaBin        ; añade un espacio cada
8 dígitos
    endif
  endfor
  if Num  0 then
    CadenaBin = "1" + CadenaBin          ; añade el bit de signo
  else
    CadenaBin = "0" + CadenaBin
  endif
  ; muestra el número
  message("El número equivalente en binario es ", CadenaBin)
endif
endmethod
```

Vea también bitAND
bitOR
bitXOR

bitOR

Método Realiza una operación binaria OR sobre dos valores.

Tipo LongInt

Sintaxis **bitOR** (const ***valOR*** LongInt) LongInt

Descripción Devuelve el resultado de una operación binaria OR sobre *valor*. **bitOR** opera sobre las representaciones binarias de dos enteros, comparándolas bit a bit. La tabla real de **bitOR** es:

a	**b**	**a bitOR b**
0	0	0
1	0	1
0	1	1
1	1	1

Ejemplo Para el ejemplo siguiente, el método **pushButton** de un botón llamado *orDeDosNúmeros* toma dos enteros y realiza un cálculo binario OR sobre ellos. El resultado del cálculo se muestra en un cuadro de diálogo.

```
; orDeDosNúmeros::pushButton
method pushButton(var eventInfo Event)
var
  a, b LongInt
endVar
a = 33333    ; en binario: 00000000 00000000 10000010 00110101
b = -77777   ; en binario: 11111111 11111110 11010000 00101111
a.bitOR(b)   ; en binario: 11111111 11111110 11010010 00111111
msgInfo( 33333 OR -77777", a.bitOR(b)) ; muestra -77249
endmethod
```

Vea también bitAND
bitXOR

bitXOR

Método Realiza una operación binaria XOR sobre dos valores.

Tipo LongInt

Sintaxis **bitXOR** (const ***valor*** LongInt) LongInt

Descripción Realiza una operación binaria XOR (OR exclusivo) sobre *valor*. **bitXOR** opera sobre las representaciones binarias de dos enteros, comparándolas bit a bit. La tabla real de **bitXOR** es:

a	**b**	**a bitXOR(b)**
0	0	0
1	0	1
0	1	1
1	1	0

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *xorDeDosNúmeros* toma dos enteros y realiza un cálculo binario XOR sobre ellos. El resultado del cálculo se muestra en un cuadro de diálogo.

```
; xorDeDosNúmeros::pushButton
method pushButton(var eventInfo Event)
var
  a, b LongInt
endVar
a = 33333     ; en binario: 00000000 00000000 10000010 00110101
b = -77777    ; en binario: 11111111 11111110 11010000 00101111
a.bitXOR(b)   ; en binario: 11111111 11111110 01010010 00011010
msgInfo( 33333 XOR -77777", a.bitXOR(b)) ; muestra -110054
endmethod
```

Vea también bitAND
bitOR

LongInt

Principiante

Procedimiento Convierte un valor a LongInt.

Tipo LongInt

Sintaxis **LongInt** (const ***valor*** AnyType) LongInt

Descripción Convierte el tipo de datos de *valor* en un número entero grande. Si se realiza la conversión desde un tipo más preciso (como Number), puede perderse precisión.

Ejemplo El ejemplo siguiente asigna un número a *x*, convierte *x* a LongInt y asigna el resultado a *l*. Observe que se pierde la precisión decimal de *x* cuando se convierte a LongInt y se asigna a *l*.

```
; convertirAEntero::pushButton
method pushButton(var eventInfo Event)
var
  x Number
  l LongInt
endVar
x = 12.34         ; da un valor a x
x.view()          ; muestra x, el título del cuadro de diálogo
será "Number"
l = LongInt(x)    ; amolda x al tipo LongInt y se lo asigna a l
l.view()          ; muestra l, nótese que no hay decimales
                  ; muestra 12
endmethod
```

Vea también AnyType::view

memo

Procedimiento Convierte un valor a Memo.

Tipo Memo

Sintaxis **memo** (const ***valor*** AnyType [, const ***valor*** AnyType]*) String

Descripción Convierte la expresión *valor* en un Memo. Si se especifican varios argumentos, este método los convertirá todos en Memo y los concatenará en un Memo.

Ejemplo Este ejemplo supone que FICHSDOC.DB existe y tiene un campo alfanumérico llamado Memo Name, un campo Date llamado Memo Date y un campo memo con formato llamado Memo Data. Para este ejemplo, una ficha tiene campos no asociados llamados *ObjetoCadena* y *ObjetoMemo*, y un botón llamado *esteBotón*. El código anexado al método **pushButton** de *esteBotón* define un TCursor para buscar un registro en particular en *FichsDoc*. A continuación, el código convierte y concatena el contenido de los tres campos *FichsDoc* en un valor de String y, a continuación, en un valor de Memo. El valor convertido a String se muestra en el objeto *ObjetoCadena*, mientras que el convertido a Memo se muestra en el objeto *ObjetoMemo*. Obsérvese que, al convertir a String, la información de formateo no se muestra en *ObjetoCadena*. Al convertir a Memo, *ObjetoMemo* muestra toda la información de formateo.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  tc TCursor
endVar

if tc.open("FichsDoc.db") then
  if tc.locate("Nombre del campo Memo", "Notas del proyecto")
then
    ; esta línea amolda los datos de los campos de FichsDoc.db
al tipo String
    ; por ello, los datos que aparecen en ObjetoCadena se
muestran SIN formato
    ObjetoCadena.value = string(tc."Nombre del campo Memo", "\
t",
                        tc."Fecha del campo Memo", "\n", tc."Datos
del campo Memo")
    ; esta línea amolda los datos de los tres campos de
FichsDoc.db al tipo Memo
    ; por ello, los datos de ObjetoMemo se muestran como texto
CON FORMATO
    ObjetoMemo.value = memo(tc."Nombre del campo Memo", "\t",
          tc."Fecha del campo Memo", "\n", tc."Datos del campo
Memo")
  else
    msgStop("Error", "No encuentro Notas del proyecto.")
  endif
else
  msgStop("Error", "No puedo abrir la tabla FichsDoc.")
endif
```

```
endmethod
```

readFromFile

Método Lee un memo de un archivo.

Tipo Memo

Sintaxis **readFromFile** (const ***nombreArchivo*** String) Logical

Descripción Lee un memo de un archivo en disco especificado en *nombreArchivo*. Este método sólo lee texto. No lee el formateo de memos con formato.

Ejemplo El ejemplo siguiente lee el contenido de un archivo de texto en un campo memo de una tabla. Supóngase que una tabla llamada *NotasPro* existe en el directorio actual y tiene los campos siguientes: *FechaProyec* (campo Date) y *NotasProyec* (campo Memo). El método **pushButton** de un botón llamado *obtenArchivo* abre, edita e inserta un nuevo registro en la tabla *NotasPro* y, a continuación, rellena el campo *FechaProyec* con la fecha actual y el campo *NotasProyec* con el texto de un archivo llamado NOTAS.TXT.

```
; obtenArchivo::pushButton
method pushButton(var eventInfo Event)
var
  Archivo Memo

  TC      TCursor
endVar

if TC.open("NotasPro.db") then    ; abre un TCursor para
NotasPro.db
  if Archivo.readFromFile("Notas.txt") then
    ; si la lectura del archivo ha tenido éxito
    TC.edit()                     ; editamos la tabla Notas del
proyecto
    TC.insertRecord()             ; insertamos un registro vacío
    TC.FechaProyec = today()      ; rellenamos el campo
FechaProyec
    TC.NotasProyec = MemoFile     ; escribimos el campo memo en
el campo NotasProyec
    TC.endEdit()                  ; desactivamos el modo de
Edición
  endif
  TC.close()                      ; cerramos el TCursor
endIf
endmethod
```

Vea también writeToFile

writeToFile

Método Escribe un memo en un archivo.

Tipo Memo

Sintaxis **writeToFile** (const ***nombreArchivo*** String) Logical

Descripción Escribe un memo en un archivo en disco especificado en *nombreArchivo*. Este método escribe sólo texto, no el formateo de memos con formato.

Ejemplo Este ejemplo escribe el contenido de un memo en un archivo de texto. Supóngase que hay en el directorio actual una tabla llamada *NotasPro* con los campos *FechaProyec* (campo Date) y *NotasProyec* (campo Memo). El método **pushButton** de un botón llamado *EscribirArchivo* abre la tabla *NotasPro*, busca un registro con la fecha actual y escribe el contenido del campo *NotasPro* de ese registro en un archivo llamado NOTASDIA.TXT.

```
; obtenArchivo::pushButton
method pushButton(var eventInfo Event)
var
  Archivo Memo
  TC      TCursor
endVar
  if TC.open("NotasPro.db") then              ; abrimos la tabla
Notas del Proyecto
    if TC.locate("FechaProyec", today()) then
      if NOT (TC.NotasProyec = blank()) then  ; ver si el campo
memo está vacío
        Archivo = TC.NotasProyec              ; si no,
escribimos en la variable Archivo
        Archivo.writeToFile("NotasDia.txt")   ; escribe Archivo
en un archivo de texto
      endif
    endif
    TC.close()                                ; cerramos el
TCursor
endIf
endmethod
```

Vea también readFromFile

abs

Principiante

Método Devuelve el valor absoluto de un número.

Tipo Number

Sintaxis **abs** ( ) Number

Descripción Elimina el signo de un valor numérico.

Ejemplo Para el ejemplo siguiente, supóngase que una ficha contiene tres objetos de campo: *CantidadPrevista*, *CantidadReal* y *PorCentajeDif*. El método **newValue** de *CantidadReal* halla la diferencia entre *CantidadPrevista* y *CantidadReal* y calcula la exactitud de la previsión. Dependiendo de los valores de *CantidadPrevista* y *CantidadReal*, el número en que varía puede ser positivo o negativo. Para averiguar el porcentaje de error, se emplea **abs** para obtener el valor absoluto del número, que se multiplica por 100 para obtener un porcentaje. Este código se anexa al método **newValue** de *CantidadReal*:

```
; CantidadReal::newValue
method newValue(var eventInfo Event)
var
  Diferencia Number
endVar
; no lo ejecutamos si newValue se llama al comenzar, o
; si uno de los campos involucrados está vacío
 if eventInfo.reason() <>ValorDeInicio then
   if NOT self.isBlank() AND
      NOT CantidadPrevista.isBlank() then
     ; encontrar cuanto difiere la previsión de lo real
     Diferencia = (CantidadPrevista -
Number(self.Value))/CantidadPrevista
     PorCentajeDif = Diferencia.abs() * 100   ; obtener la
variación como valor absoluto
   else
     msgStop("Error", "El campo CantidadPrevista no puede estar
vacío.")
   endif
endif
endmethod
```

Vea también number

acos

Principiante

Método Devuelve el arcocoseno de dos cuadrantes de un número.

Tipo Number

Sintaxis **acos** ( ) Number

Descripción Dado un número entre -1 y 1, **acos** devuelve un valor numérico entre 0 y pi, expresado en radianes. **acos** se denomina el arcocoseno de dos cuadrantes porque devuelve valores dentro los cuadrantes 1 y 4 (es decir, entre pi/2 y pi/2). **acos** es la inversa de **cos** (si **acos**(*x*) = *y*, entonces **cos**(*y*) = *x*).

Ejemplo El método **pushButton** del botón *ArcoCoseno* calcula y muestra el arcocoseno de un ángulo de 30 grados.

```
; ArcoCoseno::pushButton
method pushButton(var eventInfo Event)
var
  TreintaGrados Number
endVar
TreintaGrados = PI / 3,0
msgInfo("El arco coseno de 30 grados es", TreintaGrados.acos())
    ; muestra 1,02
endmethod
```

Vea también asin
atan

asin

Principiante

Método Devuelve el arcoseno de dos cuadrantes de un número.

Tipo Number

Sintaxis **asin** ( ) Number

Descripción Dado un número entre 1 y 1, **asin** devuelve un valor numérico entre pi/2 y pi/2, expresado en radianes.

Ejemplo Para este ejemplo, el método **pushButton** del botón *buscarArcoseno* muestra el arcoseno de un número.

```
; buscarArcoseno::pushButton
var
  x Number
endvar
x = ,5
msgInfo("Arco seno de 0,5", x.asin())   ; muestra 0,52
endmethod
```

Vea también acos
atan
cos
sin

atan

Principiante

Método Devuelve la arcotangente de dos cuadrantes de un número.

Tipo Number

Sintaxis **atan** ( ) Number

Descripción Dada una tangente en radianes, **atan** devuelve el ángulo en radianes. **atan** se denomina arcotangente de dos cuadrantes porque devuelve valores dentro de los cuadrantes 1 y 4 (es decir, entre pi/2 y pi/2). **atan** es la inversa de **tan** (si atan(x) = y, entonces tan(y) = x).

Ejemplo En este ejemplo, el método **pushButton** de *ArcoTangente* calcula la arcotangente de dos cuadrantes de *x* e *y* y muestra el resultado.

```
; ArcoTangente::pushButton
method pushButton(var eventInfo Event)
var
  x  Number
  ComprobPi, CuarentaYCincoGrados Number
endvar
x = 1
CuarentaYCincoGrados = x.atan()
msgInfo("45 grados en radianes: ", CuarentaYCincoGrados)  ; 0,79
ComprobPi = CuarentaYCincoGrados * 4          ; Pi radianes =
180 grados
msgInfo("Pi: ", format("w12,10", ComprobarPi))
endmethod
```

Vea también cos
sin
tan
tanh
atan2

atan2

Principiante

Método Devuelve la arcotangente de los cuatro cuadrantes de un número.

Tipo Number

Sintaxis **atan2** (const ***x*** Number) Number

Descripción Dado un seno en radianes, **atan** devuelve un ángulo en radianes cuyo coseno es x. **atan2** se denomina arcotangente de los cuatro cuadrantes porque devuelve valores en los cuatro cuadrantes.

Ejemplo Dado un seno en radianes, **atan** devuelve un ángulo en radianes cuyo coseno es x. **atan2** se denomina arcotangente de los cuatro cuadrantes porque devuelve valores en los cuatro cuadrantes.

```
; obtenerAtan2::pushButton
method pushButton(var eventInfo Event)
var
  x,
  y,
  ComprobarPi,
  CuarentaYCincoGrados Number
endvar
x = 1                                   ; el ángulo cuya
tangente es 1 / 1
y = 1                                   ; es un ángulo de 45
grados
CuarentaYCincoGrados = x.atan2(y)
msgInfo("45 grados en radianes: ", CuarentaYCincoGrados)  ; 0,79
ComprobarPi = CuarentaYCincoGrados * 4,0           ; Pi
radianes = 180 grados
msgInfo("Pi: ", format("w12,10", ComprobarPi))
endmethod
```

Vea también cos
sin
tan
tanh

ceil

Principiante

Método Redondea una expresión numérica hacia arriba al número entero más próximo.

Tipo Number

Sintaxis **ceil** ( ) Number

Descripción Redondea una expresión numérica hacia arriba (hacia el infinito positivo) al número entero más próximo.

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *CeilContraRound* calcula el valor entero aproximado por exceso de un número y, a continuación, muestra el valor entero redondeado del mismo número:

```
; CeilContraRound::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = 3,1
msgInfo("Aplicando Ceil a " + String(x) + " obtenemos",
ceil(x))                        ;   muestra 4,0
msgInfo("Aplicando Round a " + String(x) + " obtenemos",
x.round(0))     ; muestra 3
endmethod
```

Vea también floor

cos

Principiante

Método Devuelve el coseno de un ángulo.

Tipo Number

Sintaxis **cos** ( ) Number

Descripción Devuelve el valor entre -1 y 1 del coseno de un valor o expresión que representa el tamaño del ángulo en radianes.

Ejemplo En este ejemplo, el método **pushButton** del botón *Coseno* calcula y muestra el coseno de un ángulo de 60 grados:

```
; Coseno::pushButton
method pushButton(var eventInfo Event)
var
  SesentaGrados Number
endVar
SesentaGrados = PI / 3,0
msgInfo("El coseno de 60 grados es", SesentaGrados.cos())
    ; muestra 0,50
endmethod
```

Vea también cosh
sin
tan

cosh

Principiante

Método Devuelve el coseno hiperbólico de un ángulo.

Tipo Number

Sintaxis **cosh** ( ) Number

Descripción Devuelve el coseno hiperbólico de un valor o expresión que representa el tamaño del ángulo en radianes. La fórmula utilizada es

cosh(ángulo) = (exp(ángulo) + exp(-ángulo)) / 2

Ejemplo El método **pushButton** del botón *CosenoHiperbólico* calcula y muestra el coseno hiperbólico de 60 grados.

```
; CosenoHiperbólico::pushButton
method pushButton(var eventInfo Event)
var
  SesentaGrados Number
endVar
SesentaGrados = PI / 3.0
msgInfo("El coseno hiperbólico de " + format("W8,6",
SesentaGrados) +
        " radianes", format("W14,12", SesentaGrados.cosh()))
; muestra 1,600286857702
endmethod
```

Vea también cos
sin
tan

exp

Principiante

Método Devuelve el exponencial (en base *e*) de un número.

Tipo Number

Sintaxis **exp** ( ) Number

Descripción Calcula *ex*, donde *e* es la constante 2,7182845905. El método inverso es **ln**.

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *ObtenNumeroe* calcula y muestra la base *e* de 1:

```
; ObtenNumeroe::pushButton
method pushButton(var eventInfo Event)
msgInfo("El número e elevado a 1,0", format("W14,12",
exp(1,0)))
; mostramos el resultado de exp(1) con formato de máxima
precisión
endmethod
```

Vea también ln
log

floor

Principiante

Método Redondea una expresión numérica hacia abajo al número entero más próximo.

Tipo Number

Sintaxis **floor** ( ) Number

Descripción Redondea una expresión numérica hacia abajo (hacia el infinito negativo) al número entero más próximo.

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *floorContraRound* utiliza **floor** para aproximar *x* al número entero más próximo por defecto. Como comparación, para el mismo número, **round** produce número entero mayor.

```
; floorContraRound::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar

x = 3,9
msgInfo("Aplicando Floor a " + String(x) + " obtenemos",
floor(x))
                                    ; muestra 3,0
msgInfo("Aplicando Round a " + String(x) + " obtenemos",
x.round(0))
                                    ; muestra 4,0
endmethod
```

Vea también ceil

fraction

Principiante

Método Devuelve la parte fraccionaria de un número.

Tipo Number

Sintaxis **fraction** ( ) Number

Descripción Devuelve la parte fraccionaria de un número; es decir, la parte del número situada a la derecha de la coma decimal.

Ejemplo En este ejemplo, el método **pushButton** de *BotónDeFracción* muestra la parte fraccionaria de un valor numérico:

```
; BotónDeFracción::pushButton
method pushButton(var eventInfo Event)
var
  Num Number
endVar
Num = 12,23
msgInfo("La parte fraccionaria de " + String(Num) +  es ,
        Num.fraction())
          ; muestra 0,23
endmethod
```

Vea también mod

fv

Principiante

Método Devuelve el valor futuro de una serie de pagos iguales.

Tipo Number

Sintaxis **fv** (const ***tipoInterés*** Number, ***periodos*** Number **)** Number

Descripción Devuelve el valor futuro de una serie de *periodos* de pago iguales, invertidos con el tipo de interés *tipoInterés*. *tipoInterés* se expresa como un número decimal (como 0,12), en lugar de como porcentaje (12%). Asegúrese de que el periodo de aplicación del interés coincida con el periodo de depósito; es decir, si los depósitos son mensuales, el tipo de interés debe ser mensual también.

La fórmula empleada es

FV = *pago(pow(*1 + *interés, periodos)* - 1*) / interés*

fv también se denomina valor compuesto o de futuro de una renta, porque puede emplearse para calcular la cantidad acumulada en un plan de pensiones cuando se realizan pagos iguales regulares en el tiempo.

Ejemplo Este ejemplo calcula el valor de una cuenta de jubilación personal al 14,5% si se despositasen 166,67 pesetas cada mes durante 30 años.

```
; PlanDeFuturo::pushButton
method pushButton(var eventInfo Event)
var
  CantidadDepositada,
  Interés,
  NumPagos,
  PlanPensiones      Number
endVar
IntRate = 0,145 / 12                    ; convertimos el interés
anual en interés mensual
NumPagos = 360                          ; pagos mensuales
durante 30 años
CantidadDepositada = 166,67             ; cantidad depositada
mensualmente (200.000 Pts./año)
PlanPensiones = CantidadDepositada.fv(Interés, NumPagos)
msgInfo("Plan de Pensiones", "Depositando " +
String(CantidadDepositada) +
        " al mes durante " + String(NumPagos/12) + " años con
un" +
        String(Interés * 12 * 100) + "% se producen " +
String(iraValue) +
        ". ¡Serás anciano pero rico!")
; muestra "Depositando 166,67 al mes durante 30 años
;          con un 14.50% se producen 1.027.394,23 ..."
endmethod
```

Vea también pmt
pv

ln

Principiante

Método Devuelve el logaritmo neperiano de una expresión numérica.

Tipo Number

Sintaxis **ln** ( ) Number

Descripción Calcula el logaritmo neperiano en base *e* de un valor positivo. La constante se aproxima al valor 2,7182845905. Si el valor es 0 o negativo, el método **ln** falla.

El método inverso es **exp**. Utilice **log** para calcular logaritmos en base 10.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *LogaritmoNatural* calcula y muestra el logaritmo neperiano de varios números:

```
; LogaritmoNatural::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = 2.71828
msgInfo("Logaritmo Natural de " + Format("W10,6", x), ln(x)) ;
muestra 1,00
x = 7.3891
msgInfo("Logaritmo Natural de " + Format("W10,6", x), ln(x)) ;
muestra 2,00
x = 20.0855
msgInfo("Logaritmo Natural de " + Format("W10,6", x), ln(x)) ;
muestra 3,00
endmethod
```

Vea también exp
log

log

Principiante

Método Devuelve el logaritmo en base 10 de una expresión numérica.

Tipo Number

Sintaxis **log** ( ) Number

Descripción Devuelve el logaritmo en base 10 de un valor o expresión numérica. Si el valor es 0 o negativo, **log** falla.

Para calcular logaritmos neperianos, utilice **ln**

Ejemplo

```
; Logaritmo::pushButton

method pushButton(var eventInfo Event)
var
  x Number
endVar
x = 10
msgInfo("Logaritmo de " + String(x), log(x))  ; muestra 1,00
x = 100
msgInfo("Logaritmo de " + String(x), log(x))  ; muestra 2,00
x = 1000
msgInfo("Logaritmo de " + String(x), log(x))  ; muestra 3,00
endmethod
```

Vea también exp
ln

max

Principiante

Procedimiento Devuelve el mayor de dos números.

Tipo Number

Sintaxis **max** (const ***x1*** AnyType, const ***x2*** AnyType) AnyType

Descripción Devuelve el mayor de los valores *x1* y *x2*.

Ejemplo En el ejemplo siguiente, se desea averiguar la deducción por gastos médicos permitida a efectos fiscales. El método **pushButton** de *DeducciónSanitaria* halla el máximo del 7,5% del *IRPF* o *GastoMedico* y, a continuación, deduce el 7,5% de *IRPF* en el resultado. El cálculo del número máximo en primer lugar garantiza que no se devolverá un número negativo.

```
; DeducciónSanitaria
method pushButton(var eventInfo Event)
var
  GastoMedico,
  IRPF        Number
endVar
IRPF = 32000,45
GastoMedico = 4035,24
msgInfo("Deducción Sanitaria permitida",
        max(GastoMédico, IRPF * 0,075) - (IRPF * 0,075))
    ;muestra 1.635,21
; asume que se puede deducir sólo la parte cuyos gastos médicos
y
; dentales sean mayores que el 7,5% de IRPF
endmethod
```

Vea también min

min

Principiante

Procedimiento Devuelve el menor de dos números.

Tipo Number

Sintaxis **min** (const ***x1*** AnyType, const ***x2*** AnyType) AnyType

Descripción Devuelve el menor de los valores *x1* y *x2*.

Ejemplo En este ejemplo, se desea calcular la cantidad máxima de contribuciones a la caridad deducibles de impuestos y que no supere el 30% de la base imponible que puede deducirse. El método **pushButton** del botón *Deducción* halla y muestra el mínimo del 30% del *IRPF* y de *contribución*.

```
; Deducción::pushButton
method pushButton(var eventInfo Event)
var
  Contribución,
  IRPF          Number
endVar
IRPF = 32000,45             ; Impuesto sobre la Renta
Contribución = 12000        ; contribuciones del año al estado
msgInfo("Deducción permitida", min(Contribución, IRPF *
0,30)) ; muestra 9.600,13
; asume que las contribuciones al estado, que son un 30% a
deducir en el IRPF,
; están permitidas como deducciones
endmethod
```

Vea también max

mod

Principiante

Método Devuelve el resto cuando se divide un número por otro.

Tipo Number

Sintaxis **mod** (const ***módulo*** Number) Number

Descripción Devuelve el resto (o módulo) cuando un número se divide por el valor de *módulo*. Si *módulo* es 0, **mod** devuelve 0.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *mostrarResto* calcula y muestra el módulo de una serie de operaciones de división:

```
; mostrarResto::pushButton
method pushButton(var eventInfo Event)
var
  x, m Number
endVar
x = 8
msgInfo("El resto de " + String(x) + "/" + "3 es",
         x.mod(3))                                          ;
muestra 2
msgInfo("El resto de " + String(x) + "/" + "12 es",
        x.mod(12))                                          ;
muestra 3
x = -2
msgInfo("El resto de " + String(x) + "/" + "10 es",
        x.mod(10))                                          ;
muestra -2
x = -10
msgInfo("El resto de " + String(x) + "/" + "-100 es",
        x.mod(-100))                                        ;
muestra -10
endmethod
```

Vea también fraction

number

Principiante

Procedimiento Convierte un valor a Number.

Tipo Number

Sintaxis **number** (const ***valor*** AnyType) Number

Descripción Convierte *valor* a Número. *valor* debe estar en la forma de un número válido que pueda introducirse en un campo. **number** se utiliza para convertir un tipo no numérico a Number cuando un operando numérico es necesario en una expresión o un argumento numérico es necesario en un procedimiento o método. **number** se comporta igual que **numVal**.

Ejemplo En el ejemplo siguiente, se declara una variable *x* como un String y se le asigna una cadena de números. El método **pushButton** del botón *mostrarDoble* convierte *x* a Number antes de multiplicarlo por 2 y muestra el resultado:

```
; mostrarDoble::pushButton
method pushButton(var eventInfo Event)
var
  x String
endVar
x = "1.123,54"
; amolda x al tipo Number antes de multiplicarlo por 2
msgInfo("El Doble de " + x + " es", Number(x) * 2)        ;
muestra 2.247,08
endmethod
```

Vea también numVal

numVal

Procedimiento Convierte un valor a Number.

Tipo Number

Sintaxis **numVal** (const ***valor*** AnyType) Number

Descripción Convierte *valor* a Number. *valor* debe tener la forma de un número válido que pueda introducirse en un campo. **numVal** se utiliza con mayor frecuencia para convertir un tipo no numérico a Number cuando un operando numérico es necesario en una expresión o un argumento numérico es necesario en un procedimiento o método. **numVal** se comporta igual que **number**.

Ejemplo En el ejemplo siguiente, se declara una variable *x* como String y se le asigna una cadena de números. El método **pushButton** del botón *mostrarDoble* convierte *x* a Number antes de multiplicarlo por dos y muestra el resultado:

```
; mostrarDoble::pushButton
method pushButton(var eventInfo Event)
var
  x String
endVar
x = "1.123,54"
; amolda x al tipo Number antes de multiplicarlo por 2
msgInfo("El Doble de " + x + " es", Number(x) * 2)        ;
muestra 2.247,08
endmethod
```

Vea también number

pmt

Principiante

Método Devuelve el pago periódico que es necesario para reembolsar un préstamo.

Tipo Number

Sintaxis **pmt** (const ***tipoInterés*** Number, const ***periodos*** Number **)** Number

Descripción Devuelve el pago constante regular que es necesario para reembolsar (devolver) un préstamo. La fórmula utilizada es:

PMT = *p* * *i* / (1 - (1 + *i*) -*t*)

donde *p* = principal de la deuda, *i* = tipo de interés efectivo por periodo y *t* = plazo del préstamo (número de periodos de pago).

Los pagos se consideran vencidos al final de cada periodo.

pmt funciona con los préstamos amortizables (por ejemplo, las hipotecas convencionales) en que parte del pago consiste en el interés sobre el principal restante y el resto devuelve parte del principal del préstamo. **pmt** no funciona con préstamos al consumo, como cuentas de crédito o préstamos para adquirir automóviles.

El tipo de interés utilizado en **pmt** se expresa en *tipoInterés* como un número decimal (por ejemplo, 0,12), no como un porcentaje (12%). Asegúrese de que el periodo del interés coincide con el periodo de pago; es decir, si los pagos son mensuales, el tipo de interés debe ser mensual también. Puesto que el tipo de interés aplicado a los préstamos de amortización (hipotecas) suele ser anual, es necesario dividirlo por 12 para los pagos mensuales, por 4 para los trimestrales, y así sucesivamente.

Comience con el tipo de interés anual nominal contratado, no con el tipo anual equivalente (TAE) que lo acompaña.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *Pago* calcula el pago mensual de un préstamo a 24 meses de 1.000 pesetas al 12%:

```
; Pago::pushButton
method pushButton(var eventInfo Event)
var
  PagoMensual,
  CantidadPrestada,
  Interés,
  NumPagos Number
endVar
CantidadPrestada = 1000  ; prestadas 1000 pts.
Interés = 0,12 / 12      ; 12 por ciento de interés anual
NumPagos = 24                                ; 1 pago al mes
durante 2 años
PagoMensual = CantidadPrestada.pmt(Interés, NumPagos)
msgInfo("Pago mensual", "El pago mensual de un préstamo de " +
        String(CantidadPrestada) + " al " + String(Interés * 12
* 100) +
        "% de interés durante " + String(SmallInt(NumPagos)) +
        " meses es " + String(PagoMensual))              ; el
pago es 47,07 pts.
```

```
endmethod
```

Vea también fv
pv

pow

Principiante

Método Eleva un número a una potencia.

Tipo Number

Sintaxis **pow** (const ***exponente*** Number) Number

Descripción Devuelve el valor de un número elevado a la potencia especificada en *exponente*. Si el resultado es mayor que 10308 o menor que 10-307, se obtiene un error.

Ejemplo En el ejemplo siguiente, el método pushButton del botón *elevarDos* calcula *Baseexponente* y muestra el resultado:

```
; elevarDos::pushButton
method pushButton(var eventInfo Event)
var
  Base,
  Exponente    Number
endVar
Base = 2
Exponente = 8
msgInfo(String(Base) + " elevado a " +
        String(Exponente), Base.pow(Exponente))          ;
muestra 256
endmethod
```

Vea también pow10
ln
log

pow10

Principiante

Método Eleva 10 a una potencia especificada.

Tipo Number

Sintaxis **pow10** ( ) Number

Descripción Devuelve el valor de 10 elevado a una potencia especificada.

Ejemplo En este ejemplo, el método **pushButton** del botón *ElevarDiez* calcula 10*exponente* y muestra el resultado:

```
; ElevarDiez::pushButton
method pushButton(var eventInfo Event)
var
  Exponente,
  Resultado  Number
endVar
Exponente = 9
Resultado = Exponente.pow10()
msgInfo("Diez elevado a " + String(Exponente) + " es",
        format("EC", Resultado))                          ;
muestra 1.000.000.000
endmethod
```

Vea también pow
ln
log

pv

Principiante

Método Devuelve el valor actual de una serie de pagos iguales.

Tipo Number

Sintaxis **pv** (const ***tipoInterés*** Number, const ***periodos*** Number **)** Number

Descripción Calcula el valor actual de pagos iguales y regulares sobre un préstamo (o retirada de fondos de una inversión) a un interés especificado en *tipoInterés* para un número de periodos especificados en *periodos*. Los pagos reducen el principal, pero el saldo restante continúa generando un interés compuesto.

La fórmula empleada es

PV = *pago* * (1- (1+*tipoInterés*)-*n* / *tipoInterés*)

donde *n* es el número de periodos.

El tipo de interés empleado en **pv** se expresa como un número decimal (por ejemplo, 0,12), no como un porcentaje (12%). Asegúrese de que el periodo del interés corresponde con el de pago; es decir, si los pagos son mensuales, el tipo de interés debe ser mensual también. Es posible utilizar **pv** para calcular el máximo de hipoteca que se puede afrontar (utilice **pmt** a la inversa para hallar el pago mensual necesario para amortizar una cantidad dada). También puede emplearse **pv** para calcular la cantidad que se necesitará para adquirir una anualidad que generará pagos regulares e iguales al inversor en el tiempo. Por esta razón, **pv** se denomina, en ocasiones, el valor actual de una anualidad.

Ejemplo Supóngase que una persona puede permitirse pagar 1.200 pesetas al mes y obtener una hipoteca a 30 años a un interés anual fijo del 9% (0,75% mensual). El método **pushButton** de *valorPresente* calcula y muestra el importe del préstamo al que puede aspirar:

```
; valorPresente::pushButton
method pushButton(var eventInfo Event)
var
  CantidadAPagar,
  Interés,
  Periodo,
  Hipoteca    Number
endVar

CantidadAPagar = 1200
Interés        = 0,09 / 12                                  ;
interés mensual del 9% al año
Periodo        = 360                                        ; 30
años (expresados en meses)
Hipoteca       = CantidadAPagar.pv(Interés, Periodo)
msgInfo("Hipoteca máxima", "Si se puede pagar " +
String(CantidadAPagar) +
        " pts. al mes durante " + String(Periodo /12) + " años
al " +
        String(Interés * 12 * 100) + "% se obtienen " +
```

```
        string(Hipoteca) + " pts.")                 ; muestra
149.138 pts.
endmethod
```

Supóngase que una persona, cuando se jubile, desea disponer de 250.000 pesetas al mes durante 30 años de una cuenta de jubilación que rinde el 7,5% de interés anual. Este método **pushButton** del botón *Anualidad* calcula cuánto debería depositarse en la cuenta:

```
; Anualidad::pushButton
method pushButton(var eventInfo Event)
var
  CantidadMensual,
  Periodo,
  Interés,
  Inversión        Number
endVar

CantidadMensual = 250.000,00           ; cantidad mensual que
se desea pagar anualmente
Periodo = 360                       ; 30 años son 360 meses
Interés = 0,075/12                  ; 7.5% al año, pasado a
porcentaje mensual
Inversión = CantidadMensual.pv(Interés, Periodo)    ; lo que se
necesita para
                                    ; empezar
msgInfo("Anualidad requerida", "Para devolver una anualidad de
" +
        String(CantidadAPagar) + " pts. al mes al "  +
        format("W4,2", Interés * 12 * 100) + "% durante"+
        String(SmallInt(Periodo / 12)) +
        " años, la cantidad original debe ser " +
        String(Inversión))
    ; muestra 357.544,07
endmethod
```

Vea también fv
pmt

rand

Principiante

Procedimiento Genera un valor aleatorio entre 0 y 1.

Tipo Number

Sintaxis **rand** ( ) Number

Descripción Genera un valor aleatorio que se halla entre 0 y 1.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *obtenAzar* calcula y muestra un número aleatorio *x* entre 1 (*NumMínimo*) y 10 (*NumMáximo*).

```
; obtenAzar::pushButton
method pushButton(var eventInfo Event)
var
  x,
  NumMínimo,
  NumMáximo  SmallInt
endVar
NumMínimo = 1
NumMáximo = 10
; obtenemos un entero al azar entre NumMínimo y NumMáximo
x = SmallInt(rand() * (NumMáximo - NumMínimo + 1) + NumMínimo)
msgInfo("Un número entre " + String(NumMínimo) + " y " +
         String(NumMáximo), x)
endmethod
```

Vea también truncate

round

Principiante

Método Redondea un número con un número especificado de cifras decimales.

Tipo Number

Sintaxis **round** (const ***cifras*** SmallInt) Number

Descripción Devuelve un número redondeado con el número de cifras decimales especificado en *cifras*.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *redondeo* redondea un número a cuatro cifras decimales y muestra el resultado y, a continuación, redondea y muestra un número al millar más próximo.

```
; redondeo::pushButton
method pushButton(var eventInfo Event)
var
  Redondear Number
endVar
Redondear = 1.2356838
msgInfo(format("W9,7",Redondear) +  "redondeado a cuatro
decimales",
           format("W6,4", Redondear.round(4)))
    ; muestra 1,2357
Redondear = 678394
msgInfo(String(Redondear) + " redondeado a -3 cifras
decimales",
           Redondear.round(-3))
    ; muestra 678.000
endmethod
```

Vea también truncate
ceil
floor

sin

Principiante

Método Devuelve el seno de un ángulo.

Tipo Number

Sintaxis **sin** ( ) Number

Descripción Devuelve un valor numérico entre -1 y 1 como seno de un valor que representa el tamaño del ángulo en radianes.

Ejemplo El método **pushButton** del botón *seno* halla el seno de un ángulo de 45 grados:

```
; seno::pushButton
method pushButton(var eventInfo Event)
var
  CuarentaYCincoGrados Number
endVar
CuarentaYCincoGrados = PI / 4,0
msgInfo("El seno de 45 grados es",
         format("W14,12", CuarentaYCincoGrados.sin()))
; muestra 0,707106781187
endMethod
```

Vea también asin
cos
tan

sinh

Principiante

Método Devuelve el seno hiperbólico de un ángulo.

Tipo Number

Sintaxis **sinh** ( ) Number

Descripción Devuelve el seno hiperbólico de un valor que representa el tamaño del ángulo en radianes. La fórmula empleada es

sinh (ángulo) = (exp(ángulo) - exp(-ángulo)) / 2

Ejemplo En este ejemplo, el método **pushButton** del botón *senoHiperbólico* halla el seno hiperbólico de un ángulo de 45 grados:

```
; senoHiperbólico
method pushButton(var eventInfo Event)

var
  CuarentaYCincoGrados Number
endVar
CuarentaYCincoGrados = PI / 4,0
msgInfo("El seno hiperbólico de 45 grados",
        format("w14.12", CuarentaYCincoGrados.sinh()))
; muestra 0,868670961486
endmethod
```

Vea también sin
cos
tan

sqrt

Principiante

Método Devuelve la raíz cuadrada de un número.

Tipo Number

Sintaxis **sqrt** ( ) Number

Descripción Devuelve la raíz cuadrada de un valor o expresión numérica positivos.

Ejemplo En este ejemplo, el método **pushButton** del botón *raízCuadrada* asigna el valor de *CampoUno* (un objeto de campo no asociado) a *x*, comprueba si *x* es negativo, y, si no lo es, calcula y muestra la raíz cuadrada de *x*:

```
; raízCuadrada::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = CampoUno
if x < 0 then
  msgStop("Lo siento",
         "No puedo resolver la raíz cuadrada de un número
negativo.")
else
  msgInfo("La raíz cuadrada de " + String(x) +  "es",
           format("w14,6", sqrt(x)))               ; muestra el
resultado
endif
endmethod
```

Vea también exp

tan

Principiante

Método Devuelve la tangente de un ángulo.

Tipo Number

Sintaxis **tan** ( ) Number

Descripción Devuelve la tangente de un valor o expresión numérica que representa el tamaño del ángulo en radianes. **tan** diverge (tiende al infinito) en pi/2, pi/2 y cada ± pi radianes a partir de esos valores.

Ejemplo En este ejemplo, el método **pushButton** del botón *tangente* calcula la tangente de un ángulo de 45 grados y muestra el resultado.

```
; tangente::pushButton
method pushButton(var eventInfo Event)
var
  CuarentaYCincoGrados Number
endVar
CuarentaYCincoGrados = PI / 4,0
msgInfo("La tangente de 45 grados es",
CuarentaYCincoGrados.tan()) ; muestra 1.00
endmethod
```

Vea también atan
atan2
cos
sin

tanh

Principiante

Método Devuelve la tangente hiperbólica de un ángulo.

Tipo Number

Sintaxis **tanh** ( ) Number

Descripción Devuelve la tangente hiperbólica de un valor o expresión numérica que representa el tamaño del ángulo en radianes. La fórmula es

tanh(ángulo) = sinh(ángulo) / cosh(ángulo)

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *tangenteHiperbólica* calcula la tangente hiperbólica de un ángulo de 60 grados y muestra el resultado:

```
; tangenteHiperbólica::pushButton
method pushButton(var eventInfo Event)
var
  SesentaGrados Number
endVar
SesentaGrados = PI / 3,0
msgInfo("La tangente hiperbólica de 60 grados es",
        format("W14,12", SesentaGrados.tanh()))
; muestra 0,780714435359
endmethod
```

Vea también atan
cos
sin

truncate

Principiante

Método Trunca un número a un número especificado de cifras decimales.

Tipo Number

Sintaxis **truncate** (const ***cifras*** SmallInt) Number

Descripción Devuelve un número truncado hacia cero con el número de cifras decimales especificado en *cifras*. No redondea el valor.

Ejemplo En este ejemplo, el método **pushButton** del botón *truncarUnValor* asigna el valor de *CampoUno* (un objeto de campo no asociado) a *x*, trunca *x* a tres cifras y muestra el resultado truncado:

```
; truncarUnValor::pushButton
method pushButton(var eventInfo Event)
var
  x Number
endVar
x = CampoUno
msgInfo("x truncado a 3 cifras",
         format("W14,6", x.truncate(3)))      ; muestra una
versión truncada de x
endmethod
```

Vea también ceil
floor
round

canReadFromClipboard

Método Informa de si un objeto OLE puede pegarse desde el Portapapeles en una variable OLE.

Tipo OLE

Sintaxis **canReadFromClipboard** ( ) Logical

Descripción Devuelve True si un objeto OLE puede leerse (pegarse) desde el Portapapeles en una variable OLE; en caso contrario, devuelve False. Este método es útil en una rutina que informe al usuario de si la operación es posible; por ejemplo, se podría atenuar e inactivar una opción de menú cuando este método devuelva False.

Ejemplo En este ejemplo, una ficha tiene un objeto OLE llamado *ObjetoOLE* y un botón llamado *actualizarOLE*. El método **pushButton** de *ObjetoOLE* comprueba el contenido del Portapapeles para determinar si un documento OLE puede leerse en una variable OLE. Si el método **canReadFromClipboard** devuelve True, este ejemplo pega el contenido del Portapapeles en *ObjetoOLE*; en caso contrario, muestra un mensaje de error.

```
; actualizarOLE::pushButton
method pushButton(var eventInfo Event)
var
  VarOLE   OLE
endVar

; si se puede leer un objeto OLE del Portapapeles
if VarOLE.canReadFromClipboard() then

  ; leemos el objeto OLE del portapapeles y lo llevamos a ; la
variable VarOLE
  VarOLE.readFromClipboard()

  ; actualizamos en la ficha el objeto OLE que contieneel
portapales
  ObjetoOLE = VarOLE

else
  beep()
  msgStop("Error", "No puedo leer en el portapapeles.")
endif

endmethod
```

Vea también readFromClipboard

edit

Método Arranca el servidor OLE y permite que el usuario edite el objeto o tome alguna otra acción.

Tipo OLE

Sintaxis **edit** (const ***textoOLE*** String, const ***verbo*** SmallInt) Logical

Descripción Arranca la aplicación servidora de OLE y da el control al usuario. El argumento *textoOLE* es una cadena que Paradox pasa a la aplicación servidora. Muchas aplicaciones servidoras pueden mostrar *textoOLE* en la barra de título. **edit** pasa *verbo* a la aplicación servidora para especificar una acción debe realizarse.

verbo es un número entero que corresponde a una de las constantes de acción del servidor de OLE. El significado de *verbo* varía de una aplicación a otra, por lo que un *verbo* que sea adecuado para una aplicación puede no serlo para otra. Con el método **enumVerbs**, es posible averiguar qué verbos admite el servidor y, posteriormente, utilizar uno de ellos en la llamada a **edit**.

Si se desea arrancar un servidor de OLE sin emplear **enumVerbs** en primer lugar, utilice 0 (cero) en *verbo* este valor es el *verbo* primario y todos los servidores de OLE deberían utilizarlo.

Ejemplo Supóngase que la tabla *ArchivoDoc* contiene documentos creados con un procesador de textos en un campo OLE. La tabla tiene dos campos: Nombre del Archivo (A25) y Archivos (O). Este ejemplo busca un registro en la tabla y utiliza **edit** para ejecutar el procesador de textos, permitiendo así al usuario editar el documento en el campo OLE.

El ejemplo siguiente es el código anexado al método **pushButton** de *editarArchivo*.

```
; editarArchivo::pushButton
method pushButton(var eventInfo Event)
var
  Archivos    OLE
  tc         TCursor
  NombreArch String
endVar

NombreArch = "NotasPro.doc"

if tc.open("ArchiDoc.db") then
  ; buscamos "NotasPro.doc" en el campo Nombre del Archivo
  if tc.locate("Nombre del Archivo", NombreArch) then

    ; almacenamos en la variable OLE Archivos el
    ; contenido del campo OLE
    Archivos = tc.Archivo

    ; lanzamos el servido OLE (el procesador de texto) para
Archivo
    Archivos.edit("Administrador de documentos", 0)
```

```
else
  msgStop("Lo siento", "No encuentro " + NombreArch +".")
  endif
else
  msgStop("Lo siento", "No puedo abrir la tabla.")
endif

endmethod
```

Vea también enumVerbs

enumVerbs

Método Crea un DynArray que enumera las acciones permitidas por el servidor de OLE.

Tipo OLE

Sintaxis **enumVerbs** (var ***verbos*** DynArray[] SmallInt) Logical

Descripción Crea un DynArray que enumera los comandos de acción (llamados *verbos*) permitidos por el servidor de OLE.

Cuando se asocia una variable OLE con un objeto OLE (denominado también *documento* OLE), Paradox sabe en qué aplicación servidora se generó el objeto. Mediante métodos OLE como **enumVerbs** y **getServerName**, es posible realizar preguntas acerca del servidor. **enumVerbs** pide al servidor una lista de comandos acción permitidos y los carga en una matriz dinámica. Cada índice del DynArray corresponde al nombre que el servidor da a una acción específica; los valores del DynArray corresponden a la constante de acción utilizada por el servidor. Puesto que el significado de *verbos* varía de una aplicación a otra, es necesario saber con precisión qué verbo pasar al servidor para que éste haga lo que se desea.

Por ejemplo, Paintbrush de Windows es un servidor de OLE. Paintbrush sólo tiene un comando de acción, llamado Edit , con un valor de 0. El código siguiente lee del Portapapeles una imagen generada con Paintbrush, genera una matriz dinámica con enumVerbs y muestra el contenido del DynArray en un cuadro de diálogo.

```
var
  VarOLE OLE
  dy     DynArray [] SmallInt
endVar

VarOLE.readFromClipboard() ; leemos desde el portapapeles a
VarOLE
VarOLE.enumverbs(dy)       ; generamos un DynArray con verbos
dy.view                    : mostramos el contenido del
DynArray en un cuadro de diálogo
```

El código anterior supone que el Portapapeles contiene un objeto OLE (una imagen gráfica, en este caso) que se generó en Paintbrush. La matriz dinámica contiene un elemento cuyo índice es Editar y cuyo valor es 0. Algunos servidores de OLE utilizan más de un verbo, por lo que se generaría una lista más larga. Otros servidores de OLE utilizan Editar , pero anteponen un ampersand al nombre, como &Editar . El ampersand es útil cuando se desea mostrar nombres de acción en un menú. Paradox reconoce el ampersand como un carácter especial, muestra &Edit como Edit y designa E como tecla de acceso.

Consulte los métodos del tipo Menu en este capítulo para más información sobre menús y caracteres especiales.

Ejemplo En este ejemplo, la tabla *Sonidos* tiene un campo alfanumérico llamado NombreDelSonido y un campo OLE llamado DatosSonido. Los datos del campo OLE se copiaron de la Grabadora de sonidos de Windows. El ejemplo siguiente utiliza enumVerbs para crear un menú emergente que enumera

los verbos (acciones) de la Grabadora de sonidos. Puesto que este programa permite dos acciones (Editar y Sonar), este ejemplo permite que el usuario elija editar o reproducir el sonido contenido en el campo OLE.

```
; editarSonidos::pushButton
method pushButton(var eventInfo Event)
var
  VarOLE  OLE
  d       PopUpMenu
  verbos  DynArray[] SmallInt
  tc     TCursor
  Selección, Nombre String
endvar
NombreDelSonido = "acorde.wav"
NombreTabla = "Sonidos.db"

if tc.open(NombreTabla) then
  if tc.locate(1, NombreDelSonido) then ; buscamos en el primer
campo
    ; de acorde.wav
     VarOLE = tc.DatosSonido               ; asignamos a la
variable OLE el valor
    ;  del campo OLE
     VarOLE.enumVerbs(verbos)            ; cargamos un dynArray
con acciones
    ; de la Grabadora de Sonidos
     forEach Nombre in verbos           ; creamos un menú
desplegable con
    ; la lista de los verbos
          d.addText(Nombre)               ; los verbos de la
Grabadora de
    ; Sonidos son &Editar y &Sonar
     endForEach
     Selección = d.show()                ; mostramos "Editar" y
"Sonar" en
    ; el menú desplegable

    ; si el usuario hace una selección, pasamos el "verbo"
seleccionado al
    ; método edit, verbos[Selección] se evalúa como 0 o 1.
    ; "PdoxWin" aparece en la barra de título de la grabadora
de sonidos cuando
    ;  se selecciona Sonar
    if not Selección.isBlank() then
      VarOLE.edit("PdoxWin", verbos[Selección])
    endif

  else
    msgStop("Lo siento", "No encuentro " + NombreDelSonido +
".")
  endif
else
  msgStop("Lo siento", "No puedo abrir la tabla " + NombreTabla
+ ".")
endif

endmethod
```

Vea también readFromClipboard

getServerName

Método Informa del nombre del servidor de OLE de un objeto OLE.

Tipo OLE

Sintaxis **getServerName** ( ) String

Descripción Devuelve en forma de cadena el nombre del servidor de OLE de un objeto OLE. Este método es útil cuando se desea informar al usuario del nombre del servidor de OLE.

Ejemplo En este ejemplo, supóngase que la tabla *Media* tiene un campo alfanumérico llamado NombreMedia, un campo alfanumérico llamado ServerName y un campo OLE llamado DatosMedia. El código siguiente examina los registros de *Media*, rellenando el campo ServerName con el nombre del servidor de OLE que generó los datos del campo DatosMedia.

```
; obtenNombreDelServidor::pushButton
method pushButton(var eventInfo Event)
var
  VarOLE  OLE
  tc      TCursor
endvar

if tc.open("Media") then
  tc.edit()
  scan tc for not isBlank(tc.DatosSonido) :
     VarOLE = tc.DatosSonido
     tc.ServerName = VarOLE.getServerName()
  endScan
  tc.close()
else
  msgStop("Error", "No puedo abrir la tabla Media.")
endif

endmethod
```

Vea también edit
enumVerbs

readFromClipboard

Método Pega un objeto OLE desde el Portapapeles en una variable OLE.

Tipo OLE

Sintaxis **readFromClipboard** ( ) Logical

Descripción Devuelve True si se ha logrado leer (pegar) un objeto OLE desde el Portapapeles en una variable OLE; en caso contrario, devuelve False.

Después de que un objeto OLE se haya leído desde el Portapapeles, los cambios realizados en el objeto OLE mientras se está en Paradox no afectan al archivo subyacente.

Ejemplo Consulte el ejemplo de canReadFromClipboard.

Vea también canReadFromClipboard

writeToClipboard

Método Copia una variable OLE al Portapapeles.

Tipo OLE

Sintaxis **writeToClipboard** ( ) Logical

Descripción Copia el objeto OLE original al Portapapeles como si el servidor hubiera realizado la copia (porque Paradox no es servidor de OLE).

Este método devuelve True si se ha logrado escribir (pegar) un objeto OLE en el Portapapeles; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente lee un campo OLE en una tabla de Paradox y asigna su valor a una variable OLE. Entonces, escribe la variable en el Portapapeles, donde Paradox u otra aplicación pueden utilizarla. El código ejemplo supone que EMPLEADO.DB tiene un campo alfanumérico llamado Ultimo Nombre y un campo OLE llamado Imagen.

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  TC TCursor
  ImagenOLE OLE
endVar

TC.open("Empleado.db")        ; EMPLEADO.DB tiene imágenes OLE

if TC.locate("Ultimo Nombre", "Dibujito") then

  ImagenOLE = TC.Imagen        ; Imagen es un campo OLE
  ImagenOLE.writeToClipboard() ; escribimos el contenido del
campo OLE en
                               ; la variable

else
  msgStop("Error", "No encuentro Dibujito...")
endif
endmethod
```

Vea también readFromClipboard

distance

Método Devuelve la distancia entre dos puntos.

Tipo Point

Sintaxis **distance** (const ***punto*** Point) Number

Descripción Devuelve el número de twips entre un punto y *punto*.

Ejemplo Supóngase que una ficha contiene dos cuadros: *CajaRoja* y *CajaMarrón*. El método **pushButton** de un botón llamado *HallarDistancia* halla la distancia entre los ángulos superiores izquierdos de los cuadros:

```
; cajaMarrón::pushButton
method pushButton(var eventInfo Event)
var
  p1, p2 Point
endVar
p1 = CajaRoja.Position
p2 = CajaMarrón.Position
msgInfo("Distancia entre las cajas", p1.distance(p2))
; muestra la distancia entre la esquina superior izquierda de
; CajaRoja y la esquina superior izquierda de CajaMarrón
endmethod
```

Vea también isAbove
isBelow
isLeft
isRight
x
y

isAbove

Método Informa de si un punto está situado por encima de otro punto.

Tipo Point

Sintaxis **isAbove** (const ***punto*** Point) Logical

Descripción Devuelve True si la coordenada y de un punto es menor que la de *punto*; en caso contrario, devuelve False.

Ejemplo En este ejemplo, el método **pushButton** de *superponerCajas* acerca *CajaUno* a *CajaDos*, hasta que los dos cuadros convergen. Supóngase que *CajaUno* comienza a la izquierda y por encima de *CajaDos*. Cada vez que se hace clic sobre el botón, *CajaUno* bajará hasta que esté en el mismo plano vertical y entonces hacia la derecha hasta que esté cubierto por *CajaDos*.

```
; superponerCajas::pushButton
method pushButton(var eventInfo Event)
var
  p1, p2 Point
endVar
p1 = CajaUno.position             ; obtenemos la posición de
CajaUno
p2 = CajaDos.position             ; obtenemos la posición de
CajaDos
if p1.isAbove(p2) then            ; comparamos ambos puntos
  ; si p1 está más alto que p2, movemos CajaUno hacia abajo
  CajaUno.position = Point(p1.x(), p1.y() + 100)
else
  if p1.isLeft(p2) then
    ; si p1 está a la izquierda de p2, movemos CajaUno a la
derecha
    CajaUno.position = Point(p1.x() + 100, p1.y())
  endif
endif
endmethod
```

Vea también isBelow
isLeft
isRight

isBelow

Método Informa de si un punto está situado por debajo de otro punto.

Tipo Point

Sintaxis **isBelow** (const ***punto*** Point) Logical

Descripción Devuelve True si la coordenada y de un punto es mayor que la de *punto*; en caso contrario, devuelve False.

Ejemplo En este ejemplo, el método **pushButton** de *superponerCajas* acerca *CajaDos* a *CajaUno*, hasta que los dos cuadros convergen. Supóngase que *CajaDos* comienza a la derecha y por debajo de *CajaUno*. Cada vez que se hace clic sobre el botón, *CajaDos* subirá hasta que esté en el mismo plano vertical y entonces hacia la izquierda hasta que esté cubierto por *CajaUno*.

```
; superponerCajas::pushButton
method pushButton(var eventInfo Event)
var
  p1, p2 Point
endVar
p1 = CajaUno.position             ; obtenemos la posición de
CajaUno
p2 = CajaDos.position             ; obtenemos la posición de
CajaDos
if p2.isBelow(p1) then            ; comparamos ambos puntos
  ; si p2 está más bajo que p1, movemos CajaUno hacia arriba
  CajaUno.position = Point(p1.x(), p1.y() + 100)
else
  if p2.isRight(p1) then
    ; si p2 está a la derecha de p1, movemos CajaDos a la
izquierda
    CajaDos.position = Point(p1.x() - 100, p1.y())
  endif
endif
endmethod
```

Vea también isAbove
isLeft
isRight

isLeft

Método	Informa de si un punto está situado a la izquierda de otro punto.
Tipo	Point
Sintaxis	**isLeft** (const ***punto*** Point) Logical
Descripción	Devuelve True si la coordenada x de un punto es menor que la de *punto*; en caso contrario, devuelve False.
Ejemplo	Consulte el ejemplo de isAbove.

Vea también isAbove
isBelow
isRight

isRight

Método Informa de si un punto está situado a la derecha de otro punto.

Tipo Point

Sintaxis **isRight** (const ***punto*** Point) Logical

Descripción Devuelve True si la coordenada x de un punto es mayor que la de *punto*; en caso contrario, devuelve False.

Ejemplo Consulte el ejemplo de isBelow.

Vea también isAbove
isBelow
isLeft

point

Procedimiento Convierte una expresión a Point.

Tipo Point

Sintaxis **1. point** ([const ***x*** LongInt, const ***y*** LongInt]) Point

2. point (const ***nuevoPunto*** Point) Point

Descripción Convierte una expresión a Point.

Ejemplo En este ejemplo, se desea variar la posición de un cuadro llamado *RangoCaja*. Los valores de un objeto de campo no asociado llamado *campoRango* se hallan entre 1 y 10. La posición de *RangoCaja* se determina mediante el valor de *campoRango*. El código siguiente se anexa al método **changeValue** de *campoRango*:

```
; campoRango::changeValue
method changeValue(var eventInfo ValueEvent)
Const
  BaseX = LongInt(3000)
  BaseY = LongInt(1000)
endConst
Var
  RangoX LongInt
endVar
try
  ; esta estructura if fallará si el contenido del campo no se
puede
  ; comparar con los enteros 0 y 10 - por ejemplo, si
  ; el usuario introduce una cadena
  if eventInfo.newValue() = 0 AND eventInfo.newValue() = 10
then
    RangoX = (eventInfo.newValue() * 400) + BaseX
    RangoCaja.Position = point(RangoX, BaseY)
  else
    fail() ; si el valor es un número, pero está fuera de
rango,
           ; llamamos al bloque fail
  endif
onFail
  disableDefault
  eventInfo.setErrorCode(CanNotDepart)
  msgStop("Alto", "El rango debe estar entre 0 y 10.")
endTry

endmethod
```

Vea también setX
setY

setX

Método Especifica la coordenada x de un punto.

Tipo Point

Sintaxis **setX** (const ***nuevoValorX*** LongInt)

Descripción Define la coordenada x de un punto como *nuevoValorX*. Si *nuevoValorX* no es un LongInt, se convierte a LongInt, por lo que puede perderse precisión.

Ejemplo En este ejemplo, una ficha contiene una elipse llamada *CírculoUno* y un botón llamado *moverALaDerecha*. El método **pushButton** de *moverALaDerecha* utiliza **setX** para cambiar la coordenada horizontal de un punto y define la posición de *CírculoUno* al punto cambiado:

```
; moverseALaDerecha::pushButton
method pushButton(var eventInfo Event)
var
  p1 Point
endVar
p1 = CírculoUno.position  ; obtenemos la posición del círculo
p1.setX(p1.x() + 100)     ; añadimos 100 twips a la coordenada
X
CírculoUno.Position = p1  ; asignamos la nueva posición
message(p1)               ; mostramos las coordenadas
endmethod
```

Vea también setY

setXY

Método Especifica las coordenadas x e y de un punto.

Tipo Point

Sintaxis **setXY** (const ***nuevoValorX*** LongInt, const ***nuevoValorY*** LongInt)

Descripción Define las coordenadas x e y de un punto como *nuevoValorX* y *nuevoValorY*. Este método combina las funciones de **setX** y **setY**. Si *nuevoValorX* y *nuevoValorY* no son LongInt, se convierten a LongInt, por lo que puede perderse precisión.

Ejemplo En este ejemplo, una ficha contiene una elipse llamada *CírculoUno* y un botón llamado *moverseEnDiagonal*. El método **pushButton** de *moverseEnDiagonal* utiliza **setXY** para cambiar las coordenadas horizontal y vertical de un punto y define la posición de *CírculoUno* al punto cambiado:

```
; moverseEnDiagonal::pushButton
method pushButton(var eventInfo Event)
var
  p1 Point
endVar
p1 = CírculoUno.position              ; obtenemos la posición
del círculo
p1.setXY(p1.x() + 100, p1.y() + 100)  ; añadimos 100 twips a
cada coordenada
CírculoUno.Position = p1              ; asignamos la nueva
posición
message(p1)                           ; mostramos las
coordenadas
endmethod
```

Vea también setX
setY

setY

Método Especifica la coordenada y de un punto.

Tipo Point

Sintaxis **setY** (const ***nuevoValorY*** LongInt)

Descripción Define la coordenada y de un punto como *nuevoValorY*. Si *nuevoValorY* no es un LongInt, se convierte a LongInt, por lo que puede perderse precisión.

Ejemplo En este ejemplo, una ficha contiene una elipse llamada *CírculoUno* y un botón llamado *moverseAbajo*. El método **pushButton** de *moverseAbajo* utiliza **setY** para cambiar la coordenada vertical de un punto y define la posición de *CírculoUno* al punto cambiado:

```
; moverseAbajo::pushButton
method pushButton(var eventInfo Event)
var
  p1 Point
endVar
p1 = CírculoUno.position  ; obtenemos la posición del círculo
p1.setX(p1.x() + 100)     ; añadimos 100 twips a la coordenada
Y
CírculoUno.Position = p1  ; asignamos la nueva posición
message(p1)               ; mostramos las coordenadas
endmethod
```

Vea también setX
setXY

x

Método	Devuelve la coordenada x de un punto.
Tipo	Point
Sintaxis	**x** ( ) LongInt
Descripción	Devuelve la coordenada **x** de un punto.
Ejemplo	Consulte el ejemplo de setX.
Vea también	setX setY y

y

Método	Devuelve la coordenada y de un punto.
Tipo	Point
Sintaxis	**y** ( ) LongInt
Descripción	Devuelve la coordenada **y** de un punto.
Ejemplo	Consulte el ejemplo de setY.
Vea también	setY setX x

view

Método Muestra el valor de un registro en un cuadro de diálogo.

Tipo Record

Sintaxis **view** ([const ***título*** String])

Descripción Muestra el valor de un registro en un cuadro de diálogo modal. La ejecución de ObjectPAL se suspende hasta que el usuario cierra este cuadro de diálogo. Es posible especificar el título del cuadro de diálogo en *título*, u omitirlo para mostrar en su lugar el tipo de datos de la variable. A diferencia de muchos tipos de datos, los valores de un Record no pueden cambiarse cuando se muestran en un cuadro de diálogo de **view**. Para más información sobre **view** y otros tipos, consulte AnyType en este capítulo.

Ejemplo El ejemplo siguiente utiliza un tipo llamado Registro. El método **pushButton** de *ObtenerYVerRegistro* declara una variable llamada *Registro* del tipo Registro. Este método abre un TCursor con la tabla Clientes, rellena *Registro* con los valores de los campos *Nº de Cliente* y *Nombre* del primer registro y utiliza **view** para mostrar el registro en un cuadro de diálogo. Entonces, esta operación se repite para el segundo registro de *Clientes*.

El código siguiente se anexa a la ventana Type de *ObtenerYVerRegistro*. Este código crea un tipo definido por el usuario llamado Registro.

```
; ObtenerYVerRegistro::Type
Type
  Registro = record         ; definimos una estructura Record
               ID     String
               Nombre String
               endRecord
endType
```

Este código se anexa al método **pushButton** de un botón llamado *ObtenerYVerRegistro*:

```
; ObtenerYVerRegistro::pushButton
method pushButton(var eventInfo Event)
var
  RegUno,RegDos Registro
  tc            TCursor
endVar

if tc.open("Clientes.db") then
  RegUno.ID = tc."N° de Cliente"  ; escribimos algunos valores
en el registro
  RegUno.Nombre = tc."Nombre"
  RegUno.view("Primer Registro")  ; mostramos el registro en un
cuadro de diálogo

  tc.nextRecord()                 ; nos movemos la siguiente
registro

  RegDos.ID = tc."N° de Cliente"  ; obtenemos nuevos valores
  RegDos.Nombre = tc."Nombre"
```

```
  RegDos.view("Segundo registro") ; mostramos el segundo
registro

  msgInfo("¿Es RegUno igual a RegDos?", RegUno = RegDos)  ;
Muestra False

  RegUno = RegDos                  ; ahora ambos registros
tienen los mismos
       ; valores
  msgInfo("¿Es RegUno igual a RegDos?", RegUno = RegDos) ;
muestra True

else
  msgStop("Alto", "No puedo abrir la tabla Clientes.")
endIf
endmethod
```

Vea también Array
DynArray

bitAND

Método Realiza una operación binaria AND sobre dos valores.

Tipo SmallInt

Sintaxis **bitAND** (const ***valor*** SmallInt) SmallInt

Descripción Devuelve el resultado de una operación binaria AND sobre *valor*. **bitAND** opera sobre las representaciones binarias de dos enteros, comparándolas bit a bit. La tabla real de **bitAND** es:

a	**b**	**a bitAND b**
0	0	0
1	0	0
0	1	0
1	1	1

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *andDeDosNúmeros* toma dos números enteros y realiza un cálculo binario AND sobre ellos. El resultado del cálculo se muestra en un cuadro de diálogo.

```
; andDeDosNúmeros::pushButton
method pushButton(var eventInfo Event)
var
  a, b SmallInt
endVar
a = 30233   ; en binario: 01110110 00011001
b = 1233    ; en binario: 00000100 11010001
a.bitAND(b) ; en binario: 00000100 00010001
msgInfo("El resultado de 30233 bitAND 1233 es:", a.bitAND(b))
; muestra 1041
endmethod
```

Vea también bitOR
bitXOR

bitIsSet

Método Informa de si un bit es 1 ó 0.

Tipo SmallInt

Sintaxis **bitIsSet** (const ***valor*** SmallInt) Logical

Descripción Examina la representación binaria de un entero, informando de si el bit **valor** es 0 ó 1. Devuelve True si el bit especificado es 1, y False si es 0.

valor es un número especificado mediante 2n, siendo *n* un entero entre 0 y 14. El exponente *n* corresponde a una unidad menos que la posición del bit que se comprueba, contando desde la derecha. Por ejemplo para especificar el tercer bit desde la derecha, utilice 4 (2(3-1), que es 22).

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *esUnConjuntoDeBits* examina los valores de dos objetos de campo no asociado: *quéBit* y *QuéNúmero*. *quéBit* contiene la posición del bit (contando desde la derecha) que se comprueba. *QuéNúmero* contiene el entero que se comprueba. El método **pushButton** utiliza *quéBit* para calcular el valor de la posición y asigna el resultado a *NúmeroBit*. El método comprueba en *Número* si el bit *NúmeroBit* está definido y muestra el resultado Logical en un cuadro de diálogo **msgInfo**.

```
; esUnConjuntoDeBits::pushButton
method pushButton(var eventInfo Event)
var
  NúmeroBit,
  Número   SmallInt
endVar
; obten número de la posición del bit del campo
; quéBit y amoldarlo para multiplicarlo por 2
NúmeroBit = SmallInt(pow(2, quéBit - 1))
; obtenemos el número para comprobar en el campo QuéNúmero
Número = QuéNúmero
; ¿está el bit NúmeroBit en Número?
msgInfo("¿Está el conjunto de Bits?", Num.bitIsSet(NúmeroBit))
endmethod
```

El ejemplo siguiente ilustra cómo puede utilizarse **bitIsSet** para mostrar un número entero como binario. El método **pushButton** de *mostrarBinario* construye una cadena de ceros y unos comprobando cada bit de un entero grande de cuatro bytes. Por razones de legibilidad, se añade un blanco en la cadena después de ocho dígitos.

```
; mostrarBinario::pushButton
method pushButton(var eventInfo Event)
var
  CadenaBinaria String   ; para construir la cadena binaria
  Número        SmallInt ; número a comprobar
  i             SmallInt ; para el índice del bucle
endVar
if NOT QuéNúmero.isBlank() then
  Número = QuéNúmero             ; obtenemos la comprobación
del
    ; número con QuéNúmero
```

```
  CadenaBinaria = ""               ; iniciamos la cadena
  for i from 0 to 14
    if Número.bitIsSet(SmallInt(pow(2, i))) then
      CadenaBinaria = "1" + CadenaBinaria  ; añade 1 al
principio de la cadena
     else
      CadenaBinaria = "0" + CadenaBinaria  ; añade 0 al
principio de la cadena
    endif
    if i = 7 then
      CadenaBinaria = " " + CadenaBinaria  ; añade un espacio
cada 8 dígitos
    endif
  endfor
  if Número  0 then
    CadenaBinaria = "1" + CadenaBinaria      ; añade el bit de
signo
  else
    CadenaBinaria = "0" + CadenaBinaria
  endif
  ; muestra el número
  message("El número binario equivalente es ", CadenaBinaria)
endif
endmethod
```

Vea también bitAND
bitOR
bitXOR

bitOR

Método Realiza una operación binaria OR sobre dos valores.

Tipo SmallInt

Sintaxis **bitOR** (const ***valor*** SmallInt) SmallInt

Descripción Devuelve el resultado de una operación binaria OR sobre *valor*. **bitOR** opera sobre las representaciones binarias de dos números enteros, comparándolas bit a bit. La tabla real de **bitOR** es:

a	**b**	**a bitOR b**
0	0	0
1	0	1
0	1	1
1	1	1

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *orDeDosNúmeros* toma dos enteros y realiza un cálculo binario OR sobre ellos. El resultado del cálculo se muestra en un cuadro de diálogo.

```
; orDeDosNúmeros::pushButton
method pushButton(var eventInfo Event)
var
  a, b SmallInt
endVar
a = 30233  ; en binario: 01110110 00011001
b = 1233  ; en binario: 00000100 11010001
a.bitOR(b) ; en binario: 01110110 11011001
msgInfo("30233 OR 1233", a.bitOR(b)) ; muestra 30425
endmethod
```

Vea también bitAND
bitXOR

bitXOR

Método Realiza una operación binaria XOR sobre dos valores.

Tipo SmallInt

Sintaxis **bitXOR** (const ***valor*** SmallInt) SmallInt

Descripción Realiza una operación binaria XOR (OR exclusivo) sobre *valor*. **bitXOR** opera sobre las representaciones binarias de dos enteros, comparándolas bit a bit. La tabla real de **bitXOR** es:

a	**b**	**a bitXOR(b)**
0	0	0
1	0	1
0	1	1
1	1	0

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *xorDeDosNúmeros* toma dos enteros y realiza un cálculo binario XOR sobre ellos. El resultado del cálculo se muestra en un cuadro de diálogo.

```
; xorDeDosNúmeros::pushButton
method pushButton(var eventInfo Event)
var
  a, b SmallInt
endVar
a = 30233   ; en binario: 01110110 00011001
b = 1233    ; en binario: 00000100 11010001
a.bitXOR(b) ; en binario: 01110010 11001000
msgInfo("30233 XOR 1233", a.bitXOR(b)) ; muestra 29384
endmethod
```

Vea también bitAND
bitOR

int

Procedimiento Convierte un valor en entero.

Tipo SmallInt

Sintaxis **int** (const ***valor*** AnyType) SmallInt

Descripción Convierte la expresión numérica *valor* en un entero. Si *valor* pertenecía a un tipo más preciso (como Number), se pierde precisión.

Ejemplo El ejemplo siguiente asigna un número a *nn*, muestra el valor de *nn* en un cuadro de diálogo y, entonces, muestra *nn* como un entero. Este código se anexa al método **pushButton** del botón *parteEntera*.

```
; parteEntera::pushButton
method pushButton(var eventInfo Event)
var
  nn Number
endVar
nn = 123,12
view(nn)                              ; muestra 123,12
msgInfo("nn como Entero", int(nn))    ; muestra 123
endmethod
```

Vea también smallInt

smallInt

Principiante

Procedimiento Convierte un valor en entero pequeño.

Tipo SmallInt

Sintaxis **smallInt** (const ***valor*** AnyType) SmallInt

Descripción Convierte la expresión numérica *valor* a SmallInt. Si *valor* es de un tipo más preciso (como Number), se pierde precisión.

Ejemplo El ejemplo siguiente asigna un número a *x*, se convierte *x* a SmallInt y asigna el resultado a *s*. La precisión decimal de *x* se pierde al convertirlo en un SmallInt.

```
; amoldarAEntero::pushButton
method pushButton(var eventInfo Event)
var
  x Number
  s SmallInt
endVar
x = 12,34             ; damos un valor a x
x.view()              ; vemos x, el título del cuadro de
diálogo será "Number"
s = SmallInt(x)       ; amoldamos x al tipo LongInt y se lo
asignamos a s
s.view()              ; mostramos s, nótese que no hay
decimales
                      ; muestra 12
endmethod
```

Vea también int

advMatch

Principiante

Método Busca una cadena especificada en un texto.

Tipo String

Sintaxis **advMatch** (const ***patrón*** String [,var ***varCoincidencia*** String]*) Logical

Descripción Devuelve True si se encuentra *patrón* dentro de la cadena; en caso contrario, devuelve False. Este método diferencia las mayúsculas y minúsculas. Para especificar *patrón*, utilice una cadena y los símbolos optativos enumerados en la tabla.

varCoincidencia es una variable a la que se asignará la parte que coincida. **advMatch** asigna los patrones coincidentes a una o más variables *varCoincidencia* conforme se encuentran los patrones. Las partes de la cadena que coincidan con los elementos comodín se asignan a las variables de izquierda a derecha. Puesto que puede haber múltiples coincidencias, la primera subcadena coincidente se asigna a la primera variable, la segunda subcadena coincidente, a la segunda variable, y así sucesivamente. Si no se encuentra ninguna coincidencia, no se asignan valores a las variables.

Si suministra *patrón* desde dentro de un método, debe utilizar dos barras invertidas cuando desee indicar a **advMatch** que trate un carácter especial como literal; por ejemplo, **\\(** indica a **advMatch** que trate el paréntesis como literal. Dos barras invertidas son necesarias en esta situación debido a la ambigüedad entre la interpretación de una barra invertida por parte del compilador (utilizada en secuencias de escape, como \t para tabulador) y la comprensión de **advMatch** de una barra invertida. Cuando el compilador ve una cadena con una secuencia de escape incluida, como \tinicio , interpreta el \t como un tabulador. El carácter de barra invertida tiene un significado especial para el compilador, pero también tiene un significado especial para **advMatch**.

Por ejemplo, si desea buscar un signo de interrogación incluido en una cadena, debería ejecutar **advMatch**, como sigue:

```
c = "¿una cadena?"
advMatch(c, "\?")     ;esto no funciona
```

Podría pensar que está indicando a **advMatch** que busque el signo de interrogación literal. Sin embargo, el compilador ve la cadena primer y devuelve un error de sintaxis porque \? no es una secuencia de escape válida. Para impedir que el compilador interprete la barra invertida como comienzo de una secuencia de escape, incluya dos barras invertidas. Esto sí funcionará:

```
c = "¿una cadena?"
advMatch(c, "\\?")    ;esto sí funciona
```

Si suministra *patrón* desde un campo de una tabla o un TextStream, los símbolos especiales de **advMatch** se reconocen sin la barra invertida precedente, y una barra invertida y el símbolo más (\+) produce un carácter literal.

Símbolo	Corresponde con
\	Utilice la barra invertida para incluir uno de los caracteres especiales siguientes como un carácter normal (recuerde utilizar dos barras invertidas en las cadenas entrecomilladas).
[ ]	Coincide con el conjunto incluido entre corchetes. Por ejemplo, [aeiou0-9] corresponde con a, e, i, o, u y 0 a 9.
[^]	No corresponde con el conjunto incluido. Por ejemplo, [^aeiou0-9] corresponde con todo excepto a, e, i, o, u y 0 a 9.
()	Agrupamiento.
^^	Principio de línea (no lo confunda con [^], donde ^^ actúa como un NOT lógico).
$	Fin de cadena.
..	Corresponde con todo.
@	Corresponde con cualquier carácter individual.
*	Cero o más del carácter o expresión precedente.
++	Uno o más del carácter o expresión precedente.
?	Ninguno o uno del carácter o expresión precedente.
|	Operación OR.

Ejemplo Estas sentencias demuestran la funcionalidad de **advMatch**:

```
method pushButton(var eventInfo Event)
     var
       w, x, y, z     String
       l              Logical
     endVar

     l = advMatch("esto es", "s")
     l.view()
       ; devuelve True (diferente de match)

     l = advMatch("esto es", "^s")
     l.view()
       ; devuelve False, porque requiere que s esté en el
principio de la línea

     l = advMatch("esto es", "S")
     l.view()
       ; devuelve False, se distinguen mayúsculas de minúsculas

     l = advMatch("esto es", "[sS]")
     l.view()
       ; devuelve True, porque [sS] especifica cualquiera
dentro de este
       ; conjunto

     l = advMatch("esto es", "[a-z]")
     l.view()
       ; devuelve True, porque [a-z] especifica cualquiera
dentro de este
       ; conjunto desde a hasta z
```

```
     l = advMatch("esto es", "[a-c]")
     l.view()
       ; devuelve False, poret [a-c] especifica cualquiera
dentro de este
       ; conjunto desde a hasta c y "esto es" no contiene
ninguna a, ninguna b y
       ; ninguna c

     l = advMatch("esto es", "[a-cs]")
     l.view()
       ; devuelve True, porque [a-cc] especifica cualquiera
dentro de este
       ; conjunto desde a hasta c o hasta s y "esto es"
contiene s
       ; nótese que [a-c, s] especificaría cualquiera dentro
del conjunto desde
       ; a hasta c, una coma, un espacio o una s

     l = advMatch("esto es", "(@)s", x)
     l.view()
     x.view()
       ; devuelve True, x = "i" porque los operadores "()"
especifican un grupo,
       ; al contrario que match, advMatch sitúa sólo aquello
que se agrupa
       ; en las variables

     l = advMatch("esto es un test", "((t@@s)|(t@s))|(@s)", w,
x, y, z)
     l.view()   ; devuelve True, y
     w.view()   ; "esto", el resultado del primer conjunto de
paréntesis,
                ; es decir, para la expresión completa ((t@@s)|
(t@s))
                ; también, "esto" fue comprobado antes de
"test"
     x.view()   ; también "esto", para el resultado del segundo
conjunto de
                ; paréntesis, (t@@s)
     y.view()   ; el resultado de (t@s), vacío, porque  t@@s
                ; satisface la expresión ((t@@s)|(t@s))
     z.view()   ; también vacío, porque la expresión ((t@@s)|
(t@s)) satisface
                ; el patrón entero ((t@@s)|(t@s))|(@s)
    ; NOTA: Las variables Match se comprueban en grupos en
orden de ocurrencia,
    ;       no en orden de precedencia: El primer grupo--
comenzando por
    ;       la izquierda--se asigna a la primera variable.

     l = advMatch("esto es así", "(..)es(..)", x, y)
     l.view()
     x.view()
     y.view()
       ; devuelve True, x = "esto", y = " así"

     l = advMatch("esto es así", "[a-c]|[f-l]s" )
```

```
     l.view()
       ; devuelve True, porque una s es precedida por una a
hasta c
       ; o bien por una f hasta l

     l = advMatch("esto como aquello", "[a-c]|[t-z]o" )
     l.view()
       ; devuelve True, porque dos oes están precedidas por una
a hasta c
       ; o bien por una t hasta z

endmethod
```

Vea también match
search

ansiCode

Procedimiento Devuelve el código ANSI de una cadena de un solo carácter.

Tipo String

Sintaxis **ansiCode** (const ***carácter*** String) SmallInt

Descripción Devuelve el código ANSI de *carácter*. El código ANSI devuelto es un número entero entre 1 y 255.

Ejemplo En el ejemplo siguiente, supóngase que una ficha contiene cuatro objetos de campo: *mostrarTodosLosCaracteres*, *CampoANSI*, *CampoOEM* y *CampoNombreTecla*. El método **keyPhysical** de *mostrarTodosLosCaracteres* examina todos los caracteres y, a continuación, los traduce a su código ANSI, código OEM y equivalente de nombre de tecla. Los distintos códigos de carácter se escriben en *CampoANSI*, *CampoOEM* y *CampoNombreTecla*.

```
; mostrarTodosLosCaracteres::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  Carácter String
  ANSI     SmallInt
  NombreT  String
  OEM     SmallInt
endVar
Carácter = eventInfo.char()       ; obtenemos el carácter
tecleado
ANSI = ansiCode(Carácter)         ; lo amoldamos a su código
ANSI
CampoANSI = ANSI                  ; escribimos el código ANSI
en Campo ANSI

Código = eventInfo.vCharCode()    ; obtenemos el VK_Code del
carácter

NombreT = VKCodeToKeyName(Código) ; amoldamos el VK_Code al
nombre de la tecla

CampoNombreTecla = NombreT        ; escribimos el nombre de la
tecla en
                                  ; CampoNombreTecla

OEM = oemCode(Carácter)           ; amoldamos el carácter a su
código OEM
CampoOEM = OEM                    ; escribimos el código
OEM en CampoOEM
beep()
endmethod
```

Vea también chr
chrToKeyName

breakApart

Principiante

Método Divide una cadena en subcadenas.

Tipo String

Sintaxis **breakApart** (var ***matriz*** Array[] String [,const ***separadores*** String])

Descripción Divide una cadena en una matriz de subcadenas; cada subcadena se escribe en un elemento de la matriz *matriz*. Es posible especificar uno o más caracteres de delimitación en *separadores*. Si se omite *separadores*, las subcadenas se delimitan por un espacio. En cualquier caso, los caracteres de delimitación no se incluyen en *matriz*. Este método es útil para importar datos de un archivo de texto en una tabla.

Nota: Dos delimitadores sin nada entre ellos se procesan como una señal y producen un elemento vacío en la matriz.

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *breakALaMatriz* crea tres matrices a partir de la misma cadena. La primera vez, la llamada al método **breakApart** no especifica ningún delimitador; por defecto, el método trata el espacio como delimitador. La segunda vez, la llamada a **breakApart** especifica el asterisco como delimitador. Cada vez que un asterisco sigue inmediatamente a otro asterisco, se crea un elemento vacío en la matriz. La tercera vez, se enumeran como delimitadores el signo de interrogación, la coma y el punto y coma. El espacio no es emplea como delimitador.

```
; breakALaMatriz::pushButton
method pushButton(var eventInfo Event)
var
  Mat Array[] String ; se debe poder cambiar su tamaño
  c String
endvar

c = "esto es, una : cadena ? delimitada"

c.breakApart(ar) ; rompe la cadena en espacios por
defectos
ar.view()
{
ar = esto
     es,
     una
     :
     cadena
     ?
     delimitada
}

c = "esto*es*una*cadena**delimitada"
c.breakApart(ar, "*") ; rompe la cadena en el carácter
                      ; especificado
ar.view()
{
ar = esto
```

```
     es
     una
     cadena

     delimitada

}

c = "esto es, una : cadena ? delimitada"
c.breakApart(ar, ",:?") ; rompe la cadena en los
                        ; caracteres especificados
                        ; esta vez, no hay espacios en la
                        ; lista de limitadores
ar.view()
{
ar = esto es
     una
     cadena
     delimitada
}

endmethod
```

Vea también substr

chr

Principiante

Procedimiento Devuelve la cadena de un solo carácter representada por un código ANSI.

Tipo String

Sintaxis **chr** (const ***códigoANSI*** SmallInt) String

Descripción Devuelve una cadena de un solo carácter que contiene el carácter ANSI correspondiente a *códigoANSI*. Si *códigoANSI* no es un entero entre 1 y 255, se produce un error.

chr puede utilizarse para generar caracteres a los que no se accede fácilmente mediante el teclado.

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *mostrarCarácter* asigna el carácter ANSI 167 a la variable *carácter*, convierte el carácter 167 a su nombre de tecla y lo asigna a *NombreT*. Entonces, el método muestra ambas versiones del carácter en un cuadro de diálogo.

```
; mostrarCarácter::pushButton
method pushButton(var eventInfo Event)
var
  Carácter String
  NombreT  String
endVar
Carácter = chr(167)                       ; obtenemos el
carácter
NombreT = chrToKeyName(chr(167))          ; obtenemos el nombre
de
                                          ; la tecla
msgInfo("El nombre del carácter", Carácter + ; mostramos el
carácter y
             " es " + NombreT)               ; el nombre de la
tecla
endmethod
```

Vea también chrOEM
chrToKeyName
string

chrOEM

Procedimiento Devuelve la cadena de un sólo carácter de un código OEM.

Tipo String

Sintaxis **chrOEM** (const ***códigoOEM*** SmallInt) String

Descripción Devuelve una cadena de un solo carácter que contiene el carácter OEM correspondiente a *códigoOEM*. Si *códigoOEM* no es un entero entre 1 y 255, se produce un error.

chrOEM puede utilizarse para generar caracteres a los que no se accede fácilmente mediante el teclado. Para más información, consulte la *Guía del Programador ObjectPAL*.

Ejemplo En el ejemplo siguiente, una ficha tiene un botón llamado *mostrarOEM* y un campo llamado *CampoUno*. El método **pushButton** de *mostrarOEM* muestra el carácter OEM especificado por el número de *CampoUno*.

```
; mostrarOEM::pushButton
method pushButton(var eventInfo Event)
msgInfo("Carácter OEM descrito en CampoUno",
chrOEM(CampoUno))
endmethod
```

Vea también chr
string

chrToKeyName

Procedimiento Devuelve la cadena de código de tecla virtual equivalente de una cadena de un solo carácter.

Tipo String

Sintaxis **chrToKeyName** (const ***carácter*** String) String

Descripción Devuelve el código de tecla virtual de *carácter* como cadena. Un nombre de tecla es uno de los códigos de tecla virtual (como VK_BACK para Retroceso), pero se devuelve como cadena (por ejemplo, "VK_BACK"), no como constante. Los caracteres alfanuméricos y la mayoría de los símbolos tienen un nombre de tecla que consiste simplemente en el carácter, por ejemplo, "J" para la letra *J*. ObjectPAL proporciona constantes para los códigos de tecla virtual; consulte Keyboard en el cuadro de diálogo Constantes.

Ejemplo Consulte el ejemplo de chr.

Vea también vkCodeToKeyName

fill

Principiante

Procedimiento Devuelve una cadena que contiene repeticiones de un carácter.

Tipo String

Sintaxis **fill** (const ***carácterRelleno*** String, const ***númeroRepeticiones*** SmallInt) String

Descripción Devuelve una cadena que contiene el primer carácter de *carácterRelleno* (normalmente una cadena de un solo carácter), donde *carácterRelleno* se repite el número de veces especificado en *númeroRepeticiones*. *númeroRepeticiones* debe ser un entero no negativo; si *númeroRepeticiones* es 0, **fill** devuelve una cadena vacía.

Ejemplo En este ejemplo, el método **pushButton** del botón *rellenarYVer* crea dos cadenas con el procedimiento **fill**. El método crea la primera cadena rellenando una variable con la misma letra cinco veces. La segunda cadena se crea repitiendo la cadena Quattro Pro cuatro veces.

```
; rellenarYVer::pushButton
method pushButton(var eventInfo Event)
var
  Cadena String
endVar
Cadena = fill("X", 5)
Cadena.View()                    ; muestra XXXXX
Cadena = fill("Quattro Pro
", 4)          ; añade un fin de línea cada ocurrencia
Cadena.View()
; muestra:    Quattro Pro
;             Quattro Pro
;             Quattro Pro
;             Quattro Pro
endmethod
```

Vea también space

format

Principiante

Procedimiento Devuelve una cadena formateada para su visualización o impresión.

Tipo String

Sintaxis **format** (const ***especFormato*** String, const ***valor*** AnyType) String

Descripción Permite controlar la forma en que los valores se visualizan o se imprimen. *especFormato* es una expresión de cadena que contiene una o más especificaciones de formato para su aplicación en String.

La tabla siguiente enumera la especificación de formato por defecto para categoría de formato. Esta tabla también lista los tipos de datos válidos para cada categoría de formato. Además de los tipos de datos enumerados, es posible utilizar valores AnyType, siempre que su valor pueda interpretarse coherentemente con la categoría de formato.

Categoría de formato	Significado	Tipos de datos permitidos	Por defecto
Anchura	Define la anchura de campo y precisión decimal permitidas	Todos	Valor de datos completo
Alineación	Alineación dentro de la anchura	Todos	AR (justificación a la derecha) para todos los datos numéricos, AL (justificación a la izquierda) para todos los demás tipos (incluido el punto)
Mayús/minús	Cadenas en mayúsculas o minúsculas	Todos los tipos de cadena	Sin valor por defecto
Edición	Especifica caracteres y espaciado	Todos los tipos numéricos	Consulte los valores por defecto siguientes
	Incluye un símbolo especificado		Sin valor por defecto
	Carácter de coma decimal		ED. (punto como separador decimal)
	Separador de número entero		Sin separador
	Número de ceros iniciales		Ninguno
	Espaciado de símbolos		Ninguno
	Notación científica		No

	Ocultar espacios finales		No (mostrar espacios)
	Usar ceros como patrón de cadena		No
	Aumentar todos los números de cadena		No
	Anteponer signo de dólar		No
	Separadores estadounidenses de cadena		Estadounidenses
Signo	Formato de números positivos y negatifos	Todos los tipos numéricos	Consulte los siguientes
	Positivo		Sin signo positivo inicial (999)
	Negativo		Signo menos inicial (-999)
Fecha	Especifica formatos de fecha	mm/dd/aa(aa) para Date o hh:mm:ss am(pm), mm/dd/aa(aa) para DateTime	Date y DateTime
Hora	Especifica formatos de hora	Time y DateTime mm/dd/aa(aa) para DateTime	hh:mm:ss am(pm) para Date o hh:mm:ss am(pm), DateTime
Lógico	Representación de valores de cadena	Logical	True/False

Es posible combinar dos o más especificaciones de formato en *especFormato* separándolas con comas.

Tipo	**Especif**	**Significado**
Anchura	W*n*	*n* especifica la anchura de formato total, incluyendo todos los caracteres especiales
	W.*n*	*n* especifica el número de cifras decimales, por lo que W12.2 especifica un campo de 12 caracteres, dos de los cuales van después del carácter decimal
	W.W	Utiliza cifras decimales según el formato numérico de Windows
	W.$	Utiliza cifras decimales según el formato de moneda de Windows
Alineación	AL	Alinea el campo a la izquierda

	AR	Alinea el campo a la derecha
	AC	Centra el campo
Mayús/Minús	CU	Convierte a mayúsculas
	CL	Convierte a minúsculas
	CC	Convierte letras iniciales a mayúsculas
Edición	E(*s*)	*s* especifica un símbolo que precede al número
	E$W	Incluye el símbolo de moneda definido en Windows
	ED*d*	*d* especifica el carácter de coma decimal
	EDW	Utiliza el carácter de coma decimal definido en Windows
	EN*c*	*c* especifica el separador de número entero
	ENW	Utiliza el separador de número entero definido en Windows
	EL*n*	*n* especifica el número de ceros iniciales
	ELW	Utiliza el valor de ceros iniciales definido en Windows
	EP0	Sin espaciado de símbolo
	EP-	Define el espaciado de símbolo para números negativos
	EP+	Define el espaciado de símbolo para números positivos
	EPB	Define el espaciado de símbolo para todos los números
	EPW	Utiliza el espaciado de símbolo definido en Windows
	ES	Utiliza notación científica
	ET	Oculta espacios finales
	EZ	Utiliza ceros como patrón de relleno
	EB	Utiliza blancos como patrón de relleno
	E*	Utiliza '*' como patrón de relleno
	E+*n*	Aumenta el número
	E-*n*	Reduce el número
	E$	Igual que E($)
	EC	Igual que EN (o EN.D si EC está definido)
	EI	Igual que ED (o ED,N. si EC está definido)
Signo	S+0	Formatea los números positivos como $999
	S+1	Formatea los números positivos como +$999
	S+2	Formatea los números positivos como $+999
	S+3	Formatea los números positivos como $999+
	S+4	Formatea los números positivos como 999$
	S+5	Formatea los números positivos como +999$
	S+6	Formatea los números positivos como 999+$
	S+7	Formatea los números positivos como 999$+
	S+8	Formatea los números positivos como $999DB
	S+W	Formatea los números positivos con el formato monetario definido en Windows
	S-0	Formatea los números negativos como ($999)
	S-1	Formatea los números negativos como -$999
	S-2	Formatea los números negativos como $-999
	S-3	Formatea los números negativos como $999-
	S-4	Formatea los números negativos como (999$)
	S-5	Formatea los números negativos como -999$
	S-6	Formatea los números negativos como 999-$
	S-7	Formatea los números negativos como 999$-

	S-8	Formatea los números negativos como $999CR
	S-W	Formatea los números negativos con el formato monetario de Windows
	SP	Igual que S-0
	S-	Igual que S-1
	S+	Igual que S-1+1
	SC	Igual que S-8
	SD	Igual que S-8+8
Fecha	DW1	Día de la semana como Lun
	DW2	Día de la semana como Lunes
	DWL	Día de la semana con el formato de fecha larga definido en Windows
	DM1	Mes como 1
	DM2	Mes como 01
	DM3	Mes como Ene
	DM4	Mes como Enero
	DML	Mes con el formato de fecha larga definido en Windows
	DMS	Mes con el formato de fecha corta definido en Windows
	DD1	Día como 1
	DD2	Día como 01
	DDL	Día con el formato de fecha larga definido en Windows
	DDS	Día con el formato de fecha corta definido en Windows
	DY1	Año como 1
	DY2	Año como 01
	DY3	Año como 1901
	DYL	Año con el formato de fecha larga definido en Windows
	DYS	Año con el formato de fecha corta definido en Windows
	DO(*s*)	*s* especifica orden y separadores, utilice %W para día de la semana, %D para el día numérico, %M para mes, e %Y para año; los separadores son literales, por lo que 28/12/92 como DO(%W %D-%M-%Y) es Lun 28-12-92
	DOL	Orden y separadores con el formato de fecha larga definido en Windows
	DOS	Orden y separadores con el formato de fecha corta definido en Windows
	D1	Este es el formato de fecha por defecto
	D2	Como DM4Y3O(%M %D %Y)
	D3	Como DO(%M/%D)
	D4	Como DO(%M/%Y)
	D5	Como DM3O(%D-%M-%Y)
	D6	Como DM3O(%M %Y)
	D7	Como DM3Y3O(%D-%M-%Y)
	D8	Como DY3O(%M/%D/%Y)
	D9	Como DO(%D.%M.%Y)
	D10	Como DO(%D/%M/%Y)
	D11	Como DO(%Y-%M-%D)
Hora	TH1	Horas como 1T
	TH2	Horas como 01
	THW	Horas con el formato definido en Windows

	TM1	Minutos como 1
	TM2	Minutos como 01
	TMW	Minutos con el formato definido en Windows
	TS1	Segundos como 1
	TS2	Segundos como 01
	TSW	Segundos con el formato definido en Windows
	TNA(*s*)	*s* es la cadena que se muestra tras las horas antes del mediodía
	TNP(*s*)	*s* la cadena que se muestra tras las horas después del mediodía
	TNW	Valores relativos al mediodía definidos en Windows
	TO(*s*)	*s* especifica orden y separadores, utilice %H para horas, %M para minutos, %S para segundos, %N para am/pm, %D para incluir la fecha en el formato por defecto al formatear un valor de DateTime
	TOW	Orden y separadores definidos en Windows
Lógico	LT(*s*)	*s* especifica la representación del valor True lógico
	LF(*s*)	*s* especifica la representación del valor False lógico
	LY	Valores lógicos como Yes y No
	LO	Valores lógicos como On y Off

Ejemplo En los ejemplos siguientes, supóngase que una ficha contiene un campo llamado *CampoFormat* y un botón llamado *demostraciónDeFormat*. El método **pushButton** de *demostraciónDeFormat* demuestra muchas especificaciones de formato distintas. Para cada ejemplo, el método rellena *CampoFormat* con la cadena formateada y muestra una copia de la especificación de formato en un cuadro de diálogo (con view). El método no continuará en el ejemplo siguiente hasta que se cierre el cuadro de diálogo de view, por lo que proporciona una forma de examinar tanto la especificación de archivo como la salida formateada antes de ir al ejemplo siguiente.

```
; demostraciónDeFormat::pushButton
method pushButton(var eventInfo Event)
var
  x  AnyType
  fs String
endVar
fs = "\"w6\",\"Esto es un test\""
CampoFormat = format("w6","Esto es un test")
; muestra Esto e
fs.view("Formatos")

fs = "\"w6\",1234567"
CampoFormat = format("w6",1234567)
; muestra 1.e+6
fs.view("Formatos")

fs = "\"w1\",( =5)"
CampoFormat = format("w1",( =5))
; devuelve True, muestra T
fs.View()

fs = "\"w9.2\",1234.567"
```

```
CampoFormat = format("w9.2",1234.567)

; muestra 1234.57
fs.View()

; he aquí algunos ejemplos de alineación:
fs = "\"w20,ac\",\"Esto es\""
CampoFormat = format("w20,ac","Esto es")
; muestra                Esto es
fs.view()

fs = "\"w20,ac\",\"El Título\""
CampoFormat = format("w20,ac","El Título")
; muestra            El Título
fs.view()
fs = "\"w20,ac\",\"Del libro\""
CampoFormat = format("w20,ac","Del libro")
; muestra          Del libro
fs.view()

fs = "\"w20,al\",123456"

CampoFormat = format("w20,al",123456)
; muestra  123456
fs.view()

fs = "\"w20,ar\",123456"
CampoFormat = format("w20,ar",123456)
; muestra               123456
fs.view()

; he aquí algunos ejemplos de mayúsculas y minúsculas:
fs = "\"cu\",\"Dame la manita Pepe Luis\""
CampoFormat = format("cu","Dame la manita, Pepe Luis")

; muestra DAME LA MANITA, PEPE LUIS
fs.view()

fs = "\"cl\",\"SALTA POR ENCIMA DE L\""
CampoFormat = format("cl","SALTA POR ENCIMA DE L")

; muestra  salta por encima de l
fs.view()

fs = "\"cc\",\"pERRO.\""
CampoFormat = format("cc","pERRO.")
; muestra Perro.
fs.view()

fs = "\"cc\",\"danos pipas \" + \"también\""
CampoFormat = format("cc","danos pipas " + "también")
; muestra Danos Pipas También
fs.view()


; he aquí algunos ejemplos de edición:
x = 34567.89
```

```
fs = "\"w10.2, e$c\", x"
CampoFormat = format("w10.2, e$c", x)      ; muestra $34,567.89
fs.view()

fs = "\"w10.2, e$ci\", x"

CampoFormat = format("w10.2, e$ci", x)     ; muestra $34.567,89
fs.view()

fs = "\"w13.2, e$c\", x"
CampoFormat = format("w13.2, e$c", x)      ; muestra
$34,567.89
fs.view()

fs = "\"w14.2, e$cb, al\", x"
CampoFormat = format("w14.2, e$cb, al", x) ; muestra  $
34,567.89
fs.view()

fs = "\"w15.2, e$cz, al\", x"
CampoFormat = format("w15.2, e$cz, al", x) ; muestra
$000034,567.89
fs.view()

fs = "\"w15.2, e$c*, al\", x"
CampoFormat = format("w15.2, e$c*, al", x) ; muestra
$****34,567.89

fs.view()


;    he aquí algunos ejemplos de signo:

x = -3456.12
fs = "\"w8.2, s+\", x"
CampoFormat = format("w8.2, s+", x)          ; muestra 3456.12
fs.view()

fs = "\"w11.2, e$c, sc\", x"
CampoFormat = format("w11.2, e$c, sc", x)    ; muestra
$3,456.12CR
fs.view()

fs = "\"w14.2, e$c*, sp\", x"
CampoFormat = format("w14.2, e$c*, sp", x)   ; muestra
($***3,456.12)
fs.view()

fs = "\"w13.2, e$c*, s+\", x"
CampoFormat = format("w13.2, e$c*, s+", x)   ; muestra -
$***3,456.12
fs.view()

fs = "\"w14.2, e$c*, sd\", x"

CampoFormat = format("w14.2, e$c*, sd", x)  ; muestra
$***3,456.12CR
```

```
fs.view()

; he aquí una miscelanea de ejemplos:
fs = "\"D2\", Date(\"3/7/1948\""
CampoFormat = format("D2", Date("3/7/1948")) ; muestra
Marzo 7, 1948
fs.view()

fs = "\"W9.2, AL\", 1234.123"
CampoFormat = format("W9.2, AL", 1234.123)
; muestra 1234.12 en un campo de 9 dígitos con 2 cifras
decimales
fs.view()

fs = "\"W9.2, AR\", 1234.123"
CampoFormat = format("W9.2, AR", 1234.123)
; muestra 1234.12 alineado a la derecha en el mismo campo
fs.view()

; para mostrar fecha y hora en formato de 24 horas

fs = "\"TNA(), TNP(), TO(%H:%M:%S %D), DO(%W %M%D%Y)\","+

                " dateTime(\"2:30:00 pm 11/24/92\")"

Campoformat = format ("TNA(), TNP(), TO(%H:%M:%S %D), DO(%W %M
%D%Y)\",
                                dateTime("2:30:00 pm
11/24/92"))

; muestra 14:30:00 Tue 11/24/92

fs-view("Formatos")

endmethod
```

Vea también string

ignoreCaseInStringCompares

Procedimiento Especifica si se diferencia el uso de mayúsculas y minúsculas al comparar cadenas.

Tipo String

Sintaxis **ignoreCaseInStringCompares** (const ***yesNo*** Logical)

Descripción Especifica si se diferencia el uso de mayúsculas y minúsculas al comparar cadenas. Normalmente, se considera que las letras en mayúscula y en minúscula no se corresponden. Por ejemplo, Q y q no son lo mismo. Pero, cuando se utiliza **ignoreCaseInStringCompares(Yes)**, no se toma en cuenta este aspecto, por lo que Q sería igual a q . Una vez que se ejecuta **ignoreCaseInStringCompares(Yes)**, permanece en vigor hasta que se ejecuta **ignoreCaseInStringCompares**(No).

Para averiguar si se diferencia el uso de mayúsculas y minúsculas, utilice **isIgnoreCaseInStringCompares**.

Ejemplo En este ejemplo, el método **pushButton** del botón *intentaComparar* comprueba si Paradox está configurado para no diferenciar las mayúsculas y minúsculas en la comparación de cadenas. Si **isIgnoreCaseInStringCompares** devuelve Yes, el método emplea **ignoreCaseInStringCompares** para definirlo como No lo que implica que se diferencia el uso de mayúsculas y minúsculas y compara una cadena en mayúsculas y minúsculas. Una ventana de mensajes informa al usuario de que las cadenas no son equivalentes. A continuación, el método desactiva la diferenciación e intenta la misma comparación, que ahora devuelve True.

```
; intentaComparar::pushButton
method pushButton(var eventInfo Event)
var
  c1,
  c2  String
endVar
c1 = "gato"
c2 = "GATO"
if isIgnoreCaseInStringCompares() then
  ignoreCaseInStringCompares(No)
endif
x = (c1 = c2)               ; el primer "=" asigna, los demás
comparan
msgInfo(c1 + " = " + c2 + "?", x) ; muestra False
ignoreCaseInStringCompares(Yes)
x = (c1 = c2)
msgInfo(c1 + " = " + c2 + "?", x) ; muestra True
endmethod
```

Vea también isIgnoreCaseInStringCompares

isIgnoreCaseInStringCompares

Procedimiento Informa de si diferencia el uso de mayúsculas y minúsculas al comparar cadenas.

Tipo String

Sintaxis **isIgnoreCaseInStringCompares** ( ) Logical

Descripción Devuelve True si se diferencia el uso de mayúsculas y minúsculas al comparar cadenas; en caso contrario, devuelve False.

Para especificar si se considera el uso de mayúsculas y minúsculas, utilice **ignoreCaseInStringCompares**.

Ejemplo Consulte el ejemplo de ignoreCaseInStringCompares.

Vea también ignoreCaseInStringCompares

isSpace

Principiante

Método Informa de si una cadena sólo contiene espacios (o está vacía).

Tipo String

Sintaxis **isSpace** (const ***cadena*** String) Logical

Descripción Devuelve True si *cadena* sólo contiene espacio blanco, o si *cadena* es la cadena vacía ( ). De lo contrario, devuelve False. Entre los caracteres de espacio blanco, se incluyen espacios, tabuladores, retornos de carro, avances de línea y avances de hoja.

Ejemplo Este ejemplo crea varias cadenas y comprueba si contienen sólo espacios, o no contienen nada en absoluto. Este es el código del método **pushButton** del botón *valorDeCadenas*:

```
; valorDeCadenas::pushButton
method pushButton(var eventInfo Event)
var
  c String
endVar
c = space(3)                            ; 3 espacios
msgInfo("3 Espacios", c.isSpace())      ; True
c = ""                                  ; cadena vacía
msgInfo("Cadena Vacía", c.isSpace())    ; True
c = "Z" + space(2)                      ; Z y 2 espacios
msgInfo("Z y 2 Espacios", c.isSpace()) ; False
endmethod
```

Vea también space

keyNameToChr

Procedimiento Devuelve la cadena de un solo carácter representada por una cadena de código de tecla virtual.

Tipo String

Sintaxis **keyNameToChr** (const ***nombreTecla*** String) String

Descripción Devuelve la cadena de un solo carácter representada por el código de tecla virtual *nombreTecla*.

nombreTecla es uno de los códigos de tecla virtual (como VK_BACK para Retroceso), pero debe proporcionarse como cadena (por ejemplo, "VK_BACK"), no como constante. Los caracteres alfanuméricos y muchos símbolos tienen un nombre de tecla que consiste simplemente en el carácter, por ejemplo, "J" para la letra *J*. ObjectPAL proporciona constantes para los códigos de tecla virtual; consulte Keyboard en el cuadro de diálogo Constantes.

Ejemplo Consulte el ejemplo de keyNameToVKCode.

Vea también chrToKeyName
keyNameToVKCode

keyNameToVKCode

Procedimiento Devuelve el código VK_ numérico de una cadena de código de tecla virtual.

Tipo String

Sintaxis **keyNameToVKCode** (const ***nombreTecla*** String) SmallInt

Descripción Devuelve el código VK_ del carácter representado por la cadena *nombreTecla*.

nombreTecla es uno de los códigos de tecla virtual (como VK_BACK para Retroceso), pero debe proporcionarse como cadena (por ejemplo, "VK_BACK"), no como constante. Los caracteres alfanuméricos y muchos símbolos tienen un nombre de tecla que consiste simplemente en el carácter, por ejemplo, "J" para la letra *J*. ObjectPAL proporciona constantes para los códigos de tecla virtual; consulte Keyboard en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, el método **pushButton** de *mostrarCódigo* asigna un corchete derecho ([) a la variable de cadena *Cadena* y muestra el código ANSI y el nombre de tecla de *Cadena* en un cuadro de diálogo.

```
; mostrarCódigo::pushButton
method pushButton(var eventInfo Event)
var
  Cadena  String
endVar
Cadena = "["              ; asignamos al nombre de la
                          ; tecla abrir corchete
msgInfo("Código_VK/Carácter", "Código_VK: " +       ; Código_VK
91
             String(keyNameToVKCode(Cadena)) +
             "\nCarácter: " + keyNameToChr(Cadena)) ; Carácter
"["
endmethod
```

Vea también chrToKeyName
keyNameToChr
vkCodeToKeyName

lower

Principiante

Método Convierte una cadena a minúsculas.

Tipo String

Sintaxis **lower** ( ) String

Descripción Convierte una cadena a letras minúsculas. Utilice **upper** para convertirla a mayúsculas.

Ejemplo En el ejemplo siguiente, el método **pushButton** de *hacerMinúsculas* crea una cadena en mayúsculas y utililza **lower** para mostrarla en minúsculas.

```
; hacerMinúsculas::pushButton
method pushButton(var eventInfo Event)
var
  Texto String
endVar
Texto = "¡HOLA A TODOS! ES HORA DE CERRAR"
msgInfo("Noticia oficial", Texto.lower())
; muestra "¡hola a todos! es hora de cerrar"
endmethod
```

Vea también upper

lTrim

Principiante

Método Elimina los blancos iniciales de una cadena.

Tipo String

Sintaxis **lTrim** ( ) String

Descripción Elimina espacios y caracteres de tabulador del extremo izquierdo de una cadena.

Ejemplo En este ejemplo, el método **pushButton** de *eliminarIzquierda* crea una cadena con espacios iniciales y un tabulador inicial (la secuencia de escape \t). El método muestra la cadena original, emplea **lTrim** para eliminar los caracteres no imprimibles iniciales y muestra la versión truncada.

```
; eliminarIzquierda::pushButton
method pushButton(var eventInfo Event)
var
   Original, Cortada String
endVar
Original = "  \t   Primera palabra"  ; cadena con espacios y un
tabulador
msgInfo("Cadena original", Original)

Cortada = Original.lTrim()           ; elimina los espacios y
el tabulador
msgInfo("Una versión ligeramente más corta", Cortada)
; muestra "Primera palabra"
endmethod
```

Vea también rTrim

match

Principiante

Método Compara una cadena con un patrón.

Tipo String

Sintaxis **match** (const ***patrón*** String [, var ***varCoincidencia*** String]*) Logical

Descripción Comprueba si una cadena coincide con un patrón y, si es así, extrae los componentes de la cadena que coinciden con los elementos comodín. El valor de *patrón*, al igual que la patrones en las consultas, consta de caracteres entremezclados con los operadores comodín .. y @. El .. equivale a la coincidencia de cualquier número de caracteres (o de ninguno), mientras que el operador @ equivale a la coincidencia de cualquier carácter individual. También como en las consultas, **match** no toma en cuenta por defecto el uso de mayúsculas y minúsculas (pero puede emplearse **ignoreCaseInStringCompares(No)** para que sí lo diferencie).

varCoincidencia es una variable a la que se asignará la parte coincidente. **match** asigna los patrones coincidentes a una o más variables *matchVar* conforme se encuentran los patrones. Las partes de la cadena que coinciden con los elementos comodín se asignan a las variables de izquierda a derecha. Puesto que puede haber múltiples coincidencias, la primera subcadena coincidente se asigna a la primera variable, la segunda subcadena coincidente a la segunda variable, y así sucesivamente. Si no se encuentra ninguna coincidencia, no se asignan valores a las variables.

Para incluir una comilla en el patrón, antepóngale una barra invertida (\). Para incluir un punto, rodéelo con comillas y anteponga barras invertidas a las comillas (\".\").

Ejemplo Estas sentencias demuestran la funcionalidad de match.

```
var
  c, x, y, z String
endVar

c = "esto y aquello"

msgInfo("¿Coinciden?", c.match("e.."))         ; muestra True
msgInfo("¿Coinciden?", c.match("@sto.."))      ; muestra True
msgInfo("¿Coinciden?", c.match("@ y aquello")) ; muestra False
msgInfo("¿Coinciden?", c.match("..y.."))       ; muestra True

msgInfo("¿Coinciden?", c.match("..y..", x, y))
                      ; muestra True (x = esto, y = aquello)

msgInfo("¿Coinciden?", c.match("E..", z))
  ; si isIgnoreCaseInString() es False, esta sentencia muestra
  ; False, y z no se asigna. Utilice
  ; ignoreCaseInStringCompares(Yes) para verlo en la pantalla
  ; True, y asigna a z "sto y aquello"
```

Vea también advMatch
search

oemCode

Procedimiento Devuelve el código OEM de una cadena de un solo carácter.

Tipo String

Sintaxis **oemCode** (const ***carácter*** String) SmallInt

Descripción Devuelve el código OEM de *carácter*. El código OEM devuelto es un entero entre 1 y 255.

Ejemplo Consulte el ejemplo de ansiCode.

Vea también ansiCode
chrToKeyName

rTrim

Principiante

Método Elimina los blancos finales de una cadena.

Tipo String

Sintaxis **rTrim** ( ) String

Descripción Elimina caracteres de espacio, tabulador, retorno de carro y avance de línea del extremo derecho de una cadena.

Ejemplo En este ejemplo, el método **pushButton** de *eliminarDerecha* crea una cadena con espacios finales. El método muestra la cadena original, utiliza **rTrim** para eliminar los caracteres no imprimibles finales y muestra la versión truncada.

```
; eliminarDerecha::pushButton
method pushButton(var eventInfo Event)
var
  Original, Cortada String
endVar
Original = "Ultima palabra     "     ; cadena con espacios al
final
msgInfo("Cadena original", Original + "Fin")
; muestra "Ultima palabra     Fin"

Original = Original.rTrim()      ; elimina los espacios
msgInfo("Una versión ligeramente más corta", Cortada + "Fin")
; muestra "Ultima palabraFin"
endmethod
```

Vea también lTrim

search

Principiante

Método Devuelve la posición de una cadena dentro de otra.

Tipo String

Sintaxis **search** (const ***cad*** String) SmallInt

Descripción Comprueba la aparición de *cad* dentro de una cadena de destino. Si se encuentra *cad*, **search** devuelve la posición del carácter inicial de *cad* dentro de la cadena de destino; en caso contrario, devuelve 0. La búsqueda siempre comienza en el primer carácter de la cadena de destino.

Por defecto, **search** no tiene en cuenta el uso de mayúsculas y minúsculas, pero es posible utilizar **ignoreCaseInStringCompares** para que sí lo diferencie.

Ejemplo El ejemplo siguiente busca partes de la cadena Goliath y Golgolítico . El código siguiente se anexa al método **pushButton** del botón *buscarCadena*:

```
; buscarCadena::pushButton
method pushButton(var eventInfo Event)
var
  c String
endVar
c = "Goliath"
msgInfo("¿Dónde está lia en Goliath?", c.search("lia")) ;
muestra 3
msgInfo("¿Dónde está lai en Goliath?", c.search("lai")) ;
muestra 0
ignoreCaseInStringCompares(No)
c = "Golgolítico"
msgInfo("¿Dónde está gol en Golgolítico?", c.search("gol"))
; muestra 4
; Nota: si ignoreCaseInStringCompares está activado, la última
; búsqueda produce un 1 en vez de 4.
endmethod
```

Vea también advMatch
match

size

Principiante

Método Devuelve la longitud de una cadena.

Tipo String

Sintaxis **size** ( ) SmallInt

Descripción Devuelve el número de caracteres (incluidos los espacios) existentes en una cadena.

Ejemplo En este ejemplo, el método **pushButton** de *obtenTamaño* asigna una cadena a la variable *Texto* y muestra la frase y su tamaño en un cuadro de diálogo. Entonces, el método emplea **size** para obtener la primera mitad de *Texto* y la asigna de nuevo a *Texto*. El tamaño del *Texto* y del *Texto* más pequeño se muestra en un cuadro de diálogo.

```
; obtenTamaño::pushButton
method pushButton(var eventInfo Event)
var
  Texto String
endVar
Texto = "Esta es una sentencia corta."
msgInfo("Tamaño", "Longitud: " + String(Texto.size()) +
                  "\n" + Texto)
; muestra    Longitud: 28
;            Esta es una sentencia corta.

; ahora cortamos la sentencia por la mitad
Texto = subStr(Texto, 1, SmallInt(Texto.size()/2))
msgInfo("La mitad del tamaño", "Longitud: " +
strVal(Texto.size())
                           + "\n" + Texto)

; muestra   Longitud: 14
;           Esta es una se
endmethod
```

Vea también string

space

Principiante

Método	Crea una cadena de un número especificado de espacios.
Tipo	String
Sintaxis	**space** (const ***númeroEspacios*** SmallInt) String
Descripción	Devuelve una cadena de *númeroEspacios* espacios.
Ejemplo	Consulte el ejemplo de isSpace.
Vea también	isSpace

string

Principiante

Procedimiento Convierte un valor a String.

Tipo String

Sintaxis **string** (const ***valor*** AnyType [, const ***valor*** AnyType]*) String

Descripción Convierte la expresión *valor* a String. Si se especifican varios argumentos, **string** los convierte a String y los concatena en una sola cadena.

Ejemplo En el ejemplo siguiente, el método **pushButton** de *NúmeroACadena* solicita al usuario un número, lo convierte a cadena y la concatena con otra cadena para su visualización en un cuadro de diálogo de **msgInfo**.

```
; NúmeroACadena::pushButton
method pushButton(var eventInfo Event)
var
  nn Number
endVar
nn = 0,0                      ; inicializar el número
nn.View("Escriba un número") ; lo mostramos y pedimos que se
escriba otro

; Nota: Como se puede escribir un solo argumento para el texto
; del cuadro de diálogo msgInfo, si se tiene algún elemento que
no
; sea de tipo cadena, debe amoldarse a dicho tipo, y ser
; concatenado. En nuestro caso, nn es amoldado al tipo
; String antes de ser concatenado con "Usted ha escrito "

msgInfo("Estado", "Usted ha escrito " + string(nn))
msgInfo("Estado", string("Usted ha escrito ", nn))  ;también
funciona
endmethod
```

Vea también format

strVal

Procedimiento Convierte un valor a cadena.

Tipo String

Sintaxis **strVal** (const ***valor*** AnyType) String

Descripción Convierte *valor* a cadena. El tipo de datos de *valor* puede ser cualquiera de los tipos representados por AnyType.

Ejemplo Consulte el ejemplo de size.

Vea también string
size

substr

Principiante

Método Devuelve una parte de una cadena.

Tipo String

Sintaxis **substr** (const ***inicioIndice*** Number [, const ***númeroCaracteres*** SmallInt]) String

Descripción Devuelve la parte de la cadena que comienza en *inicioIndice* y continúa durante *númeroCaracteres* caracteres. El valor de *inicioIndice* debe ser mayor que 0 y menor o igual que el tamaño de la cadena. Si *númeroCaracteres* es 0, **substr** devuelve una cadena nula. Si se omite *númeroCaracteres*, **substr** devuelve el carácter situado en la posición *inicioIndice*.

Ejemplo En este ejemplo, supóngase que una ficha contiene un botón llamado *obtenerTeléfono* y cuatro campos llamados *TelefCompleto*, *Prefijo*, *Area* y *número*. El método de este ejemplo utiliza **substr** para extraer los tres grupos de dígitos de un número telefónico. El código siguiente se anexa al método **pushButton** de *obtenerTeléfono*.

```
; obtenerTeléfono::pushButton
method pushButton(var eventInfo Event)
var
  NumTeléfono String
endVar
NumTeléfono = TelefCompleto.Value
; asume que el número de teléfono se ha escrito con el
; formato (###) ###-##-##
; comenzamos por la primera posición, tomando tres caracteres
Prefijo.Value = NumTeléfono.substr(2, 3)  ; obtiene el número
de prefijo
Area.Value    = NumTeléfono.substr(7, 3)  ; obtiene el número
de área
número.Value  = NumTeléfono.substr(11, 5) ; obtiene el número
beep()
endmethod
```

Vea también breakApart
search
size

toANSI

Método Convierte una cadena de caracteres OEM a caracteres ANSI.

Tipo String

Sintaxis **toANSI** ( ) String

Descripción Convierte una cadena de caracteres OEM a caracteres ANSI.

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *mostrarANSI* muestra una cadena de dos formas: en el título del cuadro de diálogo, la cadena se muestra como tal; en la ventana del cuadro de diálogo, la cadena se convierte antes a ANSI. El último carácter de la cadena es el símbolo de copyright (©). Este símbolo se imprime en el título del cuadro de diálogo; sin embargo, en la ventana del cuadro de diálogo, se sustituye el símbolo por un carácter de subrayado (_).

```
; mostrarANSI::pushButton
method pushButton(var eventInfo Event)
var
  c String
endVar
; la cadena más el símbolo de copyright
c = "Una cadena de caracteres " + chr(169)
msgInfo(c, c.toANSI())
; muestra la cadena más "_" en un cuadro de diálogo - depende
del sistema
endmethod
```

Vea también toOEM

toOEM

Método Convierte una cadena de caracteres ANSI a caracteres OEM.

Tipo String

Sintaxis **toOEM** ( ) String

Descripción **toOEM** convierte una cadena de caracteres ANSI a caracteres OEM.

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *MostrarOEM* muestra una cadena de dos formas: en el título del cuadro de diálogo, la cadena se muestra como tal; en la ventana del cuadro de diálogo, la cadena se convierte antes a OEM. El último carácter de la cadena es el símbolo de copyright (©). Este símbolo se imprime en el título del cuadro de diálogo; sin embargo, en la ventana del cuadro de diálogo, se sustituye el símbolo por la letra *c*.

```
; MostrarOEM::pushButton
method pushButton(var eventInfo Event)
var
  c String
endVar
; la cadena más el símbolo de copyright
c = "Una cadena de caracteres " + chr(169)
msgInfo(c, c.toOEM())
; muestra la cadena más "c" en un cuadro de diálogo
endmethod
```

Vea también toANSI

upper

Principiante

Método Convierte una cadena a mayúsculas.

Tipo String

Sintaxis **upper** ( ) String

Descripción Convierte una cadena a letras mayúsculas. Utilice **lower** para convertirla a minúsculas.

Ejemplo En este ejemplo, el método **pushButton** de *hacerMayúsculas* obtiene una cadena del usuario y la convierte a mayúsculas. La cadena convertida se compara entonces con una constante de cadena en mayúsculas.

```
; hacerMayúsculas::pushButton
method pushButton(var eventInfo Event)
const
  TIPOPEDIDO = "PEDIDOOFERTA"   ; concatena dos tipos válidos
endConst
var
  Texto String
  x     SmallInt
endVar
Texto = ""                                ; inicializa la
cadena
Texto.view("Escriba 'Pedido' o 'Oferta'") ; obtener una
respuesta
Texto = Texto.upper()                     ; transforma en
mayúsculas
if search(TIPOPEDIDO, Texto)  0 then
  ; buscamos una cadena que coincida -- devuelve el lugar
  ; de coincidencia, o cero si no coinciden

  msgInfo("Estado", "Ha introducido un tipo válido.")
else
  msgStop("Alto", "Debe introducir Oferta o Pedido.")
endif
endmethod
```

Vea también lower

vkCodeToKeyName

Procedimiento Convierte una constante de código de tecla virtual a una cadena de código de tecla virtual.

Tipo String

Sintaxis **vkCodeToKeyName** (const ***códigoVK*** SmallInt) String

Descripción Devuelve, como String, el nombre de código de tecla virtual del carácter representado por *códigoVK*.

Un nombre de tecla es uno de los códigos de tecla virtual (como VK_BACK para Retroceso), pero se devuelve como cadena (por ejemplo, "VK_BACK"), no como constante. Los caracteres alfanuméricos y muchos símbolos tienen un nombre de tecla que consiste simplemente en el carácter, por ejemplo, "J" para la letra *J*. ObjectPAL proporciona constantes para los códigos de tecla virtual; consulte Keyboard en el cuadro de diálogo Constantes.

Ejemplo Consulte el ejemplo de ansiCode.

Vea también ansiCode
chr
chrToKeyName
keyNameToChr

time

Principiante

Procedimiento Convierte un valor a hora, o devuelve la hora actual.

Tipo Time

Sintaxis **time** ([const ***valor*** AnyType]) Time

Descripción Convierte *valor* a hora o devuelve la hora actual según el reloj del ordenador. *valor*, si se proporciona, debe tener el mismo formato que el actual de Paradox. Para más información, consulte el procedimiento **formatSetTimeDefault** del tipo System.

Ejemplo Este ejemplo ejecuta time para convertir un valor de cadena a un valor horario:

```
var
  Cadena String
  Hora   Time
endVar

Cadena =  12:21:33 am
Hora = time(Cadena)
```

El ejemplo siguiente muestra la hora actual en un cuadro de diálogo. El formato de visualización varía en función del formato de hora actual del usuario. Este código se anexa al método **pushButton** de un botón.

```
; botónTiempo::pushButton
method pushButton(var eventInfo Event)

  ; visualiza la hora actual en un cuadro de diálogo
   msgInfo( Hora actual , time())
endmethod
```

Vea también DateTime::hour
DateTime::milliSec
DateTime::minute
DateTime::second

addArray

Método Añade elementos de una matriz en un menú.

Tipo Menu

Sintaxis **addArray (** const ***elementos*** Array[] String **)**

Descripción Añade *elementos* de una matriz en un menú. Los *elementos* de la matriz se muestran de izquierda a derecha en la barra de menús. Para crear un menú desplegable o en cascada, utilice **addPopUp**.

Ejemplo Este ejemplo crea y muestra una barra de menús de una aplicación cuando se abre una ficha y podría ser el menú principal de la aplicación. En toda la aplicación, el menú mostrado aquí puede ser modificado por métodos de otros objetos.

```
; estaFicha::open
method open(var eventInfo Event)
var
  MenúPrincipal  Menu             ; menú principal
  Items Array[3] String     ; items del menú principal
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
elemento de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; el menú aparece cuando se abre la primera Ficha
    Items[1] = "Archivo"                ; Rellenamos la matriz
    Items[2] = "Editar"
    Items[3] = "Ventanas"
    MenúPrincipal.addArray(Items)       ; lo mismo que
MenúPrincipal.addText(...)
          ; 3 veces
    MenúPrincipal.show()                ; mostramos el menú
endif
endmethod
```

Vea también addBreak
addPopUp
addStaticText
addText

addBreak

Método Comienza una nueva fila en un menú.

Tipo Menu

Sintaxis **addBreak ( )**

Descripción Comienza una nueva fila en un menú. **addBreak** permite que el programador "ajuste" explícitamente las construcciones de menú grandes en dos o más filas.

Ejemplo Este ejemplo construye y muestra la barra de menús de una aplicación cuando se abre una ficha. Utiliza uses **addBreak** para añadir una segunda fila en la barra de menús.

```
; esteFormato::open ;
method open(var eventInfo Event)
var
  MenúPrincipal Menu
endVar

if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; el menú aparece cuando se abre la primera Ficha
    MenúPrincipal.addText("Archivo")
    MenúPrincipal.addText("Editar")
    MenúPrincipal.addBreak()
    MenúPrincipal.addText("Acerca de...")     ; esto aparece en
la segunda fila
    MenúPrincipal.show()                      ; mostramos el
menú
endif

endmethod
```

Vea también addArray
addPopUp
addStaticText
addText

addPopUp

Método Añade un menú emergente en una opción de la barra de menús.

Tipo Menu

Sintaxis **addPopUp (** const ***nombreMenú*** String, const ***menúEmergenteCascada*** PopUpMenu **)**

Descripción Añade la cabecera *nombreMenú* y el menú emergente *menúEmergenteCascada* en un menú. Este método es útil para crear menús emergentes y menús en cascada.

Es necesario declarar y crear el menú emergente antes de usar **addPopUp**.

Ejemplo El código de este ejemplo se anexa al método estándar **arrive** de cada una de las dos páginas de una ficha. El método **arrive** de *páginaUno* crea y muestra un menú personalizado. El método **arrive** de *páginaDos* de la misma ficha elimina el menú personalizado. **addPopUp** se utiliza para crear un menú emergente en cascada y un menú desplegable.

Este es el método **arrive** de *páginaUno*:

```
; páginaUno::arrive
method arrive(var eventInfo MoveEvent)
var
  d1, d2, d3  PopUpMenu
  m1          Menu
endVar

d1.addText("Contraseñas...")      ; añade items al menú
desplegable d1
d1.addText("Atributos...")
d2.addText("Basica...")           ; añade items al menú
desplegable d2
d2.addText("Científica...")
d1.addPopUp("Calculadora", d2)    ; añade otro ítem al menú
desplegable d1,
    ; y muestra el menú d2 cuando se
;
    ; selecciona el item
d3.addText("Acerca de...")        ; añade un ítem al tercer menú
desplegable
m1.addPopUp("Utilidades", d1)     ; añade un ítem a la barra de
menú,
    ; y desplega d1 cuando se selecciona
m1.addPopUp("Ayuda", d3)          ; añade el ítem a la barra de
menú,
    ; y desplega d3 cuando se selecciona
m1.show()                         ; muestra la barra de menú (No
es de tipo
    ; PopUpMenu)
endmethod
```

Este es el método **arrive** de *páginaDos*:

```
; páginaDos::arrive
```

```
method arrive(var eventInfo MoveEvent)  removemenu()
; elimina el menú personalizado-aparece en su lugar
; el menú por defecto
endmethod
```

Vea también addArray
addBreak

addStaticText

Método Añade una cadena de texto no seleccionable en un menú.

Tipo Menu

Sintaxis **addStaticText (** const ***elemento*** String **)**

Descripción Añade *elemento* en un menú como texto no seleccionable.

Ejemplo En este ejemplo, el código anexado al método **open** de una ficha crea una barra de menús. Este ejemplo emplea **addStaticText** para añdir una opción de menú estática en la barra de menús.

```
;estaFicha::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    MenúPrincipal.addStaticText("Menú Principal")   ; el primer
ítem es estático
    MenúPrincipal.addText("Archivo")                ; añadimos
dos items más
    MenúPrincipal.addText("Editar")
    MenúPrincipal.show()                            ; mostramos
el menú
endif
endmethod
```

Vea también addText

addText

Método Añade una cadena de texto seleccionable en un menú.

Tipo Menu

Sintaxis **1. addText (** const ***nombreMenú*** String **)**

2. addText (const ***nombreMenú*** String, const ***atributo*** SmallInt **)**

3. addText (const ***nombreMenú*** String, const ***atributo*** SmallInt, const ***id*** SmallInt **)**

Descripción Añade la opción *nombreMenú* en un menú. Las opciones de menú se muestran de izquierda a derecha en la barra de menús. Es posible utilizar *atributo* para predefinir el atributo de visualización de *nombreMenú*. ObjectPAL proporciona constantes (como MenuEnabled) para *atributo*; consulte MenuChoiceAttributes en el cuadro de diálogo Constantes.

En la tercera forma de la sintaxis, puede especificar un número *id* (un SmallInt) que identifique el menú por su número en lugar de por *nombreMenú*. Entonces, en el método estándar **menuAction**, utilice el número *id* para determinar qué menú elige el usuario. Cuando especifique un *id* de menú, debería utilizar la constante estándar UserMenu como constante base y, posteriormente, añada su propio número. Por ejemplo, la línea siguiente añade "Archivo" en el menú *myMenu* y especifica un número *id* para esa opción de menú:

```
MiMenú.addText ("Archivo", MenuEnabled, UserMenu + 1)
```

Para más información sobre las constantes definidas por el usuario, consulte el Capítulo 7 de la *Guía del Programador ObjectPAL*.

Es posible utilizar un ampersand (&) en una opción de menú para designar una letra subrayada. Por ejemplo, la opción "&Archivo" se mostraría como ***Archivo*** y el usuario podría elegirla pulsando *Alt+A*. Si confía en *nombreMenú* para comprobar la elección del usuario, recuerde incluir el ampersand en la cadena de comparación. En este ejemplo, el valor devuelto es "&Archivo", no "Archivo".

Si desea alinear a la derecha las opciones de menú, puede anteponer el valor de cadena "\008" a *nombreMenú*. Una vez que incluya "\008" en *nombreMenú*, las opciones de menú siguientes aparecen alineadas a la derecha; no es necesario utilizar "\008" de nuevo. Por ejemplo, estas líneas muestran Archivo a la izquierda, y Ayuda y Utilidades a la derecha:

```
MiMenú.addText ("Archivo")
MiMenú.addText ("\008Ayuda")
MiMenú.addText ("Utilidades")
MiMenú.Show
```

Ejemplo Los ejemplos siguientes demuestran cómo influye la sintaxis de **addText** en la forma en que se comprueba la elección en el menú por parte del usuario.

El primer ejemplo utiliza la primera forma de la sintaxis **addText** para crear un menú sencillo. Este ejemplo no usa *id* en las sentecias **addText**. El código anexado al método estándar **menuAction** debe evaluar la cadena

especificada en *nombreMenú* para determinar la elección en el menú por parte del usuario. El código siguiente se anexa al método **open** de *páginaUno*.

```
; páginaUno::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
  UtilDespleg   PopUpMenu
endVar

  ; construimos un menú desplegable
UtilDespleg.addText("&Hora")
UtilDespleg.addText("&Fecha")

  ; asociamos el menú desplegable al ítem Utilidades del menú
principal
MenúPrincipal.addPopUp("&Utilidades", UtilDespleg)

  ; añadimos "Ayuda" al menú y alineamos "Ayuda" con \008
MenúPrincipal.addText("\008&Ayuda")

  ; ahora mostramos el menú
MenúPrincipal.show()

endmethod
```

El código siguiente se anexa al método **menuAction** de *páginaUno*. Este código utiliza el método **menuChoice** para obtener el valor de cadena definido mediante *nombreMenú*.

```
; páginaUno::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Selección String
endVar

Selección = eventInfo.menuChoice() ; asignamos el valor de la
cadena a Selección

; ahora utilizamos el valor Selección para determinar qué menú
se ha
; seleccionado
switch
  case Selección = "&Hora"  :
    msgInfo("Hora actual", time())
  case Selección = "&Fecha"  :
     msgInfo("Fecha de hoy", today())
  case Selección = "\008&Ayuda" :
    ; abrimos el sistema de ayuda estándar
    action(EditHelp)
endSwitch

endmethod
```

El ejemplo siguiente demuestra cómo puede utilizarse la cláusula *id* con **addText** para referenciar a opciones de menú por su número en lugar de

por su nombre. Este código establece constantes definidas por el usuario para que se puedan recordar con mayor facilidad las asignaciones de *id* en el menú. El código siguiente va en la ventana Const de *páginaUno*:

```
; páginaUno::Const
Const            ; definimos constantes para los
identificadores del menú
                 ; los valores reales (1, 2 y 3) son
arbitrarios
MenúDeHora = UserMenu + 1
MenúDeFecha = UserMenu + 2
MenúDeAyuda = UserMenu + 3
endConst
```

El código siguiente se anexa al método **open** de *páginaUno*. Para controlar los atributos de visualización del menú, este código emplea constantes estándar como MenuEnabled. Para identificar cada opción de menú por su nombre, el código emplea las constantes definidas en la ventana Const de *páginaUno* (TimeMenu, DateMenu y HelpMenu).

```
; páginaUno::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
  UtilDespleg   PopUpMenu
endVar

  ; construimos un menú desplegable y utilizamos las constantes
  ; (p.ej.: MenúDeHora) definidas en la ventana de constantes
(Const)
  ; de esta misma página
UtilDespleg.addText("&Hora", MenuEnabled, MenúDeHora)
UtilDespleg.addText("&Fecha", MenuEnabled, MenúDeFecha)

  ; asociamos el menú desplegable al ítem Utilidades del menú
principal
MenúPrincipal.addPopUp("&Utilidades", UtilDespleg)

  ; añadimos "Ayuda" a la barra de menú y alineamos "Ayuda" con
\008
MenúPrincipal.addText("\008&Ayuda", MenuEnabled, MenúDeAyuda)

mainMenu.show()                          ; mostramos el menú

endmethod
```

El código siguiente se anexa al método **menuAction** de *páginaUno*. Este método evalúa las selecciones en el menú por su número *id*, y no por el nombre especificado en *nombreMenú*.

```
; páginaUno::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Selección SmallInt
endVar
```

```
Selección = eventInfo.id()      ; asignamos un valor constante
(p.ej.:900) a
                                ; Selección

  ; ahora utilizamos las constantes para determinar qué menú se
ha seleccionado
switch
  case Selección = MenúDeHora :
    msgInfo("Hora Actual", time())
  case Selección = MenúDeFecha :
    msgInfo("Fecha de hoy", today())
  case Selección = MenúDeAyuda :
    ; abrimos el sistema de ayuda estándar
    action(EditHelp)
endSwitch

endmethod
```

Vea también addStaticText
addArray

contains

Método Informa de si una opción está en un menú.

Tipo Menu

Sintaxis **contains (** const ***opción*** AnyType) Logical

Descripción Devuelve True si *opción* está en la lista de opciones de un menú; en caso contrario, devuelve False. **contains** diferencia el uso de mayúsculas o minúsculas.

Ejemplo Este ejemplo supone que un objeto multirregistro está en la ficha. Cuando el usuario cambia el valor de un campo contenido en el objeto multirregistro, se añade una opción Deshacer en la barra de menús personalizada existente. Cuando el usuario se desplaza a otro registro, se elimina Deshacer. El ejemplo utiliza **contains** para averiguar si Deshacer está presente antes de añadir o eliminar la opción. La variable de menú se define en la ventana Var de la ficha. La barra de menús se crea mediante el método **open** de la ficha.

El código siguiente va en la ventana Var de la ficha:

```
; estaFicha::var
Var
  m1 Menu
endVar
```

El código siguiente corresponde al método **open** de la ficha:

```
; estaFicha::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    m1.addText("&Insertar")
    m1.addText("&Eliminar")
    m1.show()                                    ; mostrar
los dos items del menú
endif
endmethod
```

El código siguiente corresponde al método **action** de la ficha:

```
; estaFicha::action
method action(var eventInfo ActionEvent)
if eventInfo.isPreFilter() then
  ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
  switch
    ; cuando el usuario bloquea un registro (comienza por
cambiar el valor
    ; del campo)
    case eventInfo.id() = DataLockRecord :
```

```
      ; añadimos Deshacer y volvemos a mostrar el menú
      m1.addText("&Deshacer")
      m1.show()
    ; cuando el usuario almacena el registro (lo mueve a otro
registro)
    case eventInfo.id() = DataUnlockRecord :
      ; eliminamos Deshacer y volvemos a mostrar el menú
      m1.remove("&Deshacer")
      m1.show()
  endswitch
endif
endmethod
```

El código siguiente corresponde al método **menuAction** de la ficha:

```
; estaFicha::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Selección String
endVar

if eventInfo.isPreFilter() then
  ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
  Selección = eventInfo.menuChoice()
  switch
    case Selección = "&Insertar" :
      active.action(DataInsertRecord)   ; insertamos un nuevo
registro
    case Selección = "&Eliminar" :
      active.action(DataDeleteRecord)   ; eliminamos el registro
actual
    case choice = "&Deshacer"   :
      active.action(DataCancelRecord)   ; volver a dejar el
registro en
                ; su estado original
      m1.remove("&Deshacer")            ; eliminar el ítem
Deshacer del menú
      m1.show()                         ; volver a mostrar el
menú sin Deshacer
  endswitch
endif
endmethod
```

Vea también count

count

Método Devuelve el número de opciones de un menú.

Tipo Menu

Sintaxis **count ( )** SmallInt

Descripción Devuelve el número de opciones de un menú, incluyendo separadores, barras y cambios de columna.

count devuelve el número de opciones de un solo menú. Si se anexa un menú emergente con una opción de una barra de menús mediante **addPopUp**, **count** devuelve el número de opciones del menú emergente o el número de opciones de la barra de menús, pero no el número total de opciones de ambos menús.

Ejemplo El ejemplo siguiente construye una menú y un menú emergente, y muestra el número de opciones de cada menú. Obsérvese que **count** devuelve el número de opciones de un menú tanto si el menú se muestra como si no.

```
; RecuentoDeMenús::pushButton
method pushButton(var eventInfo Event)
var
  m Menu
  p PopUpMenu
endVar

p.addText("&Uno")
p.addBar()
p.addText("&Dos")
p.addText("&res")           ; 3 items + 1 barra de menú = 4
elementos

m.addText("&Primero")
m.addText("&Segundo")
m.addPopUp("&Tercero", p)         ; 3 items en la barra de menú

msgInfo("Items de la barra de Menú", m.count())
                ; muestra 3 (sólo cuenta; la barra de menú)
msgInfo("Items de los menús desplegables", p.count())
    ; muestra 4( sólo cuenta el menú desplegable)

endmethod
```

Vea también contains

empty

Método Borra todas las opciones de un menú.

Tipo Menu

Sintaxis **empty ( )**

Descripción Elimina todas las opciones de un menú personalizado. Utilice **empty** cuando necesite borrar un menú existente antes de regenerarlo.

Ejemplo El ejemplo siguiente utiliza dos botones que muestran menús alternativos. Ambos métodos afectan al mismo menú, declarado con la variable *MenúPrincipal* en la ventana Var de la ficha.

El código siguiente va en la ventana Var de la ficha:

```
; estaFicha::Var
Var
  MenúPrincipal Menu        ; barra de menú personalizada
endVar
```

Este es el código del método **pushButton** de *mostrarPrimerMenú*:

```
; mostrarPrimerMenú::pushButton
method pushButton(var eventInfo Event)
MenúPrincipal.empty()                   ; vaciamos el menú
MenúPrincipal.addText("&Uno")           ; Lo reconstruimos
MenúPrincipal.addText("&Dos")
MenúPrincipal.show()                    ; mostramos el menú que
hemos cambiado
endmethod
```

Este es el código del método **pushButton** de *mostrarSegundoMenú*:

```
; mostrarSegundoMenú::pushButton method pushButton(var
eventInfo Event)
MenúPrincipal.empty()                   ; vaciamos el menú
MenúPrincipal.addText("Archivo")        ; lo reconstruimos
MenúPrincipal.addText("Editar")
MenúPrincipal.show()                    ; Mostrarlo de nuevo
endmethod
```

Vea también remove

getMenuChoiceAttribute

Procedimiento Informa de los atributos de visualización de una opción de menú.

Tipo Menu

Sintaxis **getMenuChoiceAttribute (** const ***opciónMenú*** String **)** SmallInt

Descripción Devuelve un número entero que representa el atributo de visualización de la opción de menú especificada en *opciónMenú*. El valor entero representa la combinación de atributos aplicados. Use **getMenuChoiceAttribute** con **hasMenuChoiceAttribute** para determinar si un atributo de visualización específico se aplica a una opción de menú.

ObjectPAL proporciona constantes (como MenuEnabled) para los atributos de visualización; consulte MenuChoiceAttributes en el cuadro de diálogo Constantes.

Este procedimiento devuelve el atributo del menú visualizado actualmente; si no se ha creado un menú personalizado, **getMenuChoiceAttribute** opera sobre el menú estándar.

Ejemplo En este ejemplo, el método **open** de *páginaUno* construye y muestra un menú simple. El botón *obtenEstadoDelMenú* informa de si la opción de menú Time está activada o no.

El código siguiente se anexa al método **open** de *páginaUno*:

```
;páginaUno::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
  UtilDespleg   PopUpMenu
  Atributo      SmallInt
endVar

  ; construimos un menú desplegable, desactivando la opción
Hora     UtilDespleg.addText("&Hora", MenuDisabled +
MenuGrayed)
UtilDespleg.addText("&Fecha")
  ; asociamos el menú desplegable y mostramos la barra de menú
MenúPrincipal.addPopUp("&Utilidades", UtilDespleg)
MenúPrincipal.addText("&Ayuda")
MenúPrincipal.show()

endmethod
```

Este es el código del método **pushButton** de *obtenEstadoDelMenú*:

```
;obtenEstadoDelMenú::pushButton
method pushButton(var eventInfo Event)
var
Atributos SmallInt
endVar

  ; almacenar los atributos de Hora en Atributos
Atributos = getMenuChoiceAttribute("&Hora")
  ; esto muestra False porque Hora está activado
```

```
msgInfo("¿Está activado Hora?",
HasMenuChoiceAttribute(Atributos, MenuEnabled))
  ; esto muestra True porque Hora está en gris
msgInfo("¿Está Hora en gris?",
hasMenuChoiceAttribute(Atributos, MenuGrayed))

endmethod
```

Vea también getMenuChoiceAttributeById
hasMenuChoiceAttribute
setMenuChoiceAttribute
setMenuChoiceAttributeById

getMenuChoiceAttributeById

Procedimiento Informa del atributo de visualización de una opción de menú especificada por su ID de menú.

Tipo Menu

Sintaxis **getMenuChoiceAttributeById (** const ***idMenú*** SmallInt **)** SmallInt

Descripción Devuelve un número entero que representa el atributo de la opción de menú especificada en *idMenú*. El valor entero representa la combinación de atributos que se aplican. Utilice **getMenuChoiceAttributeById** con **hasMenuChoiceAttribute** para determinar si un atributo de visualización específico se aplica a una opción de menú.

ObjectPAL proporciona constantes (como MenuEnabled) para los atributos de menú; consulte MenuChoiceAttributes en el cuadro de diálogo Constantes.

Este procedimiento devuelve los atributos del menú visualizado actualmente; si no se ha creado un menú personalizado, **getMenuChoiceAttributeById** opera sobre el menú estándar.

Este procedimiento es similar a **getMenuChoiceAttribute** en el sentido de que ambos informan de los atributos de visualización de una opción de menú especificada. La diferencia es que, para **getMenuChoiceAttributeById**, se especifica el ID real de menú (un valor de SmallInt) mientras que, para **getMenuChoiceAttribute**, se especifica el nombre de menú (un valor de String). **getMenuChoiceAttributeById** es especialmente útil cuando se especifica un ID de menú como parte de la sintaxis de **addText**.

Ejemplo El ejemplo siguiente demuestra cómo puede utilizarse **getMenuChoiceAttributeById** con **hasMenuChoiceAttribute** para averiguar si una opción de menú está desactivada. En este ejemplo, el método **open** de *páginaUno* construye un pequeño menú. El método **pushButton** del botón *obtenEstadoDelMenú* informa del estado de la opción de menú Deshacer.

El código siguiente va en la ventana Var de la ficha:

```
;estaFicha::Var
Var
  m1     Menu
  p1, p2 PopUpMenu
endVar
```

El código siguiente va en la ventana Const de la ficha:

```
;estaFicha::Const
Const
  MenúDeDeshacer = UserMenu + 1
  MenúDeInsertar = UserMenu + 2
  MenúDeEliminar = UserMenu + 3
  MenúDeIndexar  = UserMenu + 4
  MenúAcercaDe   = UserMenu + 5
endConst
```

Este es el código del método **open** de la página:

```
;páginaUno::open
method open(var eventInfo Event)

p1.addText("Deshacer",   MenuDisabled + MenuGrayed,
MenuDeDeshacer+UserMenu)     p1.addText("Insertar",
MenuEnabled, MenuDeInsertar+UserMenu)
p1.addText("Eliminar", MenuEnabled, MenuDeEliminar+UserMenu)
p2.addText("Indexar",  MenuEnabled, MenuDeInsertar+UserMenu)
p2.addText("Acerca de...",  MenuEnabled, MenuAcercaDe+UserMenu)

m1.addPopUp("&Registro", p1)
m1.addPopUp("&Ayuda", p2)
m1.show()

endmethod
```

El código siguiente se anexa al método **pushButton** de *obtenEstadoDeMenú*:

```
;obtenEstadoDeMenú::pushButton
method pushButton(var eventInfo Event)

  ; almacenamos los atributos del menú Deshacer en Atributos
Atributos = getMenuChoiceAttributeById(MenúDeDeshacer)
  ; esto muestra Falso porque Deshacer está desactivado
msgInfo("¿Deshacer activado?",
hasMenuChoiceAttribute(Atributos, MenuEnabled))
  ; esto muestra Cierto porque Deshacer está en gris
msgInfo("¿Deshacer en gris?", hasMenuChoiceAttribute(Atributos,
MenuGrayed))

endmethod
```

Vea también getMenuChoiceAttribute
hasMenuChoiceAttribute
setMenuChoiceAttribute

hasMenuChoiceAttribute

Procedimiento Informa de si una opción de menú contiene un atributo de visualización dado.

Tipo Menu

Sintaxis **hasMenuChoiceAttribute (** const ***atributo*** SmallInt **,** const ***conjAtributos*** SmallInt **)** Logical

Descripción Devuelve True si *conjAtributos* contiene el atributo especificado en *atributo*; de lo contrario, devuelve False.

Utilice **hasMenuChoiceAttribute** con **getMenuChoiceAttribute** o **getMenuChoiceAttributeById** para averiguar si un atributo de visualización en particular de una opción de menú está representado en *conjAtributos*.

ObjectPAL proporciona constantes (como MenuEnabled) para *atributo*; consulte MenuChoiceAttributes en el cuadro de diálogo Constantes.

Ejemplo El ejemplo siguient demuestra cómo puede usarse **hasMenuChoiceAttribute** con **getMenuChoiceAttribute** para averiguar si un atributo en particular se aplica en el menú mostrado actualmente. El código siguiente se anexa al método **open** de *páginaUno*.

```
;páginaUno::open
method open(var eventInfo Event)
var
  m1 Menu
  p1 PopUpMenu
endVar

p1.addText("&Inserta")   ; creamos un menú simple
p1.addText("&Eliminar")
p1.addText("&Deshacer")
m1.addPopUp("&Registro", p1)
m1.show()
endmethod
```

El código siguiente se anexa al método **pushButton** del botón *IntercambiarEstadoDelMenú*:

```
;IntercambiarEstadoDelMenú::pushButton
method pushButton(var eventInfo Event)
var
  Atributos SmallInt
endVar

 ; almacenar los atributos del menú compuesto en Atributos
Atributos = getMenuChoiceAttribute("&Deshacer")

  ; esto es True si Deshacer está activado
if hasMenuChoiceAttribute(Atributos, MenuEnabled) then
  setMenuChoiceAttribute("&Deshacer", MenuDisabled +
MenuGrayed)
else
  setMenuChoiceAttribute("&Deshacer", MenuEnabled)
```

```
endif

endmethod
```

Vea también getMenuChoiceAttribute
getMenuChoiceAttributeById

remove

Método Elimina una opción de un menú.

Tipo Menu

Sintaxis **remove (** const ***opción*** String)

Descripción Borra la primera aparición de *opción* en un menú. Este método es útil para cambiar una opción de un menú sin tener que regenerarlo por completo.

Ejemplo El código mostrado en este ejemplo cambia un menú inmediatamente eliminando una opción y añadiendo otra en su lugar.

```
;cambiarMenú::pushButton
method pushButton(var eventInfo Event)
var
  MenúPrincipal Menu
endVar

 ; Primero, asumimos que el usuario está trabajando con una
Ficha.
 ; Se podría mostrar un menú así:
MenúPrincipal.addText("Archivo")
Menú.addText("Editar")
MenúPrincipal.addText("Ficha")
MenúPrincipal.show()
msgInfo("Estado", "Al cambiar el menú, tenga cuidado.")

 ; luego, suponemos que el usuario trabajará con un informe.
 ; se podría cambiar el menú así:
MenúPrincipal.remove("Ficha")
MenúPrincipal.addText("Informe")
MenúPrincipal.show()
msgInfo("Estado", "Al eliminar menús, tenga cuidado.")

 ; eliminar el menú completo, mostrando el menú estándar
removeMenu()
endmethod
```

Vea también contains
empty

removeMenu

Procedimiento Elimina un menú personalizado y restaura el menú por defecto.

Tipo Menu

Sintaxis **removeMenu ( )**

Descripción Sustituye un menú creado utilizando ObjectPAL por el menú por defecto de Paradox.

Ejemplo En el ejemplo siguiente, el método **open** de la ficha construye un menú (pero no lo muestra). El método **arrive** de *páginaUno* muestra el menú con **show**. El método **arrive** de *páginaDos* elimina el menú y muestra el menú estándar de Paradox.

El código siguiente va en la ventana Var de la ficha:

```
;estaFicha::var
Var
  m1 Menu
EndVar
```

El código siguiente se anexa al método **open** de la ficha:

```
;estaFicha::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ; el código que haya aquí se ejecuta para cada objeto de la
Ficha
  else
    ; el código que hay aquí se ejecuta sólo para la propia
Ficha

m1.addText("&Archivo")   ; construimos un menú
m1.addText("&Editar")
m1.addText("&Ficha")

endif

endmethod
```

El código siguiente se anexa al método **arrive** de *pageOne*:

```
;páginaUno::arrive
 method arrive(var eventInfo MoveEvent)
m1.show()  ; muestra el menú de la aplicación
endmethod
```

El código siguiente se anexa al método **arrive** de *pageTwo*:

```
;páginaDos::arrive
method arrive(var eventInfo MoveEvent)
removeMenu()   ; eliminamos el menú de la aplicación,
               : mostrando el menú estándar
endmethod
```

Vea también empty
remove

setMenuChoiceAttribute

Procedimiento Define el atributo de visualización de una opción de menú.

Tipo Menu

Sintaxis **setMenuChoiceAttribute (** const ***opciónMenú*** String, const ***atributoMenú***
SmallInt **)**

Descripción Define como *atributoMenú* el atributo de visualización de *opciónMenú*. Este procedimiento afect al menú mostrado actualmente; si no se ha creado un menú personalizado, **setMenuChoiceAttribute** afecta al menú estándar.

ObjectPAL proporciona constantes (como MenuGrayed) para *atributoMenú*; consulte MenuChoiceAttributes en el cuadro de diálogo Constantes.

Ejemplo En el ejemplo siguiente, se cambia el atributo de la opción Deshacer, dependiendo de si hay algo que anular. Cuando el usuario realiza cambios en el registro, la opción Deshacer se convierte en seleccionable. Después de almacenar los cambios, Deshacer deja de estar disponible.

El código siguiente va en la ventana Var de la ficha:

```
;estaFicha::var
Var
  m1 Menu
  p1 PopUpMenu
endVar
```

El código siguiente corresponde al método **open** de la ficha:

```
;estaFicha::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha

    ; creamos un menú y lo mostramos
    p1.addText("&Deshacer", MenuDisabled + MenuGrayed)
    p1.addText("&Inserta")
    p1.addText("&Eliminar")
    m1.addPopUp("&Registro", p1)
    m1.show()

endif

endmethod
```

Este es el código del método **action** de la ficha:

```
;estaFicha::action
method action(var eventInfo ActionEvent)
```

```
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha

    switch
         : cuando el usuario bloquea un registro (comienza por
         : cambiar el valor del campo)
      case eventInfo.id() = DataLockRecord :
         ; activamos el ítem Deshacer
        setMenuChoiceAttribute("&Deshacer", MenuEnabled)

         ; cuando el usuario almacena el registro (lo mueve a
otro registro)
      case eventInfo.id() = DataUnlockRecord :
         ; desactivamos y ponemos en gris el ítem Deshacer
        setMenuChoiceAttribute("&Deshacer", MenuDisabled +
MenuGrayed)
    endswitch

  else
    ; el código fuente que haya aquí sólo se ejecuta para la
propia Ficha
  endif

endmethod
```

Este es el código del método **menuAction** de la ficha:

```
;estaFicha::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Selección String
endVar

if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    Selección = eventInfo.menuChoice()
    switch
      case Selección = "&Insertar" :
        active.action(DataInsertRecord)  ; insertamos un nuevo
registro
      case Selección = "&Eliminar" :
        active.action(DataDeleteRecord)  ; eliminamos el
registro actual
      case Selección = "&Deshacer"   :
        active.action(DataCancelRecord)  ; volvermos a dejar el
registro en
                                                  : su estado
original
        setMenuChoiceAttribute("&Deshacer", MenuDisabled +
MenuGrayed)
    endswitch
  else
```

```
    ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif

endmethod
```

Vea también getMenuChoiceAttribute
getMenuChoiceAttributeById
hasMenuChoiceAttribute
setMenuChoiceAttributeById

setMenuChoiceAttributeById

Procedimiento Define el atributo de visualización de una opción de menú.

Tipo Menu

Sintaxis **setMenuChoiceAttributeById (** const ***idMenú*** SmallInt, const ***atributoMenú*** SmallInt **)**

Descripción Define como *atributoMenú* el atributo de visualización de *idMenú*. Este procedimiento afecta al menú mostrado actualmente; si no se ha creado un menú personalizado, **setMenuChoiceAttributeById** afecta al menú estándar.

ObjectPAL ofrece constantes (como MenuGrayed) para *atributoMenú*; consulte MenuChoiceAttributes en el cuadro de diálogo Constantes.

Ejemplo En el ejemplo siguiente, se cambia el atributo de la opción Deshacer, dependiendo de si hay algo que anular. Cuando el usuario realiza cambios en el registro, la opción se hace seleccionable. Después de almacenar los cambios, Deshacer deja de estar disponible. Este ejemplo utiliza la cláusula *idMenú* de *addText* para que el código pueda referenciar a las opciones de menú por su número, no por su nombre.

El código siguiente va en la ventana Var de la ficha:

```
;estaFicha::var
Var
  m1 Menu
  p1 PopUpMenu
endVar
```

El código siguiente va en la ventana Const de la ficha:

```
;estaFicha::const
Const
  MenúDeInsertar = UserMenu + 1   ; utilizamos constantes para
         : los identificadores del menú
  MenúDeEliminar = UserMenu + 2
  MenúDeDeshacer = UserMenu + 3
<+>endConst
```

El código siguiente se anexa al método **open** de la ficha:

```
;estaFicha::open
method open(var eventInfo Event)

if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha

     ; construmos un menú y lo mostramos
     p1.addText("&Deshacer",   MenuDisabled + MenuGrayed,
MenúDeDeshacer)
```

```
     p1.addText("&Eliminar", MenuEnabled, MenúDeEliminar)
     p1.addText("&Insertar", MenuEnabled, MenúDeInsertar)
     m1.addPopUp("&Registro", p1)
     m1.show()
  endif
endmethod
```

El código siguiente se anexa al método **action** de la ficha:

```
;estaFicha::action
method action(var eventInfo ActionEvent)

if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    switch
          ; cuando el usuario bloquea un registro
          : (primero cambia el valor del campo)
      case eventInfo.id() = DataLockRecord :
          ; activamos el ítem Deshacer
        setMenuChoiceAttributeById(MenúDeDeshacer, MenuEnabled)
          ; cuando el usuario almacena el registro (lo mueve a
otro registro)
      case eventInfo.id() = DataUnlockRecord :
          ; desactivamos y oscurecemos el ítem Deshacer
        setMenuChoiceAttributeById(MenúDeDeshacer, MenuGrayed +
MenuDisabled)
    endswitch
else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha     endif

endmethod
```

El código siguiente se anexa al método **menuAction** de la ficha:

```
;estaFicha::menuAction
method menuAction(var eventInfo MenuEvent)
var
  ítemDelMenú SmallInt
endVar

if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    ítemDelMenú = eventInfo.id()
  switch
    case ítemDelMenú = MenúDeInsertar :
      active.action(DataInsertRecord)    ; insertamos un nuevo
registro
    case temDelMenú = MenúDeEliminar :
      active.action(DataDeleteRecord)    ; eliminamos el
registro actual
     case menuItem = MenúDeDeshacer :
```

```
      active.action(DataCancelRecord)    ; volvemos a dejar el
registro en su
                  : estado original
      setMenuChoiceAttributeById(MenúDeDeshacer, MenuDisabled +
MenuGrayed)
  endswitch
else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha     endif

endmethod
```

Vea también getMenuChoiceAttribute
getMenuChoiceAttributeById
hasMenuChoiceAttribute
setMenuChoiceAttribute

show

Método Muestra un menú.

Tipo Menu

Sintaxis **show ( )**

Descripción Muestra un menú.

La elección del usuario se gestiona mediante el método estándar **menuAction** y el método **menuChoice** del tipo MenuEvent. Consulte el Capítulo 7 de la *Guía del Programador ObjectPAL* para más información sobre la utilización de los menús.

Ejemplo En este ejemplo, el método **open** de una ficha construye un menú sencillo y, a continuación, lo muestra mediante **show**. El método **menuAction** de la ficha gestiona la elección del usuario en el menú. A continuación, se presenta el código anexado al método **open** de *estaFicha*.

```
;estaFicha::open
method open(var eventInfo Event)
var
  d1 PopUpMenu
  m1 Menu
endVar

if eventInfo.isPreFilter()
  then
  ; el código fuente que haya aquí se ejecuta para cada objeto
de la Ficha
  else
  ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha

    d1.addText("&Hora")            ; construímos un menú
desplegable
    d1.addText("&Fecha")
    m1.addPopUp("&Utilidades", d1) ; asociamos el menú
desplegable
       : al ítem del menú
    m1.show()                      ; mostramos el menú m1

endif

endmethod
```

El código siguiente se anexa al método **menuAction** de la ficha:

```
;estaFicha::menuAction method
menuAction(var eventInfo MenuEvent)
var
  NombreMenú String
endVar

if eventInfo.isPreFilter() then
```

```
  ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha

  NombreMenú = eventInfo.menuChoice()
  switch
    case NombreMenú = "&Hora" : msgInfo("Hora actual", time())
    case NombreMenú = "&Fecha" : msgInfo("Fecha de hoy",
date())
  endSwitch

else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha     endif

endmethod
```

Vea también addText

addArray

Método Añade elementos de una matriz en un menú emergente.

Tipo PopUpMenu

Sintaxis **addArray (** const ***elementos*** Array[] String **)**

Descripción Añade *elementos* de una matriz en un menú emergente.

Ejemplo El código siguiente se anexa al método estándar **mouseRightUp** de un objeto de campo. Cuando el usuario hace clic con el botón derecho sobre el campo, aparece en un menú emergente una lista de tipos de pago disponibles. El código siguiente se anexa al método **mouseRightUp** de *TipoDePago*.

```
; TipoDePago::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  items  Array[4] String
  d1     PopUpMenu          ;  addArray se llama para esta
variable
  Selección String
endVar

disableDefault              ; no muestra el menú por defecto

items[1] = "Visa"
items[2] = "Red 6000"
items[3] = "Talón"
items[4] = "Efectivo"

d1.addArray(items)          ; añadimos los items de la matriz a
d1
Selección = d1.show()       ; mostramos el menú, recordando la
selección
if not Selección.isBlank() then
  self.value = Selección
endif

endmethod
```

Vea también addBar
addBreak
addSeparator
addStaticText
addText

addBar

Método Añade una barra vertical en un menú emergente.

Tipo PopUpMenu

Sintaxis **addBar ( )**

Descripción Añade una barra vertical en un menú emergente. El método **addBar** marca el comienzo de una nueva columna de opciones e inserta una barra vertical inmediatamente antes de la nueva columna. **addBar** es el equivalente vertical de **addSeparator**.

Ejemplo El código siguiente muestra un menú emergente con dos columnas de opciones. Las dos primeras opciones se muestra en la columna izquierda. Las demás opciones se muestran en la columna derecha. Este código se anexa al método **mouseRightUp** de un campo.

```
; campoNavegante::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  DesplegNaveg        PopUpMenu         ; para mostrar el menú
desplegable
  naveganteSelección String             ; almacena la selección
del menú
endVar

disableDefault                            ; no muestra el menú
estándar
                                          ; para el campo
DesplegNaveg.addText("Registro Anterior")  ; menú a la
izquierda
DesplegNaveg.addText("First record")
DesplegNaveg.addBar()                     ; añade la barra
vertical
DesplegNaveg.addText("Registro Siguiente") ; menú a la derecha
DesplegNaveg.addText("Ultimo registro")

Selección = DesplegNaveg.show()           ; llama al menú
; ...
; proceso de la selección
; ...

endmethod
```

Vea también addBreak
addSeparator

addBreak

Método Comienza una nueva columna en un menú emergente.

Tipo PopUpMenu

Sintaxis **addBreak ( )**

Descripción Comienza una nueva columna en un menú emergente. La primera opción añadida después de la llamada a **addBreak** se muestra en la parte superior de una columna situada a la derecha de la columna anterior, y las opciones sucesivas se muestra bajo ella. El método **addBreak** se comporta como **addBar**, en el sentido de que marca el comienzo de una nueva columna de opciones. Sin embargo, **addBreak** no crea una barra vertical entre las columnas. **addBreak** no crea un menú en cascada; para ello, utilice **addPopUp**.

Ejemplo El código siguiente muestra un menú emergente con nueve opciones en tres columnas verticales. Este código se anexa al método **pushButton** de *Dónde*.

```
; Dónde::pushButton
method pushButton(var eventInfo Event)
var
  DesplegNaveg PopUpMenu        ; un menú desplegable de
selecciones
  Selección   String            ; selección
endVar

navPopUp.addText("Inicio")      ; menú a la izquierda
navPopUp.addText("Izquierda")
navPopUp.addText("Fin")

navPopUp.addBreak()             ; inicia la segunda columna
navPopUp.addText("Arriba")
navPopUp.addText("Centro")
navPopUp.addText("Abajo")

navPopUp.addBreak()             ; inicia la tercera columna
navPopUp.addText("RePág")       ; menú a la derecha
navPopUp.addText("Derecha")

navPopUp.addText("AvPág")

Selección = DesplegNaveg.show() ; llama al menú

; ... proceso de la selección

endmethod
```

Vea también addBar
addSeparator

addPopUp

Método Añade un menú emergente en la estructura.

Tipo PopUpMenu

Sintaxis **addPopUp (** const ***nombreMenú*** String, const ***menúEmergenteEnCascada*** PopUpMenu **)**

Descripción Añade *nombreMenú* y *menúEmergenteEnCascada* en la estructura actual de menú emergente, creando un menú en cascada. *nombreMenú* se muestra como una opción del menú emergente original, y la primera opción de *menúEmergenteEnCascada* se muestra junto a ella. Las opciones sucesivas de *menúEmergenteEnCascada* se muestran en una columna por debajo de la primera opción.

Ejemplo Este ejemplo emplea **addPopUp** para anexar un menú en cascada a una opción de la barra de menús (un menú del tipo Menu). En este ejemplo, el código anexado al método estándar **open** de *estaPágina* crea y muestra la estructura de menús. El código anexado a **menuAction** de *estaPágina* gestiona la selección del usuario, porque los menús emergentes están anexados a una opción de la barra de menús.

El código siguiente se anexa al método **open** de *estaPágina*:

```
; estaPágina::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
  subMenu1, subMenu2 PopUpMenu
endVar

  ; creamos el segundo nivel de submenús
subMenu2.addText("&Hora")
subMenu2.addText("&Fecha")

  ; añadimos el segundo nivel al primer nivel
subMenu1.addPopUp("&Utilidades", subMenu2)
  ; añadimos el primer nivel a la barra de menú
MenúPrincipal.addPopUp("&Archivo", subMenu1)

  ; mostramos la barra de menú
MenúPrincipal.show()

endmethod
```

El ejemplo siguiente se anexa al método **menuAction** de *estaPágina:*

```
; estaPágina::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Selección String
endVar

Selección = eventInfo.menuChoice()
switch
  case Selección = "&Hora"  : msgInfo("Hora actual", time())
```

```
  case Selección = "&Fecha" : msgInfo("Fecha de hoy", date())
endSwitch

endmethod
```

El ejemplo siguiente emplea **addPopUp** para crear un menú emergente en cascada. Esta estructura de menús no se anexa a una opción de la barra de menús. El código que sigue a la llamada a **show** realiza acciones en función de la selección del usuario; el método estándar **menuAction** no se emplea.

El código siguiente se anexa al método **mouseRightUp** de *páginaDos*:

```
; páginaDos::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  d1, d2, d3 PopUpMenu
  Selección String
endVar

disableDefault                ; no muestra el menú desplegable
estándar

d2.addText("&Hora")           ; construimos los submenús d2 y
d3
d2.addText("&Fecha")
d3.addText("&Rojo")
d3.addText("&Verde")
d3.addText("&Azul")

d1.addPopUp("&Utilidades", d2) ; creamos el item Utilidades y
lo asociamos con d2
d1.addPopUp("&Colores", d3)    ; creamos el item Colores y lo
asociamos con d3

Selección = d1.show()          ; mostramos el menú y
almacenamos la selección en
                               ; Selección

switch                         ; ahora tamamos una acción
basándonos en la
                               ; seleccion
  case Selección = "&Rojo"  : self.color = Rojo
  case Selección = "&Verde" : self.color = Verde
  case Selección = "&Azul"  : self.color = Azul
  case Selección = "&Hora"  : msgInfo("Hora Actual",time())
  case Selección = "&Fecha" : msgInfo("Fecha de hoy", date())
endSwitch

endmethod
```

Vea también addBreak

addSeparator

Método Añade una barra horizontal en un menú emergente.

Tipo PopUpMenu

Sintaxis **addSeparator ( )**

Descripción Añade una barra horizontal para separar grupos de opciones en un menú emergente. **addSeparator** sirve para agrupar comandos similares dentro de un menú.

Ejemplo El ejemplo siguiente utiliza **addSeparator** para agrupar comandos del menú emergente. El código siguiente se anexa al método estándar **open** de *estaPágina*:

```
; estaPágina::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
  subMenu1, Desplegable PopUpMenu
endVar
Desplegable.addText("&Rojo")
Desplegable.addText("&Azul")
Desplegable.addText("&Blanco")
subMenu1.addText("&Hora")
subMenu1.addText("&Fecha")
subMenu1.addSeparator()
subMenu1.addPopUp("&Colores para la página", Desplegable)
subMenu1.addSeparator()
subMenu1.addText("&Acerca de")

MenúPrincipal.addPopUp("&Utilidades", subMenu1)
MenúPrincipal.show()
endmethod
```

El código siguiente se anexa al método estándar **menuAction** de *estaPágina*:

```
; estaPágina::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Selección String
endVar
Selección = eventInfo.menuChoice()
switch
  case Selección = "&Rojo"      : self.color = Rojo
  case Selección = "&Azul"      : self.color = Azul
  case Selección = "&Blanco"    : self.color = Blanco
  case Selección = "&Hora"      : msgInfo("Hora Actual",
time())
  case Selección = "&Fecha"     : msgInfo("Fecha de hoy",
date())
  case Selección = "&Acerca de" :
eventInfo.setId(MenuHelpAbout)
endSwitch
endmethod
```

Vea también addBar
addBreak

addStaticText

Método Añade una cadena de texto no seleccionable en un menú emergente.

Tipo PopUpMenu

Sintaxis **addStaticText (** const ***opción*** String **)**

Descripción Añade una opción en un menú emergente como texto no seleccionable. El texto estático se utiliza generalmente como título (primera opción) de un menú emergente.

Ejemplo El código siguiente se anexa al método estándar **mouseRightUp** de un objeto de campo. Cuando el usuario hace un clic con el botón derecho del ratón sobre el campo, aparece una lista de tipos de pago disponibles en un menú emergente. Este ejemplo muestra la primera opción como texto estático. El código siguiente se anexa al método **mouseRightUp** de *TipoDePago*.

```
; TipoDePago::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  items  Array[4] String
  d1     PopUpMenu          ;  addArray se llama para esta
variable
  Selección String
endVar

disableDefault              ; no mostramos el menú estándar

items[1] = "Visa"
items[2] = "Red 6000"
items[3] = "Talón"
items[4] = "Efectivo"
                            ; mostramos el primer item como
texto estático
d1.addStaticText("Método de pago")
d1.addSeparator()           ; añadimos un separador horizontal
d1.addArray(items)          ; añadimos la matriz items a d1
Selección = d1.show()       ; mostramos el menú, recordando la
selección
if not Selección.isBlank() then
  self.value = Selección
endif

endmethod
```

Vea también addText

addText

Método Añade una cadena de texto seleccionable en un menú emergente.

Tipo PopUpMenu

Sintaxis **1. addText (** const ***nombreMenú*** String **)**

2. addText (const ***nombreMenú*** String, const ***atributo*** SmallInt)

3. addText (const ***nombreMenú*** String, const ***atributo*** SmallInt, const ***id*** SmallInt **)**

Descripción Añade *nombreMenú* en un menú emergente como opción seleccionable. Es posible emplear *atributo* para predefinir el atributo de visualización de *nombreMenú*. ObjectPAL proporciona constantes (como MenuDisabled) para *atributo*; consulte MenuChoiceAttributes en el cuadro de diálogo Constantes.

La tercera forma de la sintaxis de **addText** sólo se utiliza cuando el menú emergente se anexa a un objeto Menu. Es posible especificar un número *id* (del tipo SmallInt) para identificar el menú por su número, en lugar de por su *nombreMenú*. Entonces, en el método estándar **menuAction**, se emplea el número *id* para determinar qué menú elige el usuario.

Cuando especifique un *id* de menú, debería utilizar la constante estándar UserMenu como constante básica y, a continuación, añadir su propio número. Por ejemplo, la línea siguiente añade "Archivo" en el PopUpMenu *Desplegable* y especifica un número *id* para esa opción de menú:

```
Desplegable.addText ("Archivo", MenuEnabled,
    UserMenu + 1)
```

Para más información sobre las constantes definidas por el usuario, consulte el Capítulo 7 de la *Guía del Programador ObjectPAL*. Consulte también la sección "Menu" en este capítulo para más información sobre los objetos Menu y la cláusula *id*.

Es posible utilizar un ampersand (&) en una opción para que el usuario pueda seleccionarla mediante el teclado. Por ejemplo, la opción "&Archivo" se mostraría como Archivo y el usuario podría elegirla pulsando **A**. Recuerde incluir el ampersand al comprobar la elección del usuario. En el ejemplo anterior, el valor devuelto sería "&Archivo", no "Archivo".

Además, es posible utilizar "\t" para incluir un tabulador entre una opción y su tecla rápida. Por ejemplo, la opción "&Editar datos\tF9" se mostraría con "Editar datos" alineado a la izquierda y "F9" alineado a la derecha. El valor de cadena devuelto en este caso sería "&Editar datos\tF9".

Ejemplo El ejemplo siguiente demuestra algunas variaciones de la sintaxis de **addText**.

En el primer ejemplo, supóngase que una ficha tiene un campo no asociado llamado *CampoDePago*. Cuando el usuario hace clic con el botón derecho del ratón sobre el campo, aparece en un menú emergente una lista de métodos de pago disponibles. Entonces, el usuario puede elegir en la lista para insertar el valor elegido en el campo o pulsar *Esc* para cancelar. El código siguiente va en la ventana Var de *CampoDePago*:

```
; campoDePago::var
var
  Desplegable PopUpMenu
  Selección   String
endVar
```

El código siguiente se anexa al método **open** de *CampoDePago*. Cuando se abre el campo por primera vez, este código añade cuatro opciones en el PopUpMenu *Desplegable*. Este código no muestra el menú emergente; sólo prepara el menú para su visualización posterior.

```
; campoDePago::open
method open(var eventInfo Event)

Desplegable.addText("Visa")
Desplegable.addText("Red 6000")
Desplegable.addText("Talón")
Desplegable.addText("Efectivo")

endmethod
```

El código siguiente se anexa al método estándar **mouseRightUp** de *CampoDePago*. Cuando el usuario hace clic con el botón derecho sobre el campo, este método muestra el menú con **show** e inserta la elección del usuario en el campo no asociado.

```
; campoDePago::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)

disableDefault                 ; no mostramos el menú
desplegable estándar

Selección = payPopUp.show()    ; mostramos el menú, almacenando
la selección
                               ; en Selección
if not isBlank(Selección) then ; si el usuario no pulsa Esc
  self.value = Selección       ; insertamos Selección en un
campo no asociado
endif
endmethod
```

El ejemplo siguiente demuestra cómo puede utilizarse la cláusula *id* en los menús emergentes anexados a un objeto Menu. Este código establece constantes definidas por el usuario para que las asignaciones de *id* de menu se recuerden más fácilmente. El código siguiente va en la ventana Var de *estaPágina*.

```
; estaPágina::const
Const
   Rojo     = 1  ; definimos valores constantes para los
identificadores del menú
   Azul     = 2
   Blanco   = 3
  Hora     = 4
  Fecha    = 5
  AcercaDe = 6
endConst
```

El código siguiente se anexa al método **open** de *estaPágina*. Para controlar los atributos de visualización del menú, este código emplea constantes estándar como MenuEnabled. Para identificar cada opción de menú por su número, el código utiliza las constantes definidas en la ventana Const de *estaPágina* (*Rojo*, *Azul*, etc.).

```
; estePágina::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
  subMenu1, Desplegable PopUpMenu
endVar

  ; añadimos el texto a menús desplegables y utilizamos las
constantes personalizadas
Desplegable.addText("&Rojo", MenuEnabled, Rojo+UserMenu)
Desplegable.addText("&Azul", MenuEnabled, Azul+UserMenu)
Desplegable.addText("&Blanco", MenuEnabled, Blanco+UserMenu)

subMenu1.addText("&Hora", MenuEnabled, Hora+UserMenu)
subMenu1.addText("&Fecha", MenuEnabled, Fecha+UserMenu)
subMenu1.addSeparator()
subMenu1.addPopUp("&Colores para la página", Desplegable)
subMenu1.addSeparator()
subMenu1.addText("&Acerca de", MenuEnabled, AcercaDe+UserMenu)

  ; asociamos los menús desplegables al menú principal y
mostramos la barra de menú
MenúPrincipal.addPopUp("&Utilidades", subMenu1)
MenúPrincipal.show()
endmethod
```

El código siguiente se anexa al método **menuAction** de *estaPágina*. Este ejemplo evalúa las selecciones en el menú por su número ID, y no por el nombre especificado en *nombreMenú*.

```
; estaPágina::menuAction
method menuAction(var eventInfo MenuEvent)
var
  IdMenú SmallInt
endVar

IdMenú = eventInfo.id()   ; almacenamos el número de
identificador del menú en IdMenú

switch
  case IdMenú = Rojo     : self.color = Rojo
  case IdMenú = Azul     : self.color = Azul
  case IdMenú = Blanco   : self.color = Blanco
  case IdMenú = Hora     : msgInfo("Hora actual", time())
  case IdMenú = Fecha    : msgInfo("Fecha de hoy",date())
  case IdMenú = AcercaDe : eventInfo.setId(MenuHelpAbout)
endSwitch

endmethod
```

Vea también addStaticText

show

Método Muestra un menú emergente y devuelve la opción seleccionada.

Tipo PopUpMenu

Sintaxis **show ([** const ***twipsX*** SmallInt, const ***twipsY*** SmallInt]) String

Descripción Muestra un menú emergente y devuelve la opción seleccionada. Si el usuario pulsa *Esc* en lugar de realizar una selección, el valor devuelto es una cadena de longitud cero. Los argumentos optativos *twipsX* y *twipsY* especifican las coordenadas, en twips, del ángulo superior izquierdo del menú emergente. Si no se especifican, se utilizan las coordenadas *x* e *y* del puntero.

Ejemplo En el ejemplo siguiente, supóngase que una ficha tiene un campo no asociado llamado *CampoDePago*. Cuando el usuario hace clic con el botón derecho del ratón sobre el campo, aparece una lista de tipos de pago en un menú emergente. El usuario puede elegir en la lista para insertar el valor elegido en el campo o pulsar *Esc* para cancelar. El código siguiente va en la ventana Var de *CampoDePago*:

```
; campoDePago::var
var
  Desplegable PopUpMenu
  Selección   String
endVar
```

El código siguiente se anexa al método **open** de *CampoDePago*. Cuando se abre el campo por primera vez, este código añade cuatro opciones en el PopUpMenu *Desplegable*. Este código no muestra el menú emergente; sólo lo prepara para su visualización posterior.

```
; campoDePago::open
method open(var eventInfo Event)

Desplegable.addText("Visa")
Desplegable.addText("Red 6000")
Desplegable.addText("Talón")
Desplegable.addText("Efectivo")

endmethod
```

El código siguiente se anexa al método estándar **mouseRightUp** de *CampoDePago*. Cuando el usuario hace clic con el botón derecho del ratón sobre el camnpo, este método muestra el menú con **show** e inserta la elección del usuario en el campo no asociado.

```
; campoDePago::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)

disableDefault                  ; no mostramos el menú
desplegable por defecto

Selección = Desplegable.show() ; mostramos el menú, y
almacenamos la selección
                                ; en Selección
```

```
if not isBlank(Selección) then ; si el usuario no pulsa Esc
  self.value = Selección       ; insertamos Selección en un
campo no asociado
endif
endmethod
```

Vea también Menu::empty

switchMenu

Palabra clave Crea y muestra un menú emergente, y gestiona la elección en el menú.

Tipo PopUpMenu

Sintaxis **switchMenu** *listaCase* **[otherwise :** *Sentencias* **] endSwitchMenu**

listaCase es cualquier número de sentencias con la forma siguiente:

CASE *opciónMenú* : *Sentencias*

Descripción **switchMenu** utiliza los valores del argumento *opciónMenú* de cada *listaCase* para crear y mostrar un menú emergente. Las *Sentencias* que siguen a cada *opciónMenú* especifican cómo gestionar la elección de cada opción de menú. La cláusula optativa *otherwise* especifica lo que debe hacerse si el usuario cierra el menú sin realizar una elección (por ejemplo, pulsando *Esc*).

Ejemplo El ejemplo siguiente utiliza **switchMenu** para crear, mostrar y procesar la elección en un menú emergente. En la ventana de mensajes de la línea de estado, se muestra una cadena que describe la selección.

```
; botónDeAcción::pushButton
method pushButton(var eventInfo Event)
switchMenu
  case "Añadir"   : message("Se ha elegido Añadir.")
  case "Editar"   : message("Se ha elegido Editar.")
  case "Eliminar" : message("Se ha elegido Eliminar.")
  otherwise       : message("No se ha elegido nada del menú.")
endSwitchMenu
endmethod
```

Vea también addText
show

action

Principiante

Método Realiza una acción especificada.

Tipo UIObject

Sintaxis **action (** const ***idAcción*** SmallInt **)** Logical

Descripción Especifica un *idAcción* que se realizará en respuesta a un suceso. ObjectPAL proporciona una constante para *idAcción*; consulte una de las categorías que comienzan con Action (por ejemplo, ActionDataCommands) en el cuadro de diálogo Constantes.

Ejemplo El código de este ejemplo se anexa al método **mouseUp** de un botón y realiza lo siguiente: si el usuario pulsa y mantiene pulsada la tecla *Mayús* y hace clic sobre el botón, el puntero se desplaza al siguiente grupo de registros. Si hace clic sobre el botón sin pulsar *Mayús*, el puntero se desplaza al siguiente registro.

Las constantes de acción DataFastForward y DataNextRecord se comportan como los botones Grupo de registros siguiente y Registro siguiente de la barra rápida. Supóngase que *CLIENTES* referencia a un marco de tabla de la ficha y que *siguienteRegistroRápido* es un botón de la misma ficha. El botón *siguienteRegistroRápido* no está en la misma jerarquía de contenedores que *CLIENTES*, por lo que la acción no se lanzará a *CLIENTES* automáticamente. Así, la acción debe enviarse al objeto *CLIENTES* explícitamente.

```
; siguienteRegistroRápido::mouseUp
method mouseUp(var eventInfo MouseEvent)
; si la tecla SHIFT está pulsada, va al siguiente conjunto de
registros
; en otro caso va al siguiente registro
if eventInfo.isShiftKeyDown() then
 CLIENTES.action(DataFastForward)
else
  CLIENTES.action(DataNextRecord)
endif
endmethod
```

Vea también menuAction
ActionEvent**::**id
ActionEvent**::**setId

atFirst

Método Informa de si el puntero está en el primer registro de una tabla.

Tipo UIObject

Sintaxis **atFirst ( )** Logical

Descripción Devuelve True si el puntero se halla en el primer registro de una tabla; si no es así, devuelve False. **atFirst** respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro.

Ejemplo En el ejemplo siguiente, supóngase que *CLIENTES* referencia a un marco de tabla de la ficha y que *botónDeIrAlPrimero* es un botón de la misma ficha. El método comprueba la posición del puntero. Si éste no se halla en el primer registro de *CLIENTES*, el método lo desplaza a ese registro.

```
; botónDeIrAlPrimero::pushButton
method pushButton(var eventInfo Event)
if NOT CLIENTES.atFirst() then
  CLIENTES.home()
  ; tiene el mismo efecto que : CLIENTES.action(DataBegin)
endif
endmethod
```

Vea también atLast

atLast

Método Informa de si el puntero está en el último registro de una tabla.

Tipo UIObject

Sintaxis **atLast ( )** Logical

Descripción Devuelve True si el puntero se halla en el último registro de una tabla; en caso contrario, devuelve False. **atLast** respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro.

Ejemplo En el ejemplo siguiente, supóngase que *CLIENTES* referencia a un marco de tabla de la ficha y que *botónDeIrAlUltimo* es un botón de la misma ficha. El método comprueba la posición del puntero. Si éste no se halla en el último registro de *CLIENTES*, el método lo desplaza a ese registro.

```
; botónDeIrAlUltimo::pushButton
method pushButton(var eventInfo Event)
if NOT CLIENTES.atLast() then
  CLIENTES.end()
; tiene el mismo efecto que : CLIENTES.action(DataEnd)
endif
endmethod
```

Vea también atFirst

attach

Método Asocia una variable UIObject con un objeto de diseño especificado.

Tipo UIObject

Sintaxis **1. attach ( )** Logical**2. attach (** const ***objeto*** UIObject **)** Logical

3. attach (const ***ficha*** Form **[,** const ***nombreObjeto*** String **])** Logical

4. attach (const ***ficha*** Report **[,** const ***nombreObjeto*** String **])** Logical

Descripción Asocia una variable UIObject con el objeto que la activa (sintaxis 1), con otro UIObject (*objeto* en la sintaxis 2), con un Form (*ficha* en la sintaxis 3) o con un UIObject de otra ficha (*nombreObjeto* en la sintaxis 3). También es posible emplear **attach** para asociar un UIObject con un informe abierto o con un objeto de un informe abierto (sintaxis 4).

Nota: Algunos de los métodos del tipo UIObject pueden emplearse con las fichas, pero sólo si se anexa una variable UIObject con la ficha. La sintaxis 3 del método **attach** permite anexar una variable UIObject con una ficha de forma que sea posible acceder a estos métodos. Por ejemplo, para enviar un suceso **mouseUp** al método estándar **mouseUp** a nivel de ficha de otra ficha, es necesario anexar un UIObject (una variable Form no funcionará) a una ficha abierta.

Ejemplo El ejemplo siguiente muestra las tres formas de la sintaxis. Con la sintaxis 1, el método anexa la variable *Caja* al objeto actual (self) y cambia su color. Con la sintaxis 2, el método anexa *Caja* a otro objeto y cambia su color mediante *Caja*. Un segundo ejemplo de la sintaxis 2 abre otra ficha, anexa *Caja* a un cuadro de la ficha y cambia el color del objeto de la otra ficha mediante *Caja*.

En este ejemplo, supóngase que la ficha actual contiene dos cuadros, *estaCaja* y *aquellaCaja*. El método se anexa a *estaCaja*. La ficha secundaria contiene un cuadro, llamado *otraCaja*.

```
; estaCaja::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  Caja,
  Ficha UIObject
  OtraFicha Form
endVar

; sintaxis 1
Caja.attach()                    ; asocia Caja a estaCaja
Caja.color = DarkMagenta

; sintaxis 2
Caja.attach(AquellaCaja)         ; asocia Caja a aquellaCaja
Caja.color = Magenta

; asumimos que la Ficha UIAsocia.fsl existe y que tiene
; un objeto llamado otraCaja
if OtraFicha.open("UIAsocia.fsl") then
  Caja.attach(OtraFicha.OtraCaja)
  Caja.color = DarkBlue
```

```
  sleep(2000)
  OtraFicha.close()
endif

; sintaxis 3
if OtraFicha.open("UIAsocia.fsl") then
  ; nótese que el nombre del objeto viene dado como una cadena
  Caja.attach(OtraFicha, "otraCaja")
  Caja.color = LightBlue
  sleep(2000)
  OtraFicha.close()
endif

endmethod
```

Vea también moveTo

broadcastAction

Método	Transmite una acción a un objeto.
Tipo	UIObject
Sintaxis	**broadCastAction (**const ***idAcción*** SmallInt)
Descripción	Notifica que se ha producido un suceso a un objeto y a todos los objetos que dicho objeto contiene.
Vea también	action menuAction

cancelEdit

Principiante

Método Cancela los cambios en un registro sin salir del modo editar.

Tipo UIObject

Sintaxis **cancelEdit ( )** Logical

Descripción Deja una tabla en modo editar pero cancela los cambios en el registro actual. Devuelve True si es satisfactorio; en caso contrario, devuelve False. Para cancelar los cambios en el registro actual, es necesario utilizar **cancelEdit** antes de desplazar el puntero desde el registro actual, puesto que una vez que se desplaza el puntero, se consignan los cambios en el registro.

cancelEdit tiene el mismo efecto que la constante de acción DataCancelEdit, por lo que las sentencias siguientes son equivalentes:

```
obj.cancelEdit()
obj.action(DataCancelEdit)
```

Ejemplo El método siguiente anexa una variable UIObject, *NoCambiar*, a un marco de tabla, *CLIENTES* (desde ese momento, se emplea *NoCambiar* como gestor del marco de tabla). El método busca un valor en la tabla *CLIENTES* y, si lo encuentra, lo cambia. Antes de salir del registro, el cambio se cancela con el método **cancelEdit**. En este ejemplo, supóngase que se tiene una página en la ficha, llamada *páginaUno*, un marco de tabla anexado a la tabla *CLIENTES* y un botón llamado *botónDeCancelarEdición*.

```
; botónDeCancelarEdición::pushButton
method pushButton(var eventInfo Event)
var
  NoCambiar UIObject
endVar
NoCambiar.attach()
NoCambiar.attach(PáginaUno.CLIENTES)
NoCambiar.edit()
if NoCambiar.locate("Nombre", "Unisco") then
  NoCambiar."Nombre" = "Carmen"         ; nos preparamos para
cambiar el registro
  msgInfo("NoCambiar.'Nombre'", NoCambiar."Nombre".value)
  NoCambiar.cancelEdit()                ; ¡asegurar esta orden!
         ; el registro no se ha cambiado
endif
NoCambiar.endEdit()                     ; finalizamos el modo de
Edición

endmethod
```

Vea también currRecord
edit
endEdit

convertPointWithRespectTo

Método Cambia el marco de referencia para el cálculo de las coordenadas de un punto.

Tipo UIObject

Sintaxis **convertPointWithRespectTo (** const ***otroUIObject*** UIObject, const ***puntoAnterior Point, var puntoConvertido*** Point)

Descripción Cambia el marco de referencia para el cálculo de la posición de un punto. Normalmente, las coordenadas se calculan respecto al ángulo superior izquierdo del contenedor del objeto (o del marco del contenedor, en el caso de una elipse). Sin embargo, este método calcula la posición de un punto respecto al ángulo superior izquierdo del objeto especificado en *otroUIObject*.

Ejemplo Este ejemplo obtiene y muestra la posición de un objeto llamado *CajaInterior*, que está contenido en *CajaExterior* y se halla en una página llamada *páginaUno*. En primer lugar, se obtiene y se muestra la posición de *CajaExterior* respecto al ángulo superior izquierdo de la página. A continuación, se toma la posición de *CajaInterior* respecto al ángulo superior izquierdo de *CajaExterior*. Por último, la posición de *CajaInterior* se convierte con respecto a la página, para que pueda apreciarse la distancia de *CajaInterior* con los bordes superior e izquierdo de la página.

```
; alinearCajaInterior::pushButton
method pushButton(var eventInfo Event)
var
  PosiciónInterior,
  PosiciónExterior,
  PosiciónTransformada Point
  x, y, w, h           LongInt
  ObjetoInterior       UIObject
endVar


CajaExterior.getPosition(x, y, w, h)
PosiciónExterior = point(x, y)                ; transformar x e
y desde
PosiciónExterior.view("Posición de la Caja Exterior") ;
CajaExterior a un punto
; CajaExterior.CajaInterior.getPosition(x, y, w, h)
ObjetoInterior.attach("CajaExterior.CajaInterior")
ObjetoInterior.getPosition(x, y, w, h)
PosiciónInterior = point(x, y)
PosiciónInterior.view("Posición de la Caja Interior sin
transformar")
; ¿A qué distancia está PosiciónInterior de la esquina superior
izquierda
; de la página?
ObjetoInterior.convertPointWithRespectTo(PáginaUno,
PosiciónInterior, PosiciónTransformada)
PosiciónTransformada.view("Posición de la Caja Interior
transformada")

endmethod
```

Vea también Point

copyFromArray

Método Copia datos de una matriz en un registro de una tabla.

Tipo UIObject

Sintaxis **copyFromArray (** const ***mat*** Array[] AnyType) Logical

Descripción Copia los elementos de la matriz mat en un UIObject (normalmente, un marco de tabla o un objeto multirregistro). El primer elemento de la matriz se copia en el primer campo de la tabla, el segundo elemento en el segundo campo, y así sucesivamente hasta que se termine la matriz o el registro esté lleno.

El método falla si se intenta copiar un elemento no asignado de una matriz o si las estructuras no coinciden (es imposible que suceda esto si la matriz se creó mediante **copyToArray**, que asigna un valor vacío si un campo está vacío). Además, el método falla si la ficha no está en modo editar. Si hay más elementos en la matriz que campos en el registro, los elementos sobrantes no se copian.

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla llamado *NOMCLIEN*. La tabla *NOMCLIEN* tiene tres campos: Apellidos, A20; Nombre, A20; Inicial, A1. Este método comienza editando *NOMCLIEN*, crea una matriz con tres elementos, crea un nuevo registro en *NOMCLIEN* y copia los datos de la matriz en el registro.

```
; crearRegistro::pushButton
method pushButton(var eventInfo Event)
var
  matrizNombres Array[3] String
endvar


NOMCLIEN.edit()                          ; activamos el modo de
Edición     matrizNombres[1] = "Recibidor"          ; rellenamos
la matriz con el registro
               ; a insertar
matrizNombres[2] = "Roberto"
matrizNombres[3] = "A"
NOMCLIEN.action(DataInsertRecord)             ; primero
insertamos un registro en blanco
NOMCLIEN.copyFromArray(matrizNombres)   ; luego copiamos la
matriz en el nuevo
               ; registro
NOMCLIEN.endEdit()
endmethod
```

Vea también copyToArray

copyToArray

Método Copia datos de un registro en una matriz.

Tipo UIObject

Sintaxis **copyToArray (** var ***mat*** Array **[]** AnyType) Logical

Descripción Copia los campos del registro actual de un UIObject (normalmente un marco de tabla o un objeto multirregistro) en los elementos de la matriz especificada en *mat*. Es necesario declarar que la matriz sea del tipo AnyType o de un tipo que coincida con cada campo de la tabla. Si la matriz es redimensionable, crece automáticamente para albergar el número de campos del registro. Si no lo es, sólo recibe tantos campos como pueda, desechándose el resto.

El valor del primer campo se copia en el primer elemento de la matriz, el valor del segundo campo en el segundo elemento, y así sucesivamente. El tamaño de la matriz es igual al número de campos del registro. El campo de número de registro y cualquier campo de sólo visualización o calculado que aparezca en una vista de ficha de la tabla no se copian a la matriz.

Ejemplo El ejemplo siguiente supone que hay dos marcos de tabla en una ficha, llamados *CLIENTES* y *ARCCLIEN*, y un botón, llamado *botónDeArchivar*. La ficha en sí se denomina *estaFicha*. Cuando se pulsa *botónDeArchivar*, se desplaza el registro actual de *CLIENTES* a *ARCCLIEN*.

En primer lugar, el método comprueba la propiedad Editing de la ficha; si es False, el método activa el modo editar. Entonces, el método copia el registro actual de *CLIENTES* en la matriz *matrizNombres* e intenta borrar el registro actual. Si no es posible bloquearlo y borrarlo, el registro no se copia a la tabla destino *ARCCLIEN*. Si se logra borrar el registro, el método añade un nuevo registro vacío en la tabla destino y escribe el contenido de la matriz en el registro.

```
; botónDeArchivar::pushButton
method pushButton(var eventInfo Event)
var
  matrizNombres Array[] String
endVar

; comprobamos si la Ficha está en modo de Edición
if EsteFormato.Editing = False then     ; si no, comenzar
  CLIENTES.action(DataBeginEdit)
endif

; trasladamos el registro actual desde CLIENTES al archivo
ARCCLIEN
CLIENTES.copyToArray(matrizNombres)
matrizNombres.view()              ; echamos un vistazo a la
matriz                                      ; si el registro
no se puede bloquear, no se elimina
if CLIENTES.deleteRecord() = True then
  ; si se elimina, lo copiamos en la tabla ARCCLIEN
  ARCCLIEN.insertRecord()                    ; insertamos un
registro en blanco     ARCCLIEN.copyFromArray(matrizNombres)
    ; copiamos la matriz en el registro
```

```
                    ; en blanco
        endif

        endmethod
```

Vea también copyFromArray

create

Método Crea un objeto.

Tipo UIObject

Sintaxis **create (** const ***tipoObjeto*** SmallInt, const ***x*** LongInt,

const ***y*** LongInt, const ***w*** LongInt, const ***h*** LongInt

[, const ***contenedor*** UIObject **])**

Descripción Crea el objeto especificado en *tipoObjeto* en la posición especificada en *x* e *y*, con la anchura y altura especificadas en *w* y *h*, respectivamente (se supone que *x*, *y*, *w* y *h* se expresan en twips). El argumento optativo *contenedor* especifica un objeto contenedor para el objeto que se crea.

ObjectPAL proporciona constantes (como ButtonTool) para *tipoObjeto*; consulte UIObjectTypes en el cuadro de diálogo Constantes.

Nota: Cuando emplea **create** para crear un objeto y ejecutar una ficha o informe, el objeto es invisible. Para que sea visible, defina su propiedad Visible como True. Los objetos también pueden borrarse durante la ejecución con el método **delete**.

Ejemplo El código siguiente se anexa al método **mouseUp** de *páginaUno* en una ficha. Este ejemplo crea un cuadro, lo llama *Alfredo*, lo colorea de azul y lo define como visible. Entonces se crea una elipse en *Alfredo*, y su contenedor se define como *Alfredo*.

```
; páginaUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  ui UIObject
endvar

; creamos una caja Azul, llamada Alfredo y la hacemos visible
ui.create(BoxTool, 144, 144, 2880, 2880)
ui.Name = "Alfredo"
ui.Color = Blue       ; color Azul
ui.Visible = True
; definimos una elipse Verde dentro de Alfredo, llamada María
ui.create (EllipseTool, 288, 288, 1440, 1440, self.Alfredo)
ui.Name = "María"
ui.Color = Green      ; color Verde
ui.Visible = True
endmethod
```

Vea también delete
methodSet

currRecord

Método Lee el registro actual en la memoria intermedia de registro.

Tipo UIObject

Sintaxis **currRecord ( )** Logical

Descripción Cancela los cambios en el registro actual, actualizándolo con los datos almacenados. Los cambios que se hayan realizado en el registro no se almacenan. Si el registro está bloqueado, **currRecord** lo deja bloqueado. Devuelve True si es satisfactorio; en caso contrario, devuelve False.

currRecord tiene el mismo efecto que la constante de acción DataRefresh, por lo que las sentencias siguientes son equivalentes:

```
obj.currRecord()
obj.action(DataRefresh)
```

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla asociado con *Pedidos*.

```
; refrescarRegistro::pushButton
method pushButton (var eventInfo Event)
PEDIDOS.edit()                    ; activamos el modo de Edición
PEDIDOS.Pendiente = 321,45        ; hacer un cambio
message("Ahora mira de cerca.")
sleep (2000)
PEDIDOS.currRecord()              ; refrescar el registro desde
el disco
                                  ; cualquier cambio se pierde,
porque el registro
    ; no está bloqueado

if PEDIDOS.recordStatus("Locked") then
  msgInfo ("","El registro aún está bloqueado.")
endif
endmethod
```

Vea también cancelEdit

delete

Método Borra un objeto de una ficha.

Tipo UIObject

Sintaxis **delete ( )**

Descripción Borra un objeto de una ficha durante la ejecución.

Ejemplo En el ejemplo siguiente, supóngase que una ficha contiene un método que crea un cuadro llamado *Alfredo* y una elipse dentro de *Alfredo* llamada *María*. Estos objetos se crean en el momento de la ejecución y por tanto este método no puede referenciarlos directamente, ya que no existen aún. La técnica utilizada aquí se anexa al objeto mediante una cadena evaluada en tiempo de ejecución. Consulte el ejemplo de **create** para más detalles sobre el método **mouseUp** (de la misma ficha) que crea los objetos que se borrarán.

```
; paginaUno::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  ui   UIObject
endVar

; Alfredo y María son objetos creados por el método mouseUp
; para la PáginaUno de esta Ficha. Como se crean en tiempo de
ejecución,
; no nos podemos referir directamente a ellos como objetos del
; código fuente. Por ello, attach se utiliza para asociar la
variable ui
; a la cadena "Alfredo.María", que se evalúa en tiempo de
ejecución.
; Siempre que se llame a mouseUp antes que a mouseRightUp,
; esos objetos existirán.
if ui.attach("Alfredo.María") then
  ui.delete()
  ui.attach("Alfredo")
  ui.delete()
  {Esto haría lo mismo que las cuatro líneas anteriores,
   porque Alfredo contiene a María en tiempo de ejecución:
   ui.attach("Alfredo")
    ui.delete()
  }
endif

endmethod
```

Vea también create
methodSet

deleteRecord

Principiante

Método Borra el registro actual de la tabla.

Tipo UIObject

Sintaxis **deleteRecord ( )** Logical

Descripción Borra el registro actual de una tabla sin pedir confirmación. Devuelve True si lo logra; en caso contrario, devuelve False. Esta operación no puede anularse.

deleteRecord tiene el mismo efecto que la constante de acción DataDeleteRecord, por lo que las sentencias siguientes son equivalentes:

```
obj.deleteRecord()
obj.action(DataDeleteRecord)
```

Ejemplo EL ejemplo siguiente supone que hay dos marcos de tabla en una ficha, *CLIENTES* y *ARCCLIEN*, y un botón, llamado *botónDeArchivar*. La ficha se denomina *estaFicha*. Cuando se pulsa *botónDeArchivar*, se desplaza el registro actual de *CLIENTES* a *ARCCLIEN*.

En primer lugar, el método comprueba la propiedad Editing de la ficha; si es False, el método activa el modo editar. Entonces, el método copia el registro actual de *CLIENTES* en la matriz *matrizNombres* e intenta borrar el registro actual. Si no es posible bloquearlo y borrarlo, el registro no se copia a la tabla destino *ARCCLIEN*. Si se logra borrar el registro, el método añade un nuevo registro vacío en la tabla destino y escribe el contenido de la matriz en el registro.

```
; botónDeArchivar::pushButton
method pushButton(var eventInfo Event)
var
  matrizNombres Array[] String
endVar

; comprobamos si la Ficha está en modo de Edición
if estaFicha.editing = False then       ; si no, comenzar
  CLIENTES.action(DataBeginEdit)
endif

; desplazamos el registro actual desde CLIENTES hasta ARCCLIEN
CLIENTES.copyToArray(matrizNombres)
matrizNombres.view()                    ; echamos un vistazo a
la matriz
; si el registro no puede ser bloqueado, no se elimina
if CLIENTES.deleteRecord() = True then
  ; si se elimina, se copia en ARCCLIEN
  ARCCLIEN.insertRecord()
  ARCCLIEN.copyFromArray(matrizNombres)
endif

endmethod
```

Vea también empty
insertRecord
insertAfterRecord
insertBeforeRecord

edit

Principiante

Método Pone una tabla en modo editar.

Tipo UIObject

Sintaxis **edit ( )** Logical

Descripción Pone todas las tablas de una ficha en modo editar, por lo que es posible realizar cambios. Si una ficha ya está en modo editar, un **edit** innecesario no provoca error ni desactiva el modo editar.

En modo editar, los cambios en los registros se almacenan cuando el foco sale del registro, cuando la tabla recibe una acción DataPostRecord o DataUnlockRecord o cuando se ejecuta **endEdit**. Use **cancelEdit** si necesita cancelar cambios en el registro antes de salir del mismo.

endEdit tiene el mismo efecto que la constante de acción DataEndEdit, por lo que las sentencias siguientes son equivalentes:

```
obj.endEdit()
obj.action(DataEndEdit)
```

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla asociado con la tabla *Pedidos* y un botón llamado *cambiarFecha*. El método **pushButton** de *cambiarFecha* comprueba los campos *Fecha de Venta* y *Fecha de Envío* del registro actual, y actualiza *Fecha de Venta* si *Fecha de Envío* es anterior a *Fecha de Venta*. Una vez que la transacción se ha realizado, **endEdit** almacena el registro y termina el modo editar.

```
; cambiarFecha::pushButton
method pushButton(var eventInfo Event)

; primero, comprobamos si se desea cambiar la Fecha de Envío
if PEDIDOS."Fecha de Venta".value PEDIDOS."Fecha de
Envío".value then
  ; activamos el modo de Edición para esta Ficha
  PEDIDOS.edit()
  ; si la Fecha de Venta es posterior a la fecha de Envío
  ; cambiamos la fecha de Envío
  PEDIDOS."Fecha de Envío".value = PEDIDOS."Fecha de
Venta".value + 5
  PEDIDOS.endEdit()         ; finalizamos la edición. Los
cambios
    ; del registro no se pueden cancelar
endif

endmethod
```

Vea también cancelEdit

empty

Método Elimina todos los registros de una tabla.

Tipo UIObject

Sintaxis **empty ( )** Logical

Descripción Borra todos los registros de una tabla sin pedir confirmación. No es necesario que la tabla se encuentre en modo editar. Esta operación no puede anularse.

En aplicaciones multiusuario, este método intenta, durante el periodo de reintento, aplicar un bloqueo completo sobre la tabla. Si no es posible aplicar el bloqueo, el método falla.

Ejemplo El ejemplo siguiente supone una ficha que tiene tres botones: *crearTabla*, *vaciarTabla* y *eliminarTabla*. *crearTabla* crea una copia de la tabla *Pedidos* llamada *TmpPedid*, sitúa un marco de tabla en la ficha y asocia *TmpPedid* con él. *vaciarTabla* borra todos los registros de *TmpPedid*. *eliminarTabla* elimina el marco de tabla, elimina la tabla del modelo de datos de la ficha y borra la tabla temporal.

Este es el código del botón *crearTabla*:

```
; crearTabla::pushButton
method pushButton(var eventInfo Event)
var
  Tabla Table
  ui    UIObject
endVar

Tabla.attach("Pedidos.db")
Tabla.copy("TmpPedid.db")                              ; copiamos
Pedidos en TmpPedid
ui.create(TableFrameTool, 720, 720, 4320, 1440)     ; crea el
marco de tabla ui.TableName = "TmpPedid.db"      ; tambien
añade la tabla al modelo de datos
ui.visible = True

endmethod
```

Este es el código del método *vaciarTabla*:

```
; vaciarTabla::pushButton
method pushButton(var eventInfo Event)
var
  ui  UIObject
endVar

if ui.attach("TMPPEDID") then
  if msgYesNoCancel("Vaciar",
    "¿Elimino todos los registros de esta tabla?") = "Yes" then
    ui.empty()    ; eliminamos todos los registros de la tabla
TMPPEDID
  endif
endif
```

```
endmethod
```

Este es el código del método *eliminarTabla*:

```
; eliminarTabla::pushButton
method pushButton(var eventInfo Event)
var
  Tabla Table
  ui   UIObject
endVar

; limpiar
if ui.attach("TMPPEDID") then
  ui.delete()                    ; eliminamos el marco de tabla
  DMRemoveTable("TmpPedid.db")   ; la eliminamos del modelo de
datos
  Tabla.attach("TmpPedid.db")
  Tabla.delete()                 ; eliminamos la tabla del
disco
endif

endmethod
```

Vea también deleteRecord

end

Principiante

Método Se desplaza al último registro de una tabla.

Tipo UIObject

Sintaxis **end ( )** Logical

Descripción Define el último registro de una tabla como registro actual.

end tiene el mismo efecto que la constante de acción DataEnd, por lo que las sentencias siguientes son equivalentes:

```
obj.end()
obj.action(DataEnd)
```

Ejemplo Este ejemplo se desplaza al último registro de la tabla *Clientes*. Supóngase que *Clientes* está asociado con un marco de tabla de la ficha; *saltarAlFinal* es un botón de la misma ficha.

```
; saltarAlFinal::pushButton
method pushButton(var eventInfo Event)
CLIENTES.end()        ; nos desplazamos al último registro.
                      ; es lo mismo que:
CLIENTES.action(DataEnd)
msgInfo("¿Estamos en el último registro?", CLIENTES.atLast())

endmethod
```

Vea también home
nextRecord
priorRecord
currRecord
skip
moveTo

endEdit

Principiante

Método Sale del modo editar y acepta los cambios realizados en el registro actual.

Tipo UIObject

Sintaxis **endEdit ( )** Logical

Descripción Desactiva el modo Editar para una tabla y almacena los cambios en el registro actual.

endedit tiene el mismo efecto que la constante de acción DataEndEdit, por lo que las sentencias siguientes son equivalentes:

```
obj.endEdit()
obj.action(DataEndEdit)
```

Ejemplo Consulte el ejemplo de **edit**.

Vea también cancelEdit
edit

enumFieldNames

Método Rellena una matriz con los nombres de los campos de una tabla.

Tipo UIObject

Sintaxis **enumFieldNames (** var ***matrizCampos*** Array[] String)

Descripción Rellena *matrizCampos* con los nombres de los campos de una tabla. Este método devuelve un valor de Logical. Si *matrizCampos* es redimensionable, crece automáticamente para albergar los nombres de los campos; si no lo es, almacena tantos como pueda y desecha el resto. Si *matrizCampos* ya existe, este método la sustituye sin pedir confirmación.

Ejemplo El ejemplo siguiente utiliza **enumFieldNames** para escribir los nombres de los campos de la tabla *Pedidos* en una matriz llamada *nombresDeRegistro*. Supóngase que una ficha tiene un marco de tabla asociado con *Pedidos* y un botón llamado *obtenerNombresDeRegistros*.

```
; obtenerNombresDeRegistros::pushButton
method pushButton(var eventInfo Event)
var
  NombresDeRegistro Array[] String
endVar
PEDIDOS.enumFieldNames(NombresDeRegistro)
NombresDeRegistro.view()
endmethod
```

Vea también enumObjectNames

enumLocks

Método Crea una tabla de Paradox que enumera los bloqueos aplicados actualmente sobre un UIObject; devuelve el número de bloqueos.

Tipo UIObject

Sintaxis **enumLocks (** const ***nombreTabla*** String **)** LongInt

Descripción Crea la tabla de Paradox especificada en *nombreTabla*, listando los bloqueos aplicados actualmente sobre el objeto de tabla. Si *nombreTabla* existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* está abierta, este método falla. En las tablas de dBASE, este método sólo enumera el bloqueo que se ha aplicado (no todos los bloqueos actuales de la tabla).

Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de *nombreTabla* se muestra a continuación:

Nombre de campo	Tipo	Tamaño
UserName	A	15
Lock Type	A	32
Net Session	N	
Session	N	
Record Number	N	

Ejemplo En este ejemplo, el método estándar **pushButton** del botón *mostrarBloqueos* crea una tabla que enumera los bloqueos aplicados actualmente sobre la tabla *Clientes*.

```
; mostrarBloqueos::pushButton
method pushButton(var eventInfo Event)
var
  obj     UIObject
  Cuantos LongInt
  Tabla   TableView
endVar
obj.attach(CLIENTES)                    ; marco de tabla dentro
del formato
lock("Clientes", "Write")               ; bloqueamos Clientes
para escritura
Cuantos = obj.enumLocks("Bloqueos.db")        ; enumeramos los
bloqueos
message("Hay ", Cuantos, " bloqueos en la tabla Clientes.")
Tabla.open("Bloqueos.db")               ; mostramos la tabla
resultante
Tabla.wait()
Tabla.close()

endmethod
```

Vea también lockStatus
TCursor**::**lock

TCursor**::**lockStatus

enumObjectNames

Método/ Procedimiento Rellena una matriz con los nombres de los objetos de una ficha.

Tipo UIObject

Sintaxis **enumObjectNames (** var ***nombresObjetos*** Array[] String **)**

Descripción Rellena una matriz con nombres de objeto. Si *nombresObjetos* es redimensionable, crece automáticamente para albergar los nombres de los objetos; si no lo es, almacena tantos como pueda y desecha el resto. Si la matriz *nombresObjetos* ya existe, este método la sustituye sin pedir confirmación.

Este método devuelve los nombres de objetos asociados y no asociados, comenzando en el objeto que activó este método e incluyendo vías de acceso a los objetos que contiene. Por tanto, para enumerar todos los objetos de una ficha, incluya **enumObjectNames** en un método anexado a la ficha.

Ejemplo En el código siguiente, supóngase que un método personalizado llamado **obtenObjetoNUsuario** está definido para la ficha. La ficha también contiene un botón llamado *obtenNombresDeObjetos* y todos los campos de la tabla*Clientes*. El método **pushButton** de *obtenNombresDeObjetos* ejecuta el método personalizado de la ficha para escribir los nombres de todos los objetos de la ficha en una matriz y, a continuación, en una tabla.

El código siguiente se anexa al método personalizado **obtenObjetoNUsuario**:

```
; Ficha::obtenObjetoNUsuario (método personalizado)
method obtenObjetoNUsuario()
var
  matrizObjetos Array [] String
endVar
enumObjectNames(matrizObjetos)          ; escribimos los nombres
en el array
matrizObjetos.view()                    ; mostramos el array
enumUIObjectNames("TablaObj.db")        ; escribimos los nombres
en la tabla

endmethod
```

Este código es para el método **pushButton** de *obtenNombresDeObjetos*:

```
; obtenNombresDeObjetos::pushButton
method pushButton(var eventInfo Event)
obtenObjetoNUsuario()             ; llamamos al método de
usuario desde la Ficha

endmethod
```

Vea también enumUIObjectProperties
enumUIClasses

enumSource

Método Rellena una tabla con el código fuente de los métodos de una ficha.

Tipo UIObject

Sintaxis **enumSource (** const ***nombreTabla*** String, [const ***recursivo*** Logical **])** Logical

Descripción Rellena una tabla con el código fuente de los métodos de una ficha. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
Object	A	128
MethodName	A	128
Source	M	64

Cuando *recursivo* es True, **enumSource** devuelve las definiciones de los métodos anulados, comenzando en el objeto que activó este método e incluyendo vías de acceso a los objetos que contiene. Por tanto, para enumerar el código fuente de todos los objetos de una ficha, incluya **enumSource** en un método anexado a la ficha.

Ejemplo En el ejemplo siguiente, supóngase que se define un método personalizado llamado **obtenOrigenATablaPersonal** para la ficha. La ficha también contiene un botón llamado *obtenOrigenATabla*. El método **pushButton** de *obtenOrigenATabla* ejecuta el método personalizado de la ficha, que escribe en la tabla *UsuaOrig* todo el código fuente de todos los objetos de la ficha. Este código es el método **pushButton** de *obtenOrigenATablaPersonal*:

```
; obtenOrigenATablaPersonal (método de usuario)
method obtenOrigenATabla()
self.enumSource("UsuaOrig.db", True)
endmethod
```

Este el método **pushButton** de *obtenOrigenATabla*:

```
; obtenOrigenATabla::pushButton
method pushButton(var eventInfo Event)
obtenOrigenATablaPersonal()
endmethod
```

Vea también enumSourceToFile
enumUIObjectProperties
enumUIClasses

enumSourceToFile

Método Escribe el código fuente de una ficha o un objeto en un archivo de texto.

Tipo UIObject

Sintaxis **enumSourceToFile (** const ***nombreArchivo*** String **[,** const ***<Rrecursivo*** Logical **])** Logical

Descripción Escribe el código fuente de los métodos de una ficha en un archivo de texto. Si *nombreArchivo* ya existe, este método lo sustituye sin pedir confirmación. Es posible incluir un alias o una vía de acceso en *nombreArchivo*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreArchivo* en el directorio de trabajo (:TRABAJO:).

Si *recursivo* es False, este método devuelve las definiciones de los métodos anulados del objeto actual solamente. Para incluir el código fuente de los métodos anulados de los objetos contenidos en el objeto actual, *recursivo* debería ser True.

Cuando *recursivo* es True, **enumSourceToFile** devuelve las definiciones de los métodos anulados, comenzando en el objeto que activó este método e incluyendo vías de acceso a los objetos que contiene. Por tanto, para enumerar el código fuente de todos los objetos de una ficha, incluya **enumSourceToFile** en un método anexado a la ficha.

Ejemplo En el ejemplo siguiente, supóngase que se define un método personalizado llamado **obtenOrigenAArchivoPersonal** para la ficha. La ficha también contiene un botón llamado *obtenOrigenAArchivo*. El método **pushButton** de *obtenOrigenAArchivo* ejecuta el método personalizado de la ficha, que escribe en el archivo USUAORIG.TXT todo el código fuente de todos los objetos de la ficha.

A continuación, se muestra el método **pushButton** del botón *obtenOrigenAArchivoPersonal*:

```
; obtenOrigenAFicheroPersonal (método personalizado)
method obtenOrigenAFichero()
self.enumSourceToFile("UsuaOrig.txt", True)
endmethod
```

El código siguiente es el método **pushButton** del botón *obtenOrigenAArchivo*:

```
; obtenOrigenAFichero::pushButton
method pushButton(var eventInfo Event)
obtenOrigenAArchivoPersonal()
endmethod
```

Vea también enumSource
enumUIObjectProperties
enumUIClasses

enumUIClasses

Procedimiento Escribe una lista de clases de UIObject en una tabla.

Tipo UIObject

Sintaxis **enumUIClasses (** const ***nombreTabla*** String **)** Logical

Descripción Crea la tabla *nombreTabla* con una lista de todas las clases de UIObject (como un mapa de bits, cuadro y campo) y sus nombres de propiedad. Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
ClassName	A	32
PropertyName	A	64

Ejemplo Este ejemplo escribe las clases y propiedades en una tabla llamada *ClaseTmp*.

```
; escribirClases::pushButton
method pushButton(var eventInfo Event)
enumUIClasses("ClaseTmp.db")

endmethod
```

Vea también enumUIObjectNames
enumUIObjectProperties

enumUIObjectNames

Método/ Procedimiento Obtiene los nombres de cada objeto de una ficha y los escribe en una tabla.

Tipo UIObject

Sintaxis **enumUIObjectNames (** const ***nombreTabla*** String **)** Logical

Descripción Rellena una tabla con nombres de objeto. Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
ObjectName	A	128
ObjectClass	A	32

Este método devuelve los nombres de objetos asociados y no asociados, comenzando en el objeto que activó este método e incluyendo vías de acceso a los objetos que contiene. Por tanto, para enumerar todos los objetos de una ficha, incluya **enumUIObjectNames** en un método anexado a la ficha.

Ejemplo En el ejemplo siguiente, supóngase que se define un método personalizado llamado **obtenObjetoNUsuario** para la ficha. La ficha también contiene un botón llamado *obtenNombresDeObjetos* y todos los campos de la tabla *CLIENTES*. El método **pushButton** de *obtenNombresDeObjetos* ejecuta el método personalizado de la ficha para escribir todos los nombres de objeto de la ficha en una matriz y, a continuación, en una tabla. El código siguiente se anexa al método personalizado **obtenObjetoNUsuario**:

```
; diseñoFormato::obtenObjetoNUsuario (método de usuario)
method obtenObjetoNUsuario()
var
  matrizObjetos Array [] String
endVar
enumObjectNames(matrizObjetos)          ; escribimos los nombres
a la matriz
matrizObjetos.view()                    ; muestra la matriz
enumUIObjectNames("TablaObj.db")        ; escribimos los nombres
en la tabla
endmethod
```

Este es el código del método **pushButton** de *obtenNombresDeObjetos*:

```
; obtenNombresDeObjetos::pushButton
method pushButton(var eventInfo Event)
obtenObjetoNUsuario()       ; llamamos al método de usuario
desde la Ficha
endmethod
```

Vea también enumObjectNames

enumUIObjectProperties
enumUIClasses

enumUIObjectProperties

Método/ Procedimiento Obtiene las propiedades de cada objeto de una ficha y escribe los datos en una tabla de Paradox.

Tipo UIObject

Sintaxis **enumUIObjectProperties (** const ***nombreTabla*** String **)** Logical

Descripción Obtiene las propiedades de cada objeto de una ficha y escribe los datos en la tabla de Paradox especificada en *nombreTabla*. Es posible incluir un alias o una vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de la tabla es:

Nombre de campo	Tipo	Tamaño
ObjectName	A	128
PropertyName	A	64
PropertyValue	A	255

Ejemplo En el ejemplo siguiente, supóngase que *obtenPropiedades* es un botón de una ficha diseñado para mostrar campos de la tabla *CLIENTES*. El método **pushButton** de *obtenPropiedades* utiliza **enumUIObjectProperties** para escribir valores de todas las propiedades de cada objeto de la ficha en la tabla *UsuaProp*.

```
; obtenPropiedades::pushButton
method pushButton(var eventInfo Event)
enumUIObjectProperties("UsuaProp.db")
endmethod
```

Vea también enumUIObjectNames
enumUIClasses

execMethod

Método/ Procedimiento Ejecuta un método personalizado que no toma argumentos.

Tipo UIObject

Sintaxis **execMethod (** const ***nombreMétodo*** String **)**

Descripción Ejecuta el método personalizado indicado por la cadena nombreMétodo. El método mencionado en nombreMétodo no puede tomar argumentos. execMethod permite ejecutar un método en función del contenido de una variable, lo que implica que el compilador no conoce el método que se activará hasta el momento de la ejecución.

Ejemplo En el ejemplo siguiente, supóngase que una ficha contiene varios campos, *campoUno*, *campoDos* y *campoTres*. La ventana Var de la ficha declara una matriz dinámica llamada *objPreProc*. El método personalizado de la ficha se llama *campoUnoPreProc*. El método **open** de la ficha (en la cláusula **isPreFilter=False**) crea elementos en la matriz *objPreProc*: se crea un elemento para cada objeto de la ficha para el que hay un método personalizado que se procesó previamente.

En este ejemplo, supóngase que *campoUno* requiere algún proceso previo. Se crea un elemento en la matriz con un índice del nombre de objeto "pageOne.fieldOne"; el valor del método personalizado es "campoUnoPreProc". La cláusula **isPreFilter=True** del método **open** de la ficha –ejecutado para cada objeto de la ficha– se ordena si un elemento de la matriz *objPreProc* corresponde al objeto actual; si es así, se ejecuta el método personalizado para ese objeto. El código siguiente define el método personalizado **campoUnoPreProc**:

```
; diseñoFormato::campoUnoPreProc (método de usuario)
; este método se llama durante la cláusula preFilter del
formato,
; cuando el objeto actual sea campoUno.
method campoUnoPreProc()
campoUno.color = "Red"      ; cambiamos el color del campo
campoUno.Value = "Iniciado por el método open del formato"
endmethod
```

Este código se anexa a la ventana Var de la ficha:

```
; ventana de variables del formato
Var
  ObjPreProc DynArray[] String    ; indexado por el nombre del
objeto,                                    ; contendrá los
nombres de los métodos a ejecutar
    ; cuando isPreFilter sea cierto (True)
endVar
```

Este es el código del método **open** de la ficha:

```
method open(var eventInfo Event)
var
  ObjetoDestino   UIObject        ; contiene el objeto de
destino
```

```
  NombreDestino   String          ; Nombre del objeto de destino
  Elemento        AnyType         ; índice del array dinámico
objPreProcs
endVar
if eventInfo.isPreFilter()
 then
    ; este código fuente se ejecuta para todos los objetos del
formato
    eventInfo.getTarget(ObjetoDestino)  ; encontramos quién es
el destino actual
    NombreDestino = ObjetoDestino.name        ; obtenemos el
nombre del destino
   foreach Elemento in objPreProc       ; iteramos a través del
array
      if Elemento = NombreDestino then        ; ¿Existe un
elemento del destino?
     execMethod(objPreProc[NombreDestino])    ; si es así
ejecutamos el
                              ;  método de usuario
      endif
   endforeach
  else
    ; este código fuente lo ejecuta sólo el propio formato.

    ; asignamos los elementos al array objPreProc para indicar
    ; los objetos para los que existe un método de úsuario
preprocesado
    objPreProc["campoUno"] = "campoUnoPreProc"
endif

endmethod
```

Vea también methodDelete
methodGet
methodSet

getBoundingBox

Método Devuelve las coordenadas del marco que rodea un objeto.

Tipo UIObject

Sintaxis **getBoundingBox (** var ***superiorIzquierdo*** Point, var ***inferiorDerecho*** Point **)**

Descripción Devuelve las coordenadas del ángulo superior izquierdo (*superiorIzquierdo*) y del inferior derecho (*inferiorDerecho*) del cuadro (marco) invisible que rodea un objeto, respecto al contenedor del objeto. En sentido estricto, el cuadro circundante sólo es visible en una ventana de diseño. Cuando se selecciona un objeto en una ventana de diseño, sí se puede ver su cuadro circundante.

Ejemplo Este ejemplo dibuja un cuadro alrededor de una elipse en función del cuadro circundante de la elipse. Supóngase que una ficha contiene una elipse llamada *círculoRojo*.

```
; círculoRojo::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  ArribaIzquierda,
  AbajoDerecha    Point           ; para almacenar los puntos
proporcionados
    ; por getBoundingBox
  ui               UIObject       ; para crear un nuevo objeto
endVar

self.getBoundingBox(ArribaIzquierda, AbajoDerecha)
ui.create(BoxTool, ArribaIzquierda.x(),
                  ArribaIzquierda.y(),
                   AbajoDerecha.x() ArribaIzquierda.x(),
                   AbajoDerecha.y() ArribaIzquierda.y())
ui.Color = Green
ui.Translucent = Yes
ui.Visible = Yes

endmethod
```

Vea también convertPointWithRespectTo

getPosition

Método Busca la posición de un objeto.

Tipo UIObject

Sintaxis **getPosition (** const ***x*** LongInt, const ***y*** LongInt,

const ***w*** LongInt, const ***h*** LongInt)

Descripción Busca la posición de un objeto en la pantalla. Las variables *x* e *y* especifican las coordenadas (en twips) del ángulo superior izquierdo del objeto. Las variables *w* y *h* especifican la anchura y altura (en twips) del objeto. Si no se especifica el objeto, implica self.

Ejemplo El ejemplo siguiente desplaza un círculo por la pantalla en respuesta a sucesos del temporizador. El método **pushButton** de *botónIntercambio* emplea **setTimer** y **killTimer** para iniciar o detener un temporizador, dependiendo de la condición del botón. Cuando se inicia el temporizador, emite un TimerEvent cada 100 milisegundos. Cada TimerEvent provoca la ejecución del método **timer** de *botónIntercambio*. El método **timer** busca la posición actual de la elipse mediante **getPosition** y, entonces, la desplaza 100 twips a la derecha mediante **setPosition**.

Este es el código del método **pushButton** de *botónIntercambio*:

```
; BotónIntercambio::pushButton
method pushButton(var eventInfo Event)        ; EtiquetaBotón ha
de ser el nombre
                       ; asociado al texto del botón
if EtiquetaBotón = "Activar Temporizador" then      ; si está
desactivado, lo
                             ; activamos
  EtiquetaBotón = "Desactivar Temporizador"         ; cambiamos
la etiqueta
  self.setTimer(100)                                ; indicamos
al temporizador que
                             ; provoque una pulsación cada
                             ; 100 milisegundos
else
  EtiquetaBotón = "Activar Temporizador"            ; cambiamos
la etiqueta
  self.killTimer()                                  ; detenemos
el temporizador
endif

endmethod
```

Este es el código del método **timer** de *botónIntercambio*:

```
; botónIntercambio::timer
; este método se llama una vez por cada pulso del temporizador
method timer(var eventInfo TimerEvent)
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar
```

```
ui.attach(floatCircle)            ; lo asociamos al círculo
ui.getPosition(x, y, w, h)        ; asignamos las coordenadas a
variables
if x  4320 then                          ; si no estamos en el
borde derecho del área
  ui.setPosition(x + 100, y, w, h)       ; nos desplazamos a la
derecha
else
  ui.setPosition(1440, y, w, h)          ; volvemos a la
izquierda
endif

endmethod
```

Vea también setPosition

getProperty

Método Devuelve el valor de una propiedad especificada.

Tipo UIObject

Sintaxis **getProperty (** const ***nombrePropiedad*** String **)** AnyType

Descripción Devuelve el valor de la propiedad especificada en *nombrePropiedad*. No todas las propiedades toman cadenas como valores. Por ejemplo, si un valor de propiedad es un número, este método devuelve un número. Para devolver una cadena en todos los casos, utilice **getPropertyAsString**.

getProperty es una alternativa a la obtención directa de una propiedad; es útil cuando *nombrePropiedad* es una variable. De lo contrario, acceda a la propiedad directamente, como en

```
EsteColor = MiCaja.Color
```

Ejemplo El ejemplo siguiente crea una matriz dinámica, indexada por nombres de propiedad, para que contenga valores de propiedad. La matriz se rellena empleando el índice de la matriz como argumento del comando **getProperty**.

```
; cajaUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  NomsDeProp DynArray[] AnyType         ; para almacenar los
nombres de las
          ; propiedades y sus valores
  indiceMatriz   String                 ; Indice al array
dinámico
endVar

NomsDeProp["Color"] = ""
NomsDeProp["Visible"] = ""
NomsDeProp["Nombre"] = ""

foreach indiceMatriz in NomsDeProp      ; asignamos las
propiedades al array
  NomsDeProp[indiceMatriz] = self.getProperty(indiceMatriz)
endforeach

NomsDeProp["Color"] = "DarkBlue"

foreach indiceMatriz in NomsDeProp      ; obtener las
propiedades que hay
          ; en el array
  self.setProperty(indiceMatriz, NomsDeProp[indiceMatriz])
endforeach

endmethod
```

Vea también getPropertyAsString
setProperty

getPropertyAsString

Método Devuelve el valor de una propiedad especificada como una cadena.

Tipo UIObject

Sintaxis **getPropertyAsString (** const ***nombrePropiedad*** String **)** String

Descripción Devuelve una cadena que contiene el valor de la propiedad especificada en *nombrePropiedad*.

Ejemplo Este ejemplo asigna el valor de la propiedad Color a una variable AnyType mediante el método **getProperty**. El valor devuelto es un LongInt, ya que los colores son constantes de números enteros grandes. A continuación, se obtiene la propiedad de Color mediante **getPropertyAsString**. El valor devuelto es del tipo String, como "Blue".

```
; cajaUno::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  Color AnyType
endVar

Color = self.getProperty("Color")
Color.view()                              ; muestra el dato según
el tipo LongInt
Color = self.getPropertyAsString("Color")
Color.view()                              ; muestra el dato según
el tipo String

endmethod
```

Vea también getProperty
setProperty

getRGB

Método Halla los componentes rojo, verde y azul de un color.

Tipo UIObject

Sintaxis **getRGB (** const ***color*** LongInt, var ***rojo*** SmallInt, var ***verde*** SmallInt, var ***azul*** SmallInt **)**

Descripción Descompone un color compuesto, *color*, en sus valores componentes de *rojo*, *verde* y *azul*.

Ejemplo Este ejemplo determina los componentes rojo, verde y azul de la constante Brown.

```
; descomponerMarrón::pushButton
method pushButton(var eventInfo Event)

var
  Rojo, Azul, Verde SmallInt
endVar
getRGB(Brown, Rojo, Verde Azul)
msgInfo("Marrón tiene los siguientes componentes",
        String("Rojo ", Rojo, "  Verde ", Verde,
                " Azul ", Azul))
endmethod
```

Vea también rgb

hasMouse

Método Informa de si el ratón está situado sobre un objeto.

Tipo UIObject

Sintaxis **hasMouse ( )** Logical

Descripción Devuelve True si el puntero está situado dentro de los límites de un objeto; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente supone que una ficha tiene un objeto de mapa de bits llamado *gato*. El método **open** de *gato* define el intervalo de temporizador como 250 milisegundos. El método **timer** emplea **hasMouse** para averiguar si *gato* tiene el ratón; si no es así, desplaza *gato* a la posición del ratón. Este es el código del método **open** de *gato*:

```
; gato::open
method open(var eventInfo Event)
; Ajustar el intervalo del temporizador a 250 milisegundos
self.setTimer(250)
endmethod
```

Este es el código del método **timer** de *gato*:

```
; gato::timer
method timer(var eventInfo TimerEvent)
var
  Ratón Point                          ; para obtener la
posición del ratón     endVar
if NOT Gato.hasMouse() then            ; ¿Tengo ratón?
  Ratón = getMouseScreenPosition()     ; encontrar el ratón
  gato.setPosition(Ratón.x()  350,
                 Ratón.y()  2880,
                 4320, 1750)           ; perseguir el ratón
  ; mueve Gato arriba y ligéramente a la izquierda.
  ; asume que Gato es un mapa de bits con una anchura de 4320,
y una altura
  ; de 1750. Como getMouseScreenPosition proporciona la
posición del ratón en el
 ; entorno, estos números asumen que el formato está
maximizado. El desplaza
  ; miento (28801750) acepta la altura del menú y de la barra
de botones
Endif
endmethod
```

Vea también MouseEvent

home

Principiante

Método Se desplaza al primer registro de una tabla.

Tipo UIObject

Sintaxis **home ( )** Logical

Descripción Define el primer registro de una tabla como registro actual. **home** respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro; **home** se desplaza al primer registro de una vista restringida.

home tiene el mismo efecto que la constante de acción DataBegin, por lo que las sentencias siguientes son equivalentes:

```
obj.home()
obj.action(DataBegin)
```

Ejemplo Este ejemplo se desplaza al primer registro de la tabla *Clientes*. Supóngase que *Clientes* está asociada con un marco de tabla de la ficha; *moveToHome* es un botón de la misma ficha.

```
; desplazarseAlOrigen::pushButton
method pushButton(var eventInfo Event)
CLIENTES.home()       ; salta al primer registro
          ; es lo mismo que CLIENTES.action(DataBegin)
msgInfo("¿Estamos en el primer registro?", CLIENTES.atFirst())
endmethod
```

Vea también end
nextRecord
priorRecord

insertAfterRecord

Método Inserta un registro en una tabla después del registro actual.

Tipo UIObject

Sintaxis **insertAfterRecord ( )** Logical

Descripción Inserta un registro nuevo y vacío en una tabla después del registro actual. La tabla debe estar en modo editar.

Ejemplo En este ejemplo, supóngase que *OrdClien* es una copia de la tabla *Clientes*, ordenada por el campo Nombre. La ficha contiene el marco de tabla *ORDCLIEN* asociado con la tabla *OrdClien*, un campo no definido llamado *nuevoCampo* y un botón llamado *botóndeInsertarRegistro*. Para añadir un nuevo nombre en la tabla, teclee el nombre en *NuevoCampo* y pulse *botónDeInsertarRegistro.*

El código siguiente se anexa al método **pushButton** de *insRecButton*. Este método comprueba un valor de *newField* y, a continuación, comprueba si la ficha está en modo editar. Si lo está, el método anexa el TCursor *custTC* a *CUSTSORT*, y busca en *custTC* un valor mayor que la cadena dada en *newField*. Si encuentra un nombre mayor que el nuevo, el método emplea **insertRecord** para insertar un nuevo registro vacío antes del nombre encontrado; en caso contrario, emplea **insertAfterRecord** para insertar un nuevo registro vacío al final de la tabla.

```
; botónDeInsertarRegistro::pushButton
method pushButton(var eventInfo Event)
var
  ClienTC  TCursor
  Nombre   String
endvar

if NuevoCampo.Value = "" then     ; salir si el campo está en
blanco
  RETURN
endif
Nombre = NuevoCampo.Value              ; obtenemos el nombre
que se va a añadir
ORDCLIEN."Nombre".moveTo()
if EsteFormato.Editing then       ; primero comprobamos si se
está en modo
          ; de Edición
  ClienTC.attach(ORDCLIEN)
  scan ClienTC for ClienTC."Nombre"  Nombre:
    quitloop                      ; detenemos cuando se
encuentra el nombre
  endscan
  msgInfo("Número del registro actual", ClienTC.recno())
  ORDCLIEN.resync(ClienTC)        ; sincronizamos el cursor en
ORDCLIEN con
          ; el TC
  if NOT ORDCLIEN.atLast() then
    ORDCLIEN.insertBeforeRecord()       ; insertamos un registro
en blanco antes
          ; del actual
```

```
          ; funciona igual que insertRecord()
  else     ORDCLIEN.insertAfterRecord()      ; insertamos un
registro en blanco
                      ; después del actual
  endif
  ; ... llenamos el registro con el resto de la información del
cliente
else
  msgInfo("Lo siento", "Debes entrar en modo de Edición antes
de insertar un
         registro.")
endif
endmethod
```

Vea también insertRecord
insertBeforeRecord

insertBeforeRecord

Método Inserta un registro en una tabla antes del registro actual.

Tipo UIObject

Sintaxis **insertBeforeRecord ( )** Logical

Descripción Inserta un registro en una tabla antes del registro actual. La tabla debe estar en modo editar. Este método no está disponible para las tablas con formato de dBASE.

insertBeforeRecord tiene el mismo efecto que la constante de acción DataInsertRecord, por lo que las sentencias siguientes son equivalentes:

```
obj.insertBeforeRecord()
obj.action(DataInsertRecord)
```

Ejemplo En este ejemplo, supóngase que *OrdClien* es una copia de la tabla *Clientes*, ordenada por el campo Nombre. La ficha contiene el marco de tabla *ORDCLIEN* asociado con *OrdClien*, un campo no definido llamado *nuevoCampo* y un botón llamado *botónDeInsertarRegistro*. Para añadir un nuevo nombre en la tabla, teclee el nombre en *nuevoCampo* y pulse *botónDeInsertarRegistro.*

El método siguiente anula el método **pushButton** de *botónDeInsertarRegistro*. Este método comprueba un valor de *nuevoCampo* y, a continuación, comprueba si la ficha está en modo editar. Si lo está, el método anexa el TCursor *clienTC* a *ORDCLIEN*, y busca en *clienTC* un valor mayor que la cadena dada en *nuevoCampo*. Si encuentra un nombre mayor que el nuevo, el método emplea **insertBeforeRecord** para insertar un nuevo registro vacío antes del nombre encontrado; en caso contrario, emplea **insertAfterRecord** para insertar un nuevo registro vacío al final de la tabla.

```
; botónDeInsertarRegistro::pushButton
method pushButton(var eventInfo Event)
var
  ClienTC  TCursor
  Nombre   String
endvar

if NuevoCampo.Value = "" then     ; salimos si el campo está en
blanco
  RETURN
endif
Nombre = NuevoCampo.Value         ; obtenemos el nombre que se
va a añadir
ORDCLIEN."Nombre".moveTo()
if EsteFormato.Editing then       ; primero comprobamos si se
está en
    ; modo Edición
  ClienTC.attach(ORDCLIEN)
  scan ClienTC for ClienTC."Nombre"  Nombre:
    quitloop                      ; se detiene cuando se
encuentra el nombre
  endscan
```

```
  msgInfo("Número del registro actual", ClienTC.recno())
  ORDCLIEN.resync(ClienTC)        ; sincronizamos el cursor en
ORDCLIEN con el TC
  if NOT ORDCLIEN.atLast() then
    ORDCLIEN.insertBeforeRecord()       ; insertamos un registro
en blanco antes
          ; del actual
                ; funciona igual que insertRecord()
  else
    ORDCLIEN.insertRecord()       ; insertamos un registro en
blanco después
          ; del actual

  endif
  ; ... llenamos el registro con el resto de la información del
cliente
else
  msgInfo("Lo siento", "Debes entrar en modo de Edición antes
de insertar un
         registro.")
endif

endmethod
```

Vea también insertRecord
insertAfterRecord

insertRecord

Principiante

Método Inserta un registro en una tabla.

Tipo UIObject

Sintaxis **insertRecord ( )** Logical

Descripción Inserta un registro antes del registro actual de una tabla. Este método no está disponible para las tablas con formato de dBASE.

insertRecord tiene el mismo efecto que **insertBeforeRecord** y la constante de acción DataInsertRecord, por lo que las tres sentencias siguientes son equivalentes:

```
obj.insertRecord()
obj.insertBeforeRecord()
obj.action(DataInsertRecord)
```

Ejemplo Consulte el ejemplo de **insertBeforeRecord**.

Vea también insertAfterRecord
insertBeforeRecord

isContainerValid

Procedimiento Informa de si el contenedor de un objeto es válido.

Tipo UIObject

Sintaxis **isContainerValid ( )** Logical

Descripción Informa de si el contenedor de un objeto es válido. Por ejemplo, si una ficha no tiene ningún contenedor, la propiedad Container de una ficha no es válida.

Ejemplo En este ejemplo, el método **arrive** de una ficha utiliza **isContainerValid** para comprobar si hay un contenedor:

```
; esteFormato::arrive
method arrive(var eventInfo MoveEvent)
  if eventInfo.isPreFilter() then

    ; este código fuente se ejecuta antes de cada objeto
  else
   ; este código fuente se ejecuta después (o para el propio
formato)
    if NOT isContainerValid() then
     msgInfo("Formato", "Este objeto no tiene un contenedor
válido.")
   endif
endif

endmethod
```

Vea también isLastMouseClickedValid

isEdit

Principiante

Método	Informa de si un objeto está en modo editar.
Tipo	UIObject
Sintaxis	**isEdit ( )** Logical
Descripción	Informa de si un objeto está en modo editar.
Ejemplo	Consulte el ejemplo de **lockRecord**.
Vea también	edit endEdit lockRecord

isEmpty

Método Informa de si una tabla contiene algún registro.

Tipo UIObject

Sintaxis **isEmpty ( )** Logical

Descripción Devuelve True si no hay registros de la tabla asociados con el marco de tabla. **isEmpty** respeta los límites de vistas restringidas mostradas en un marco de tabla enlazada u objeto multirregistro.

También es posible averiguar si un tabla está vacía comprobando el valor devuelto por el método **nRecords** o comprobando el valor de la propiedad NRecords del objeto.

Ejemplo El botón *eliminarCascada* de este ejemplo borra un pedido y todos los registros de detalle enlazados con ese pedido. Supóngase que una ficha contiene un objeto de un solo registro asociado con la tabla *Pedidos* y un marco de tabla asociado con la tabla *Artículo*. *Pedidos* tiene un enlace unovarios con *Artículo.*

```
; eliminarCascada::pushButton
method pushButton(var eventInfo Event)
var
  ui        UIObject
endVar

if EsteFormato.Editing then
  if msgQuestion("Confirme", "¿Eliminar este pedido?") = "Yes"
then
   ui.attach(articulo)
    while NOT ui.isEmpty()              ; comprobar si la tabla
enlazada está
                ; vacía. Respeta lo que se ve
      ui.deleteRecord()                 ; eliminar los detalles
de los registros
    endwhile
   PEDIDOS.action(DataDeleteRecord)     ; eliminar el registro
maestro
  endif
else
  msgInfo("Estado", "Debes estar en modo edición para eliminar
un registro.")
endif

endmethod
```

Vea también empty
nRecords

isLastMouseClickedValid

Procedimiento Informa de si el último objeto que recibió un clic es válido.

Tipo UIObject

Sintaxis **isLastMouseClickedValid ( )** Logical

Descripción Informa de si la ficha actual ha recibido un clic del ratón desde que se abrió.

Ejemplo Este método comprueba si ya se ha hecho clic sobre una ficha.

```
; esteFormato::arrive
method arrive(var eventInfo MoveEvent)
  if eventInfo.isPreFilter() then
    ; este código fuente se ejecuta antes de cada objeto
  else
    ; este código fuente se ejecuta después (o para el propio
formato)
   if NOT isLastMouseClickedValid() then
      msgInfo("", "En este formato aún no se ha pulsado el
botón del ratón.")
   endif
  endif

endmethod
```

Vea también islastMouseRightClickedValid

isLastMouseRightClickedValid

Procedimiento Informa de si el último objeto que recibió un clic derecho del ratón es válido.

Tipo UIObject

Sintaxis **isLastMouseRightClickedValid ( )** Logical

Descripción Informa de si la ficha actual ha recibido un clic del botón derecho del ratón desde que se abrió.

Ejemplo Este método comprueba si ya se ha hecho un clic derecho sobre una ficha.

```
; esteFormato::arrive
method arrive(var eventInfo MoveEvent)
  if eventInfo.isPreFilter() then
    ; este código fuente se ejecuta antes de cada objeto
  else
    ; este código fuente se ejecuta después (o para el propio
formato)
    if NOT isLastMouseClickedValid() then
      msgInfo("", "En este formato aún no se ha pulsado el
botón derecho
              del ratón.")
    endif
  endif

endmethod
```

Vea también isLastMouseClickedValid

isRecordDeleted

Método Informa de si se ha borrado el registro actual (sólo en tablas de dBASE).

Tipo UIObject

Sintaxis **isRecordDeleted ( )** Logical

Descripción Informa de si el registro actual ha sido borrado. **isRecordDeleted** sólo funciona con tablas de dBASE, porque los registros borrados de Paradox no pueden mostrarse. Este método devuelve True si el registro actual se ha borrado; en caso contrario, devuelve False.

Por defecto, no se muestran los registros borrados de una tabla de dBASE. Para que **isRecordDeleted** funcione correctamente, es necesario ejecutar **showDeleted** para mostrar los registros borrados de la tabla; de lo contrario, dichos registros no son visibles para **isRecordDeleted**.

Vea también unDeleteRecord
TCursor::isShowDeletedOn
TCursor::showDeleted

keyChar

Principiante

Método Envía un suceso al método **keyChar** de un objeto.

Tipo UIObject

Sintaxis

1. keyChar (const ***caracteres*** String **[,** const ***estado*** SmallInt **])** Logical

2. keyChar (const ***valorTeclaAnsi*** SmallInt **)** Logical

3. keyChar (const ***valorTeclaAnsi*** SmallInt, const ***vCarácter*** SmallInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **keyChar** de un objeto con ese suceso.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo El ejemplo siguiente anula el método **pushButton** de un botón llamado *enviarCarácterDeTeclado*. Este método envía pulsaciones de teclas a un campo, llamado *campoUno*, de la misma ficha.

```
; enviarCarácterDeTeclado::pushButton
method pushButton(var eventInfo Event)
var
  x  SmallInt
endVar
campoUno.keyChar("Envíame un ")         ; enviamos una cadena
campoUno.keyChar(65, 65, Shift)         ; enviamos una caracter
ANSI, decimal
          ; equivalente a VK_Char,
; y
          ; EstadoDelTeclado
campoUno.keyChar(" y un ", Shift)       ; enviamos una cadena
con el
          ; EstadoDelTeclado
x = 98                                  ; asignamos el código
campoUno.keyChar(x)                     ; enviamos el código
ANSI del carácter

endmethod
```

Vea también keyPhysical
KeyEvent

keyPhysical

Método Envía un suceso al método **keyPhysical** de un suceso.

Tipo UIObject

Sintaxis **keyPhysical (** const ***valorTeclaAnsi*** SmallInt,

const ***vCarácter*** SmallInt, const ***estado*** SmallInt **)** Logical

Descripción Envía un suceso al método **keyPhysical** de un objeto.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo El código siguiente se anexa al método **pushButton** de un botón llamado *enviarTeclaFísica*. Este método envía el carácter "a" al campo *campoUno*.

```
; enviarTeclaFísica::pushButton
method pushButton(var eventInfo Event)
campoUno.keyPhysical(97, 97, Shift)     ; enviamos una "a"
endmethod
```

Vea también keyChar
KeyEvent

killTimer

Método Detiene el temporizador asociado con un objeto.

Tipo UIObject

Sintaxis **killTimer ( )**

Descripción Detiene un temporizador asociado con un objeto.

Ejemplo El ejemplo siguiente desplaza un círculo por la pantalla en respuesta a varios TimerEvent. El método **pushButton** de *botónDeIntercambio* utiliza **setTimer** y **killTimer** para iniciar o detener un temporizador, dependiendo de la condición del botón. Cuando se inicia el temporizador, emite un TimerEvent cada 100 milisegundos. Cada TimerEvent provoca la ejecución del método **timer** de *botónDeIntercambio*. El método **timer** obtiene la posición actual de la elipse mediante **getPosition** y, entonces, la desplaza 100 twips a la derecha mediante **setPosition**. El código siguiente se anexa al método **pushButton** de *botónDeIntercambio*:

```
; botónDeIntercambio::pushButton
method pushButton(var eventInfo Event)
if EtiquetaBotón = "Activar Temporizador" then      ; si está
desactivado,
                              ; activarlo
  EtiquetaBotón = "Desactivar Temporizador"         ; cambiamos
la etiqueta
  self.setTimer(100)              ; indicamos al Temporizador
que provoque una
          ; pulsación cada 100 milisegundos
else
  EtiquetaBotón = "Activar Temporizador"      ; cambiamos la
etiqueta
  self.killTimer()                                  ; detenemos
el temporizador
endif

endmethod
```

Este es el código del método **timer** de *botónDeIntercambio*:

```
; botónDeIntercambio::timer
  ; este método se llama una vez cada pulsación del
temporizador
method timer(var eventInfo TimerEvent)
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar

ui.attach(floatCircle)                  ; asociamos al círculo
ui.getPosition(x, y, w, h)        ; asignamos las coordenadas a
variables
if x  4320 then                         ; si no estamos en el
borde derecho del área
  ui.setPosition(x + 100, y, w, h)      ; nos desplazamos a la
derecha
```

```
else
  ui.setPosition(1440, y, w, h)         ; volvemos a la
izquierda
endif
endmethod
```

Vea también setTimer

locate

Principiante

Método Busca un valor de campo especificado.

Tipo UIObject

Sintaxis **1. locate (** const ***nombreCampo*** String, const ***coincidenciaExacta*** AnyType **[,**const ***nombreCampo*** String, const ***coinicidenciaExacta*** AnyType **]*)** Logical

2. locate (const ***númeroCampo*** SmallInt, const ***coinicidenciaExacta*** AnyType **[,**const ***númeroCampo*** SmallInt, const ***coincidenciaExacta*** AnyType **]*)** Logical

Descripción Busca en un marco de tabla u objeto multirregistro registros cuyos valores coincidan con los criterios especificados en uno o más pares de campo/valor. Este método utiliza los índices activos, cuando puede, para acelerar la búsqueda, y respeta los límites de vistas restringidas en tablas de detalle enlazadas.

La búsqueda siempre comienza desde el principio de la tabla, pero si no se encuentra ninguna coincidencia, el puntero vuelve al registro actual. Si se encuentra una, el puntero se desplaza a ese registro. Esta operación falla si el registro actual no puede consignarse y desbloquarse (por ejemplo, por una violación de clave).

Ejemplo En el ejemplo siguiente, supóngase que una ficha contiene un marco de tabla asociado con la tabla *Clientes* y un botón llamado *botónDeLocalizar*. El método **pushButton** de *botónDeLocalizar* intenta encontrar al cliente llamado "El Escafandrista" de la ciudad "Benicásim". Si lo encuentra, el nombre del cliente se cambia a "El Escafandrador".

```
; botónDeLocalizar::pushButton
method pushButton(var eventInfo Event)
var
  Cliente UIObject
endVar

Cliente.attach(CLIENTES)          ; buscamos un cliente llamado
"El Escafandrista"
    en Benicásim
if Cliente.locate("Nombre", "El Escafandrista", "Ciudad",
"Benicásim") then
  Cliente.edit()
  Cliente."Nombre" = "El Escafandrador"
  Cliente.endEdit()
endIf

endmethod
```

Vea también locateNext
locatePattern
locateNextPattern

locateNext

Principiante

Método Busca un valor de campo especificado hacia delante desde el registro actual.

Tipo UIObject

Sintaxis **1. locateNext (** const ***nombreCampo*** String, const ***coincidenciaExacta*** AnyType **[,** const ***nombreCampo*** String, const ***coincidenciaExacta*** AnyType **]*)** Logical

2. locateNext (const ***númeroCampo*** SmallInt, const ***coincidenciaExacta*** AnyType **[,** const ***númeroCampo*** SmallInt, const ***coincidenciaExacta*** AnyType **]*)** Logical

Descripción Busca en una tabla los registros cuyos valores coincidan con los criterios especificados en uno o más pares de campo/valor. Este método utiliza los índices activos, cuando puede, para acelerar la búsqueda, y respeta los límites de vistas restringidas en tablas de detalle enlazadas.

La búsqueda comienza en el registro siguiente al actual. Si se encuentra una coincidencia, el puntero se desplaza a ese registro. Si no se encuentra, el puntero vuelve al registro actual. Para comenzar una búsqueda desde el principio de una tabla, utilice **locate**.

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla asociado con la tabla *Clientes* y un botón llamado *botónDeLocalizar*. El método **pushButton** de *botónDeLocalizar* busca clientes de la ciudad de Salobreña. Si el primer **locate** es satisfactorio, el método emplea **locateNext** para buscar registros sucesivos.

```
; botónDeLocalizar::pushButton
method pushButton(var eventInfo Event)
var
  Cliente   UIObject
 Buscar    String
  Encontrados SmallInt
endVar
Cliente.attach(CLIENTES)
Buscar = "Salobreña"
if Cliente.locate("Ciudad", Buscar) then
  Encontrados = 1
  message("")
  while Cliente.locateNext("Ciudad", Buscar)
   Encontrados = Encontrados + 1
  endWhile
  msgInfo("He encontrado " + Buscar, strval(Encontrados) + "
veces.")
endIf
endmethod
```

Vea también locate
locateNextPattern

locateNextPattern

Método Busca el siguiente registro que contiene un campo con un patrón especificado de caracteres.

Tipo UIObject

Sintaxis **1. locateNextPattern ([** const ***nombreCampo*** String,const ***coincidenciaExacta*** AnyType,] * const ***nombreCampo*** String, const ***patrón*** String **)** Logical

2. locateNextPattern ([const ***númeroCampo*** SmallInt, const ***coincidenciaExacta*** AnyType,] * const ***númeroCampo*** SmallInt, const ***patrón*** String **)** Logical

Descripción Busca subcadenas (por ejemplo, "orde" en "ordenador"). La búsqueda comienza en el registro siguiente al actual. Si se encuentra una coincidencia, el puntero se desplaza a ese registro. Si no se encuentra, el puntero vuelve al registro actual. Este método utiliza los índices activos, cuando puede, para acelerar la búsqueda, y respeta los límites de vistas restringidas en tablas de detalle enlazadas.

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave). Para comenzar la búsqueda al comienzo de una tabla, utilice **locatePattern**.

Para buscar registros en función del valor de un solo campo, especifique el campo en *nombreCampo* o *númeroCampo*, y especifique un patrón de caracteres en *patrón*.

Es posible incluir los operadores de patrón @ y .. en el argumento *patrón*. El operador .. representa cualquier cadena de caracteres (incluso, ninguno); @ implica cualquier carácter individual. Cualquier combinación de caracteres literales y comodines puede emplearse en la construcción de una cadena de búsqueda. Si **advancedWildcardsInLocate** (del tipo Session) está activado, es posible emplear operadores de patrón de coincidencia avanzados, no los operadores de patrón estándar. Consulte la descripción de **advMatch** del tipo String para más información sobre operadores de patrón de coincidencia avanzados, y **advancedWildcardsInLocate** e **isAdvancedWildcardsInLocate** del tipo Session.

Para buscar registros en función de los valores de más de un campo, especifique coincidencias exactas en todos los campos *excepto* el último de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas con "Borland" en el campo Nombre, "Paradox" en el campo Producto y palabras que comienzan con "data" (por ejemplo, "database") en el campo Keywords.

```
tc.locateNextPattern("Nombre", "Borland"  "Producto", "Paradox"
"Palabras clave", "datos..")
```

Ejemplo El ejemplo siguiente busca múltiples apariciones de la letra "C" en el campo Nombre de la tabla *Clientes* y escribe los nombres que coincidan en una matriz. Supóngase que el marco de tabla CLIENTES está asociado con *Clientes* y que *botónDeLocalizar* es un botón de la misma ficha.

```
; botónDeLocalizar::pushButton
method pushButton(var eventInfo Event)
var
  Cliente     UIObject           ; para asociar CLIENTES a la
tabla
  Buscar      String             ; el patrón de cadena que se
busca
  Encontrados SmallInt           ; el número de equivalencias
que se localicen
  Nombres     Array[] String     ; las equivalencias
encontradas
endVar

Cliente.attach(CLIENTES)
Buscar = "C.."                   ; buscamos clientes cuyo
nombre comience por C
if Cliente.locatePattern("Nombre", Buscar) then     ; si se
encuentra uno
  Encontrados = 1                                   ; lo
almacenamos en el array
  Nombres.grow(1)                                   ; después
seguimos buscando
 Nombres[Encontrados] = Cliente."Nombre"
  while Cliente.locateNextPattern("Nombre", Buscar)
    Encontrados = Encontrados + 1
    Nombres.grow(1)
    Nombres[Encontrados] = Cliente."Nombre"
  endWhile
endIf
if Nombre.size()  0 then                            ; si hay
algo en el array
  Nombres.view()                                ; se visualiza
endIf

endmethod
```

Este ejemplo es similar al anterior, excepto en que busca registros en función del valor del campo Ciudad y de un patrón en el campo Nombre:

```
; botónDeLocalizar::pushButton
method pushButton(var eventInfo Event)
var
  Cliente   UIObject                  ; para asociar CLIENTES
a la tabla
  Buscar    String               ; el patrón de cadena que se
busca
  Encontrados SmallInt           ; el número de equivalencias
que se localicen
  Nombres Array[] String               ; las equivalencias que
se encuentren
endVar

Cliente.attach(CLIENTES)     Buscar = "..C.."
; buscamos clientes cuyo nombre contenga una C

if Cliente.locatePattern("Ciudad", "Granada", "Nombre", Buscar)
then
```

```
  ; si encontramos alguno
  Encontrados = 1                ; lo almacenamos en el array
  Nombres.grow(1)                ; después seguimos buscando
  Nombres[Encontrados] = Cliente."Nombre"
  while Cliente.locateNextPattern("Ciudad", "Granada",
"Nombre", Buscar)
    Encontrados = Encontrados + 1
    Nombres.grow(1)
   Nombres[Encontrados] = Cliente."Nombre"
  endWhile
endIf
if Nombre.size() <> 0 then             ; si hay algo en el
array
  Nombres.view()                 ; se visualiza
endIf

endmethod
```

Vea también locate
locatePattern
locateNext
String**::**advMatch
String**::**match

locatePattern

Método Busca un registro que contiene un campo con un patrón especificado de caracteres.

Tipo UIObject

Sintaxis **1. locatePattern ([** const ***nombreCampo*** String, const ***coincidenciaExacta*** AnyType,] * const ***nombreCampo*** String, const ***patrón*** String **)** Logical

2. locatePattern ([const ***númeroCampo*** SmallInt, const ***coincidenciaExacta*** AnyType,] * const ***númeroCampo*** SmallInt, const ***patrón String) Logical***

Descripción Busca subcadenas (por ejemplo, "orde" en "ordenador"). Este método utiliza los índices activos, cuando puede, para acelerar la búsqueda, y respeta los límites de vistas restringidas en tablas de detalle enlazadas.

La búsqueda siempre comienza al principio de la tabla, pero si no encuentra ninguna coincidencia, el puntero vuelve al registro actual. Si se encuentra una coincidencia, el puntero se desplaza a ese registro. Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Para buscar registros en función del valor de un solo campo, especifique el campo en *nombreCampo* o *númeroCampo,* e indique un patrón de caracteres en patrón.

Es posible incluir los operadores de patrón @ y .. en el argumento *patrón*. El operador .. representa cualquier cadena de caracteres (incluso, ninguno); @ implica cualquier carácter individual. Cualquier combinación de caracteres literales y comodines puede emplearse en la construcción de una cadena de búsqueda. Si **advancedWildcardsInLocate** (del tipo Session) está activado, es posible emplear operadores de patrón de coincidencia avanzados, no los operadores de patrón estándar. Consulte la descripción de **advMatch** del tipo String para más información sobre operadores de patrón de coincidencia avanzados, y **advancedWildcardsInLocate** e **isAdvancedWildcardsInLocate** del tipo Session.

Para buscar registros en función de los valores de más de un campo, especifique coincidencias exactas en todos los campos excepto el último de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas con "Borland" en el campo Nombre, "Paradox" en el campo Producto y palabras que comienzan con "datos" (por ejemplo, "datosimportantes") en el campo Palabras clave.

```
tc.locateNextPattern("Nombre", "Borland"  "Producto", "Paradox"
"Palabras clave", "datos..")
```

Para comenzar la búsqueda después del registro actual, utilice ***locateNextPattern.***

Ejemplo Consulte el ejemplo de locateNextPattern.

Vea también locateNextPattern
locate

String::advMatch
String::match

locatePrior

Método Busca hacia atrás un valor de campo especificado.

Tipo UIObject

Sintaxis **1. locatePrior *(const nombreCampo*** String, const ***valorBúsqueda*** AnyType **[,**const ***nombreCampo*** String, const ***valorBúsqueda*** AnyType **]*)** Logical

2. locatePrior (const ***númeroCampo*** SmallInt, const ***valorBúsqueda*** AnyType **[,**const ***númeroCampo*** SmallInt, const ***valorBúsqueda*** AnyType **]*)** Logical

Descripción Busca hacia atrás desde el registro actual en una tabla los registros cuyos valores coincidan con los criterios especificados en uno o más pares de campo/valor. Este método utiliza los índices activos, cuando puede, para acelerar la búsqueda, y respeta los límites de vistas restringidas en tablas de detalle enlazadas.

La búsqueda comienza desde el registro anterior al actual y se desplaza hacia arriba por la tabla. Si se encuentra una coincidencia, el puntero se desplaza a ese registro; en caso contrario, vuelve al actual. Para comenzar la búsqueda desde el principio de la tabla, utilice **locate**.

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave).

Ejemplo El método mostrado en este ejemplo busca la última aparición de un valor en la tabla yendo hasta el final de la tabla y empleando **locatePrior** para buscar hacia atrás. Supóngase que la ficha contiene un marco de tabla asociado con la tabla *Clientes* y un botón llamado *botónDeLocalizar*.

```
; botónDeLocalizar::pushButton
method pushButton(var eventInfo Event)
var
  Cliente   UIObject                    ; para asociar CLIENTES
a la tabla
  Buscar String                         ; la cadena que se busca
endVar
Cliente.attach(CLIENTES)                ; asociamos a la tabla
Cliente.end()         ; saltamos al final de la tabla
Buscar = "Salobreña"
if Cliente.locatePrior("Ciudad", Buscar) then       ; buscamos
el registro
  msgInfo("Estado", "El último registro que tiene como Ciudad "
+
       Buscar + " es el número " + Cliente.recno + ".")
endIf
endmethod
```

Vea también locate, locatePriorPattern

locatePriorPattern

Método Busca el registro anterior que contiene un campo con un patrón especificado de caracteres.

Tipo UIObject

Sintaxis **1. locatePriorPattern ([** const ***nombreCampo*** String,const ***coincidenciaExacta*** AnyType,] * const ***nombreCampo*** String, const ***patrón*** String **)** Logical

2. locatePriorPattern ([const ***númeroCampo*** SmallInt, const ***coincidenciaExacta*** AnyType,] * const ***númeroCampo*** SmallInt, const ***patrón*** String **)** Logical

Descripción Busca subcadenas (por ejemplo, "orde" en "ordenador"). Este método utiliza los índices activos, cuando puede, para acelerar la búsqueda, y respeta los límites de vistas restringidas en tablas de detalle enlazadas.

La búsqueda comienza en el registro anterior al actual y continúa hacia arriba por la tabla. Si se encuentra una coincidencia, el puntero se desplaza a ese registro. Si no se encuentra ninguna coincidencia, vuelve al registro actual. Este método utiliza los índices activos, cuando puede, para acelerar la búsqueda.

Esta operación falla si no puede consignarse el registro actual (por ejemplo, por una violación de clave). Para comenzar la búsqueda al comienzo de la tabla, utilice **locatePattern**.

Para buscar registros en función del valor de un solo campo, especifique el campo en *nombreCampo* o *númeroCampo*, y especifique un patrón de caracteres en *patrón*.

Es posible incluir los operadores de patrón @ y .. en el argumento *patrón*. El operador .. representa cualquier cadena de caracteres (incluso, ninguno); @ implica cualquier carácter individual. Cualquier combinación de caracteres literales y comodines puede emplearse en la construcción de una cadena de búsqueda. Si **advancedWildcardsInLocate** (del tipo Session) está activado, es posible emplear operadores de patrón de coincidencia avanzados, no los operadores de patrón estándar. Consulte la descripción de **advMatch** del tipo String para más información sobre operadores de patrón de coincidencia avanzados, y **advancedWildcardsInLocate** e **isAdvancedWildcardsInLocate** del tipo Session.

Para buscar registros en función de los valores de más de un campo, especifique coincidencias exactas en todos los campos *excepto* el último de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas con "Borland" en el campo Nombre, "Paradox" en el campo Producto y palabras que comienzan con "datos" (por ejemplo, "datosimportantes") en el campo Palabras clave.

```
obj.locatePriorPattern("Nombre", "Borland" "Producto",
"Paradox" "Palabras clave", "datos*")
```

Ejemplo El método mostrado en este ejemplo busca la última aparición de un valor en una tabla yendo hasta el final de la tabla y empleando

locatePriorPattern. Supóngase que la ficha contiene un marco de tabla asociado con la tabla *Clientes* y un botón llamado *botónDeLocalizar*.

```
; botónDeLocalizar::pushButton
method pushButton(var eventInfo Event)
var
  Cliente      UIObject     ; para asociar CLIENTES con el marco
de tabla
  Buscar       String             ; la cadena que se busca
endVar
Cliente.attach(CLIENTES)    ; asociamos a la tabla
Cliente.end()                      ; saltamos al final de la
tabla
Buscar = "Salobreña"
if Cliente.locatePrior("Ciudad", Buscar, "Nombre", "..C..")
then
    ; buscamos el registro
  msgInfo("Estado", "El último registro cuya Ciudad es " +
Buscar +
  "y que tiene un nombre con C es el " + Cliente.recno + ".")
endIf

endmethod
```

Vea también locatePattern
locatePrior
String**::**advMatch
String**::**match

lockRecord

Principiante

Método Aplica un bloqueo de escritura sobre el registro actual.

Tipo UIObject

Sintaxis **lockRecord ( )** Logical

Descripción Devuelve True si logra aplicar un bloqueo de escritura explícito sobre el registro actual; en caso contrario, devuelve False. Si el registro ya existe, se bloquea y se convierte en el registro actual.

Nota: También es posible examinar la propiedad Locked de un objeto para averiguar si está bloqueado, pero no puede cambiarse la propiedad para bloquear o desbloquear un objeto. La propiedad Locked es de sólo lectura.

Ejemplo Este ejemplo comprueba, en primer lugar, si la tabla *Clientes* está en modo editar. Si es así, el método busca un registro, intenta bloquearlo con **lockRecord** y comprueba el estado del bloqueo con **recordStatus**. Supóngase que una ficha contiene un marco de tabla asociado con la tabla *Clientes* y un botón llamado *botónDeBloqueo*. Supóngase también que el registro incluido en el marco de tabla CLIENTES se llama *RegClien.*

```
; botónDeBloqueo::pushButton
method pushButton(var eventInfo Event)
var
  obj  UIObject
endVar
obj.attach(CLIENTES)
obj.locate("Nombre", "El Escafandrista")
if CLIENTES.isEdit() then
  if NOT obj.lockRecord() then
    msgStop("Fallo en el bloqueo", "recordStatus(\"Locked\") es
" +
           String(obj.recordStatus("Locked")))
  else
   msgStop("Exito en el bloqueo", "recordStatus(\"Locked\") es
" +
            String(obj.recordStatus("Locked")))
   obj.RegClien."Nombre" = "El Escafandrista"  ; las comillas
en Nombre
                            ; indican el nombre del campo
                            ; en vez de propiedad
   obj.unlockRecord()
  endif
else
  msgInfo("Estado", "Debes estár en modo Edición para poder
bloquear y
         cambiar registros.")
endif

endmethod
```

El ejemplo siguiente muestra cómo puede examinarse la propiedad Locked de un objeto de registro para determinar si el registro está bloqueado. Este

ejemplo se comporta, en líneas generales, igual que el anterior.

```
; botónDeBloqueo::pushButton
method pushButton(var eventInfo Event)
var
  obj  UIObject
endVar
obj.attach(CLIENTES)
obj.locate("Nombre", "El Escafandrista")

if EsteFormato.editing then
 obj.lockRecord()
  recObj.attach(CLIENTES.RegClien)
  if NOT obj.lockRecord() then
    msgStop("Fallo en el bloqueo", "recordStatus(\"Locked\") es
" +
            String(obj.recordStatus("Locked")))
  else
   msgStop("Éxito en el bloqueo", "recordStatus(\"Locked\") es
" +
            String(obj.recordStatus("Locked")))
    obj.RegClien."Nombre" = "El Escafandrista"     ; las
comillas en Nombre
                            ; indican el nombre del campo
                            ; en vez de propiedad
   obj.unlockRecord()
  endif
else
  msgInfo("Estado", "Debes estár en modo Edición para poder
bloquear y cambiar
          registros.")
endif

endmethod
```

Vea también postRecord
recordStatus
unlockRecord

lockStatus

Método Devuelve el número de bloqueos de una tabla.

Tipo UIObject

Sintaxis **lockStatus (**const ***tipoBloqueo*** String **)** SmallInt

Descripción Devuelve el número de veces que se ha aplicado un bloqueo del tipo *tipoBloqueo* en una tabla, donde *tipoBloqueo* es uno de los siguientes: "Write" (Escritura), "Read" (Lectura) o "Any" (Cualquiera).

Si se especifica "Any" como *tipoBloqueo,* **lockStatus** devuelve el número total de bloqueos aplicados sobre la tabla. **lockStatus** sólo informa de los bloqueos aplicados explícitamente, no de los aplicados por Paradox o por otros usuarios o aplicaciones.

Si no se ha aplicado ningún bloqueo de un tipo dado, **lockStatus** devuelve 0.

Ejemplo Este ejemplo supone que una ficha tiene un marco de tabla asociado llamado *CLIENTES* asociado con la tabla *Clientes* y un botón llamado *botónDeBloqueo*. El método **pushButton** de *botónDeBloqueo* elimina todos los bloqueos de *CLIENTES*, comprueba los bloqueos mediante **lockStatus**, aplica un bloqueo y, a continuación, informa de los bloqueos mediante **lockStatus** de nuevo.

```
; botónDeBloqueo::pushButton
method pushButton(var eventInfo Event)
var
  ClienTC TCursor                ; variable para emplazar el
bloqueo de la tabla
  Cliente UIObject
     l    Logical
endVar
ClienTC.attach(CLIENTES)         ; conectamos el TCursor con
CLIENTES
l = unlock(ClienTC, "ALL")  ; eliminamos todos los bloqueos
l.view("Exito en el desbloqueo:")
Cliente.attach(CLIENTES)         ; conectar el UIObject con
CLIENTES
if Cliente.lockStatus("ANY") = 0 then        ; comprobamos los
bloqueos
  l = lock(ClienTC, "WL")              ; asignamos el bloqueo
de escritura
  l.view("Exito en el bloqueo:")       ; de comprobación
endif
msgInfo("Estado", "La tabla " + Cliente.Name + " tiene " +
       String(Cliente.lockStatus("WL")) + " bloqueo(s) de
escritura.")
unlock(ClienTC, "ALL")                         ; eliminamos todos
los bloqueos
endmethod
```

Vea también enumLocks
lockRecord

menuAction

Método/ Procedimiento Envía un suceso al método **menuAction** de un objeto.

Tipo UIObject

Sintaxis **menuAction (** const ***acción*** SmallInt **)** Logical

Descripción Construye un MenuEvent y lo envía a un método estándar **menuAction**. *acción* es una de las constantes de MenuCommand o una constante definida por el usuario. ObjectPAL proporciona constantes para *action*; consulte MenuCommands en el cuadro de diálogo Constantes. Para más información sobre las constantes definidas por el usuario, consulte el Capítulo 6 de la *Guía del Programador ObjectPAL*.

Nota: No es posible utilizar **menuAction** para enviar una constante de comando de menú que sea equivalente a un comando del menú Archivo. Para simular un comando del menú Archivo, emplee una de las constantes Action normales, manipule una propiedad o utilice uno de los muchos métodos del tipo System que emulan los comandos del menú Archivo.

Ejemplo En este ejemplo, el botón *enviarMosaico* de la ficha actual envía a la ficha (*EsteFormato*) una acción MenuWindowTile.

```
; enviarMosaico::pushButton
method pushButton(var eventInfo Event)
EsteFormato.menuAction(MenuWindowTile)
endmethod
```

Vea también action

methodDelete

Método Borra un método especificado.

Tipo UIObject

Sintaxis **methodDelete (** const ***nombreMétodo*** String **)** Logical

Descripción Borra el método especificado en *nombreMétodo*. La ficha que contiene el objeto debe estar en la ventana Diseñar ficha.

Ejemplo Este ejemplo utiliza **methodGet**, **methodSet** y **methodDelete** para copiar métodos de un objeto a otro. El método mostrado aquí anula el método **pushButton** de un botón llamado *copiaMétodos*. Otros cuatro objetos están en la misma ficha: el campo *FormatoDestino* permite especificar el nombre de la ficha que contiene los objetos que se van a copiar; el campo *ObjetoOrigen* alberga el nombre del objeto que contiene los métodos que se van a copiar; el campo *ObjetoDestino* contiene el nombre del objeto en que se copian los métodos; y un campo de botones de radio, *CopiaOMueve*, permite especificar si los métodos desde el origen deberían copiarse, o copiarse y después borrarse.

```
; copiaMétodos::pushButton
method pushButton(var eventInfo Event)
var
  OtroFormato              Form        ; apuntador a un formato
  ObjetoOrigen,                        ; Objeto del que
copiamos
  ObjetoDest              UIObject     ; Objeto en el que
copiamos
  Método                  String       ; almacena la definición
del método
  matrizDeMétodos Array[]  String ; contiene los nombres de los
métodos a copiar
  i                       SmallInt     ; índice del array
endvar

; abrimos el formato y lo conectamos a los objetos
if FormatoDestino = "" OR ObjetoOrigen = "" OR ObjetoDestino =
"" then
  msgStop("Error", "Por favor, rellene el formato, origen y
destino.")
  return
endif
if NOT OtroFormato.load(FormatoDestino.value) then
  msgStop("Error", "No puedo abrir el formato nombrado.")
  return
endif
if NOT ObjetoOrigen.attach(OtroFormato, ObjetoOrigenect.value)
then
  OtroFormato.close()
  msgStop("Error", "No encuentro el objeto origen. Por favor
especifique la
          ruta completa.")
  return
endif
if NOT ObjetoDest.attach(OtroFormato, ObjetoDestino.value) then
```

```
  OtroFormato.close()
  msgStop("Error", "No encuentro el objeto destino. Especifique
la ruta
          completa.")
  return
endif

; rellenamos el array con los nombres de los métodos a copiar
matrizDeMétodos.addLast("mouseUp")
matrizDeMétodos.addLast("mouseDown")
matrizDeMétodos.addLast("mouseDouble")
matrizDeMétodos.addLast("mouseEnter")
matrizDeMétodos.addLast("mouseExit")
matrizDeMétodos.addLast("mouseRightUp")
matrizDeMétodos.addLast("mouseRightDown")
matrizDeMétodos.addLast("mouseRightDouble")
matrizDeMétodos.addLast("mouseMove")
matrizDeMétodos.addLast("open")
matrizDeMétodos.addLast("close")
matrizDeMétodos.addLast("canArrive")
matrizDeMétodos.addLast("arrive")
matrizDeMétodos.addLast("setFocus")
matrizDeMétodos.addLast("canDepart")
matrizDeMétodos.addLast("depart")
matrizDeMétodos.addLast("removeFocus")
matrizDeMétodos.addLast("depart")
matrizDeMétodos.addLast("timer")
matrizDeMétodos.addLast("keyPhysical")
matrizDeMétodos.addLast("keyChar")
matrizDeMétodos.addLast("action")
matrizDeMétodos.addLast("menuAction")
matrizDeMétodos.addLast("error")
matrizDeMétodos.addLast("status")

; añadimos los nombres específicos a los campos y a los botones
if ObjetoOrigen.class = "Field" AND ObjetoDest.class = "Field"
then
  matrizDeMétodos.addLast("CambiarValor")
  matrizDeMétodos.addLast("NuevoValor")
endif

if ObjetoOrigen.class = "Button" AND ObjetoDest.class =
"Button" then
  matrizDeMétodos.addLast("Botón")
endif
if ObjetoOrigen.class  "Button" AND ObjetoDest.class
<> "Button" then
  matrizDeMétodos.addLast("ClicDelRatón")
endif

; Copiamos los métodos desde ObjetoOrigen a ObjetoDest en el
formato
; OtroFormato
for i from 1 to matrizDeMétodos.size()
  ; escribimos el método nombrado en matrizDeMétodos en la
cadena
; msgInfo("matrizDeMétodos es", matrizDeMétodos[i])
```

```
  try
    Método = ObjetoOrigen.methodGet(matrizDeMétodos[i])
    msgInfo("", "Se ha recuperado el método " +
matrizDeMétodos[i] + ".")
    ; escribimos la cadena al método nombrado en
matrizDeMétodos
   ObjetoDest.methodSet(matrizDeMétodos[i], Método)
    if CopiaOMueve.Value = "Move" then
     ObjetoOrigen.methodDelete(matrizDeMétodos[i])
    endif
  onfail
  ;  bucle
  endTry
endfor

endmethod
```

Vea también create
methodGet
methodSet

methodGet

Método	Devuelve el texto de un método especificado.
Tipo	UIObject
Sintaxis	**methodGet (** const ***nombreMétodo*** String **)** String
Descripción	Devuelve una cadena que contiene el texto del método especificado en *nombreMétodo*.
Ejemplo	Consulte el ejemplo de **methodDelete**.

Vea también

create
methodDelete
methodSet

methodSet

Método Define el texto de un método especificado.

Tipo UIObject

Sintaxis **methodSet (** const ***nombreMétodo*** String, const ***textoMétodo*** String **)** Logical

Descripción Especifica en *textoMétodo* el código fuente del método mencionado en *nombreMétodo*.

Ejemplo Consulte el ejemplo de **methodDelete**.

Vea también create
methodDelete
methodGet

mouseClick

Método Genera un suceso mouseClick y lo envía a un objeto.

Tipo UIObject

Sintaxis **mouseClick ( )** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseClick** de un objeto con ese suceso.

Ejemplo El ejemplo siguiente envía un MouseEvent **mouseClick** a *campoDos* en la misma ficha:

```
; enviarClicDelRatón::pushButton
method pushButton(var eventInfo Event)
; enviamos un ClicDelRatón al CampoDos
CampoDos.mouseClick()
endmethod
```

Vea también mouseUp

mouseDouble

Método Envía un suceso al método **mouseDouble** de un objeto.

Tipo UIObject

Sintaxis **mouseDouble (** const ***x*** LongInt, const ***y*** LongInt,

const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseDouble** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo El ejemplo siguiente envía un MouseEvent **mouseDouble** a *CampoDos* en la misma ficha:

```
; enviarDobleClic::pushButton
method pushButton(var eventInfo Event)
;envia un DobleClicDelRatón al CampoDos
CampoDos.mouseDouble(100, 100, LeftButton)
endmethod
```

Vea también mouseRightDouble
mouseDown
mouseUp

mouseDown

Método Envía un suceso al método **mouseDown** de un objeto.

Tipo UIObject

Sintaxis **mouseDown (** const ***x*** LongInt, const ***y*** LongInt,

const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseDown** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo El ejemplo siguiente envía un sends MouseEvent **mouseDown** y **mouseUp** al objeto *CampoUno* en la misma ficha:

```
method pushButton(var eventInfo Event)
var
  PtC  Point
endVar
PtC = campoUno.Position
campoUno.mouseDown(fPt.x(), fPt.y(), LeftButton)
sleep(500)
campoUno.mouseUp(fPt.x(), fPt.y(), LeftButton)
endmethod
```

Vea también mouseUp
mouseDouble
mouseRightDown
mouseRightUp

mouseEnter

Método Envía un suceso al método **mouseEnter** de un objeto.

Tipo UIObject

Sintaxis **mouseEnter (** const ***x*** LongInt, const ***y*** LongInt, const ***estado<RSmallInt) Logical***

Descripción Construye un suceso y ejecuta el método estándar **mouseEnter** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en ***x*** e ***y*** (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo El ejemplo siguiente envía un MouseEvent **mouseEnter** a un campo llamado *CampoSeis* en la misma ficha:

```
; enviarAceptarDelRatón::pushButton
method pushButton(var eventInfo Event)
; enviar un AceptarDelRatón al CampoSeis
CampoSeis.mouseEnter(100,100,LeftButton)
endmethod
```

Vea también mouseExit
mouseMove

mouseExit

Método Envía un suceso al método mouseExit de un objeto.

Tipo UIObject

Sintaxis **mouseExit** ***(*** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseExit** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo Este ejemplo envía un MouseEvent **mouseExit** a *CampoSiete* en la misma ficha:

```
; enviarSalidaDelRatón::pushButton
method pushButton(var eventInfo Event)
; enviar una SalidaDelRatón al CampoSiete
CampoSiete.mouseExit(100, 100, LeftButton)
endmethod
```

Vea también mouseEnter
mouseMove

mouseMove

Método Envía un suceso al método **mouseMove** de un objeto.

Tipo UIObject

Sintaxis **mouseMove (** const ***x*** LongInt, const ***y*** LongInt,

const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseMove** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo Este ejemplo envía un HomeEvent **mouseDown**, un **mouseUp** y un **mouseMove** a un campo llamado *CampoCinco* en la misma ficha:

```
; enviarMovimientoDelRaton::pushButton
method pushButton(var eventInfo Event)
CampoCinco.mouseDown(100, 100, LeftButton)
CampoCinco.mouseUp(100, 100, LeftButton)
; enviar un MovimientoDelRatón al CampoCinco
CampoCinco.mouseMove(100, 100, LeftButton)
endmethod
```

Vea también mouseEnter
mouseExit

mouseRightDouble

Método Envía un suceso al método **mouseRightDouble** de un objeto.

Tipo UIObject

Sintaxis **mouseRightDouble (** const ***x*** LongInt, const ***y*** LongInt,

const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseRightDouble** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado. La ejecución de este método activa el método **mouseRightDouble** del objeto receptor.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo Este ejemplo envía un MouseEvent **mouseRightDouble** a un campo llamado *CampoDos* en la misma ficha:

```
; enviarDobleClic::pushButton
method pushButton(var eventInfo Event)
; enviar un DobleClic al CampoDos
CampoDos.mouseDouble(100, 100, LeftButton)
endmethod
```

Vea también mouseDouble
mouseRightUp
mouseRightDown

mouseRightDown

Método Envía un suceso al método **mouseRightDown** de un objeto.

Tipo UIObject

Sintaxis **mouseRightDown (** const ***x*** LongInt, const ***y*** LongInt,

const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseRightDown** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo Este ejemplo envía un MouseEvent **mouseRightDown** y un **mouseRightUp** a un campo llamado *CampoTres* en la misma ficha:

```
; enviarBotónDerechoRatónArriba::pushButton
method pushButton(var eventInfo Event)
var
  PtC Point
endVar
PtC = CampoTres.position    ; obtenemos la posición, enviamos un
    ; BotónDerechoRatónAbajo
CampoTres.mouseRightDown(PtC.x(), PtC.y(), LeftButton)
sleep(500)                          ; hacemos una pausa, después
enviamos
    ; BotónDerechoRatónArriba
CampoTres.mouseRightUp(PtC.x(), PtC.y(), LeftButton)
endmethod
```

Vea también mouseRightUp
mouseRightDouble
mouseUp
mouseDown

mouseRightUp

Método Envia un suceso al método **mouseRightUp** de un objeto.

Tipo UIObject

Sintaxis **mouseRightUp (** const ***x*** LongInt, const ***y*** LongInt,

const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseRightUp** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en twips). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo Este ejemplo envía un MouseEvent **mouseRightDown** y un **mouseRightUp** a un campo llamado *CampoTres* en la misma ficha:

```
; enviar BotónDerechoRatónArriba::pushButton
method pushButton(var eventInfo Event)
var
  PtC Point
endVar
PtC = CampoTres.position    ; obtenemos la posición, enviamos
    ; BotónDerechoRatónAbajo
CampoTres.mouseRightDown(PtC.x(), PtC.y(), LeftButton)
sleep(500)                  ; hacemos una pausa, después
enviamos
    ; BotónDerechoRatónArriba
CampoTres.mouseRightUp(PtC.x(), PtC.y(), LeftButton)
endmethod
```

Vea también mouseRightUp
mouseRightDouble
mouseUp
mouseDown

mouseUp

Método Envía un suceso al método **mouseUp** a un objeto.

Tipo UIObject

Sintaxis **mouseUp (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un suceso y ejecuta el método estándar **mouseUp** de un objeto con ese suceso. El suceso tendrá las coordenadas especificadas en *x* e *y* (expresadas en tipo pointers). Utilice *estado* para especificar el estado del ratón o del teclado.

ObjectPAL proporciona constantes (por ejemplo, LeftButton) para su uso con *estado*; consulte KeyboardStates en el cuadro de diálogo Constantes. Las constantes de estado del teclado pueden sumarse para crear estados de teclas combinadas, como Alt+Control.

Ejemplo El ejemplo siguiente envía un MouseEvent **mouseDown** y un **mouseUp** al objeto *CampoUno* en la misma ficha:

```
method pushButton(var eventInfo Event)
var
  PtC  Point
endVar
PtC = campoUno.Position
campoUno.mouseDown(PtC.x(), PtC.y(), LeftButton)
sleep(500)
campoUno.mouseUp(PtC.x(), PtC.y(), LeftButton)
endmethod
```

Vea también mouseDown
mouseDouble
mouseRightDown
mouseRightUp
mouseRightDouble

moveTo

Principiante

Método/ Procedimiento Sitúa el foco en un objeto especificado.

Tipo UIObject

Sintaxis **1.** (Método) **moveTo** ( ) Logical

2. (Procedimiento) **moveTo** (const ***nombreObjeto*** String) Logical

Descripción Desplaza el foco a un objeto especificado. Si se ejecuta moveTo como procedimiento, especifique el nombre del objeto en cuestión como una cadena en nombreObjeto.

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla asociado con Pedidos y otro asociado con Artículo. Pedidos tiene un enlace unovarios con Artículo. Un botón llamado encontrarDetalles también se halla en la ficha. Supóngase que se desea poder buscar en toda la tabla Artículo –no sólo en los registros enlazados con el pedido actual–. En este caso, el método pushButton de encontrarDetalles busca pedidos que incluyen el número de pieza actual.

Este código se anexa a la ventana Var de *encontrarDetalles*:

```
; encontrarDetalles::Var
Var
  LineaTC TCursor     ; busca articulo
endVar

; encontrarDetalles::open
method open(var eventInfo Event)
LineaTC.open("articulo.db")
endmethod
```

Este es el código del método **pushButton** de *encontrarDetalles*:

```
; encontrarDetalles::pushButton
method pushButton(var eventInfo Event)
var
  existencias  Number
  PedidoTC     TCursor
  pedidoNum    Number
endVar

; Obtenemos las existencias del artículo actual
existencias = articulo.lineRecord."existencias"
; LineaTC se declaró en la ventana Var y se activó mediante el
método open
if NOT LineaTC.locateNext("Número de Stock", existencias) then
  LineaTC.locate("Número de Stock", existencias)
endif
PedidoTC.attach(PEDIDOS)
PedidoTC.locate("Número de pedido", LineaTC."Número de Pedido")
```

```
PEDIDOS.moveToRecord(PedidoTC)          ; mueve a clientes y
sincroniza con el
          ; TCursor
articulo.lineRecord."Número de Stock".moveTo()
    ; movemos el puntero al detalle de artículo
    ; desplazamos el apuntador al registro coincidente
articulo.locate("existencias", StockNum)
endmethod
```

Este es el código del método **close** de *encontrarDetalles*:

```
; encontrarDetalles::close
method close(var eventInfo Event)
LineaTC.close()   ; cerramos el TCursor de artículo
endmethod
```

Vea también attach

moveToRecNo

Método Se desplaza a un registro específico de una tabla de dBASE.

Tipo UIObject

Sintaxis **moveToRecNo (** const ***númeroRegistro*** LongInt **)** Logical

Descripción Define el registro *númeroRegistro* como registro actual. Devuelve un error si *númeroRegistro* no está en la tabla. Utilice el método **nRecords** o examine la propiedad NRecords para averiguar cuántos registros contiene una tabla. Este método sólo está recomendado para las tablas de dBASE.

Ejemplo Este ejemplo se desplaza al registro central de una tabla. Supóngase que una ficha contiene un marco de tabla asociado con la tabla *Artículo* y un botón llamado *Mitad*.

```
; mitad::pushButton
method pushButton(var eventInfo Event)
var
  Mitad LongInt
endVar

Mitad = LongInt(articulo.nRecords()/2)
articulo.moveToRecNo(Mitad)

endmethod
```

Vea también nRecords
moveToRecord
resync

moveToRecord

Método Se desplaza a un registro específico de una tabla.

Tipo UIObject

Sintaxis **1. moveToRecord (** const ***númeroRegistro*** LongInt **)** Logical

2. moveToRecord (const ***tc*** TCursor **)** Logical

Descripción Define el registro *númeroRegistro* o el registro señalado por el TCursor *tc* como registro actual. Devuelve un error si *númeroRegistro* es mayor que el número de registros existentes en la tabla. Utilice el método **nRecords** o examine la propiedad NRecords para averiguar cuántos registros contiene una tabla. Este método puede ser muy lento en las tablas de dBASE; use **moveToRecNo** en su lugar.

Ejemplo Para un ejemplo de cómo utilizar **moveToRecord** con un TCursor, consulte **moveTo**.

Este ejemplo se desplaza al registro central de una tabla. Supóngase que una ficha contiene un marco de tabla asociado con la tabla *Artículo* y un botón llamado *Mitad*.

```
; mitad::pushButton
method pushButton(var eventInfo Event)
var
  Mitad LongInt
endVar

Mitad = LongInt(articulo.nRecords()/2)
articulo.moveToRecord(Mitad)

endmethod
```

Vea también nRecords
moveTo
moveToRecNo
resync

nextRecord

Principiante

Método Se desplaza al siguiente registro de una tabla.

Tipo UIObject

Sintaxis **nextRecord ( )** Logical

Descripción Define el registro siguiente de una tabla como registro actual. Devuelve un error si el puntero ya está en el último registro.

nextRecord tiene el mismo efecto que la constante de acción DataNextRecord, por lo que las sentencias siguientes son equivalentes:

```
obj.nextRecord()
obj.action(DataNextRecord)
```

Ejemplo Este ejemplo se desplaza al registro siguiente de la tabla *Clientes*. Supóngase que *Clientes* está asociada con un marco de tabla de la ficha; *desplazarseAlSiguiente* es un botón de la misma ficha.

```
; desplazarseAlSiguiente::pushButton
method pushButton(var eventInfo Event)
if NOT CLIENTES.atLast() then
  CLIENTES.nextRecord()     ; nos desplazamos al siguiente
registro
  ; sería lo mismo que: CLIENTES.action(DataNextRecord)
  msgInfo("¿En qué registro estamos?", CLIENTES.recno)
else     msgInfo("Estado", "Ya estamos en el último registro.")
endif
endmethod
```

Vea también home
end
priorRecord
moveToRecord

nFields

Método Devuelve el número de campos de una tabla.

Tipo UIObject

Sintaxis **nFields ( )** LongInt

Descripción Devuelve el número de campos de una tabla.

Nota: Para averiguar el número de columnas mostradas en un objeto asociado con una tabla, examine el valor de la propiedad NCols de ese objeto.

Ejemplo El ejemplo siguiente informa del número de campos y campos clave de la tabla *Artículo*. Supóngase que una ficha tiene el marco de tabla *ARTICULO* asociado con la tabla *Artículo*, y un botón llamado *estadísticasDeTabla*.

```
; estadísticasDeTabla::pushButton
method pushButton(var eventInfo Event)
msgInfo("Estado", "La tabla artículo tiene " +
        String(articulo.nFields()) + " campos y " +
       String(articulo.nKeyFields()) + " campos clave." +
       "\nHay " + String(articulo.NCols) +
        " columnas en la tabla.")

endmethod
```

Vea también nKeyFields
nRecords

nKeyFields

Método	Devuelve el número de campos clave de una tabla.
Tipo	UIObject
Sintaxis	**nKeyFields ( )** LongInt
Descripción	Devuelve el número de campos clave de una tabla.
Ejemplo	Consulte el ejemplo de **nFields**.
Vea también	nFields nRecords

nRecords

Principiante

Método Devuelve el número de registros de una tabla.

Tipo UIObject

Sintaxis **nRecords ( )** LongInt

Descripción Devuelve el número de registros de una tabla. Puede ser un proceso largo con las tablas de dBASE. También es posible examinar la propiedad NRecords de un objeto para averiguar el número de registros de la tabla asociada con ese objeto.

El método **nRecords** y la propiedad NRecords respetan los límites de vistas restringidas. Si un objeto basado en tabla es la tabla de detalle en una relación unovarios, **nRecords** informa del número de registros de detalle enlazados, no del número total de registros existentes en toda la tabla.

Ejemplo Este ejemplo se desplaza al registro central de una tabla. Supóngase que una ficha contiene el marco de tabla *ARTICULO* asociado con la tabla *Artículo* y un botón llamado *Mitad*.

```
; mitad::pushButton
method pushButton(var eventInfo Event)
var
  Mitad LongInt
endVar

Mitad = LongInt(articulo.nRecords()/2)
articulo.moveToRecord(Mitad)

endmethod
```

Vea también moveToRecord
nKeyFields
nFields

pixelsToTwips

Método Convierte las coordenadas de pantalla de pixels a twips.

Tipo UIObject

Sintaxis **pixelsToTwips (** const ***pixels*** Point **)** Point

Descripción Convierte las coordenadas de pantalla especificadas en *pixels* de pixels a twips. Un pixel es un punto en la pantalla, mientras que twip es una unidad independiente del dispositivo que es igual a 1/1440 de pulgada (1/20 de un punto de la impresora).

Ejemplo Este ejemplo supone que una ficha contiene un cuadrado de dos pulgadas llamado *dosCuadrados*. El cuadrado *dosCuadrados* contiene dos cuadros de texto: *pixNum* que muestra la anchura del cuadro expresada en pixels y *twipNum* que muestra la anchura en twips.

```
; dosCuadrados::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  SupIzqTw,                ; punto superior izquierdo en twips
  InfDerTw,                ; punto inferior derecho en twips
  SupIzqPx,                ; superior izquierdo en pixels
  InfDerPx,                ; inferior derecho en pixels
  PosicPropia Point   ; propiedad de la posición actual
endvar
self.getBoundingBox(SupIzqTw, InfDerTw)            ;
proporciona los puntos en twips
twipNum.Text = InfDerTw.x()  SupIzqTw.x()     ; obtenemos el
ancho en twips
SupIzqPx = TwipsToPixels(SupIzqTw)                 ;
convertimos a pixels
InfDerPx = TwipsToPixels(InfDerTw)
pixNum.Text = InfDerPx.x()  SupIzqPx.x()      ; obtenemos el
ancho en pixels
; comprobamos los cruces
SupIzqTw = PixelsToTwips(SupIzqPx)      ; convertimos de pixels
a twips
SupIzqTw.view("Top left in twips")      ; SupIzqTw debería
coincidir con PosicPropia

PosicPropia = self.Position             ; obtenemos PosicPropia,
por defecto en twips
PosicPropia.view("Posicion de la caja en twips")    ; mostrar el
resultado

endmethod
```

Vea también twipsToPixels

postAction

Método Almacena una acción en una cola de acciones para su ejecución demorada.

Tipo UIObject

Sintaxis **postAction (** const ***idAcción*** SmallInt **)**

Descripción Funciona como **action**, excepto que la acción no se ejecuta inmediatamente. Paradox espera hasta que haya terminado de ejecutarse todo el método y Paradox esté en estado de espera. La acción especificada en *idAcción* se almacena en una cola de acciones en el momento de la ejecución del método; Paradox realiza la acción después de que haya terminado de ejecutarse el método actual.

Vea también action
Form**::**postAction

postRecord

Principiante

Método Almacena un registro pendiente en una tabla.

Tipo UIObject

Sintaxis **postRecord ( )** Logical

Descripción Devuelve True si el registro actual se almacena satisfactoriamente en la tabla subyacente; en caso contrario, devuelve False. **postRecord** no desbloquea un registro bloqueado.

postRecord tiene el mismo efecto que la constante de acción DataPostRecord, por lo que las sentencias siguientes son equivalentes:

```
obj.postRecord()
obj.action(DataPostRecord)
```

Ejemplo Este ejemplo busca un registro, intenta bloquearlo mediante **lockRecord** y, entonces, comprueba el estado del bloqueo mediante **recordStatus**. El método cambia el registro y lo almacena mediante **postRecord**. Supóngase que una ficha contiene un marco de tabla asociado con la tabla *Cliente* y un botón llamado *botónDeBloqueo*.

```
; botónDeBloqueo::pushButton
method pushButton(var eventInfo Event)
var
  obj  UIObject
endVar
obj.attach(CLIENTES)
obj.locate("Nombre", "El Escafandrista")
if EsteFormato.Editing then
  if NOT obj.lockRecord() then
    msgStop("Fallo en el bloqueo", "recordStatus(\"Locked\") es
" +
            String(obj.recordStatus("Locked")))
  else
    msgStop("Exito en el bloqueo", "recordStatus(\"Locked\") es
" +
            String(obj.recordStatus("Locked")))
    obj.RegClien."Nombre" = "El Escafandrista"      ; las
comillas del Nombre
                                    ; indican el nombre del campo
                                    ; en vez de propiedad
   obj.postRecord()
    message("He bloqueado el registro: ", obj.RegClien.locked)
  endif
else
  msgInfo("Estado", "Debes estár en modo Edición para bloquear
y cambiar
          registros.")
endif

endmethod
```

Vea también recordStatus
lockRecord
TCursor**::**attachToKeyViol

priorRecord

Principiante

Método Se desplaza al registro anterior de una tabla.

Tipo UIObject

Sintaxis **priorRecord ( )** Logical

Descripción Define el registro anterior de una tabla como registro actual. Devuelve un error si el puntero ya se encuentra en el primer registro.

priorRecord tiene el mismo efecto que la constante de acción DataPriorRecord, por lo que las sentencias siguientes son equivalentes:

```
obj.priorRecord()
obj.action(DataPriorRecord)
```

Ejemplo Este ejemplo se desplaza al primer registro de la tabla *Clientes*. Supóngase que *Clientes* está asociada con un marco de tabla de la ficha; *desplazarseAlAnterior* es un botón de la misma ficha.

```
; desplazarseAlAnterior::pushButton
method pushButton(var eventInfo Event)
if NOT CLIENTES.atFirst() then
  CLIENTES.priorRecord()    ; nos desplazamos al registro
anterior
  ; sería lo mismo que CLIENTES.action(DataPriorRecord)
  msgInfo("¿En qué registro estamos?", CLIENTES.recno)
else
  msgInfo("Estado", "Ya estamos en el primer registro.")
endif

endmethod
```

Vea también home
end
nextRecord
currRecord
skip
moveToRecord

pushButton

Método Genera un Event **pushButton** y lo envía a un objeto.

Tipo UIObject

Sintaxis **pushButton ( )** Logical

Descripción Construye un suceso y ejecuta el método estándar **pushButton** de un objeto con ese suceso.

Ejemplo El ejemplo siguiente envía un suceso **pushButton** a *BotónDos* en la misma ficha:

```
; enviarPulsaciónDeBotón::pushButton
method pushButton(var eventInfo Event)
; enviar una PulsaciónDeBotón al BotónDos
BotónDos.pushButton()

endmethod
```

Vea también mouseClick
mouseUp

recordStatus

Método Informa acerca del estado de un registro.

Tipo UIObject

Sintaxis **recordStatus (** const ***tipoEstado*** String **)** Logical

Descripción Devuelve True o False a una pregunta sobre el estado de un registro. Utilice el argumento *tipoEstado* para especificar el estado en cuestión, donde *tipoEstado* es "New" (nuevo), "Locked" (bloqueado) o "Modified" (modificado).

"New" implica que el registro se acaba de insertar en la tabla. "Locked" significa que se ha aplicado un bloqueo (implícito o explícito) sobre el registro. "Modified" implica que se ha cambiado al menos uno de los valores de campo. Es posible obtener una información similar sobre el registro actual examinando las propiedades Inserting, Locked, Focus y Touched del registro.

Ejemplo Este ejemplo busca un registro, intenta bloquearlo mediante **lockRecord** y, entonces, comprueba el estado del bloqueo mediante **recordStatus**. El método cambia el registro y los debloquea mediante **unlockRecord**. Supóngase que una ficha contiene un marco de tabla asociado con la tabla *Clientes* y un botón llamado *botónDeBloqueo*.

```
; botónDeBloqueo::pushButton
method pushButton(var eventInfo Event)
var
  Cliente   UIObject                                    ; para
conectar al marco de tabla
  ClaveNueva Number

endVar
Cliente.attach(CLIENTES)                              ; conectamos
CLIENTES a la tabla
Cliente.locate("Nombre", "El Escafandrista")  ; encontramos el
registro
if NOT EsteFormato.editing then                       ;
comprobamos si el formáto está
                              ; en el modo de Edición
  msgInfo("Estado", "Debes estár en modo de Edición para esta
operación.")
else
  if NOT Cliente.lockRecord() then                    ;
intentamos el bloqueo del registro
    msgStop("Estado", "Fallo en el bloqueo.
recordStatus(\"Locked\") es " +
           String(Cliente.recordStatus("Locked")))
  else
   msgInfo("¿Está bloqueado el registro?",
Cliente.recordStatus("Locked"))
    ClaveNueva = 1384
    Cliente.RegClien."Número de Cliente" = ClaveNueva     ;
cambiamos el valor de la clave
    msgInfo("¿Se ha modificado el registro",
Cliente.recordStatus("Modified"))
```

```
   Cliente.unlockRecord()                          ; intentamos
desbloquear el
                              ; registro. si esto causa una
                                                   ; violación de
clave, Paradox
                                                   ; deja el registro
bloqueado
   if Cliente.recordStatus("Locked") then
     msgInfo("Estado", "Se ha producido una violación de clave.
Cambiando
                la clave.")
      ClaveNueva = 1451
      Cliente.custRec."Número de Cliente"=ClaveNueva     ;
cambiando a nueva clave
      Cliente.postRecord()                             ; la
almacenamos
      ; el registro se desplaza a una nueva posición basándose
en la clave
   endif
    Cliente.locate("Número de Cliente", ClaveNueva)      ;
encontramos el lugar
                                             ; dónde se desplazó
  endif
endif

endmethod
```

Vea también lockRecord
unlockRecord

resync

Método Resincroniza un objeto en un TCursor.

Tipo UIObject

Sintaxis **resync (** const ***tc*** TCursor **)** Logical

Descripción Cambia el puntero de registro actual de un UIObject al registro actual del TCursor *tc*. Cuando se resincroniza un objeto de tabla en un TCursor, los filtros e índices de la tabla cambiarán a los del TCursor (en tablas de dBASE, la tabla también tomará el valor de Mostrar borrados del TCursor).

Ejemplo Consulte el ejemplo de insertRecord.

Vea también attach
moveToRecord
insertRecord

rgb

Procedimiento Define un color.

Tipo UIObject

Sintaxis **rgb (**const ***rojo*** SmallInt, const ***verde*** SmallInt, const ***azul*** SmallInt) LongInt

Descripción Define un color en función de los valores de *rojo*, *verde* y *azul*, que pueden ser enteros entre 0 y 255 o constantes (por ejemplo, LightBlue).

Las constantes de color se enumeran en el cuadro de diálogo Constantes bajo Colores.

Ejemplo El ejemplo siguiente emplea **rgb** para definir el color de los cuadros cuando se crean. El método crea una paleta de colores. Supóngase que los títulos existen en la ficha en las posiciones adecuadas. La ficha tiene un botón, llamado **showPalette**.

```
; dibujarPaleta::pushButton
method pushButton(var eventInfo Event)
var
  matrizRGB Array[5] SmallInt                ; array que
almacena los valores rgb
  BaseX             LongInt             ; posición base
  BaseY             LongInt             ; posición base
  ui                UIObject            ; apuntador a las cajas
que se crean
endVar
const
  IncHoriz = 1440                             ; cantidad a
desplazarse horizontalmente
                ; (en twips)
  IncVert =  1080                        ; cantidad a
desplazarse verticalmente
endConst

matrizRGB[1] = 0
matrizRGB[2] = 64
matrizRGB[3] = 128
matrizRGB[4] = 192
matrizRGB[5] = 255

for i from 1 to matrizRGB.size()              ; rojo (posición
diagonal)
  BaseX = 720 + ((i  1) * 150)                ; cambiamos la
base según
                    ; i vaya incrementándose
  BaseY = 720 + ((i  1) * 150)
  for j from 1 to matrizRGB.size()            ; verde
(posicionamiento vertical)
    for k from 1 to matrizRGB.size()          ; azul
(posicionamiento horizontal)
      ui.create(boxTool, BaseX + (IncHoriz * (k  1)),
                BaseY + (IncVert * (j  1)), 250, 250)
     ; ajustamos el color utilizando los valores RGB del array
```

```
      ui.Color = rgb(matrizRGB[i], matrizRGB[j], matrizRGB[k])
      ui.Visible = Yes
    endfor                                          ; k (azul,
horizontal)
  endfor                                            ; j (verde,
vertical)
endfor                                              ; i (rojo,
diagonal)

endmethod
```

Vea también getProperty
getRGB
setProperty

setFilter

Método Define el rango de registros que un objeto de tabla puede señalar.

Tipo UIObject

Sintaxis **setFilter ([** const ***valCoincidenciaExacta*** AnyType,] *

const ***valMin*** AnyType, const ***valMax*** AnyType) Logical

Descripción Especifica las condiciones para incluir un rango de registros. Los registros que cumplen las condiciones se incluyen; los que no, se excluyen. Esta operación falla si no puede consignarse el registro actual o si el objeto de tabla no está asociado con una tabla con clave.

Este método compara los criterios especificados con los valores de los campos correspondientes del índice de una tabla. Para filtrar los registros por el valor de un solo campo, especifique valores en *valMin* y *valMax*. Por ejemplo, la sentencia siguiente comprueba los valores del primer campo del índice de cada registro. Si un valor es menor que 14 o mayor que 88, ese registro se excluye.

```
TablaObj.setFilter(14, 88)
```

Para especificar una coincidencia exacta en un solo campo, asigne el mismo valor a *valMin* y *valMax*. Por ejemplo, la sentencia siguiente descarta todos los valores excepto 55:

```
TablaObj.setFilter(55, 55)
```

Es posible filtrar registros por los valores de más de un campo. Para ello, especifique coincidencias exactas *excepto* para el último elemento de la lista. Por ejemplo, la sentencia siguiente busca coincidencias exactas de "Borland" y "Paradox" (suponiendo que son los primeros campos del índice) y valores entre 100 y 500, ambos incluidos, en el tercer campo.

```
TablaObj.setFilter("Borland", "Paradox", 100, 500)
```

La ejecución de **setFilter** sin argumentos redefine el filtro para incluir toda la tabla.

Ejemplo En este ejemplo, supóngase que el primer campo de la clave de *Artículo* es "Número de Pedido.". Cuando el usuario pulsa el botón *obtenerDetalles*, el método **pushButton** limita el número de registros incluidos en el objeto ARTICULO a los que contienen 1005 en el primer campo clave.

```
; obtenerDetalles::pushButton
method pushButton(var eventInfo Event)
var
  TablaObj  UIObject
endVar
if TablaObj.attach(articulo) then
  ; limitamos la visión de TablaObj a registros que tienen
  ; 1005 como valor clave (Número de Pedido 1005).
  TablaObj.setFilter(1005, 1005)
  ; ahora mostramos el número de registros cuyo Número de
Pedido sea 1005
```

```
  msgInfo("Número total de registros para el pedido 1005",
TablaObj.nRecords())
else
  msgStop("Lo siento", "No puedo conectar con la tabla.")
endif

endmethod
```

Vea también switchIndex
TCursor**::**setFilter

setPosition

Método Define la posición de un objeto.

Tipo UIObject

Sintaxis **setPosition (** const ***x*** LongInt, const ***y*** LongInt,

const ***w*** LongInt, const ***h*** LongInt)

Descripción Define la posición de un objeto en la pantalla. Las variables *x* e *y* especifican las coordenadas del ángulo superior izquierdo del objeto (en twips), mientras que *w* y *h* indican la anchura y la altura del objeto (en twips). Si no se especifica el objeto, implica self.

También es posible definir y examinar la posición y tamaño de un objeto mediante las propiedades Position y Size. Por ejemplo,

```
self.Position = Point(100, 150)
self.Size = Point(2000, 2500)
```

equivale a

```
self.setPosition(100, 150, 2000, 2500)
```

Ejemplo El ejemplo siguiente desplaza un círculo por la pantalla en respuesta a los TimerEvent. El método **pushButton** de *botónDeIntercambio* utiliza **setTimer** y **killTimer** para iniciar o detener un temporizador, dependiendo de la condición del botón. Cuando se inicia el temporizador, emite un TimerEvent cada 100 milisegundos. Cada TimerEvent provoca la ejecución del método **timer** de *botónDeIntercambio*. El método **timer** obtiene la posición actual de la elipse mediante **getPosition** y, entonces, la desplaza 100 twips a la derecha mediante **setPosition**.

El código siguiente corresponde al método **pushButton** de *botónDeIntercambio*:

```
; botónDeIntercambio::pushButton
method pushButton(var eventInfo Event)
; la etiqueta del botón se renombra según EtiquetaBotón
if EtiquetaBotón = "Activar Temporizador" then ; si está
desactivado,
                          ; lo activamos
  EtiquetaBotón = "Desactivar Temporizador"         ; cambiamos
la etiqueta
  self.setTimer(10)                              ; activar el
Temporizador
else                                             ; si está
desactivado,
                          ; lo activamos
  EtiquetaBotón = "Activar Temporizador"            ; cambiamos
la etiqueta
  self.killTimer()                               ; detenemos el
temporizador
endif
endmethod
```

A continuación, se muestra el código del método **timer** de

botónDeIntercambio:

```
; botónDeIntercambio::timer
method timer(var eventInfo TimerEvent)
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar
ui.attach(floatCircle)                  ; asociamos al círculo
ui.getPosition(x, y, w, h)              ; asignamos las
coordenadas a variables

if x = 4320 then                        ; si no estamos en el
borde izquierdo del área
  ui.setPosition(x + 100, y, w, h)      ; nos desplazamos a la
izquierda
else
  ui.setPosition(1440, y, w, h)         ; volvemos a la derecha
endif

endmethod
```

Vea también getPosition

setProperty

Método Define una propiedad con un valor especificado.

Tipo UIObject

Sintaxis **setProperty (** const ***nombrePropiedad*** String,

const ***valorPropiedad*** AnyType **)**

Descripción Define la propiedad *nombrePropiedad* de un objeto como *valorPropiedad*. Si el objeto no tiene una propiedad *nombrePropiedad* o si *valorPropiedad* no es válido, se produce un error.

setProperty es una alternativa a la definición directa de una propiedad; es útil cuando *nombrePropiedad* es una variable. En caso contrario, acceda a la propiedad directamente, como en

```
miCaja.Color = Red
```

Ejemplo El ejemplo siguiente crea una matriz dinámica, indexada por nombres de propiedad, para albergar valores de propiedad. La matriz se rellena empleando el índice la matriz como argumento del comando **getProperty**. El método cambia uno de los valores de las propiedades y restaura las propiedades del objeto desde la matriz dinámica con el método **setProperty**.

```
; cajaUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  Propiedades DynArray[] AnyType  ; contiene los nombres y
valores de
          ; las propiedades
  indiceMatriz           String   ; índice del array dinámico
endVar

Propiedades["Color"] = ""
Propiedades["Visible"] = ""
Propiedades["Nombre"] = ""

foreach indiceMatriz in Propiedades
  Propiedades[indiceMatriz] = self.getProperty(indiceMatriz)
endforeach

Propiedades["Color"] = "DarkBlue"



foreach indiceMatriz in Propiedades
  self.setProperty(indiceMatriz, Propiedades[indiceMatriz])
endforeach

endmethod
```

Vea también getProperty
getPropertyAsString

setTimer

Método Inicia el temporizador de un objeto.

Tipo UIObject

Sintaxis **setTimer (** const ***milisegundos*** LongInt **[,** const ***repetir*** Logical **])**

Descripción Inicia un temporizador para un objeto. El intervalo del temporizador se especifica mediante *milisegundos*. El argumento optativo *repetir* especifica si el temporizador se repite automáticamente. Si *repetir* es True o se omite, el temporizador se repite; en caso contrario, el TimerEvent se envía una vez. Normalmente, **setTimer** se anexa al método **open** de un objeto, y la respuesta del objeto se define en su método **timer**.

Nota: Windows permite un máximo de 16 temporizadores para todas las aplicaciones. Sin embargo, Paradox no tiene ningún límite. Los recursos del sistema pueden limitar el número de temporizadores que pueden definirse, por lo que tal vez agote los temporizadores de Windows, aunque Paradox no esté restringido por el límite de 16 temporizadores.

Ejemplo El ejemplo siguiente desplaza un círculo por la pantalla en respuesta a los TimerEvent. El método **pushButton** de *botónDeIntercambio* utiliza **setTimer** y **killTimer** para iniciar o detener un temporizador, dependiendo de la condición del botón. Cuando se inicia el temporizador, emite un TimerEvent cada 100 milisegundos. Cada TimerEvent provoca la ejecución del método **timer** de *botónDeIntercambio*. El método **timer** obtiene la posición actual de la elipse mediante **getPosition** y, entonces, la desplaza 100 twips a la derecha mediante **setPosition**.

El código siguiente corresponde al método **pushButton** de *botónDeIntercambio*:

```
; botónDeIntercambio::pushButton
method pushButton(var eventInfo Event)
if EtiquetaBotón = "Activar Temporizador" then      ; si está
desactivado,
                                    ; lo activamos
  EtiquetaBotón = "Desactivar Temporizador"         ; cambiamos
la etiqueta
  self.setTimer(10)                                 ; activamos
el Temporizador
else                                                ; si está
desactivado,
                                    ; lo activamos
  EtiquetaBotón = "Activar Temporizador"            ; cambiamos
la etiqueta
  self.killTimer()                                  ; detenemos
el temporizador
endif

endmethod
```

Este es el código del método **timer** de *botónDeIntercambio*:

```
; botónDeIntercambio::timer
method timer(var eventInfo TimerEvent)
```

```
var
  ui          UIObject
  x, y, w, h  SmallInt
endVar

ui.attach(floatCircle)                  ; asociamos al círculo
ui.getPosition(x, y, w, h)              ; asignamos las
coordenadas a variables
if x = 4320 then                        ; si no estamos en el
borde izquierdo del área
  ui.setPosition(x + 100, y, w, h)      ; nos desplazamos a la
izquierda
else
  ui.setPosition(1440, y, w, h)         ; volvemos a la derecha
endif

endmethod
```

Vea también killTimer

skip

Método Avanza o retrocede un número especificado de registros en una tabla.

Tipo UIObject

Sintaxis **skip (** const ***nRegistros*** LongInt **)** Logical

Descripción Define como registro actual el registro situado a *nRegistros* del actual. Si *skip* intenta ir más allá de los límites de la tabla, se obtiene un error.

Los valores positivos de *nRegistros* avanzan en la tabla (*nRegistros* = 1 equivale a **nextRecord**), mientras que los negativos retroceden (*nRegistros* = 1 equivale a **priorRecord**); un valor de 0 no produce ningún desplazamiento (*nRegistros* = 0 equivale a **currRecord**).

Ejemplo El ejemplo siguiente rellena una tabla con un muestreo de registros de la tabla *Pedidos*. Supóngase que la tabla *PediEjem* ya existe con la misma estructura que*Pedidos*. El botón *CreaciónDeEjemplos*, cuyo método **pushButton** se muestra a continuación, existe en una ficha junto con un marco de tabla asociado con *Pedidos*. El método desplaza el puntero por la tabla *Pedidos*, avanza o retrocede un número aleatorio de registros y copia el registro en que se sitúa a la tabla de muestreo.

```
; creaciónDeEjemplos::pushButton
method pushButton(var eventInfo Event)
var
  EjemploTC              TCursor        ; puntero a la tabla
ejemplo
  CopiaRegistro Array[] String          ; contiene un registro
copiado de Pedidos
  AzarEntero             SmallInt       ; número al azar al que
desplazarse
  ObjPedidos             UIObject       ; puntero a Pedidos
endVar

ObjPedidos.attach(PEDIDOS)              ; conectamos PEDIDOS a
la tabla
ObjPedidos.home()                       ; saltamos al primer
registro
if EjemploTC.open("PediEjem.db")  then
  EjemploTC.empty()                     ; vaciamos la tabla
ejemplo
  EjemploTC.edit()                      ; activamos el modo de
Edición
  while NOT ObjPedidos.atLast()
    AzarEntero = int(rand() * 20) + 1   ; obtenemos un entero
entre el 1 y el 20
    AzarEntero.view()                           ; mostramos el
número obtenido
   ObjPedidos.skip(AzarEntero)                  ; saltamos un
número de registros al azar
    ObjPedidos.copyToArray(CopiaRegistro)       ; obtenemos el
registro
    EjemploTC.insertRecord()                    ; abrimos un
espacio para él
```

```
   EjemploTC.copyFromArray(CopiaRegistro)     ; insertamos el
registro
  endwhile                          ; finalizamos el modo de
Edición
  msgInfo("Estado", "La tabla PediEjem tiene ahora " +
          String(EjemploTC.nRecords()) + " registros.")
  EjemploTC.close()                                  ; cerramos
la tabla
else
  msgStop("Lo siento", "No encuentro la tabla PediEjem.")
endif

endmethod
```

Vea también

home
end
nextRecord
priorRecord
currRecord
moveToRecord

switchIndex

Método Especifica otro índice para su uso en la visualización de los registros de una tabla.

Tipo UIObject

Sintaxis **1. switchIndex ([** const ***nombreIndice*** String][, const ***permanecerEnRegistro*** Logical **])** Logical

2. switchIndex ([const ***nombreArchivoIndice*** String **[,** const ***nombreEtiqueta*** String **]] [,** const ***permanecerEnRegistro*** Logical **])** Logical

Descripción Especifica en *nombreIndice* un archivo de índice para su uso con una tabla. En la sintaxis 1, *nombreIndice* especifica un índice para su uso con una tabla de Paradox. La sintaxis 2 está destinada a las tablas de dBASE, donde *nombreArchivoIndice* puede especificar un archivo .NDX o .MDX, y un argumento optativo *nombreEtiqueta* especifica una etiqueta de índice de un archivo de índice de producción (.MDX).

En ambas sintaxis, si el argumento optativo *permanecerEnRegistro* es Yes, este método mantiene el registro actual después del cambio de índice; si es No, el primer registro de la tabla se convierte en el actual. Si se omite, *permanecerEnRegistro* es No por defecto.

Ejemplo En este ejemplo, supóngase que *Clientes* es un tabla de Paradox con clave que tiene un índice secundario llamado "NombreYEstado". Este ejemplo se anexa a un marco de tabla asociado con *Clientes* y ejecuta **switchIndex** para cambiar del índice primario al índice "NombreYEstado".

```
; esteBotón::pushButton
method pushButton(var eventInfo Event)
var
  TablaObj UIObject
endvar

TablaObj.attach(CLIENTES)                              ; conectamos
con Clientes
TablaObj.switchindex("NombreYEstado")          ; cambiamos al
índice NombreYEstado
TablaObj.home()                                        ; nos aseguramos
de que se está en
                       ; el primer registro
msgInfo("Primer Registro", TablaObj.Nombre)    ; mostramos el
valor del campo
                       ; Nombre

endmethod
```

Vea también setFilter
TCursor::reIndex
TCursor::reIndexAll
Table::setIndex

twipsToPixels

Método	Convierte las coordenadas de pantalla de twips a pixels.
Tipo	UIObject
Sintaxis	**twipsToPixels (** const ***twips*** Point **)** Point
Descripción	Convierte las coordenadas de pantalla especificadas en *twips* de twips a pixels. Un pixel es un punto en la pantalla, mientras que twip es una unidad independiente del dispositivo que es igual a 1/1440 de pulgada (1/20 de un punto de la impresora).
Ejemplo	Consulte el ejemplo de pixelsToTwips.**.**
Vea también	pixelsToTwips

unDeleteRecord

Método Anula el borrado del registro actual de una tabla de dBASE.

Tipo UIObject

Sintaxis **unDeleteRecord ( )** Logical

Descripción Anula el borrado del registro actual de una tabla de dBASE. Esta operación sólo puede ser satisfactoria si **showDeleted** se ha definido como True, el registro actual es un registro borrado y el objeto de tabla está en modo editar.

Vea también deleteRecord
isRecordDeleted
TCursor::isShowDeletedOn
Table::showDeleted

unlockRecord

Principiante

Método Elimina un bloqueo de escritura del registro actual.

Tipo UIObject

Sintaxis **unlockRecord ( )** Logical

Descripción Devuelve True si logra eliminar un bloqueo de escritura explícito del registro actual; en caso contrario, devuelve False.

Nota: También es posible examinar la propiedad Locked para averiguar si un registro está bloqueado, pero no puede cambiarse dicha propiedad para bloquear o desbloquear un objeto. La propiedad Locked es de sólo lectura.

Ejemplo Consulte el ejemplo de recordStatus.

Vea también
recordStatus
lockRecord
TCursor**::**attachToKeyViol
TCursor**::**didFlyAway
TCursor**::**setFlyAwayControl

view

Principiante

Método Muestra el valor de un objeto en un cuadro de diálogo.

Tipo UIObject

Sintaxis **view ([** const ***título*** String **])**

Descripción Muestra el valor de un objeto en un cuadro de diálogo. La ejecución de ObjectPAL se suspende hasta que el usuario cierra este cuadro de diálogo. Existe la opción de especificar, en *título*, un título para el cuadro de diálogo. Si no se especifica *título*, el título es el tipo de datos del valor.

Este método sólo funciona con los siguientes UIObject:

Botones como casillas de verificación o botones de radio

Campos no asociados sólo como listas o botones de radio

Campos asociados con una tabla; el tipo de datos del campo puede ser cualquiera excepto Memo e Imagen

La ejecución de **view** con otro UIObject provoca un error durante la ejecución.

Ejemplo Para este ejemplo, supóngase que una ficha contiene el marco de tabla *CLIENTES* asociado con la tabla *Clientes* y un botón. El código siguiente se anexa al método **pushButton** del botón. Crea una matriz de siete UIObjectS, e intenta mostrar cada elemento de la matriz.

```
; pagina::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  obj          UIObject
  arr Array[7] UIObject
  i            SmallInt
endVar
arr[1].attach(CLIENTES.Teléfono)  ; el campo Teléfono (A15) en
un marco de tabla

          ; mostramos el número de teléfono
arr[2].attach(UnGráfico)               ; un mapa de bit (no
válido)
arr[3].attach(UnTexto)                 ; un objeto de texto (no
válido)
arr[4].attach(UnaLista)                ; una lista de campos
sin determinar
          ; mostramos la lista del elemento seleccionado

arr[5].attach(UnCampo)            ; un campo sin determinar (no
válido)
arr[6].attach(UnBotónDeSelección)      ; un campo sin
determinar como un Botón
          ; de selección

          ; mostramos el valor del Botón de selección activo
```

```
arr[7].attach(UnBotón)                  ; un campo sin
determinar como Botón
          ; de comprobación

          ; mostramos True si está activado,
          ; False si no lo está.
for i from 1 to arr.size()
  arr[i].view(arr[1].Class + ": Item " + String(i))
endFor

endmethod
```

Vea también attach

wasLastClicked

Método Indica si un objeto fue el último objeto en recibir un clic del ratón.

Tipo UIObject

Sintaxis **wasLastClicked ( )** Logical

Descripción Devuelve True si un objeto fue el último objeto que recibió un clic del ratón; en caso contrario, devuelve False. Este método sólo puede emplearse con objetos de la ficha actual.

Ejemplo El código siguiente se anexa al método **mouseUp** de un objeto llamado *CajaUno*. Si *CajaUno* recibe el clic, aparece el mensaje; si se envió un suceso **mouseUp** a *CajaUno* desde otro objeto, el método emite una señal sonora en lugar de mostrar el mensaje.

El código siguiente corresponde al método **mouseUp** de *CajaUno*:

```
; cajaUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
if self.wasLastClicked() then
  msgInfo("", "Terminar haciendo clic sobre mí.")   ; se invocó
el método
                                    ; mediante un clic
else
  beep()                                             ; se invocó
el metodo
                                    ; indirectamente
endif

endmethod
```

Este es el código del método **mouseUp** de *enviarUnClic*:

```
;enviarUnClic::mouseUp method mouseUp(var eventInfo MouseEvent)
cajaUno.mouseUp(eventInfo)  ; cuando mouseUp de la cajaUno
obtenga esto                              ; sonará un pitido
endmethod
```

Vea también wasLastRightClicked
hasMouse

wasLastRightClicked

Método Indica si un objeto fue el último objeto en recibir un clic del botón derecho del ratón.

Tipo UIObject

Sintaxis **wasLastRightClicked ( )** Logical

Descripción Devuelve True si un objeto fue el último objeto que recibió un clic del botón derecho del ratón; en caso contrario, devuelve False. Este método sólo puede emplearse con objetos de la ficha actual.

Ejemplo El código siguiente se anexa al método **mouseRightUp** de un objeto llamado *CírculoUno*. Si la elipse recibió el clic derecho, se muestra el mensaje; si se envío un suceso **mouseRightUp** a la elipse desde otro objeto, el método emite una señal sonora en lugar de mostrar el mensaje.

Este es el código del método **mouseUp** de *CírculoUno*:

```
; círculoUno::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
if self.wasLastRightClicked() then
  ; se invocó el método pulsando el botón derecho del ratón
  msgInfo("Clic en el botón derecho", "Pulsa en alguien de tu
tamaño.")
else
  beep()        ; se invocó el método indirectamente
endif

endmethod
```

El código siguiente corresponde al método **mouseUp** de *enviarUnClicDelBotónDerecho*:

```
; enviarUnClicDelBotónDerecho::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
círculoUno.mouseRightUp(eventInfo)                ; cuando el
círculoUno obtenga esto,                    ; sonará un pitido

endmethod
```

Vea también wasLastClicked
hasMouse

action

Método/ Procedimiento Ejecuta un comando de acción.

Tipo Form

Sintaxis **action (** const ***idAcción*** SmallInt **)** Logical

Descripción Realiza la acción representada por la constante *idAcción*. Este método **action** construye y envía un ActionEvent al método estándar **action** del objeto que el programador especifique. ObjectPAL proporciona constantes para *idAcción*; consulte una de las categorías que comienzan con Action en el cuadro de diálogo Constantes.

También puede utilizarse **action** para enviar una constante de acción definida por el usuario a un método estándar **action**. Las constantes de acción definidas por el usuario son simplemente números enteros que no interfieren con ninguna constante de ObjectPAL, y pueden emplearse para señalar a otras partes de una aplicación. Por ejemplo, supóngase que la ventana Const de una ficha declara una constante llamada *miAcción*. En el método estándar **action** de una página de la ficha, podría comprobarse el valor de todos los ActionEvent que entran (con el método **id**); si el valor es igual a *miAcción*, puede responderse a esa acción correspondientemente. La respuesta por defecto de Paradox para las constantes de acción definidas por el usuario es pasar la acción al método **action**. Para más información sobre la definición de constantes, consulte el Capítulo 6 de la *Guía del Programador ObjectPAL*.

Este método **action** es distinto del método estándar **action** de una ficha o de otro UIObject. El método estándar **action** de un objeto *responde* a un suceso de acción; este método *provoca* un suceso de acción.

Nota: Cuando se ejecuta el método **action** como un procedimiento, ObjectPAL debe realizar una conjetura educada sobre a qué objeto debe afectar la acción. El suceso se lanza por la jerarquía de contenedores hasta que llegue a un contenedor que pueda gestionar la acción o hasta que el suceso lleguea la ficha. Si el suceso llega a la ficha, y la acción es una acción sobre datos, la ficha envía el suceso a la tabla principal de la ficha.

Ejemplo En este ejemplo, una ficha llamada *Lugares* contiene objetos de campo asociados con la tabla *Lugares*. La ficha actual contiene un botón llamado *AbrirYEditarLugares*; el método **pushButton** de *AbrirYEditarLugares* abre *Lugares*, inicia el modo editar y espera a que se cierre *Lugares*:

```
; AbrirYEditarLugares::pushButton
method pushButton(var eventInfo Event)
var
  Ficha   Form
endVar
Ficha.open("Lugares.fsl")              ; abrimos Lugares
Ficha.action(DataBeginEdit)            ; iniciamos el modo de
Edición para Ficha
message("Para volver, cierre la Ficha Lugares.")
Ficha.wait()                           ; esta Ficha estará
inactiva hasta que
          ; Lugares vuelva
```

```
Ficha.close()                          ; esta sentencia debe
cerrar Lugares

endmethod
```

Vea también UIObject::action

attach

Método Asocia una variable Form con una ficha abierta.

Tipo Form

Sintaxis **attach ([** const ***nombreFicha*** String **])** Logical

Descripción Asocia una variable Form con una ficha abierta. Es posible utilizar *nombreFicha* para especificar el título actual de una ficha u omitir *nombreFicha* para realizar la asociación con la ficha en que se esté ejecutando **attach**.

Ejemplo En este ejemplo, una ficha tiene dos botones: *AbrirLugares* y *AsociarConLugares*. El método **pushButton** de *AbrirLugares* se ocupa de abrir la ficha *Lugares*. El método **pushButton** de *AsociarConLugares* asocia la variable Form *Ficha* con la ficha abierta por medio del título actual de la ficha. En este caso, el título de la ficha no se ha cambiado, por lo que el botón *AsociarConLugares* puede asociarse con *Lugares* mediante el título por defecto. Una vez asociado, el método **pushButton** utiliza el gestor *Ficha* para minimizar, maximizar y restaurar *Lugares*.

El código siguiente se anexa al método **pushButton** de *AbrirLugares*:

```
; AbrirLugares::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.open("Lugares.fsl")         ; abrimos Lugares, el título
          ; por defecto será "Form : Lugares.FSL"

endmethod
```

Este código se anexa al método **pushButton** de *AsociarConLugares*:

```
; AsociarConLugares::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.attach("Form : Lugares.FSL")      ; conecta con Lugares
por su título

; Nótese que esto no funcionará:  Ficha.attach("Lugares")

; ciclo de tamaños
Ficha.minimize()                        ; minimiza la Ficha
sleep(2000)                             ; pausa
Ficha.maximize()                        ; maximiza la Ficha
sleep(2000)                             ; pausa
Ficha.show()                            ; restaurar el tamaño
original

endmethod
```

Vea también getTitle

setTitle
open

bringToTop

Principiante

Método/ Procedimiento Sitúa la ventana en la parte superior de la pila de visualización y la activa.

Tipo Form

Sintaxis **bringToTop ( )**

Descripción Cuando se muestran varias ventanas, parecen solaparse dando un aspecto de capas. Utilice **bringToTop** para mostrar una ventana sobre la pila, sin que quede solapada por otras ventanas. **bringToTop** convierte una ficha en la ventana activa.

Si una sentencia **hide** ha hecho invisible una ficha, **bringToTop** la hace visible de nuevo.

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *AbrirVarios* abre la ficha *Lugares* y, a continuación, una ventana de tabla para la tabla *Pedidos*. La ventana de tabla, *PedidoTV*, se abre sobre la ficha *Lugares*, *Ficha*. El método detiene su ejecución durante unos segundos y convierte *Ficha* en capa superior:

```
; abrirVarios::pushButton
method pushButton(var eventInfo Event)
var
  Ficha     Form
  PedidoTV  TableView
endVar
Ficha.open("Lugares.fsl")         ; abrimos la Ficha Lugares
PedidoTV.open("Pedidos")          ; abrimos Pedidos sobre
Lugares
message("Intentamos hacer que la Ficha Lugares sea la ventana
más destacada.")
beep()
sleep(5000)           ; pausa
Ficha.bringToTop()    ; hacemos de Lugares la ventana más
destacada

endmethod
```

Vea también isVisible
hide
show

close

Principiante

Tipo Form

Método/ Procedimiento Cierra una ventana.

Sintaxis **1. (** Método) **close ( )**

2. (Procedimiento) **close** ([const ***valorDevuelto*** AnyType **])**

Descripción Cierra una ficha como si el usuario hubiese elegido Cerrar en el menú de control. Cuando se ejecuta **close** como procedimiento, es posible especificar el valor que se devuelve en *valorDevuelto*. El valor se devuelve a la ficha que ha realizado la llamada, si hay una.

Ejemplo En este ejemplo, una ficha contiene un botón llamado *BotónDeVolver*. La ficha *Lugares* contiene un botón llamado *BotónDeVolver*. El método **pushButton** de *BotónDeVolver* utiliza **formReturn** para devolver el control a la ficha que ha realizado la llamada:

```
; BotónDeVolver::pushButton
method pushButton(var eventInfo Event)
formReturn()    ; devuelve el control al objeto o Ficha que
hizo la llamada
endmethod
```

El código siguiente se anexa al método **pushButton** de un botón de la ficha actual llamado *AbrirYCerrar*. Este método abre la ficha *Lugares* como *Ficha* y espera a que vuelva. Una vez que *Ficha* vuelve (porque el usuario ha hecho clic sobre *BotónDevolver*), este método muestra un mensaje, realiza una pausa y cierra *Ficha*:

```
; AbrirYCerrar::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.open("Lugares")
message("Para regresar, pulse el botón Volver.")
Ficha.wait()                ; espera a que se vuelva a la Ficha
msgInfo("estado", "Cerraré la Ficha Lugares en tres segundos.")
sleep(3000)                 ; pausa
Ficha.close()               ; cierra la Ficha Lugares

endmethod
```

Vea también open

create

Método Crea una ficha vacía en una ventana de diseño.

Tipo Form

Sintaxis **create ( )** Logical

Descripción Crea una ficha vacía y la deja en la ventana Diseñar ficha. Es posible utilizar los métodos **create** y **methodSet** del tipo UIObject para situar objetos en la nueva ficha y anexar métodos a ella. Utilice **run** para ejecutar una ficha.

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *CrearUnaFicha* crea una nueva ficha mediante el método **create** y define el valor del método **mouseUp** de la nueva ficha mediante **setMethod**. Entonces, el método **pushButton** de *CrearUnaFicha* guarda la nueva ficha en un archivo llamado SALUDO.FSL, ejecuta la ficha y activa el método **mouseUp** de la nueva ficha (suministrando los argumentos correctos). El método **mouseUp** de la ficha *Saludo* abre un cuadro de diálogo que muestra "Hola". Una vez que el usuario cierra el cuadro de diálogo, el método **pushButton** de *CrearUnaFicha* cierra la ficha *Saludo*.

```
; CrearUnaFicha::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.create()              ; creamos una nueva Ficha vacía (una
ventana de diseño)
Ficha.methodSet("mouseUp",        ; asigna el método mouseUp
para la Ficha
"method mouseUp(var eventInfo MouseEvent)
msgInfo(\"Saludos\", \"Hola\")
endMethod")                       ; la barra delimita las
comillas escritas
Ficha.save("Saludo")        ; salvamos la Ficha
Ficha.run()                       ; ejecutamos la nueva Ficha
(ventana de Ver Datos)
                                  ; llamada al método mouseUp
para la Ficha
Ficha.mouseUp(100, 100, LeftButton )    ; el cuadro de diálogo
muestra "Hola"
Ficha.close()                     ; cerramos la Ficha
endmethod
```

Vea también design
load
open
UIObject::create
UIObject::methodGet
UIObject::methodSet

delayScreenUpdates

Procedimiento Activa o desactiva las actualizaciones de pantalla demoradas.

Tipo Form

Sintaxis **delayScreenUpdates (** const ***yesNo*** Logical **)**

Descripción Aplaza la actualización de las áreas de la pantalla pero sin bloquear la pantalla. Es necesario especificar Yes o No. Si especifica Yes, aplaza las actualizaciones de pantalla hasta que termine una operación. Si especifica No, las actualizaciones de pantalla se producen normalmente.

En algunas operaciones, no se notará ninguna diferencia cuando **delayScreenUpdates** está definido como Yes, especialmente cuando la aplicación se ejecute en un ordenador rápido.

Ejemplo Los métodos siguientes anulan los métodos **pushButton** de sus botones respectivos. El botón *DibujarUnoAUno* dibuja cierto número de cuadros sin cambiar **delayScreenUpdates**. El botón *DibujarTodosALaVez* dibuja el mismo número de cuadros, en otra posición, pero primero define **delayScreenUpdates** como Yes. Si ejecuta este código, observará que los cuadros creados por *DibujarUnoAUno* aparecen uno cada vez, pero con bastante rapidez. Los creados por *DibujarTodosALaVez* se dibujan detrás de la escena –lo cual provoca una breve pausa– y aparecen todos al mismo tiempo.

```
; DibujarUnoAUno::pushButton
method pushButton(var eventInfo Event)
var
  ui UIObject
endVar

; por defecto está delayScreenUpdates(No)
; creamos y mostramos un conjunto de cuadros, mostrándolos como
; fueron creados.
for i from 750 to 2550 step 300
  for j from 750 to 2550 step 300
    ui.create(boxTool, i, j, 150, 150)
    ui.Color = Blue
    ui.Visible = Yes
  endfor
endfor
endmethod
```

El botón *DibujarTodosALaVez* de la misma ficha crea el mismo número de cuadros, pero lo hace estando **delayScreenUpdates** definido como Yes. En ordenadores muy rápidos, posiblemente tampoco pueda apreciarse la diferencia.

```
; DibujarTodosALaVez::pushButton
method pushButton(var eventInfo Event)
var
  ui UIObject
endVar
```

```
delayScreenUpdates(Yes)
; Este código fuente creará todos los cuadros, luego los
muestra
; todos a la vez.
for i from 4950 to 6750 step 300
  for j from 750 to 2550 step 300
    ui.create(boxTool, i, j, 150, 150)
    ui.Color = Red
    ui.Visible = Yes
  endfor
endfor
; restaurar los datos por defecto
delayScreenUpdates(No)

endmethod
```

Vea también System::sleep

deliver

Método Envía una ficha.

Tipo Form

Sintaxis **deliver ( )** Logical

Descripción Se comporta como Ficha|Enviar. Este método guarda una copia de una ficha con la extensión .FDL, impidiendo así que los usuarios editen la ficha en la ventana Diseñar ficha. Los usuarios sólo pueden abrir la ficha en una ventana Ficha. También está prohibido que cambien una ficha abierta y enviada a la ventana Diseñar ficha. Para más información sobre el almacenamiento y envío de fichas, consulte la *Guía del Usuario*.

Ejemplo En este ejemplo, el botón *CrearDeliver* crea una nueva ficha, la guarda con el nombre Saludo y la envía (lo cual almacena otra versión como SALUDO.FDL). Cuando el método intenta cargar la ficha en una ventana Diseñar ficha, **load** devuelve False.

```
; CrearDeliver::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.create()              ; creamos una nueva Ficha vacía (una
ventana de diseño)
Ficha.save("Saludo")              ; salvamos la Ficha
Ficha.deliver()             ; definimos la Ficha recien creada
Ficha.close()               ; cerramos la Ficha
if NOT Ficha.load("Saludo.fdl") then    ; la carga proporciona
el dato False
  msgStop("Lo siento", "No puedo cargar la Ficha definida.")
endif

endmethod
```

Vea también save

design

Método Cambia una ficha en ejecución a la ventana Diseñar ficha.

Tipo Form

Sintaxis **design ( )** Logical

Descripción Cambia una ficha en ejecución a la ventana Diseñar ficha. Este método sólo funciona con fichas guardadas (.FSL), no con fichas enviadas (.FDL). Para más información sobre el almacenamiento y envío de fichas, consulte la *Guía del Programador ObjectPAL*.

Utilice **run** para ejecutar una ficha desde la ventana Diseñar ficha.

Nota Algunas acciones sobre fichas requieren un uso intensivo del procesador. En algunas situaciones, puede ser necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el método **sleep** del tipo System.

Ejemplo Consulte el ejemplo de create.

Vea también create
open
run

disableBreakMessage

Procedimiento Impide la interrupción del programa mediante *Ctrl+Pausa*.

Tipo Form

Sintaxis **disableBreakMessage (** const ***yesNo*** Logical **)** Logical

Descripción Impide o permite que el usuario interrumpa la ejecución de un programa mediante *Ctrl+Pausa*.

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla asociado con la tabla *Pedidos*. El código siguiente impide que el bucle se interrumpa mediante *Ctrl+Pausa*.

```
; aTravesDeLaTabla::pushButton
method pushButton(var eventInfo Event)
; sólo un bucle para comprobar
Ctrl+Interrumpir
disableBreakMessage(Yes)                    ; no se permite
Ctrl+Interrumpir
while NOT PEDIDOS.atLast()
  PEDIDOS.action(DataNextRecord)
endwhile
endmethod
```

dmAddTable

Método/ Procedimiento Añade una tabla al modelo de datos de una ficha.

Tipo Form

Sintaxis **dmAddTable (** const ***nombreTabla*** String **)** Logical

Descripción Añade la tabla *nombreTabla* al modelo de datos de una ficha.

Ejemplo En este ejemplo, una ficha contiene un botón llamado *IntercambiarLugares*, y un campo de lista llamado *MostrarNombresDeLugares*. Los datos de la lista para el campo *MostrarNombresDeLugares* se definen con la propiedad DataSource de su objeto de lista, *ListaNombres*. El método **pushButton** de *IntercambiarLugares* comprueba si la tabla *Lugares* está en el modelo de datos de la ficha. Si es así, la referencia a *Lugares* se elimina de la propiedad DataSource de *ListaNombres*; a continuación, se elimina *Lugares* del modelo de datos. En caso contrario, la tabla *Lugares* se añade al modelo de datos, y la propiedad DataSource de *ListaNombres* se define como el campo *Nombre del Lugar* de *Lugares* (para referenciar el campo de una tabla en la propiedad DataSource de una lista, la tabla referenciada debe formar parte del modelo de datos de la ficha).

Este es el código para el método **pushButton** de *IntercambiarLugares*:

```
; IntercambiarLugares::pushButton
method pushButton(var eventInfo Event)
; Llevamos Lugares.db dentro y fuera del modelo de datos
if dmHasTable("Lugares") then     ; ¿Está Lugares en el modelo
de datos?
  ; si es así, eliminamos las dependencias, luego eliminamos la
tabla
  ; eliminamos Lugares como fuente de
MostrarNombresDeLugares.ListaNombres
  MostrarNombresDeLugares.ListaNombres.DatosOrigen = ""
  MostrarNombresDeLugares.Visible = False
  ; eliminar Lugares del modelo de datos
  dmRemoveTable("Lugares")
  whichTable = ""
else
  ; si no está preparada en el modelo de datos, añadimos
Lugares
   dmAddTable("Lugares")          ; asignamos los datos para la
lista de
    ;la tabla Lugares
  MostrarNombresDeLugares.ListaNombres.DatosOrigen =
"[Lugares.Nombre
                                     del Lugar]"
  MostrarNombresDeLugares.Visible = True
  whichTable = "Lugares"
endIf

endmethod
```

Vea también dmHasTable

dmRemoveTable

dmGet

Método/ Procedimiento Recupera un valor de campo de una tabla del modelo de datos.

Tipo Form

Sintaxis **dmGet (** const ***nombreTabla*** String, const ***nombreCampo*** String,

var ***datoD*** AnyType **)** Logical

Descripción Proporciona acceso a los datos de una tabla del modelo de datos de una ficha. **dmGet** escribe en *dato* el valor de un campo de una tabla especificada. La tabla especificada por *nombreTabla* debe ser una de las existentes en el modelo de datos de la ficha. *nombreCampo* debe ser un campo de *nombreTabla*.

Ejemplo En el ejemplo siguiente, una ficha contiene un marco de tabla asociado con la tabla *Lugares*. El marco de tabla contiene sólo dos campos: Lugar y Nombre del Lugar. El método **pushButton** de un botón llamado *obtenCaracterísticas* utiliza **dmGet** para encontrar el valor del campo Características Lugar en el registro actual. Entonces, el método muestra el valor de Características Lugar en un cuadro de diálogo y pregunta al usuario si desea cambiar el valor. Si el usuario responde "Sí" en el cuadro de diálogo, el método muestra el valor original de Características Lugar en un cuadro de diálogo y solicita un nuevo valor al usuario. A continuación, el método emplea **dmPut** para escribir el valor cambiado en la tabla *Lugares*:

```
; obtenCaracterísticas::pushButton
method pushButton(var eventInfo Event)
var
  Característica AnyType
  qRespuesta     String
endVar
; obtenemos el valor del campo Característica para el registro
actual
if dmGet("Lugares", "Características", Característica) then
  ; muestra las características y pregunta al usuario si desea
cambiarlas
  qRespuesta = msgQuestion("¿Cambio el principal?",
                           "En el lugar " +
Lugares.Nombre_del_lugar +
                           " el principal es " +
                           String(Carcaterística) + ". Cambio
las
                           características?")
  if qRespuesta = "Yes" then
    ; comprobar el modo de Edición
    if estaFicha.Editing = True then
      action(DataBeginEdit)
    endif
    ; pregunta al usuario si desea reemplazar el valor de las
características
    ; existentes en el cuadro de diálogo View
  Característica.view("Escribir una nueva Característica:")
  ; escribir la característica cambiado de nuevo en el campo
característica
```

```
  dmPut("Lugares", "Características", Característica)
  endif
else
  msgStop("Lo siento", "No encuentro las características.")
endif
endmethod
```

Vea también dmPut

dmHasTable

Método/ Procedimiento Informa de si una tabla es parte del modelo de datos de una ficha.

Tipo Form

Sintaxis **dmHasTable (** const ***nombreTabla*** String **)** Logical

Descripción Informa de si *nombreTabla* es una tabla asociada con una ficha.

Ejemplo Para una ilustración de cómo emplear **dmHasTable** como procedimiento, consulte el ejemplo de **dmAddTable**.

Este ejemplo muestra cómo se utiliza **dmHasTable** como método. El método **pushButton** de un botón llamado *EstáExistencEnMD* trabaja con la ficha especificada en la variable *Ficha*. Este método abre la ficha *IntPedid* y comprueba si la tabla *Existenc* está en el modelo de datos de *Ficha*. Si no está, la tabla *Existenc* se añade al modelo de datos de *Ficha*:

```
; EstáExistencEnMD::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.load("IntPedid")                                ; abrimos la
Ficha IntPedid
if not Ficha.dmHasTable("Existenc") then      ; está Existenc en
modelo de datos
  msgInfo("Estado", "Añadiendo Existenc al modelo de datos para
la Ficha.")
  Ficha.dmAddTable("Existenc")                ; si no, la
añadimos
  Ficha.save()
else
  msgInfo("Estado", "Existenc ya está en modelo de datos para
la Ficha.")
endif
Ficha.close()

endmethod
```

Vea también dmAddTable
dmRemoveTable

dmPut

Método/ Procedimiento Escribe datos en una tabla del modelo de datos de una ficha.

Tipo Form

Sintaxis **dmPut (** const ***nombreTabla*** String, const ***nombreCampo*** String, const ***dato*** AnyType **)** Logical

Descripción Proporciona acceso a los datos de una tabla del modelo de datos de una ficha. **dmPut** escribe *dato* en un campo de una tabla específica. La tabla especificada en *nombreTabla* debe ser una de las tablas del modelo de datos de la ficha. *nombreCampo* debe ser un campo de *nombreTabla*.

Ejemplo Consulte el ejemplo de dmGet.

Vea también dmGet

dmRemoveTable

Método/ Procedimiento Elimina una tabla del modelo de datos de una ficha.

Tipo Form

Sintaxis **dmRemoveTable (** const ***nombreTabla*** String **)** Logical

Descripción Elimina *nombreTabla* de la lista de tablas asociadas con una ficha. Los objetos de la ficha que dependan de la tabla quedarán indefinidos cuando se elimine la tabla.

Ejemplo Consulte el ejemplo de dmAddTable.

Vea también dmAddTable
dmHasTable

enumSource

Método/ Procedimiento Crea una tabla que enumera los métodos existentes para cada objeto de una ficha.

Tipo Form

Sintaxis **enumSource (** const ***nombreTabla*** String **[,** const ***recursivo*** Logical **])** Logical

Descripción Crea una tabla de Paradox que enumera todos los objetos para los que se ha escrito un método, junto con el código fuente de ObjectPAL para el método. Utilice el argumento *nombreTabla* para especificar un nombre para la tabla. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* ya está abierta, este método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de la tabla que se crea es:

Nombre de campo	Tipo	Tamaño
Object	A	128
MethodName	A	128
Source	M	64

El campo Object contiene la vía de acceso completa del objeto.

Si *recursivo* es False, este método devuelve las definiciones de los métodos de la ficha solamente. Para incluir el código fuente de los métodos de todos los objetos contenidos en la ficha, *recursivo* debería ser True.

Ejemplo En este ejemplo, una ficha contiene un botón llamado *obtenOrigen*. El método **pushButton** de *obtenOrigen* utiliza **enumSource** como procedimiento para enumerar el código fuente de la ficha actual en una tabla llamada ORIGTEMP.DB. A continuación, el método abre una ventana de tabla para la tabla *Origtemp* y espera a que el usuario la cierre. Entonces, el método abre la ficha *Lugares* para *Ficha*, utiliza **enumSource** como método para escribir el código fuente de *Ficha* en una tabla llamada ORIGLUGR.DB y muestra la tabla:

```
; obtenOrigen::pushButton
method pushButton(var eventInfo Event)
var
  Ficha  Form
  Tabla  TableView
endVar
;enumSource("origtemp.db", True)  ; escribe todo el origen para
la
          ; Ficha actual en ORIGTEMP.DB
;Tabla.open("origtemp.db")
;Tabla.wait()
Ficha.open("Lugares.fsl")         ; abrimos otra Ficha
; escribimos el origen de Ficha en ORIGTEMP.DB
Ficha.enumSource("origtemp.db", True)
Ficha.close()                     ; cerramos la Ficha
```

```
Tabla.open("origlugr.db")          ; vemos la nueva tabla
Tabla.wait()                       ; esperamos a que el usuario
cierre
          ; la tabla
endmethod
```

Vea también enumUIObjectNames
enumUIObjectProperties
enumSourceToFile

enumSourceToFile

Método/ Procedimiento Crea un archivo que enumera los métodos de cada objeto de una ficha.

Tipo Form

Sintaxis **enumSourceToFile (** const ***nombreArchivo*** String **[,** const ***recursivo*** Logical **])** Logical

Descripción Crea un archivo de texto que enumera cada objeto para el que se ha escrito un método, junto con el código fuente de ObjectPAL para el método. Utilice el argumento *nombreArchivo* para especificar un nombre para el archivo. Si *nombreArchivo* ya existe, este método lo sustituye sin pedir confirmación. Es posible incluir un alias o vía de acceso en *nombreArchivo*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreArchivo* en el directorio de trabajo (:TRABAJO:).

Si *recursivo* es False, este método devuelve las definiciones de los métodos de la ficha solamente. Para incluir el código fuente de los métodos de todos los objetos contenidos en la ficha, *recursivo* debería ser True.

Ejemplo El código siguiente se anexa al método **pushButton** de un botón llamado *obtenOrigenParaArchivo*. Este método escribe todo el código fuente de la ficha actual en ORIGTEMP.TXT. Entonces, el método abre la ficha *Lugares* y escribe todo el código de esa ficha en un archivo llamado ORIGLUGR.TXT:

```
; ObtenOrigenParaArchivo::pushButton
method pushButton(var eventInfo Event)
var
  Ficha   Form
endVar
enumSourceToFile("origtemp.txt", True)  ; escribe todo el origen
para la
               ; Ficha actual en ORIGTEMP.TXT

Ficha.open("Lugares.fsl")               ; abrimos otra Ficha
; escribimos el origen de Ficha en ORIGLUGR.TXT
Ficha.enumSourceToFile("origlugr.txt", True)
Ficha.close()                           ; cerramos la Ficha

endmethod
```

Vea también enumUIObjectNames
enumUIObjectProperties

enumTableLinks

Método/ Procedimiento Crea una tabla que enumera las tablas enlazadas con una ficha.

Tipo Form

Sintaxis **enumTableLinks (** const ***nombreTabla*** String **)** Logical

Descripción Crea una tabla de Paradox que enumera los nombres de las tablas enlazadas con una ficha y los tipos de enlaces. Utilice el argumento *nombreTabla* para especificar un nombre para la tabla. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* ya está abierta, este método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de *nombreTabla* es:

Nombre de campo	Tipo	Tamaño
Name	A	81*
Link	A	81*
LinkType	A	24*

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *MostrarEnlacesDeTabla* escribe enlaces de tabla para la ficha actual en una tabla llamada ENLACTMP.DB. A continuación, el método abre la ficha *Lugares* y escribe los enlaces de tabla para esa ficha en una tabla llamada LUGARES.DB:

```
; MostrarEnlacesDeTabla::pushButton
method pushButton(var eventInfo Event)
var
  Ficha   Form
  Tabla   TableView
endVar
enumTableLinks("EnlacTmp.db")          ; lista los enlaces de
la Ficha actual
Tabla.open("EnlacTmp")
Tabla.wait()
Ficha.open("Lugares.fsl")
Ficha.enumTableLinks("Lugares.db")     ; lista los enlaces con
Lugares
Ficha.close()
Tabla.open("Lugares.fsl")
Tabla.wait()
Tabla.close()

endmethod
```

Vea también enumUIObjectNames
enumUIObjectProperties
TCursor::enumRefIntStruct

enumUIObjectNames

Método Crea una tabla que enumera los UIObject contenidos en una ficha.

Tipo Form

Sintaxis **enumUIObjectNames (** const ***nombreTabla*** String **)** Logical

Descripción Crea una tabla de Paradox que enumera el nombre y tipo de cada objeto contenido en una ficha. Utilice el argumento *nombreTabla* para especificar un nombre para la tabla. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* ya está abierta, este método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox crea *nombreTabla* en el directorio de trabajo (:TRABAJO:).

La estructura de *nombreTabla* es:

Nombre de campo	Tipo	Tamaño
ObjectName	A	128
ObjectClass	A	32

El campo ObjectName incluye toda la vía de acceso del objeto.

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *ObtenNombresDeObjeto* abre la ficha *Lugares* y enumera los nombres de todos los objetos de la ficha en una tabla llamada *ObjLugar*. Entonces, el método abre la tabla *ObjLugar* y espera a que el usuario la cierre:

```
; ObtenNombresDeObjeto::pushButton
method pushButton(var eventInfo Event)
var
  Ficha   Form
  Tabla   TableView
endVar
if Ficha.open("Lugares.fsl") then       ; abrimos la Ficha
Ficha.enumUIObjectNames("ObjLugar.db")  ; escribimos los nombres
de los objetos
                ; OBJLUGAR.DB
Ficha.close()                           ; cerramos la Ficha
Tabla.open("ObjLugar")                  ; abrimos la nueva tabla
Tabla.wait()                            ; esperar a que se
vuelva
Tabla.close()                           ; cerramos después de la
vuelta
endIf

endmethod
```

Vea también enumUIObjectProperties
enumSource
enumSourceToFile

enumUIObjectProperties

Método Crea una tabla que enumera las propiedades de cada UIObject contenido en una ficha.

Tipo Form

Sintaxis **enumUIObjectProperties (** const ***nombreTabla*** String **)** Logical

Descripción Crea una tabla de Paradox que enumera el nombre, nombre de propiedad y valor de propiedad de cada objeto contenido en una ficha. Utilice el argumento *nombreTabla* para especificar un nombre para la tabla. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* ya está abierta, este método falla.

La estructura de *nombreTabla* es:

Nombre de campo	Tipo	Tamaño
ObjectName	A	128
PropertyName	A	64
PropertyType	A	48
PropertyValue	A	255

Ejemplo En este ejemplo, el método **pushButton** de un botón llamado *ObtenPropiedades* escribe, en una tabla llamada *PropTemp*, las propiedades de todos los objetos contenidos en la ficha actual:

```
; ObtenPropiedades::pushButton
method pushButton(var eventInfo Event)
var
  Ficha  Form
  Tabla  TableView
endVar
if Ficha.open("Lugares.fsl") then
  message("Enumerando las propiedades en la tabla PropLugr.")
  Ficha.enumUIObjectProperties("PropLugr.db")
  Tabla.open("PropLugr")
  message("Cierre la tabla para continuar.")
  Tabla.wait()
  Tabla.close()
endIf
; para enumerar los objetos de la Ficha actual, utilizamos el
método
; enumUIObjectProperties de tipo UIObject
; estaFicha es el identificador del objeto de la Ficha actual
message("Enumerando las propiedades en la tabla PropTemp.")
estaFicha.enumUIObjectProperties("PropTemp.db")
Tabla.open("PropTemp")
message("Cierre la tabla para continuar.")
Tabla.wait()
Tabla.close()
endmethod
```

Vea también UIObject::enumUIObjectProperties

formCaller

Procedimiento Crea un gestor para la ficha que ha realizado la llamada.

Tipo Form

Sintaxis **formCaller (** var ***llamador*** Form **)** Logical

Descripción Asigna a *llamador* el gestor de la ficha que ha llamado a la ficha actual, si la ficha está en un **wait**. Si la ficha actual no fue abierta por otra ficha, y la ficha que abrió la ficha actual no espera a la ficha actual, el método devuelve False, y *llamador* queda sin asignación.

Ejemplo En este ejemplo, el método **pushButton** de *quiénMeLlama* averigua qué ficha ejecutó la ficha actual:

```
; llamarAOtraFicha::pushButton  (llamada a Ficha)
method pushButton(var eventInfo Event)
var
  Ficha  Form
endVar
Ficha.open("Lugares.fsl")   ; abrimos Lugares
Ficha.wait()                ; esperamos a que vuelva Lugares
Ficha.close()               ; cerramos Lugares
endmethod
```

Este es el código de *quiénMeLlama* en la ficha actual.

```
; quiénMeLlama::pushButton
method pushButton(var eventInfo Event)
var
  QuienLlama Form
  Título     AnyType
endVar
if formCaller(QuienLlama) then          ; intentamos obtener un
puntero a
         ; la Ficha que nos llama
Título = QuienLLama.getTitle()          ; obtenemos el título de
la Ficha
msgInfo("", "Me llamó: \n" + Título)
endif
endmethod
```

Vea también wait

formReturn

Procedimiento Devuelve el control a un método suspendido.

Tipo Form

Sintaxis **formReturn ([** const ***valorDevuelto*** AnyType **])**

Descripción Cuando una ficha ha sido llamada por **wait**, el método que ha realizado la llamada suspende su ejecución hasta que **formReturn** devuelve el control. Es posible que devuelva un valor a la ficha llamadora en *valorDevuelto*.

Ejemplo Este ejemplo consta de tres métodos. El método **pushButton** de *AbrirDiálogo* abre otra ficha en la forma de cuadro de diálogo y espera a que devuelva un valor. Los otros dos métodos se anexan a botones del cuadro de diálogo. Utilizan **formReturn** para devolver el control y valores a la ficha que realizó la llamada.

```
; AbrirDiálogo::pushButton
method pushButton(var eventInfo Event)
var
  Ficha  Form
  QuéBotón String
endVar
if Ficha.openAsDialog("Ficha.fsl", WinStyleDefault,
                      1440, 1440, 7200, 5760) then
  ; espera hasta que Ficha llama a formReturn o hasta que se
cierra
  ; el valor devuelto se almacena en QuéBotón
  QuéBotón = String(Ficha.wait())
  Ficha.close()
  ; el valor proporcionado es una cadena, de modo que será del
tipo
  ; correcto incluso si el usuario cierra el cuadro de diálogo
desde
  ; el menú de control
  msgInfo("Botón que se ha pulsado", QuéBotón)
else
  msgStop("Alto", "No puedo abrir la Ficha.")
endIf

endmethod
```

Este método se anexa al método **pushButton** de *BotónAceptar* en *Ficha*. Devuelve un valor de "OK" cuando devuelve el control al método que ejecutó **wait**:

```
; BotónAceptar::pushButton
method pushButton(var eventInfo Event)
formReturn("OK")           ; devuelve "OK" a la Ficha que lo
llame
endmethod
```

Este método se anexa al botón *BotónCancelar* en *otraFicha*. Devuelve un valor de "Cancel" cuando devuelve el control al método que ejecutó **wait**.

```
; BotónCancelar::pushButton
```

```
method pushButton(var eventInfo Event)
formReturn("Cancel")        ; devuelve "Cancel" a la Ficha que
lo llame
endmethod
```

Vea también formCaller
wait

getFileName

Método/ Procedimiento Devuelve el nombre de archivo de una ficha.

Tipo Form

Sintaxis **getFileName ( )** String

Descripción Como método, devuelve el nombre de archivo de la ficha asociada. Como procedimiento, devuelve el nombre de archivo de la ficha actual.

getPosition

Método/ Procedimiento Informa de la posición de una ventana en pantalla.

Tipo Form

Sintaxis **getPosition (** var ***x*** LongInt, var ***y*** LongInt,

var ***w*** LongInt, var ***h*** LongInt)

Descripción Escribe la posición de una ventana en la pantalla en argumentos de posición y tamaño. Los argumentos *x* e *y* contienen las coordenadas horizontal y vertical del ángulo superior izquierdo de la ficha (en twips), mientras que *w* y *h* contienen la anchura y la altura (en twips).

Ejemplo En este ejemplo, el método **pushButton** de *MoverOtraFicha* abre una ficha y obtiene su posición. Entonces, el método abre por segunda vez la misma ficha y define su posición de forma que ninguna parte de la segunda ficha solapea a la primera:

```
; MoverOtraFicha::pushButton
method pushButton(var eventInfo Event)
var
  FichaUno,
  FichaDos     Form
  x, y, w, h   LongInt
endVar
if FichaUno.open("Lugares") then
  FichaUno.getPosition(x, y, w, h)
  FichaDos.open("Lugares.fsl")                    ; abre otra
ventana
  ; asignar la posición de forma que ninguna parte se sobrepone
a otra ventana
  FichaDos.setPosition(x + w, y + h, w, h)
endif
endmethod
```

Vea también setPosition

getTitle

Método/ Procedimiento Devuelve el texto de la barra de título de la ventana.

Tipo Form

Sintaxis **getTitle ( )** String

Descripción Devuelve el texto de la barra de título de la ventana que contiene al objeto.

Ejemplo En el ejemplo siguiente, el método **pushButton** de *MostrarTitulo* abre una ficha, obtiene el título de la nueva ficha y lo muestra en un cuadro de diálogo. Entonces, este método cambia la ficha abierta a la ventana Diseñar ficha y recupera su título de nuevo:

```
; MostrarTitulo::pushButton
method pushButton(var eventInfo Event)
var
  Ficha   Form
  Título  String
endVar
Ficha.open("Lugares.fsl")
Título = Ficha.getTitle()   ; lee el título de la ventana y lo
asigna a Titulo
msgInfo("Título:", Título)  ; muestra "Form : Lugares.FSL"
Ficha.design()              ; cambiamos a la ventana de diseño
sleep()                     ; pausa
Título = Ficha.getTitle()   ; obtenemos el título de la ventana
de diseño
msgInfo("Título:", Título)  ; muestra "Form Design: Lugares.FSL"
Ficha.close()

endmethod
```

Vea también setTitle
attach

hide

Principiante

Método/ Procedimiento Convierte una ventana en invisible.

Tipo Form

Sintaxis **hide ( )**

Descripción Oculta una ventana, pero sin cerrarla.

Ejemplo En este ejemplo, el método **pushButton** de *OcultarFicha* abre una ficha, la oculta y, a continuación, la muestra:

```
; OcultarFicha::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
Ficha.open("Lugares.fsl")         ; muestra la Ficha Lugares
Ficha.hide()                      ; hace invisible la Ficha
Ficha.action(DataEnd)             ; moverse al final de la tabla
Ficha.action(DataBeginEdit)       ; iniciar el modo de Edición
Ficha.action(DataInsertRecord)    ; insertar un nuevo registro
vacío
if NOT Ficha.isVisible() then
  msgInfo("Estado", "Está oculto.")
endif
message("¡Sal, sal de donde estés!")
Ficha.show()                      ; hacemos visible a la Ficha
de nuevo
if Ficha.isVisible() then
  msgInfo("Estado", "Se puede ver.")
endif

endmethod
```

Vea también show
bringToTop
open
openAsDialog

hideSpeedBar

Procedimiento Convierte la barra rápida en invisible.

Tipo Form

Sintaxis **hideSpeedBar ( )**

Descripción Elimina la barra rápida del Escritorio. Para restaurarla, es necesario ejecutar **showSpeedBar**.

Ejemplo En este ejemplo, el método **pushButton** del botón *CambiarBarraRápida* comprueba si se muestra la barra rápida. Si es visible, este método la oculta; si no lo es, este método la muestra:

```
; CambiarBarraRápida::pushButton
method pushButton(var eventInfo Event)
if isSpeedBarShowing() then      ; si la barra de botones está
visible
  hideSpeedBar()                 ; la ocultamos
else                             ; si no
  showSpeedBar()                 ; la mostramos
endif
endmethod
```

Vea también isSpeedBarShowing
showSpeedBar

isMaximized

Método/ Procedimiento Informa de si una ventana se está mostrando con su tamaño máximo.

Tipo Form

Sintaxis **isMaximized ( )** Logical

Descripción Devuelve True si una ficha se muestra a toda pantalla; en caso contrario, devuelve False.

Ejemplo En este ejemplo, el método **pushButton** del botón *CambioDeTamaño* (de la ficha actual) abre o anexa la ficha *Lugares* con la variable *Ficha*, cambiándola de minimizada a maximizada y, entonces, a un tamaño intermedio.

```
; CambioDeTamaño::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar

; Intentamos asociar a la Ficha, ya que puede ser abierta
if NOT Ficha.attach("Form : Lugares.FSL") then
  ; si la conexión falla, intentamos abrir la Ficha
  if NOT Ficha.open("Lugares.fsl") then
    msgStop("Fallo", "No puedo abrir Lugares.")
    return
    ; si falla la apertura, abandonamos
  endif
endif
; si llegamos a este punto, tenemos un puntero correcto a la
Ficha
switch
  case isMaximized()  :                      ; si las Fichas
están maximizadas
    msgInfo("Estado", "Lugares está maximizado.")
    Ficha.show()                             ; restaurar el
tamaño
  case Lugares.isMinimized() :               ; si la Ficha está
minimizada
    msgInfo("Estado", "Lugares está minimizado.")
    Ficha.maximize()
  case NOT (Ficha.isMaximized() OR Ficha.isMinimized()):
    msgInfo("Estado", "la Ficha no está maximizada ni
minimizada.")
    Ficha.minimize()                   ; minimizamos
  otherwise :
    msgStop("Alto", "Imposible cambiar el tamaño de la Ficha.")
endswitch
endmethod
```

Vea también maximize
minimize
isMinimized

isMinimized

Método/ Procedimiento Informa de si una ventana se muestra como un icono.

Tipo Form

Sintaxis **isMinimized ( )** Logical

Descripción Devuelve True si una ficha se muestra como una icono; en caso contrario, devuelve False.

Ejemplo Consulte el ejemplo de isMaximized.

Vea también isMaximized
minimize
maximize

isSpeedBarShowing

Procedimiento Informa de si la barra rápida está visible.

Tipo Form

Sintaxis **isSpeedBarShowing ( )** Logical

Descripción Devuelve True si la barra rápida está visible; en caso contrario, devuelve False.

Ejemplo Consulte el ejemplo de hideSpeedBar.

Vea también hideSpeedBar
showSpeedBar

isVisible

Método/ Procedimiento Informa de si hay alguna parte de una ventana que se muestre.

Tipo Form

Sintaxis **isVisible ( )** Logical

Descripción Devuelve True si alguna parte de una ventana se visualiza (no está oculta); de lo contrario, devuelve False.

Ejemplo En el ejemplo siguiente, el método **pushButton** del botón *LugaresAlInicio* intenta anexarse a una ficha abierta. Si **attach** es satisfactorio, el método comprueba si la ficha es visible. Si es así, el método la sitúa como ventana superior:

```
; LugaresAlInicio::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
; si la Ficha está ahora en el entorno
if Ficha.attach("Form : Lugares.FSL") then
  if Ficha.isVisible() then                          ; si la
Ficha está visible
    Ficha.bringToTop()                               ; hacer de
su ventana la más destacada
  else
    msgStop("Lo siento", "No veo la Ficha Lugares.")
  endif
endif
endmethod
```

Vea también hide
show
bringToTop

keyChar

Método Envía un suceso al método **keyChar** de una ficha.

Tipo Form

Sintaxis **1. keyChar (** const ***carA*** SmallInt, const ***carV*** SmallInt, const ***estado*** SmallInt **)** Logical

2. keyChar (const ***caracteres*** String **[,** const ***estado*** SmallInt **])** Logical

Descripción Envía un suceso al método **keyChar** de una ficha. En la sintaxis 1, es necesario especificar el código de carácter ANSI en *carA*, el código de tecla virtual en *carV* y la constante de estado del teclado en *estado*. En la sintaxis 2, es posible especificar una cadena de uno o más caracteres y, optativamente, incluir una constante de estado del teclado.

ObjectPAL proporciona constantes para *carV* y *estado*; consulte Keyboard y KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, una ficha llamada *OtraFich* ya está abierta y contiene un campo llamado *campoUno*. El método **keyChar** a nivel de ficha para *OtraFich* repite los caracteres a *campoUno*. El método **pushButton** de un botón llamado *LlamarAOtraTeclaC* de la ficha actual se anexa a *OtraFich* como *Ficha* y ejecuta el método **keyChar** de *Ficha*, pasándole una cadena. Este es el código del método **pushButton** de *LlamarAOtraTeclaC* en la ficha actual:

```
; LlamarAOtraTeclaC::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
; conectar con otra Ficha (asume que ya está abierta)
if Ficha.attach("Form : OtraFich.FSL") then
  Ficha.keyChar("¡Hola! ")              ; enviar una cadena
else
  msgStop("Error", "La otra Ficha no está disponible.")
endif

endmethod
```

Este código se anexa al método **keyChar** a nivel de ficha de *OtraFich*:

```
; estaFicha::keyChar  (OTRAFICH.FSL)
method keyChar(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecutará para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; enviamos una tecla al CampoUno
    msgInfo("Estado", "Ejecutando el keyChar de OtraFich.")
    CampoUno.keyChar(eventInfo.char())
endif
```

```
endmethod
```

Vea también keyPhysical

keyPhysical

Método Envía un suceso al método **keyPhysical** de una ficha.

Tipo Form

Sintaxis **keyPhysical (** const ***carA*** SmallInt, const ***carV*** SmallInt, const ***estado*** SmallInt **)** Logical

Descripción Envía un suceso al método **keyPhysical** de una ficha. Es necesario especificar el código de carácter ANSI en *carA*, el código de tecla virtual en *carV* y la constante de estado del teclado en *estado*.

ObjectPAL proporciona constantes *carV* y *estado*; consulte Keyboard y KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, una ficha llamada *Ficha* ya está abierta y contiene un campo llamado *CampoUno*. El método **keyPhysical** a nivel de ficha para *OtFicha* repite los caracteres a *CampoUno*. El método **keyPhysical** de un campo llamado *CampoUnoAquí* de la ficha actual se anexa a *OtFicha* como *Ficha*. El método ejecuta, entonces, el método **keyPhysical** de *Ficha*, pasándole el código ANSI del carácter o pulsación, el código virtual del carácter o pulsación y el estado del teclado. Este es el código del método **keyPhysical** de *campoUnoAquí* en la ficha actual:

```
; campoUnoAquí::keyPhysical    (Ficha actual)
method keyPhysical(var eventInfo KeyEvent)
var
  Ficha Form
endVar
; conectar con la otra Ficha (asume que ya está abierta)
if Ficha.attach("Form : Ficha.FSL") then
  ; la estructura switch ajusta keyBoardState
  switch
    case eventInfo.isShiftKeyDown() :
      Ficha.keyPhysical(eventInfo.charAnsiCode(),
                        eventInfo.vCharCode(), Shift)
    case eventInfo.isAltKeyDown() :
      Ficha.keyPhysical(eventInfo.charAnsiCode(),
                        eventInfo.vCharCode(),
                        Alt)
    case eventInfo.isControlKeyDown() :
      Ficha.keyPhysical(eventInfo.charAnsiCode(),
                        eventInfo.vCharAnsiCode(),
                        Control)
    otherwise:
      Ficha.keyPhysical(eventInfo.charAnsiCode(),
                         eventInfo.vCharAnsiCode(), 0)
  endSwitch
else
  msgStop("Error", "La otra Ficha no está disponible.")
endif

endmethod
```

El código siguiente se anexa al método **keyPhysical** de *Ficha*:

```
; estaFicha::keyPhysical   (Ficha)
```

```
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ;el código fuente que haya aquí se ejecuta para cada objeto
de la Ficha
  else
    ;el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; pasa keyPhysical a CampoUno
    ; la estructura switch ajusta keyBoardState
    switch
      case eventInfo.isShiftKeyDown() :
        CampoUno.keyPhysical(eventInfo.charAnsiCode(),
                             eventInfo.vCharCode(), Shift)
      case eventInfo.isAltKeyDown() :
        CampoUno.keyPhysical(eventInfo.charAnsiCode(),
                              eventInfo.vCharCode(), Alt)
      case eventInfo.isControlKeyDown() :
        CampoUno.keyPhysical(eventInfo.charAnsiCode(),
                             eventInfo.vCharCode(), Control)
      otherwise :
        CampoUno.keyPhysical(eventInfo.charAnsiCode(),
                             eventInfo.vCharCode(), 0)
    endSwitch
endif
endmethod
```

Vea también keyChar

load

Método Abre una ficha en la ventana Diseñar ficha.

Tipo Form

Sintaxis **load (** const ***nombreFicha*** String **)** Logical

Descripción Abre *nombreFicha* en la ventana Diseñar ficha. Este método sólo funciona con fichas o informes guardados (.FSL o .RSL), no con fichas enviadas (.FDL o .RDL). Para más información sobre el almacenamiento y envío de fichas, consulte la *Guía del Programador ObjectPAL*.

Al diseñar o ejecutar una ficha, es posible utilizar los métodos **create** y **methodSet** del tipo UIObject para situar objetos en la nueva ficha y anexar métodos a ellos. Sin embargo, si se crean objetos mientras la ficha se ejecuta, los objetos recién creados no se guardarán automáticamente al cerrar la ficha.

Nota: Algunas acciones sobre fichas requieren un uso intensivo del procesador. En algunas situaciones, puede ser necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el método **sleep** del tipo System.

Ejemplo En el ejemplo, siguiente el método **pushButton** de un botón llamado *dibujarUnaCaja* carga la ficha *Lugares* en una ventana de diseño. Entonces, el método define la posición de la ficha, crea un cuadro pequeño, lo denomina *NuevaCaja* y define su color como Blue. Mientras la ficha se ejecuta, el cuadro no será visible; por defecto, la propiedad Visible de los objetos creados de esta forma es False.

```
; dibujarUnaCaja::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
  Objeto UIObject
endVar
; abrimos Lugares en una ventana de diseño
if Ficha.load("Lugares.fsl") then
  Ficha.setPosition(720, 720, 1440*6, 1440*5)             ; 6"
por 5"
  Ficha.create(BoxTool, 1440, 1440*3, 360, 360, myForm)
  Ficha.name = "Nueva Caja"
  Ficha.color = Blue
else
   msgStop("Alto", "No puedo cargar la Ficha.")
endif
endmethod
```

Vea también create
open
openAsDialog
design

maximize

Principiante

Método/ Procedimiento Maximiza una ventana.

Tipo Form

Sintaxis **maximize ( )**

Descripción Muestra una ventana con su tamaño completo. Es equivalente a elegir Maximizar en el menú de control.

Ejemplo En este ejemplo, el método **pushButton** del botón *irALugares* abre la ficha *Lugares* (que se supone que está en la base de datos actual), minimiza la ficha actual y espera una respuesta. Si *Lugares* devuelve "OK", este método maximiza la ficha actual; de lo contrario, restaura la ficha actual a su tamaño anterior:

```
; irALugares::pushButton
method pushButton(var eventInfo Event)
var
  Ficha   Form
  Cadena  String
endVar
; abre la Ficha Lugares, la minimiza, y luego espera
Ficha.open("Lugares.fsl")
minimize()
Cadena = String(Ficha.wait())
; si Lugares devuelve "OK", entonces maximizamos,si no
restauramos el tamaño
if Cadena = "OK" then
  maximize()
  Ficha.close()
else
  show()
  Ficha.close()
endif
endmethod
```

Este código se anexa a un botón llamado *botónAceptar* de *Lugares*:

```
; botónAceptar::pushButton
method pushButton(var eventInfo Event)
formReturn("OK")    ; devuelve la cadena "OK" a la Ficha que lo
llama
endmethod
```

Vea también minimize
isMinimized
isMaximized
show

menuAction

Método/ Procedimiento Envía un suceso al método **menuAction** de una ficha.

Tipo Form

Sintaxis **menuAction (** const ***acción*** SmallInt **)** Logical

Descripción Construye un MenuEvent y lo envía al método estándar **menuAction** de una ficha. *acción* es una de las constantes de MenuCommand o una constante definida por el usuario. ObjectPAL proporciona constantes para *acción*; consulte MenuCommands en el cuadro de diálogo Constantes. Para más información sobre las constantes definidas por el usuario, consulte el Capítulo de la *Guía del Programador ObjectPAL*.

Nota: No es posible utilizar **menuAction** para enviar una constante de comando de menú que sea equivalente a un comando del menú Archivo. Para simular un comando del menú Archivo, utilice una de las constantes de acción normales, manipule una propiedad o utilice uno de los muchos métodos del tipo System que emulan los comandos del menú Archivo.

Ejemplo En este ejemplo, el botón *enviarTítulo* de la ficha actual abre la ficha *Lugares* y le envía una acción MenuWindowTile.

```
; enviarTítulo::pushButton
method pushButton(var eventInfo Event)
var
  Ficha Form
endVar
if Ficha.open("Lugares.fsl") then
  Ficha.menuAction(MenuWindowTile)
endif
endmethod
```

Vea también action

methodDelete

Método Borra un método a nivel de ficha de una ficha.

Tipo Form

Sintaxis **methodDelete (** const ***nombreMétodo*** String **)** Logical

Descripción Borra un método estándar o personalizado especificado en *nombreMétodo* de una ficha. También es posible especificar "Var", "Proc", "Uses" o "Const" en *nombreMétodo* para borrar la ventana Va**r,** Proc, Uses o Const de una ficha. Si *nombreMétodo* es un método estándar, se restaura el comportamiento estándar de ese método.

Este método sólo funciona con fichas guardadas (.FSL), no con fichas enviadas (.FDL). Para más información sobre el almacenamiento y envío de fichas, consulte la *Guía del Usuario*.

Ejemplo En este ejemplo, hay dos fichas en el escritorio en una ventana de diseño: *FichaUno* y *FichaDos*. El método **pushButton** de un botón llamado *moverMétodo* (en la ficha actual) desplaza un método de *FichaUno* a *FichaDos*:

```
; moverMétodo::pushButton
method pushButton(var eventInfo Event)
var
  FichaOrigen,
  FichaDestino Form
  Método       String
endVar
; intentamos conectar ambas Fichas origen y destino
; asumimos que origen y destino están abiertas en el entorno en
; una ventana de diseño
if FichaOrigen.attach("Form Design : FichaUNO.FSL") AND
   FichaDestino.attach("Form Design : FichaDOS.FSL") then
  ; obtenemos la definición de mouseRightUp de la Ficha origen,
  ; luego la eliminamos
  Método = FichaOrigen.methodGet("mouseRightUp")
  FichaOrigen.methodDelete("mouseRightUp")
  ; copiamos el método mouseRightUp en la Ficha destino
  FichaDestino.methodSet("mouseRightUp", Método)
else
  msgStop("Error", "No puedo asociar las Fichas origen y
destino.")
endif

endmethod
```

Vea también methodGet
methodSet
UIObject::create
UIObject::methodGet
UIObject::methodSet

methodGet

Método Obtiene un método a nivel de ficha.

Tipo Form

Sintaxis **methodGet (**const ***nombreMétodo*** String **)** String

Descripción Obtiene el texto del método a nivel de ficha estándar o personalizado especificado en *nombreMétodo* anexado a una ficha. También es posible especificar "Var", "Proc", "Uses" o "Const" en *nombreMétodo* para obtener el contenido de la ventana Var, Proc, Uses o Const de una ficha.

Este método sólo funciona con fichas guardadas (.FSL), no con fichas enviadas (.FDL). Para más información sobre el almacenamiento y envío de fichas, consulte la *Guía del Usuario*.

Ejemplo Consulte el ejemplo de methodDelete.

Vea también methodDelete
methodSet
UIObject::create
UIObject::methodGet
UIObject::methodSet

methodSet

Método Establece la definición de un método a nivel de ficha.

Tipo Form

Sintaxis **methodSet (**const ***nombreMétodo*** String, const ***textoMétodo*** String **)** Logical

Descripción Escribe el texto de *textoMétodo* en el método a nivel de ficha estándar o personalizado especificado en *nombreMétodo*, sustituyendo cualquier definición de método existente. También es posible especificar "Var", "Proc", "Uses" o "Const" en *nombreMétodo* para definir el contenido de la ventana Var, Proc, Uses o Const de una ficha.

Este método sólo funciona con fichas guardadas (.FSL), no con fichas enviadas (.FDL). Para más información sobre el almacenamiento y envío de fichas, consulte la *Guía del Usuario*.

Ejemplo Consulte el ejemplo de methodDelete.

Vea también methodDelete
methodGet
UIObject::create
UIObject::methodGet
UIObject::methodSet

minimize

Método/ Procedimiento Minimiza una ventana.

Tipo Form

Sintaxis **minimize ( )**

Descripción Muestra una ventana como un icono. Es equivalente a elegir Minimizar en el menú de control.

Ejemplo Consulte el ejemplo de maximize.

Vea también maximize
isMaximized
isMinimized
show

mouseDouble

Método Envía un suceso al método **mouseDouble** de una ficha.

Tipo Form

Sintaxis **mouseDouble (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseDouble** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo supóngase que la ficha *Ficha* ya está abierta y ejecutándose. El método **pushButton** de un botón llamado *enviarMouseDouble* de la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseDouble** de *Fichas*:

```
; enviarMouseDouble::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar
; intentamos conectar con la Ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseDouble a la Ficha destino en las
coordenadas 1000, 1000
  Fichas.mouseDouble(1000, 1000, LeftButton)
else
  msgStop("Alto", "No encuentro la Ficha destino.")
endif
endmethod
```

Este código se anexa al método **mouseDouble** de *Fichas* (*Ficha*):

```
; otroRatón::mouseDouble  (FICHA)
method mouseDouble(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; escribimos el nombre del método en el campo UltimoMetodo
    UltimoMétodo = "mouseDouble"
    ; Obtenemos el destino y escribimos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
```

```
endmethod
```

Vea también mouseDown
mouseRightDouble

mouseDown

Método Envía un suceso al método **mouseDown** de una ficha.

Tipo Form

Sintaxis **mouseDown (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseDown** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseDown* de la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseDown** de *Fichas*:

```
; enviarMouseDown::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar
; intentamos conectar con la Ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseDown a la ficha destino en las coordenadas
1000, 1000
  Fichas.mouseDown(1000, 1000, LeftButton)

else
  msgStop("Alto", "No encuentro la ficha destino.")
endif

endmethod
```

Este código se anexa al método **mouseDown** de *Fichas* (*Ficha*):

```
; otroRatón::mouseDown  (FICHA)
method mouseDown(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; escribimos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseDown"
    ; obtenemos el destino y escribimos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
endmethod
```

Vea también mouseRightDown
mouseUp

mouseEnter

Método Envía un suceso al método **mouseEnter** de una ficha.

Tipo Form

Sintaxis **mouseEnter (** const ***x*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseEnter** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseEnter* en la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseEnter** de *Fichas*:

```
; enviarMouseEnter::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar
; intentamos conectar con la Ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseEnter a la Ficha destino en las
coordenadas 1000, 1000
  Fichas.mouseEnter (1000, 1000, LeftButton)
else
  msgStop("Alto", "No encuentro la Ficha destino.")
endif
endmethod
```

Este código se anexa al método **mouseEnter** de *Fichas* (*Ficha*):

```
; otroRatón::mouseEnter  (FICHA)
method mouseEnter(var eventInfo MouseEvent)
var
  Objeto UIObject
endVar

if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; escribimos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseEnter"
    ; obtenemos el destino y escribir el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif

endmethod
```

Vea también mouseExit

mouseExit

Método Envía un suceso al método **mouseExit** de una ficha.

Tipo Form

Sintaxis **mouseExit (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseExit** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseExit* de la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseExit** de *Fichas*:

```
; enviarMouseExit::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar

; intentamos conectar con la ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseExit a la Ficha destino en las coordenadas
1000, 1000
  Fichas.mouseExit(1000, 1000, LeftButton)
else
  msgStop("Alto", "No encuentro la ficha destino.")
endif

endmethod
```

Este código se anexa al método **mouseExit** de *Fichas* (*Ficha*):

```
; otroRatón::mouseExit  (FICHA)
method mouseExit(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia ficha
    ; escribirmos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseDown"
    ; obtenemos el destino y escribimos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
```

```
endmethod
```

Vea también mouseEnter

mouseMove

Método Envía un suceso al método **mouseMove** de una ficha.

Tipo Form

Sintaxis **mouseMove (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt) Logical

Descripción Construye un MouseEvent y lo envía al método **mouseMove** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseMove* de la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseMove** de *Fichas*:

```
; enviarMouseMove::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar
; intentamos conectar con la ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseMove a la Ficha destino en las coordenadas
1000, 1000
  Fichas.mouseMove(1000, 1000, LeftButton)
else
  msgStop("Alto", "No encuentro la Ficha destino.")
endif

endmethod
```

Este código se anexa al método **mouseMove** de *Fichas* (*Ficha*):

```
; otroRatón::mouseMove  (FICHA)
method mouseMove(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; escribimos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseMove"
    ; obtenemos el destino y escribimos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
endmethod
```

Vea también mouseEnter
mouseExit

mouseRightDouble

Método Envía un suceso al método **mouseRightDouble** de una ficha.

Tipo Form

Sintaxis **mouseRightDouble (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseRightDouble** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseRightDouble* en la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseRightDouble** de *Fichas*:

```
; enviarMouseRightDouble::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar
;intentamos conectar con la ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseRightDouble a la Ficha destino en las
coordenadas
  ;1000, 1000
  Fichas.mouseRightDouble(1000, 1000, RightButton)
else
  msgStop("Alto", "No encuentro la Ficha destino.")
endif
endmethod
```

Este código se anexa al método **mouseRightDouble** de *Fichas* (*Ficha*):

```
; otroRatón::mouseRightDouble  (FICHA)
method mouseRightDouble(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia ficha
    ; escribimos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseRightDouble"
    ; obtenemos el destino y escribiemos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
```

```
endmethod
```

Vea también mouseDouble
mouseRightDown

mouseRightDown

Método Envía un suceso al método **mouseRightDown** de una ficha.

Tipo Form

Sintaxis **mouseRightDown (** const ***x*** LongInt, const ***y*** LongInt,

const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseRightDown** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseRightDown* en la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseRightDown** de *Fichas*:

```
; enviarMouseRightDown::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar     ;
intentamos conectar con la ficha destino
if Fichas.attach("Form : FICHA.FSL") then

  ; enviamos un mouseRightDown a la ficha destino en las
coordenadas 1000, 1000
  Fichas.mouseRightDown(1000, 1000, RightButton)
else
  msgStop("Alto", "No encuentro la Ficha destino.")
endif
endmethod
```

Este código se anexa al método **mouseRightDown** de *Fichas* (*Ficha*):

```
; otroRatón::mouseRightDown  (FICHA)
method mouseRightDown(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia ficha
    ; escribimos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseRightDown"
    ; obtenemos el destino y escribiemos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
```

```
endmethod
```

Vea también mouseRightDouble
mouseDown

mouseRightUp

Método Envía un suceso al método **mouseRightUp** de una ficha.

Tipo Form

Sintaxis **mouseRightUp (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseRightUp** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseRightUp* en la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseRightUp** de *Fichas*:

```
; mouseRightUp::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar
; intentamos conectar con la ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseRightUp a la Ficha destino en las
coordenadas 1000, 1000
  Fichas.mouseRightUp(1000, 1000, RightButton)
else
   msgStop("Alto", "No encuentro la Ficha destino.")
endif
endmethod
```

Este código se anexa al método **mouseRightUp** de *Fichas* (*Ficha*):

```
; otroRatón::mouseRightUp  (FICHA)
method mouseRightUp(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia ficha
    ; escribimos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseRightUp"
    ; obtenemos el destino y escribiemos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
endmethod
```

Vea también mouseUp
mouseRightDown

mouseUp

Método Envía un suceso al método **mouseUp** de una ficha.

Tipo Form

Sintaxis **mouseUp (** const ***x*** LongInt, const ***y*** LongInt, const ***estado*** SmallInt **)** Logical

Descripción Construye un MouseEvent y lo envía al método **mouseUp** de una ficha. Los argumentos *x* e *y* especifican (en twips) la posición del suceso y *estado* indica un estado de tecla. ObjectPAL proporciona constantes para *estado*; consulte KeyBoardStates en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que la ficha *Ficha* ya está abierta. El método **pushButton** de un botón llamado *enviarMouseUp* en la ficha actual se anexa a *Ficha* como *Fichas* y ejecuta el método **mouseUp** de *Fichas*:

```
; enviarMouseUp::pushButton
method pushButton(var eventInfo Event)

var
  Fichas Form
endVar
; intentamos conectar con la ficha destino
if Fichas.attach("Form : FICHA.FSL") then
  ; enviamos un mouseUp a la ficha destino en las coordenadas
1000, 1000
  Fichas.mouseUp(1000, 1000, LeftButton)
else
  msgStop("Alto", "No encuentro la Ficha destino.")
endif
endmethod
```

Este código se anexa al método **mouseUp** de *Fichas* (*Ficha*):

```
; otroRatón::mousetUp  (FICHA)
method mouseRightUp(var eventInfo MouseEvent)
var
  Objeto  UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia ficha
    ; escribimos el nombre del método en el campo UltimoMétodo
    UltimoMétodo = "mouseUp"
    ; obtenemos el destino y escribiemos el nombre en el campo
UltimoDestino
    eventInfo.getTarget(Objeto)
    UltimoDestino = Objeto.Name
endif
endmethod
```

Vea también mouseRightUp
mouseDown

moveTo

Método Se desplaza a una ficha.

Tipo Form

Sintaxis **moveTo ([**const ***nombreObjeto*** String]) Logical

Descripción Desplaza el foco a una ficha. Optativamente, se desplaza al objeto especificado en *nombreObjeto*.

Ejemplo En el ejemplo siguiente, supóngase que una ficha llamada *Lugares* ya está abierta. El método **pushButton** del botón *irALugares* de la ficha actual asocia la variable *Fichas* con *Lugares*, determina si *Fichas* está visible, y, si es así, se desplaza a *Fichas*. Si *Fichas* no está visible, el método emplea **show** para mostrar la ficha con su tamaño por defecto (**show** también desplaza el foco a la ficha destino):

```
; irALugares::pushButton
method pushButton(var eventInfo Event)
var
  Fichas  Form
endVar
; asumimos que la Ficha Lugares está ya abierta
if Fichas.attach("Form : LUGARES.FSL") then
  if Fichas.isVisible() then
    Fichas.moveTo()         ; si la Ficha está visible, la
movemos
  else
   Fichas.show()            ; de otro modo, la hacemos visible
  endif
else
  msgStop("Alto", "No encuentro la ficha.")
endif
endmethod
```

Vea también moveToPage

moveToPage

Método/ Procedimiento Muestra una página especificada de una ficha.

Tipo Form

Sintaxis **moveToPage (** const ***númeroPágina*** SmallInt **)** Logical

Descripción Muestra la página de una ficha especificada en *númeroPágina*. *númeroPágina* puede ser una variable entera o una constante entera, pero no puede ser un ID de objeto. Para ir a una página por su ID de objeto, emplee el método **moveTo** del tipo UIObject.

Ejemplo En este ejemplo, la ficha actual tiene dos páginas. La ficha *Lugares* existe en el directorio de trabajo y tiene cuatro páginas. El método **pushButton** de *PáginaParaLugares* (en la ficha actual) se desplaza a la segunda página de la ficha actual. A continuación, el método abre la ficha *Lugares* en la variable *Fichas* y se desplaza por las páginas de *Fichas*:

```
; páginaParaLugares::pushButton
method pushButton(var eventInfo Event)
const
  Facturación = SmallInt(4)
endConst


var
  Fichas  Form
  Página SmallInt
endVar
moveToPage(2)                          ; nos desplazamos a la
página 2 de esta ficha
if Fichas.open("Lugares.fsl") then     ; se abre en la primera
página
  sleep(2000)                          ; pausa
  Fichas.moveToPage(2)                 ; se desplaza a la
página 2 de Lugares
  sleep(2000)
  Página = 3
  Fichas.moveToPage(Página)            ; se desplaza a la
página 3
  sleep(2000)
  Fichas.moveToPage(Facturación)       ; se desplaza a la
página 4
  sleep(2000)
endIf
endmethod
```

Vea también bringToTop

open

Principiante

Método Abre una ventana.

Tipo Form

Sintaxis **1. open (** const ***nombreFicha*** String **[,** const ***estiloVentana*** LongInt]) Logical

2. open (const ***nombreFicha*** String, const ***estiloVentana*** LongInt, const ***x*** LongInt, const ***y*** LongInt, const ***w*** LongInt, const ***h*** LongInt **)** Logical

3. open (const ***infoAbrir*** FormOpenInfo **)** Logical

Descripción Muestra y ejecuta la ficha especificada en *nombreFicha*. Compare este método con **load**, que abre una ficha en una ventana de diseño.

Los argumentos optativos *x* e *y* especifican la posición del ángulo superior izquierdo de la ficha (en twips), mientras que *w* y *h* indican la anchura y la altura (en twips), y *estiloVentana* especifica los atributos de visualización. Es posible especificar más de un elemento de estilo de ventana sumando las constantes. Por ejemplo, el código siguiente abre una ficha y especifica barras de desplazamiento tanto vertical como horizontal:

```
LaFicha.open("ventas", WinStyleVScroll + WinStyleHScroll)
```

ObjectPAL proporciona constantes para *estiloVentana*; consulte WindowStyles en el cuadro de diálogo Constantes.

La sintaxis 3 permite especificar valores de ficha mediante *infoAbrir*, un registro del tipo FormOpenInfo. El registro FormOpenInfo se ha declarado previamente y tiene la estructura siguiente:

```
x, y, w, h      LongInt     ; posición y tamaño de la ficha
nombre          String      ; nombre de la ficha que se abre
TablaMaestra    String      ; Nombre de la nueva tabla maestra
CadenaQBE       String      ; consulta para ejecutar (actual
cadena QBE)
EstiloVent      LongInt     ; Constante(s) de estilo de la
ventana
```

Es posible utilizar el miembro *TablaMaestra* para especificar otra tabla principal para la ficha (es similar a elegir otra tabla para una ficha al abrir la ficha en el cuadro de diálogo Abrir ficha). Alternativamente, es posible especificar una cadena de consulta en el miembro *CadenaQBE*. Paradox ejecuta la consulta y abre la ficha; el resultado de la consulta es la tabla principal.

Nota: Algunas acciones sobre fichas requieren un uso intensivo del procesador. En algunas situaciones, puede ser necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el método **sleep** del tipo System.

Ejemplo En este ejemplo, el método **keyPhysical** de un campo llamado *CampoUno* comprueba todos los sucesos de tecla. Cuando el usuario pulsa F1, se abre la ficha FICHA. El método **keyPhysical** abre una ficha desde el directorio actual:

```
; campoUno::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  Ficha Form
endVar
message(eventInfo.vChar())
if eventInfo.vChar() = "VK_F1" then
  Ficha.open("FICHA", WinStyleDefault,
             720, 720, 1440 * 2, 1440 * 4)
  disableDefault
endIf
endmethod
```

Este ejemplo funciona como el ejemplo anterior, excepto en que utiliza un registro FormOpenInfo para definir las características de la ficha que se va a abrir.

```
; campoUno::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  AbrirFicha InfoAbrirFicha      ; un tipo registro que se
declara antes
  Ficha      Form
endVar
message(eventInfo.vChar())
if eventInfo.vChar() = "VK_F1" then
  AbrirFicha.x = 720
  AbrirFicha.y = 720
  AbrirFicha.w = 2 * 1440
  AbrirFicha.h = 4 * 1440
  AbrirFicha.name = "Ficha"
  Ficha.open(AbrirFicha)
  disableDefault
endIf
endmethod
```

Vea también close
create
load
openAsDialog
design

openAsDialog

Método Abre una ventana de ficha en forma de cuadro de diálogo.

Tipo Form

Sintaxis **1. openAsDialog (** const ***nombreFicha*** **[,** const ***estiloVentana*** LongInt]) Logical

2. openAsDialog (const ***nombreFicha*** String, const ***estiloVentanaD*** LongInt, const ***x*** LongInt, const ***y*** LongInt, const ***w*** LongInt, const ***h*** LongInt **)** Logical

3. openAsDialog (const ***infoAbrir*** FormOpenInfo **)** Logical

Descripción Ejecuta la ficha *nombreFicha* y la muestra sobre cualquier otra ventana abierta. *nombreFicha* siempre se muestra encima, esté activa o no. Los argumentos optativos *x* e *y* especifican el ángulo superior izquierdo de la ventana (en twips), mientras que *w* y *h* indican la anchura y la altura (en twips), y *estiloVentana* especifica los atributos de visualización. ObjectPAL proporciona constantes para *estiloVentana*; consulte WindowStyles en el cuadro de diálogo Constantes.

Es posible especificar más de un elemento de estilo de ventana sumando las constantes. Por ejemplo, el código siguiente abre una ficha y especifica barras de desplazamiento tanto horizontal como vertical.

```
LaFicha.open("sales", WinStyleVScroll + WinStyleHScroll)
```

La sintaxis 3 permite especificar valores de ficha mediante *infoAbrir*, un registro del tipo FormOpenInfo. El tipo de registro FormOpenInfo se ha declarado previamente y tiene la estructura siguiente:

```
x, y, w, h      LongInt     ; posición y tamaño de la ficha
nombre          String      ; nombre de la ficha que se abre
TablaMaestra    String      ; Nombre de la nueva tabla maestra
CadenaQBE       String      ; ejecuta esta consulta
EstiloVent      LongInt     ; Constante(s) de estilo de la
ventana
```

Es posible utilizar el miembro TablaMaestra para indicar otra tabla principal para la ficha (es similar a elegir otra tabla para una ficha al abrir la ficha en el cuadro de diálogo Abrir ficha). Alternativamente, es posible indicar una cadena de consulta en el miembro *CadenaQBE*. Paradox ejecuta la consulta y abre la ficha; el resultado de la consulta es la tabla principal.

Ejemplo En este ejemplo, el método **keyPhysical** de un campo llamado *CampoUno* comprueba todos los sucesos de tecla. Cuando el usuario pulsa ***F1***, se abre la ficha FICHA. El método **keyPhysical** abre una ficha en forma de cuadro de diálogo:

```
; CampoUno::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  Ficha Form
endVar
; si el usuario pulsa F1, abrimos un cuadro de diálogo de ayuda
if eventInfo.vChar() = "VK_F1" then
```

```
  Ficha.openAsDialog("Ficha", WinStyleDefault,
                     720, 720, 1440 * 4, 1440 * 3)
  Ficha.setTitle("Ayuda para la Aplicación")  ; dar la cuadro de
diálogo un
                             ; título nuevo
  Ficha.wait()
  Ficha.close()
  disableDefault                                          ; no
llama al sistema de ayuda
endIf

endmethod
```

Vea también open
wait
formCaller
formReturn

postAction

Método Consigna una acción en una cola de acciones para su ejecución demorada.

Tipo Form

Sintaxis **postAction (** const ***idAcción*** SmallInt **)**

Descripción Funciona como **action**, excepto en que la acción no se ejecuta inmediatamente. Por el contrario, Paradox espera a que todo el método haya terminado de ejecutarse y a que Paradox esté en estado de espera. La acción especifica en *idAcción* se almacena en una cola de acciones en el momento de la llamada al método; Paradox realiza la acción después de que el método actual haya terminado de ejecutarse.

Ejemplo En este ejemplo, el método **pushButton** de *AbrirLugaresNuevo* abre la ficha *Lugares* en la variable *Fichas*. Entonces, el método consigna (almacena) tres acciones en *Fichas* y muestra un mensaje en un cuadro de diálogo. Las acciones especificadas en **postAction** se producen después de que aparezca el cuadro de diálogo con el mensaje y después de que termine este método:

```
; AbrirLugaresNuevo::pushButton
method pushButton(var eventInfo Event)
; la variable Fichas es global para la ficha-continúa en el
ambito después
; de que el método finalice
if Fichas.open("Lugares.fsl") then
  ; estas acciones no se ejecutarán hasta después de que
finalice este método
  Fichas.postAction(DataEnd)                  ; se desplaza al
último registro
  Fichas.postAction(DataBeginEdit)      ; activa el modo de
Edición
  Fichas.postAction(DataInsertRecord)   ; Inserta un nuevo
registro vacío
  msgInfo("Estado", "Al realizar las acciones enviadas, esté
atento.")
else
  msgStop("Atención", "No puedo abrir la ficha.")
endif
endmethod
```

Vea también action

run

Método Ejecuta una ficha que está abierta actualmente en la ventana Diseñar ficha.

Tipo Form

Sintaxis **run ( )** Logical

Descripción Cambia una ficha de la ventana Diseñar ficha a una ventana Ficha. Este método sólo funciona con fichas guardadas (.FSL), no con fichas enviadas (.FDL). Para más información sobre el almacenamiento y envío de fichas, consulte la *Guía del Usuario*.

Para cambiar de una ficha en ejecución a la ventana Diseñar ficha, utilice **design**.

Nota: Algunas acciones sobre fichas requieren un uso intensivo del procesador. En algunas situaciones, puede ser necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el método **sleep** del tipo System.

Ejemplo Este ejemplo abre la ficha *Lugares* en una ventana de diseño, borra el método **pushButton** de la ficha y ejecuta la ficha. Supóngase que la ficha *Lugares* está en el directorio actual. Este código se anexa al método **pushButton** de *eliminarPushButton*:

```
; eliminarpushButton::pushButton
method pushButton(var eventInfo Event)
var
  Fichas Form
endVar
; cargamos la ficha Lugares, eliminamos el método
; pushButton, luego ejecutamos la ficha
if Fichas.load("Lugares") then
  Fichas.methodDelete("pushButton")
  Fichas.run()        ; como no se ha guardado la ficha, el
endif                 ; borrado no será permanente
endmethod
```

Vea también design

save

Método Guarda una ficha en el disco.

Tipo Form

Sintaxis **save ([** const ***nuevoNombreFicha*** String **])** Logical

Descripción Escribe una ficha en el disco en el directorio de trabajo actual del usuario. Este método sólo funciona cuando la ficha está en una ventana de diseño.

Es posible utilizar *nuevoNombreFicha* para especificar un nombre para la ficha, u omitirlo. Si se omite *nuevoNombreFicha* y la ficha no tiene ya un nombre, Paradox muestra un cuadro de diálogo que solicita al usuario que introduzca un nombre. Si la ficha ya tiene un nombre, Paradox la almacena utilizándolo.

Ejemplo Consulte el ejemplo de create.

Vea también create
design

setPosition

Método/ Procedimiento Sitúa una ventana en la pantalla.

Tipo Form

Sintaxis **setPosition (** const ***x*** LongInt, const ***y*** LongInt, const ***w*** LongInt, const ***h*** LongInt)

Descripción Sitúa una ventana en la pantalla. Los argumentos *x* e *y* especifican las coordenadas del ángulo superior izquierdo de la ficha (en twips), mientras que *w* y *h* indican la anchura y la altura (en twips).

En cuadros de diálogo y en el Escritorio de Paradox, la posición debe expresarse respecto a toda la pantalla; en las fichas, informes y ventanas de tabla, la posición debe expresarse respecto al Escritorio de Paradox.

Ejemplo Consulte el ejemplo de getPosition.

Vea también open

setTitle

Método/ Procedimiento Define el texto que aparece en la barra de título de la ventana.

Tipo Form

Sintaxis **setTitle (** const ***texto*** String **)**

Descripción Cambia el texto de la barra de título de la ventana a *texto*. Si cambia el título de una ficha, recuerde que debe utilizar el nuevo título al querer realizar una anexación con esa ficha (consulte la descripción de attach, para más detalles).

Ejemplo Consulte el ejemplo de openAsDialog**.**

Vea también getTitle
attach

show

Principiante

Método/ Procedimiento Muestra una ventana minimizada con su tamaño anterior; hace que una ficha oculta esté visible.

Tipo Form

Sintaxis **show ( )**

Descripción Hace que una ficha oculta esté visible. **show** restaura una ventana minimizada al tamaño que tenía antes de minimizarla. Este método es similar al comando Restaurar del menú Control del sistema.

show no sitúa una ficha como ventana superior; utilice **bringToTop** para situarla como capa superior y darle el foco.

Ejemplo Consulte el ejemplo de hide.

Vea también hide
isVisible

showSpeedBar

Procedimiento	Hace que la barra rápida esté visible.
Tipo	Form
Sintaxis	**showSpeedBar ( )**
Descripción	Muestra la barra rápida.
Ejemplo	Consulte el ejemplo de .hideSpeedBar.
Vea también	hideSpeedBar isSpeedBarShowing

wait

Principiante

Método Suspende la ejecución de un método.

Tipo Form

Sintaxis **wait ( )** AnyType

Descripción Suspende la ejecución del método actual hasta que vuelve la ficha a la que se espera (consulte **formReturn**). Este método es útil cuando se abre una segunda ficha en forma de cuadro de diálogo. La ejecución se reanuda en la primera ficha cuando la segunda ficha (a la que se espera) ejecuta **formReturn** o cuando se cierra la segunda ficha. Una vez que vuelve la ficha llamada, la ficha llamadora debería cerrarla mediante **close**. La ficha llamada no se cierra automáticamente, aunque el usuario la cierre; se mantiene abierta para que el código de la ficha llamadora pueda examinarla (por ejemplo, para ver los valores de un cuadro de diálogo).

Ejemplo Consulte el ejemplo de formReturn.

Vea también formReturn
formCaller
openAsDialog

windowClientHandle

Método/ Procedimiento Devuelve el gestor de una ventana.

Tipo Form

Sintaxis **windowClientHandle ( )** SmallInt

Descripción Un gestor de ventana es un identificador entero exclusivo asignado a una ventana por Windows. **windowClientHandle** devuelve un valor entero que representa el gestor de ventana del área cliente de una ficha. Cuando se ejecuta como procedimiento, devuelve el gestor de ventana del área cliente de la ficha actual. Este método sólo deberían utilizarlo los programadores avanzados.

Esta información es útil sólo si se utilizan funciones de una biblioteca de vínculo dinámico (DLL) escrita en C, C++ o Pascal. Para más información sobre la escritura y uso de los DLL, consulte la entrada **uses** en el Capítulo 3 en la *Guía de Referéncias de ObjectPAL.*

Ejemplo En el ejemplo siguiente, supóngase que existe un DLL llamado MYTEST.DLL y que contiene una función llamada *HacerAlgo*. La función *HacerAlgo* toma un argumento, un gestor de ventana.

```
; algúnBotón::pushButton
method pushButton(var eventInfo Event)
Uses MITEST
  HacerAlgo(const wHandle CHANDLE)
endUses
HacerAlgo(windowClientHandle())   ; llamamos a HacerAlgo y
proporcionamos
    ; el puntero al área cliente
          ; de la ficha actual
endmethod
```

Vea también windowHandle

windowHandle

Método/ Procedimiento Devuelve el gestor de una ventana.

Tipo Form

Sintaxis **windowHandle ( )** SmallInt

Descripción Un gestor de ventana es un identificador entero exclusivo asignado a una ventana por Windows. **windowHandle** devuelve un valor entero que representa el gestor de ventana de una ficha. Cuando se ejecuta como procedimiento, devuelve el gestor de ventana de la ficha actual. Este método sólo deberían utilizarlo los programadores avanzados.

Esta información es útil sólo si se utilizan funciones de una biblioteca de vínculo dinámico (DLL) escrita en C, C++ o Pascal. Para más información sobre la escritura y uso de los DLL, consulte la entrada uses en el Capítulo 3 de la *Guía de Referéncias de ObjectPAL***.**

Ejemplo En el ejemplo siguiente, supóngase que existe un DLL llamado MYTEST.DLL y que contiene una función llamada *HacerAlgo*. La función *HacerAlgo* toma un argumento, un gestor de ventana.

```
; algúnBotón::pushButton
method pushButton(var eventInfo Event)
Uses MITEST
  HacerAlgo(const wHandle CHANDLE)
endUses

HacerAlgo(windowHandle())         ; llamamos a HacerAlgo y
proporcionamos                                          ; el
puntero a la ventana de la ficha actual
endmethod
```

Vea también windowClientHandle

attach

Método Asocia una variable Report con un informe abierto.

Tipo Report

Sintaxis **attach** (const ***tituloInforme*** String) Logical

Descripción Asocia una variable Report con un informe abierto. tituloInforme especifica el título de un informe abierto.

Note: El argumento *tituloInforme* referencia al texto mostrado en la barra de título de la ventana Report, no el nombre del archivo. Es posible utilizar **getTitle** para devolver este texto, o emplear **setTitle** para especificar uno.

Ejemplo En este ejemplo, supóngase que el método **open** de la ficha abrió el informe EXISTEN.RSL y retituló la ventana como Informe de Existencias . El método **pushButton** de *imprimirExistencias* se anexa al informe abierto por medio de su título y lo imprime.

```
; imprimirExistencias::pushButton
method pushButton(var eventInfo Event)
var
  Existencias Report
endVar
; el informe Existencias se abrió y fue cambiado de título por
el método open
    ;  de la ficha
Existencias.attach("Informe de Existencias")  ; asociado por el
título del
    ;  informe
Existencias.print()                           ; imprimimos el
informe
endmethod
```

El código siguiente se anexa al método **open** de la ficha:

```
; estaFicha::open
method open(var eventInfo Event)
var
  Existencias Report
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    Existencias open("existen.rsl")
    Existencias setTitle("Informe de Existencias")
    bringToTop  ; lleva la ficha de nuevo arriba
endif
endmethod
```

Vea también open
Form::getTitle

Form::setTitle

close

Método Cierra una ventana.

Tipo Report

Sintaxis **close** ( )

Descripción Cierra una ventana Report. Es equivalente a elegir Cerrar en el menú de control.

Ejemplo En este ejemplo, supóngase que el método **open** de la ficha abrió el informe EXISTEN.RSL y retituló la ventana como Informe de Existencias . El método **close** de la ficha se anexa al informe abierto por medio de su título y lo cierra cuando se cierra la ficha.

```
; estaFicha::open
method open(var eventInfo Event)
var
  Existencias Report
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; el informe Existencias se abrió y fue cambiado de título
por
    ; el método open de la ficha
    Existencias.attach("Informe de Existencias")
    Existencias.close()
endif
endmethod
```

Vea también open

currentPage

Método Devuelve el número de la página actual de un informe.

Tipo Report

Sintaxis **currentPage** ( ) SmallInt

Descripción Devuelve el número de la página actual de un informe.

Ejemplo En este ejemplo, el método **pushButton** de *másDosPáginas* intenta anexarse a un informe abierto y, si esto falla, abre el informe. Una vez que la variable *ordersRep* señala a un informe abierto, el método desplaza el informe hacia delante dos páginas.

```
; másDosPáginas::pushbutton
method pushButton(var eventInfo Event)
var
  Existencias Report
endVar
; el informe debería estar ya abierto, así que intentamos
asociarlo primero
if NOT Existencias attach("Report : PEDIDOS.RSL")
  if NOT Existencias.open("Pedidos.rsl") then
    msgStop("", "No puedo abrir o asociar el informe.")
    return
  endif
; nos desplazamos dos páginas después de la actual
Existencias.moveToPage(Existencias currentPage + 2)
bringToTop()   ; hacemos que esta Ficha sea la ventana más
destacada
endmethod
```

Vea también moveToPage

design

Método Cambia un informe en ejecución a la ventana Diseñar informe.

Tipo Report

Sintaxis **design** ( ) Logical

Descripción Cambia un informe en ejecución a la ventana Diseñar informe. Este método sólo funciona con informes guardados (.RSL), no con informes enviados (.RDL). Para más información sobre el almacenamiento y envío de informes, consulte la *Guía del Usuario*.

Utilice **run** para ejecutar un informe que ya está cargado en la ventana Diseñar informe.

Nota: Algunas acciones sobre informes requieren un uso intensivo del procesador. En algunas situaciones, tal vez sea necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el procedimiento **sleep** en el tipo System.

Ejemplo En este ejemplo, supóngase que el método **open** de la ficha abrió el informe EXISTEN.RSL y retituló la ventana como Informe de Existencias . El método **pushButton** de *diseñarExistencias* se anexa al informe abierto por medio de su título, y cambia el informe a la ventana Diseñar informe.

```
; diseñarExistencias::pushButton
method pushButton(var eventInfo Event)
var
  Existencias Report
endVar
; el método open de la ficha abrió y cambió el título del
informe de Existencias
Existencias.attach("Informe de existencias")
Existencias.design()            ; cambiamos a modo de Diseño
endmethod
```

Vea también load
open
run

enumUIObjectNames

Método Crea una tabla que enumera losUIObject contenidos en un informe.

Tipo Report

Sintaxis **enumUIObjectNames** (const ***nombreTabla*** String) Logical

Descripción Crea una tabla de Paradox que enumera el nombre y tipo de cada objeto contenido en un informe. Utilice el argumento *nombreTabla* para especificar un nombre para la tabla. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* ya está abierta, este método falla. Es posible incluir un alias o vía de acceso en *nombreTabla*; si no se especifica un alias ni una vía de acceso, Paradox la crea en el directorio de trabajo (:TRABAJO:).

La estructura de *nombreTabla* es:

Nombre de campo	**Tipo,+**	**Tamaño**
ObjectName	A	128
ObjectClass	A	32

Ejemplo En el ejemplo siguiente, el método **pushButton** de *describirInforme* utiliza **enumUIObjectNames** y **enumUIObjectProperties** para documentar un informe.

```
; describirInforme::pushButton
method pushButton(var eventInfo Event)
var
  Informe Report
  Tabla   TableView
endVar
Informe.load("Pedidos.rsl")                    ; cargamos el
informe en
                 <~>; modo de Diseño
Informe.enumUIObjectNames("NomPedid.db")       ; escribimos los
nombres en la tabla
Informe.enumUIObjectProperties("PropPedi.db")  ; escribimos las
porpiedades en
                  ; la tabla
ordersRep.close()
Tabla.open("NomPedid")                         ;observemos
nuestra obra
Tabla.wait()
Tabla.open("PropPedi")
Tabla.wait()
Tabla.close()
endmethod
```

Vea también enumUIObjectProperties

enumUIObjectProperties

Método Crea una tabla que enumera las propiedades de cada UIObject contenido en un informe.

Tipo Report

Sintaxis **enumUIObjectProperties** (const ***nombreTabla*** String) Logical

Descripción Crea una tabla de Paradox que enumera el nombre, nombre de propiedad y valor de propiedad de cada objeto contenido en un informe. Utilice el argumento *nombreTabla* para especificar un nombre para la tabla. Si *nombreTabla* ya existe, este método la sustituye sin pedir confirmación. Si *nombreTabla* ya está abierta, este método falla.

La estructura de *nombreTabla* es:

Nombre de campo	**Tipo**	**+ Tamaño**
ObjectName	A	128
PropertyName	A	64
PropertyType	A	48
PropertyValue	A	255

Ejemplo Consulte el ejemplo de enumUIObjectNames.

Vea también enumUIObjectNames

load

Método Abre un informe en la ventana Diseñar informe.

Tipo Report

Sintaxis **load** (const ***nombreInforme*** String) Logical

Descripción Abre nombreInforme en la ventana Diseñar informe. Este método sólo funciona con informes o fichas guardadas (.RSL o .FSL), no con informes ni fichas enviadas (.RDL o .FDL). Para más información sobre el almacenamiento y envío de informes, consulte la *Guía del Usuario*.

Compare este método con **open**, que ejecuta un informe.

Nota: Algunas acciones sobre informes requieren un uso intensivo del procesador. En algunas situaciones, tal vez sea necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el procedimiento **sleep** en el tipo System.

Ejemplo En este ejemplo, el método **pushButton** del botón ***cargarPedidos*** carga el informe PEDIDOS.RSL en la ventana Diseñar informe, crea un cuadro de texto en la cabecera de página y escribe una cadena en el cuadro de texto.

```
; cargarPedidos::pushButton
method pushButton(var eventInfo Event)
var
  Informe Report
  Título  UIObject
endVar
if Informe.load("Pedidos.rsl") then
  ; asumimos que el informe está en la cabecera de la página de
un cuadro de texto
  Título.create(TextTool, 1440*3, 720, 1440*2, 360, Informe)
  Título.Name = "TextoDelNuevoTitulo"
  Título.Text = "Informe de pedidos " + String(time())
  Título.Color = LightBlue
  Título.Visible = True
  Informe.run()
endif
endmethod
```

Vea también design
run
open

moveToPage

Método Muestra una página especificada de un informe.

Tipo Report

Sintaxis **moveToPage** (const ***númeroPágina*** SmallInt) Logical

Descripción Muestra la página de un informe especificada en númeroPágina. Este método no activa el informe. Si desea activar el informe, incluya **bringToTop** después de **moveToPage** (consulte el tipo Form para más información sobre **bringToTop**).

Ejemplo Consulte el ejemplo de currentPage.

Véa también Form::bringToTop
currentPage

open

Método Abre un informe.

Tipo Report

Sintaxis **1. open** (const ***nombreInforme*** String [, ***estiloVentana*** LongInt]) Logical

2. open (const ***nombreInforme*** String, const ***estiloVentana*** LongInt, const x LongInt, const y LongInt, const w LongInt, const h LongInt) Logical

3. open (const ReportOpenInfo) Logical

Descripción Muestra el informe especificado en *nombreInforme* en una ventana de informe. Los argumentos optativos especifican la posición del ángulo superior izquierdo del informe (x e y), la anchura y la altura (*w* y *h*) y el estilo (*estiloVentana*).

ObjectPAL proporciona constantes para *estiloVentana*; consulte WindowStyles en el cuadro de diálogo Constantes.

Es posible especificar más de un estilo de ventana sumando las constantes. Por ejemplo, el código siguiente abre una ventana de informe que tiene cuadros sobre los que el usuario puede hacer clic para minimizar y maximizar la ventana:

```
informeVentas.open("ventas.rsl", WinStyleMaximizeBox +
WinStyleMinimizeBox)
```

La sintaxis 3 permite especificar los valores de la ficha en *infoAbrir*, un registro del tipo ReportOpenInfo. Un registro ReportOpenInfo tiene la misma estructura:

```
x, y, w, h                    LongInt   ; tamaño y posición del
informe
Nombre                            String         ; nombre del
informe a abrir (preView)
TablaMaestra             String        ; nombre de la tabla
maestra
CadenaQBE                     String         ; ejecutamos esta
consulta (cadena QBE actual)
ReiniciarOpciones SmallInt    ; una de las constantes
ReportPrintRestart
```

Nota: Algunas acciones sobre informes requieren un uso intensivo del procesador. En algunas situaciones, tal vez sea necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el procedimiento **sleep** en el tipo System.

Ejemplo En este ejemplo, el método **pushButton** de *abrirPequeño* abre el informe ORDERS.RSL y lo minimiza suministrando la constante de estilo de ventana WinStyleMinimize.

```
; abrirPequeño::pushButton
method pushButton(var eventInfo Event)
     var
      Informe Report
     endVar
     Informe.open("Pedidos.rsl", WinStyleMinimize)   ; abrimos
el Informe Pedidos minimizado
endmethod
```

Vea también close
design
load

print

Principiante

Método Imprime un informe.

Tipo Report
Principiante

1. print ( ) Logical

2. print (const ***nombreInforme*** String, const ***reinicioImprimirInforme*** SmallInt) Logical

3. print (const ri ReportPrintInfo) Logical

Descripción Imprime un informe. Con la sintaxis 1, Paradox abre el cuadro de diálogo Imprimir para el informe actual, que permite al usuario especificar los valores de impresión. La sintaxis 2 permite especificar un nombre de informe y definir las opciones de reinicio. La sintaxis 3 permite definir los valores de impresión con un registro ReportPrintInfo. Los registros ReportPrintInfo se declaran previamente y tienen la estructura siguiente:

```
  Nombre           String      ; ejecutamos este informe si no
está abierto aún
  TablaMaestra     String      ; nombre de la tabla maestra
  CadenaQBE        String      ; ejecutamos esta consulta
(cadena QBE actual)
  ReiniciarOpciones SmallInt   ; qué hacemos cuando los datos
cambian mientras
                               ; se imprime el informe
                               ; una de las constantes
ReportPrintRestart
  ImprimirAtrás    Logical     ; adelante FALSE, atrás TRUE,
por defecto es FALSE
  HacerCopias      Logical     ; ¿Quién hace las copias:
Paradox o la impresora?
                               : si es cierto, Paradox hace las
copias
  Opciones         SmallInt    ; una de las constantes
ReportPrintPanel
  NúmeroDeCopias   SmallInt    ; número de copias, por defecto
es 1
  PáginaDeInicio   LongInt     ; pagina de comienzo, por
defecto es 1
  PáginaDeFin      LongInt     ; definición de Página del final
  Incremento       SmallInt    ; incremento de página para
impresión
                               ; multi-pasada, por defecto es 1
  DesplazamientoX  LongInt     ; Desplazamiento horizontal de
la página
  DesplazamientoY  Longint     ; Desplazamiento vertical de la
página
  Orientación      SmallInt    ; una de las constantes
ReportOrietation
```

Ejemplo Para ejemplos de impresión utilizando la sintaxis 1, consulte el ejemplo de attach. El ejemplo siguiente muestra el uso de la sintaxis 3 para imprimir

mediante un registro ReportPrintInfo.

```
; impresiónConRegistro::pushButton
method pushButton(var eventInfo Event)
var
  Informe      Report
  DatosInforme ReportPrintInfo
endVar
; primero, actualizamos el registro DatosInforme

DatosInforme.NúmeroDeCopias = 2
DatosInforme.HacerCopias = True
DatosInforme.Nombre = "Existencias"
Informe.print(DatosInforme)
endmethod
```

Vea también open
run

run

Método Ejecuta un informe desde la ventana Diseñar informe.

Tipo Report

Sintaxis **run** ( ) Logical

Descripción Cambia un informe de la ventana Diseñar informe a la ventana Informe. Este método sólo funciona con informes guardados (.RSL), no con informes enviados (.RDL). Para más información sobre el almacenamiento y envío de informes, consulte la *Guía del Usuario*.

Para cambiar un informe en ejecución a la ventana Diseñar informe, utilice **design**.

Nota: Algunas acciones sobre informes requieren un uso intensivo del procesador. En algunas situaciones, tal vez sea necesario que incluya un **sleep** después de ejecutar **open**, **load**, **design** o **run**. Para más información, consulte el procedimiento **sleep** en el tipo System.

Ejemplo Consulte el ejemplo de load.

Vea también design
load

action

Método Ejecuta un comando de acción.

Tipo TableView

Sintaxis **action** (const ***IDAcción*** SmallInt) Logical

Descripción Realiza la acción representada por la constante *IDAcción*. ObjectPAL proporciona constantes para los identificadores de acción; consulte una de las categorías Action (por ejemplo, ActionSelectCommands) en el cuadro de diálogo Constantes.

Ejemplo Este ejemplo abre una vista de tabla para la tabla *Pedidos*, desplaza el cursor al final de la tabla, inicia el modo editar e inserta un nuevo registro vacío. Este código se anexa al método **pushButton** de un botón llamado *comenzarEditarInsertar*.

```
; comenzarEditarInsertar::pushButton
method pushButton(var eventInfo Event)
  var
    ordenTV  TableView
  endVar
  if ordenTV.open( Pedidos ) then
     ordenTV.action(DataEnd)           ; nos desplazamos al
final de la tabla<|>
     ordenTV.action(DataBeginEdit)           ; iniciamos el
modo de edición
     ordenTV.action(DataInsertRecord)   ; insertamos un nuevo
registro en blanco
     ordenTV.wait()                    ; esperamos hasta que el
objeto
                      ; TableView esté cerrado
     ordenTV.close()                   ; cerrar cuando retorne
   else
     msgStop( Estado ,  No pude encontrar la tabla Pedidos. )
   endif
endmethod
```

close

Método Cierra una ventana de tabla.

Tipo TableView

Sintaxis **close** ( )

Descripción Cierra una ventana de tabla. Es equivalente a elegir Cerrar en el menú de control.

Ejemplo En este ejemplo, el método **open** de una ficha abre un objeto TableView para la tabla *Clientes* en la variable global *clienTV*. Cuando se cierra la ficha, el método **close** de la ficha cierra el TableView *clienTV*. Este código se anexa al método **close** de la ficha:

```
; estaFicha::close
method close(var eventInfo Event)
if eventInfo.isPreFilter()
  then
      ; el código fuente que se incluya aquí se ejecutará para
cada objeto de
      ; la ficha
  else
      ; este código fuente sólo se ejecutará para el propio
formato.
      ClienTV.close()       ; cerrar la tabla Clientes que fue
          ; abierta por el método open de estaFicha
endif
endmethod
```

Este es el código de la ventana Var de la ficha.

```
; estaFicha::Var
Var
    ClienTV  TableView      ; es global para la ficha, el método
open de
    ; la ficha se encarga de abrir el objeto TableView
endVar
```

Este es el código del método **open** de la ficha.

```
; estaFicha::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
   ; el código fuente que se incluya aquí se ejecutará para
cada objeto de
   ; la ficha
  else
   ; este código fuente sólo se ejecutará para el propio
formato.
   ClienTV.open( Clientes )  ; abrir la vista Clientes
endif
endmethod
```

Vea también open

moveToRecord

Method Moves to a specific record in a table.

Type TableView

Syntax **moveToRecord** (const ***tc*** TCursor) Logical

Description **moveToRecord** sets the current record to the record pointed to by the TCursor *tc*. This method can be very slow for dBASE tables; use the **RecNo** property instead.

Example This example uses a TCursor to search for a customer named Jones, then calls **moveToRecord** to make a table view display that record. The following code is attached to a button's built-in **pushButton** method.

```
method pushButton (var eventInfo Event)
var
   custTC TCursor
   custTV TableView
endVar

custTC.open ("customer.db")
custTV open ("customer.db")

if custTC.locate ("Last Name", "Jones") then
  custTV.moveToRecord (custTC)
else
   msgInfo("Search failed", "Couldn't find Jones.")
endIf

endmethod
```

See also action

open

Método Abre una ventana de tabla.

Tipo TableView

Sintaxis **1. open** (const ***nombreVT*** String [, const ***estiloVentana*** LongInt]) Logical

2. open (const ***nombreVT*** String, const ***estiloVentana*** LongInt, const x LongInt, const y LongInt, const w LongInt, const h LongInt) Logical

Descripción Muestra la tabla especificada en *nombreVT* de una ventana de tabla. Los argumentos optativos especifican (en twips) la posición del ángulo superior izquierdo de la ficha (*x* e *y*), el ancho y altura (*w* y *h*) y el estilo (*estiloVentana*). El argumento *estiloVentana* no se tiene en cuenta, pero es necesario en la sintaxis 2. Si se desea especificar un tamaño y posición, puede usarse una constante de estilo de ventana de *WinStyleDefault*.

Ejemplo En el ejemplo siguiente, el método **pushButton** de un botón llamado *abrirEsperarPedidos* abre la tabla *Pedidos* y espera a que el usuario cierre la ventana de tabla.

```
; abrirEsperarPedidos::pushButton
method pushButton(var eventInfo Event)
var
   PedidosTV  TableView
endVar
if PedidosTV.open( Pedidos , WinStyleDefault, 100, 100,
                  1440*5, 1440*4) then
    PedidosTV.wait()        ; esperamos a que el usuario cierre
    PedidosTV.close()       ; cerramos la tabla Pedidos.
endif
endmethod
```

Vea también close

wait

Método	Suspende la ejecución de un método.
Tipo	TableView
Sintaxis	**wait** ( )
Descripción	Suspende la ejecución de un método. La ejecución se reanuda cuando se cierra el objeto TableView. Observe que es necesario incluir un **close** después de un **wait**. Cuando un objeto TableView ha sido llamado por **wait**, el método que realiza la llamada suspende su ejecución hasta que el usuario cierra el objeto TableView.
Ejemplo	Consulte el ejemplo for open.
Vea también	close open

actionClass

Método Devuelve el número de clase de un ActionEvent.

Tipo ActionEvent

Sintaxis **actionClass** ( ) SmallInt

Descripción Devuelve el número de clase de un ActionEvent. ObjectPAL define constantes para estos números de clase; consulte ActionClasses en el cuadro de diálogo Constantes.

Ejemplo Este ejemplo utiliza **actionClass** para impedir que el usuario realice cambios en un objeto de campo. Este código se anexa al método estándar **action** de un campo. Para ver un ejemplo que atrapa al usuario al acceder al modo editar, consulte **id**.

```
;Notas_Lugar::Action
method action (var eventInfo ActionEvent)
; comprueba cualquier intento de edición y lo bloquea
if eventInfo.actionClass() = EditAction then
    ; permite que el usuario ejecute y finalice la
visualización del campo
    if NOT (eventInfo.id() = EditEnterFieldView) AND
        NOT (eventInfo.id() = EditToggleFieldView) AND
        NOT (eventInfo.id() = EditExitFieldView) then
      eventInfo.setErrorCode(1)
      beep()
      message("Lo siento, en este campo no pueden realizarse
cambios.")
   endIf
endIf
endmethod
```

Vea también id
setId
Event::getTarget

id

Principiante

Método Devuelve el número ID de un ActionEvent.

Tipo ActionEvent

Sintaxis **id** ( ) SmallInt

Descripción Devuelve el ID (identificador) de acción de un ActionEvent. ObjectPAL define constantes para los números ID de acción (por ejemplo, DataBeginEdit); consulte las categorías que comienzan con Action (por ejemplo, ActionDataCommands) en el cuadro de diálogo Constantes.

También es posible enviar acciones definidas por el usuario a un método estándar **action**. Para más información sobre la creación y uso de constantes definidas por el usuario, consulte el Capítulo 6 de la *Guía del Programador ObjectPAL*.

Ejemplo Este ejemplo utiliza **id** para impedir que el usuario acceda al modo editar en una ficha. Este código se anexa al método estándar **action** de la ficha.

```
;estaFicha::action
method action(var eventInfo ActionEvent)
if eventInfo.isPreFilter()
     then
        ; el código fuente aquí incluido se ejecuta para cada
objeto
         de la Ficha
     else
        ; el código fuente aquí incluido se ejecuta sólo para
la propia Ficha
        if eventInfo.id() = DataBeginEdit then
           eventInfo.setErrorCode(1)  ; No entra en modo
edición
           msgStop("Lo siento", "sólo lectura - No puedo editar
esta Ficha")
        endif
endif
endmethod
```

Vea también actionClass
setId

setId

Método Especifica el ID (identificador) de un ActionEvent.

Tipo ActionEvent

Syntax **setId** (const ***idAcción*** SmallInt)

Descripción Especifica el ActionEvent representado mediante la constante *idAcción*. ObjectPAL proporciona constantes para *idAcción*; consulte las categorías que comienzan con Action (por ejemplo, ActionDataCommands) en el cuadro de diálogo Constantes.

También es posible enviar acciones definidas por el usuario a un método estándar **action**. Para más información sobre la creación y uso de constantes definidas por el usuario, consulte el Capítulo 6 de la *Guía del Programador ObjectPAL.*

Ejemplo En este ejemplo, los botones de la barra rápida para el movimiento entre registros se reasignan para el desplazamiento dentro de un campo memo. Supóngase que una ficha contiene un objeto multirregistro, *LUGARES*, asociado con la tabla *Lugares*. El código siguientes se anexa al método **action** del objeto de campo *Notas_Lugar*.

```
;Notas_lugar::action
method action(var eventInfo ActionEvent)
var
   IdAcc  SmallInt
endVar
; Si estamos en modo edición dentro del campo Notas_lugar, los
botones
     de movimiento
; entre registros se reasignan para poder moverse dentro del
campo memo
if self.Editing then
     IdAcc = eventInfo.id()
     switch
         case IdAcc = DataPriorRecord  :
eventInfo.setID(MoveBeginLine)
         case IdAcc = DataNextRecord   :
eventInfo.setID(MoveEndLine)
         case IdAcc = DataFastBackward :
eventInfo.setID(MoveBegin)
         case IdAcc = DataFastForward  :
eventInfo.setID(MoveEnd)
         case IdAcc = DataBegin        :
eventInfo.setID(FieldBackward)
         case IdAcc = DataEnd          :
eventInfo.setID(FieldForward)
   endswitch
endif
endmethod
```

Vea también id

reason

Método Informa de por qué se ha producido un error.

Tupo ErrorEvent

Sintaxis **reason** ( ) SmallInt

Descripción Devuelve un valor entero que informa de por qué se ha producido un ErrorEvent. ObjectPAL proporciona las constantes siguientes para comprobar el valor devuelto por **reason** (enumeradas también en el cuadro de diálogo Constantes bajo ErrorReasons):

ErrorCritical indica que aparecerá un mensaje de error en un cuadro de diálogo.

ErrorWarning implica que aparecerá un mensaje de error en la línea de estado.

Nota: No confunda **reason** con **errorCode** (para una descripción de **errorCode**, consulte el tipo Event).

Ejemplo El código siguiente se anexa al método estándar error de la ficha. Informa del código de error, la razón, y el mensaje asociado con el error.

```
;estaFicha::error
method error(var eventInfo ErrorEvent)
if eventInfo.isPreFilter()
   then
     ; el código fuente que hay aquí se ejecuta para cada
objeto de la Ficha
    ; msgInfo("Error", eventInfo.errorCode())
       if eventInfo.reason() = ErrorWarning then
          msgInfo("Error de aviso", errorMessage())
       else
          msgInfo("Error Crítico", errorMessage())
       endif
       disableDefault
   else
       ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
endif
endmethod
```

Vea también Event::errorCode
setReason

setReason

Método Especifica una razón para la generación de un ErrorEvent.

Tipo ErrorEvent

Sintaxis **setReason** (const ***idRazón*** SmallInt)

Descripción Especifica una razón para la generación de un ErrorEvent. Este método toma una de las siguientes constantes de razón como argumento (también enumeradas en el cuadro de diálogo Constantes bajo ValueReasons):

ErrorCritical indica que aparecerá un mensaje de error en un cuadro de diálogo.

ErrorWarning implica que aparecerá un mensaje de error en la línea de estado.

Ejemplo El ejemplo siguiente crea un ErrorEvent, define la razón como ErrorCritical y envía el ErrorEvent a la ficha.

```
;EnviarError::pushButton
method pushButton(var eventInfo Event)
var
   ev  ErrorEvent
endVar
ev.setErrorCode(1)         ; asignamos 1 al código de error
(cualquiera que no sea cero es aceptable)
ev.setReason(ErrorWarning) ; asignamos el tipo de error a
ErrorWarning     estaFicha.error(ev)        ; enviamos el error
a la Ficha
endmethod
```

Vea también reason

errorCode

Principiante

Método Informa acerca del estado de una marca de error.

Tipo Event

Sintaxis **errorCode** ( ) LongInt

Descripción Devuelve un código de error si hay un error; en caso contrario, devuelve 0. ObjectPAL proporciona constantes para los códigos de error de Event; consulte EventErrorCodes en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que una ficha contiene un marco de tabla asociado con la tabla *Clientes*, un botón llamado *BotónDeComenzarEdición* y otro botón llamado *enviarError*. El método **pushButton** de *BotónDeComenzarEdición* examina el suceso; si el suceso no tiene un error, el método inicia el modo editar en *CLIENTES*. A los efectos de este ejemplo solamente, el botón *EnviarError* crea un suceso, define su error como 1 y activa el método **pushButton** de *BotónDeComenzarEdición* con el suceso. El código siguiente se anexa al método **pushButton** de *BotónDeComenzarEdición*:

```
;;BotónDeComenzarEdición::pushButton
method pushButton(var eventInfo Event)
; comprobar eventInfo para saber si ha ocurrido un error
if eventInfo.errorCode() = 0 then
   CLIENTES.action(DataBeginEdit)  ; si no hay error, iniciar
el modo de Edición
endif
endmethod
```

El método siguiente se anexa al método **pushButton** de *EnviarError*:

```
;EnviarError::pushButton
method pushButton(var eventInfo Event)
var
   ev Event                ; el suceso
endVar
ev.setErrorCode(1)               ; asignamos el código 1 al
error     BotónDeComenzarEdición.pushButton(ev)     ; enviamos
el suceso a
                                          ;
BotónDeComenzarEdición
endmethod
```

Vea también setErrorCode

getTarget

Método Crea un gestor para el destino de un Event.

Tipo Event

Sintaxis **getTarget** (var ***destino*** UIObject)

Descripción Devuelve en *destino* el gestor del UIObject que fue el destino del Event más reciente.

Ejemplo Este ejemplo supone que varios campos de la tabla *Clientes* están situados en una ficha. Conforme el usuario se desplaza de un campo a otro, el método **setFocus** de la ficha identifica el destino del Event, averigua si el destino es un campo y, en ese caso, cambia el color del campo actual a azul claro. Esto proporciona al usuario una indicación visual más llamativa que el resaltado normal. El color anterior del campo se almacena en la variable global *ColorAnterior*. Cuando se elimina el foco del campo, el método **removeFocus** de la ficha restaura el campo a su color original. El color anterior del campo se almacena en una variable declarada en la ventana Var de la ficha, como se muestra en el código siguiente:

```
;estaFicha::Var
Var
  ColorAnterior LongInt     ; para almacenar el color anterior
del campo
endVar
```

El código siguiente se anexa al método **setFocus** de la ficha:

```
;estaFicha::setFocus
method setFocus(var eventInfo Event)
var
   ObjetoDest    UIObject
endVar
if eventInfo.isPreFilter()
then
   ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
   ; obtener el destino
   eventInfo.getTarget(ObjetoDest)
   if ObjetoDest.Class = "Field" then   ; si es un campo,
cambiar su color
      ColorAnterior = ObjetoDest.Color  ; almacenar el antiguo
color en una
   ; variable global de la Ficha
      ObjetoDest.Color = LightBlue            ; resaltar el
campo que tiene el foco
   endif
else
   ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif
endmethod
```

Este código se anexa al método **removeFocus** de la ficha:

```
;estaFicha::removeFocus
method removeFocus(var eventInfo Event)
var
   ObjetoDest  UIObject
endVar
if eventInfo.isPreFilter()
   then
     ; el código fuente que hay aquí se ejecuta para cada
objeto de la Ficha
     ; obtener el destino
     eventInfo.getTarget(ObjetoDest)
     if ObjetoDest.Class = "Field" then  ; si es un campo,
       ObjetoDest.Color = ColorAnterior  ; restaurar el color
almacenado en
                                         ; la variable global
     endif
else
      ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha         endif
endmethod
```

Vea también isPreFilter

isFirstTime

Método Informa de si la ficha está gestionando un Event por primera vez, antes de expedirlo.

Tipo Event

Sintaxis **isFirstTime** ( ) Logical

Descripción Informa de si la ficha está gestionando un Event antes de expedirlo al objeto destino o si el suceso se ha expedido al objeto destino y, posteriormente, se ha lanzado hacia arriba en la jerarquía de contenedores. Este método devuelve True si la ficha gestiona el Event por primera vez; en caso contrario, devuelve False. Utilice **isFirstTime** en los métodos estándar anexados a la ficha.

Ejemplo Este ejemplo muestra cómo es posible utilizar **isFirstTime** con **isTargetSelf** para evaluar un suceso en un método a nivel de ficha. Este código sustituye el código por defecto para el método **pushButton** de la ficha, que normalmente comprueba **isPreFilter**.

```
;estaFicha::pushButton
method pushButton(var eventInfo Event)
var
   ObjetoDest   UIObject
endVar
; Este ejemplo utiliza isFirstTime e
    isTargetSelf en vez de isPreFilter.
; Existen tres posibilidades.
; Suceso de la propia Ficha
    : isTargetSelf es True, isFirstTime es True
; Sucesos servidos (sucesos prefiltrados)
    : isTargetSelf es False, isFirstTime es True
; Sucesos pasados (pasados explicitamente)
    : isTargetSelf es False, isFirstTime
                                                ; es False
; Para la Ficha, isTargetSelf nunca es True cuando ;isFirstTime
es False.

eventInfo.getTarget(ObjetoDest)
    ; obtener el destino para ObjetoDest
switch
   case eventInfo.isTargetSelf() AND eventInfo.isFirstTime() :
     ; Esto ocurre sólo cuando la Ficha está manejando su
propio suceso.
   msgInfo("Estado", "Esta linea no se ejecuta para ningun
suceso pushButton.")

   case NOT eventInfo.isTargetSelf() AND
eventInfo.isFirstTime() :
     ; Esto ocurre sólo cuando la Ficha está sirviendo un
suceso
     ; para otro objeto. En esta situación isPrefilter resulta
Cierto (True).
   msgInfo("Estado", "Se ha enviado un suceso pushButton a "
    + ObjetoDest.Name + ".")
```

```
   case NOT eventInfo.isTargetSelf() AND NOT
eventInfo.isFirstTime() :
      ; Esto ocurre cuando el suceso ha sído
; explicitamente pasado a la Ficha.
      ; En esta situación isPrefilter resulta falso (False).
   msgInfo("Estado", "Este cuadro de diálogo aparece sólo
cuando un suceso
    ; pushButton " +
            "se ha pasado a la Ficha explícitamente.")
endswitch
endmethod
```

El código siguiente se anexa al método **pushButton** del botón *ComprobarSucesoPasado* de la ficha. Cuando el método **pushButton** de la ficha ha realizado el filtro previo sobre el suceso y lo ha expedido de nuevo al botón, el método **pushButton** del botón lo devuelve a la ficha con el comando **passEvent**. Cuando el suceso vuelve a la ficha, los métodos **isTargetSelf**, **isFirstTime** e **isPrefilter** devolverán False.

```
;ComprobarSucesoPasado::pushButton
method pushButton(var eventInfo Event)
passEvent     ; pasamos el suceso hacia arriba en la
 ; jerarquía
endmethod
```

Vea también isPreFilter

isPreFilter

Método Informa de si la ficha está gestionando un Event en su propio nombre.

Tipo Event

Sintaxis **isPreFilter** ( ) Logical

Descripción Informa de si la ficha está gestionando un Event en su propio nombre o en nombre de otro objeto. Devuelve True sólo cuando el destino es algún objeto distinto de la ficha y la ficha no ha gestionado ya este Event. **isPreFilter** es equivalente lógicamente a que la ficha evalúe la sentencia siguiente:

```
if (NOT eventInfo.isTargetSelf()) AND eventInfo .isFirstTime()
```

Este método devuelve True para todos los métodos internos, y para todos los métodos externos cuando llegan a la ficha por primera vez.

Cuando los métodos externos se lanzan de vuelta a la ficha, este método devuelve False.

Nota: los métodos de ficha *no* se les realiza un filtro previo. En otras palabras, cuando se produce un Event para la ficha, isPreFilter devuelve False.

Ejemplo Consulte el ejemplo de getTarget.

Vea también isFirstTime

isTargetSelf

Método	Informa de si un objeto es el destino de un Event.
Tipo	Event
Sintaxis	**isTargetSelf** ( ) Logical
Descripción	Informa de si un objeto es el destino de un Event. Utilice **isTargetSelf** en métodos estándar anexados a la ficha.
Ejemplo	Consulte el ejemplo de isFirstTime.

Vea también isFirstTime

reason

Método Informa de por qué se ha producido un Event.

Tipo Event

Sintaxis **reason** ( ) SmallInt

Descripción Devuelve un valor entero que informa de por qué se ha producido un Event. **reason** devuelve constantes de razón válidas sólo para los Event generados por el método estándar **newValue** (vea los ejemplos). ObjectPAL proporciona las constantes siguientes para comprobar el valor devuelto por **reason** (enumeradas también en el cuadro de diálogo Constantes bajo ValueReasons):

FieldValue implica que se ha cambiado el valor de un campo por un desplazamiento, por una actualización en la red, por una sentencia de ObjectPAL o porque el usuario lo ha cambiado.

EditValue implica que se ha especificado un valor mediante un clic sobre un botón de radio o mediante la elección de un elemento en una lista.

StartupValue indica que se ha especificado un valor al abrir la ficha.

Nota: Las constantes de razón también están definidas para los ErrorEvent, MenuEvent, MoveEvent y StatusEvent. Consulte **reason** en estas secciones para ver ejemplos. El método **reason** es válido para los otros tipos de suceso (ActionEvent, KeyEvent, MouseEvent y ValueEvent), pero devuelve cero. **setReason** también es válido para ActionEvent, KeyEvent, MouseEvent y ValueEvent, pero sólo puede emplearse para establecer las constantes de Reason definidas por el usuario (una técnica avanzada).

Ejemplo En este ejemplo, supóngase que una ficha contiene un objeto multirregistro asociado con la tabla Pedidos y que el campo Forma_De_Envío es un conjunto de botones de radio. El método newValue siguiente de Forma_De_Envío muestra un mensaje que indica por qué se ejecutó newValue. Cuando se abre la ficha, la constante de Reason será StartupValue.

```
;FormaDeEnvío::newValue
method newValue(var eventInfo Event)
; mostrar porqué se llamó al método newValue
msgInfo("Razón de la llamada a newValue"
     iif(eventInfo.reason() = StartupValue, "StartupValue"
     iif(eventInfo.reason() = FieldValue, "FieldValue",
"EditValue")))
endmethod
```

Cuando el usuario se desplace por la tabla o hace clic sobre el botón *SiguienteRegistro*, la constante de Reason será FieldValue.

```
;SiguienteRegistro::pushButton
method pushButton(var eventInfo Event)
action(DataNextRecord)    ; esto ejecuta un newValue para
Forma_De_Envío
                          ; con la constante FieldValue de tipo
Reason
```

```
endmethod
```

Cuando el usuario elige otro botón de radio de *Forma_De_Envío* o hace clic sobre el botón *CambiarRadio*, la constante de Reason será EditValue.

```
;CambiarRadio::pushButton
method pushButton(var eventInfo Event)
PEDIDOS.Forma_De_Envío = "US Mail"      ;<%4> esto ejecuta un
newValue para Forma_De_Envío<%0>
          ; con la constante EditValue de tipo Reason
endmethod
```

Vea también

MoveEvent
StatusEvent
ErrorEvent
ValueEvent
setReason

setErrorCode

Principiante

Método Define el código de error para un Event.

Tipo Event

Sintaxis **setErrorCode** (const ***idError*** LongInt)

Descripción Define el código de error. Si *idError* es 0, implica "sin error". Cualquier valor distinto de cero para *idError* indica un error.

ObjectPAL proporciona constantes para *idError*; consulte EventErrorCodes en el cuadro de diálogo Constantes.

Ejemplo Consulte el ejemplo de errorCode.

Vea también errorCode

setReason

Método Especifica un Reason para la generación de un movimiento.

Tipo Event

Sintaxis **setReason** (const ***idRazón*** SmallInt)

Descripción Especifica un Reason para la generación de un Event. Este método toma una de las siguientes constantes de Reason como argumento (enumeradas también en el cuadro de diálogo Constantes bajo ValueReasons):

FieldValue implica que se ha cambiado el valor de un campo por un desplazamiento, por una actualización en la red, por una sentencia de ObjectPAL o porque el usuario lo ha cambiado.

EditValue implica que se ha especificado un valor mediante un clic sobre un botón de radio o mediante la elección de un elemento en una lista.

StartupValue indica que se ha especificado un valor al abrir la ficha.

Nota: Las constantes de Reason también están definidas para los ErrorEvent, MenuEvent, MoveEvent y StatusEvent. Consulte **setReason** en estas secciones para ver ejemplos. El método **setReason** es válido para los otros tipos de suceso (ActionEvent, KeyEvent, MouseEvent, TimerEvent y ValueEvent), pero devuelve cero. **setReason** también es válido para ActionEvent, KeyEvent, MouseEvent y ValueEvent, pero sólo puede emplearse para establecer las constantes de razón definidas por el usuario (una técnica avanzada).

Ejemplo En este ejemplo, supóngase que una ficha contiene un objeto multirregistro asociado con la tabla *Pedidos* y que el campo *Forma_De_Envío* es un conjunto de botones de radio. El método **newValue** siguiente de *Forma_De_Envío* muestra un mensaje que indica por qué se ejecutó **newValue**.

```
;FormaDeEnvío::newValue
method newValue(var eventInfo Event)
; mostrar porqué se llamó al método newValue
msgInfo("Razón de la llamada a newValue",
      iif(eventInfo.reason() = StartupValue, "StartupValue",
      iif(eventInfo.reason() = FieldValue, "FieldValue",
"EditValue")))
endmethod
```

El código siguiente demuestra cómo definir un Reason para un Event y enviar el Event a un objeto.

```
;EjecutarRazón::pushButton
method pushButton(var eventInfo Event)
var
   ev  Event
endVar
ev.setReason(FieldValue)          ; asignamos una constante de
tipo Reason para
         ;  el suceso
PEDIDOS.Forma_de_Envío.newValue(ev)     ; enviamos el suceso al
campo Forma_
```

```
                ; De_Envío
endmethod
```

Vea también reason

char

Método Devuelve el carácter asociado con la pulsación de una tecla.

Tipo KeyEvent

Sintaxis **char ( )** String

Descripción Devuelve el carácter asociado con la pulsación de una tecla. Por ejemplo, si el usuario teclea **a**, **char** devuelve "a". Si pulsa *Mayús*+A, **char** devuelve A. Si una pulsación resulta en un carácter no imprimible, **char** devuelve una cadena vacía ("").

char es la forma más sencilla de comprobar la pulsación de una tecla alfanumérica cuando es importante la distinción entre mayúsculas y minúsculas. Si no lo es, utilice **vChar** para realizar la comparación con el valor de cadena de un código de tecla virtual. Por ejemplo, si es relevante si el usuario pulsa una **a** minúscula o **A** mayúscula, emplee **char** para devolver el valor de cadena del carácter pulsado, y compárelo con "a" o "A". Si desea averiguar si se pulsó **a** o **A**, utilice

vChar y compárelo con "A" (el código de tecla virtual para la pulsación de una **a** minúscula o **A** mayúscula).

Ejemplo Este ejemplo muestra el carácter tecleado en un objeto de campo como un mensaje en la parte inferior de la pantalla. El código se anexa al método estándar **keyChar** de un objeto de campo.

```
; esteCampo::keyChar
method keyChar(var eventInfo KeyEvent)
doDefault                    ; llevar el carácter al campo
message(eventInfo.char())    ; y luego mostrar el carácter como
un mensaje

endmethod
```

Vea también charAnsiCode
setChar
vChar
String::ansiCode
String::chrToKeyName
String::toANSI
String::toOEM

charAnsiCode

Método Devuelve el valor ANSI asociado con una pulsación.

Tipo KeyEvent

Sintaxis **charAnsiCode ( )** SmallInt

Descripción Devuelve un número entero que representa el valor ANSI asociado con una pulsación. Por ejemplo, si el usuario teclea **a**, **charAnsiCode** devuelve 97. Si pulsa *Mayús*+A, **charAnsiCode** devuelve 65. **charAnsiCode** también funciona con caracteres no imprimibles. Por ejemplo, si el usuario pulsa *Intro*, **charAnsiCode** devuelve 13.

Ejemplo Este ejemplo emite una señal sonora cuando el usuario pulsa Retroceso o Ctrl+H. Este código se anexa al método estándar **keyPhysical** de un objeto de campo.

```
; esteCampo::keyPysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.charAnsiCode() = 8 then          ; si el usuario
pulsa Ctrl+H o la tecla
                      ; de Borrar
  beep()                                              ; hacemos
sonar un pitido
endif

endmethod
```

Vea también
char
vChar
setVChar
String::ansiCode
String::chrToKeyName
String::toANSI
String::toOEM

isAltKeyDown

Método Informa de si la tecla Alt estaba pulsada durante un KeyEvent.

Tipo KeyEvent

Sintaxis **isAltKeyDown ( )** Logical

Descripción Devuelve True si la tecla *Alt* estaba pulsada en el momento en que se produjo un KeyEvent; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente supone que una ficha tiene un cuadro llamado *CajaUno*. Cuando el usuario pulsa *AltC*, el método **keyPhysical** de la ficha cambia el color de *CajaUno*. Este código se anexa al método **keyPhysical** de una ficha:

```
; estaFicha::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    if eventInfo.isAltKeyDown() AND     ; si el usuario pulsa
Alt+C
      eventInfo.vChar() = "C" then
      disableDefault              ; bloquea los procesos
habituales
      ; asignar el color de CajaUno entre el rojo y el azul
      CajaUno.color = iif(CajaUno.color = Red, Blue, Red)
   endif
else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif
endmethod
```

Vea también isControlKeyDown
isShiftKeyDown
setAltKeyDown

isControlKeyDown

Método Informa de si la tecla *Ctrl* estaba pulsada durante un KeyEvent.

Tipo KeyEvent

Sintaxis **isControlKeyDown ( )** Logical

Descripción Devuelve True si la tecla *Ctrl* estaba pulsada en el momento en que se produjo un KeyEvent; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente supone que una ficha tiene un cuadro llamado *CajaUno*. Cuando el usuario pulsa Ctrl+C, el método **keyPhysical** de la ficha cambia el color de *CajaUno*. Este código se anexa al método **keyPhysical** de una ficha:

```
; estaFicha::keyPhysical
method keyPhysical(var eventInfo Event)
if eventInfo.isPreFilter() then
  ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
  if eventInfo.isControlKeyDown() and   ; si el usuario pulsa
Ctrl+C
    eventInfo.vChar() = "C" then
    disableDefault                      ; bloquea los procesos
habituales
    ; asignar el color de CajaUno entre el rojo y el azul
    CajaUno.color=iif(CajaUno.color = Red, Blue, Red)
  endif
else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif

endmethod
```

Vea también setControlKeyDown
isAltKeyDown
isShiftKeyDown

isFromUI

Método Informa de si un KeyEvent fue generado por la interacción del usuario con Paradox.

Tipo KeyEvent

Sintaxis **isFromUI ( )** logical

Descripción Informa de si un KeyEvent fue generado por la interacción del usuario con Paradox o internamente (por ejemplo, por una sentencia de ObjectPAL). Este método devuelve True si el KeyEvent fue generado por el usuario; en caso contrario, devuelve False. **isFromUI** devuelve True sólo para el primer KeyEvent generado por la pulsación de una tecla; para los sucesos o acciones siguientes, **isFromUI** devuelve False. Por ejemplo, cuando el usuario pulsa una tecla, genera un KeyEvent que Paradox puede traducir en uno o más sucesos o acciones; sin embargo, **isFromUI** devuelve True sólo para el primer KeyEvent (el único generado por el usuario).

Vea también Event::isPreFilter

isShiftKeyDown

Método Informa de si la tecla Mayús estaba pulsada durante un KeyEvent.

Tipo KeyEvent

Sintaxis **isShiftKeyDown ( )** Logical

Descripción Devuelve True si la tecla Mayús estaba pulsada en el momento en que se produjo un KeyEvent; en caso contrario, devuelve False.

Ejemplo Consulte el ejemplo de **vChar**.

Vea también setShiftKeyDown
isAltKeyDown
isControlKeyDown

setAltKeyDown

Método Simula que se pulsa y se mantiene pulsada la tecla *Alt* durante un KeyEvent.

Tipo KeyEvent

Sintaxis **setAltKeyDown (** const ***yesNo*** Logical **)**

Descripción Añade información sobre el estado de Alt a un KeyEvent. Es necesario especificar Yes o No. Yes implica que la tecla Alt estaba pulsada durante un KeyEvent; No indica que no estaba pulsada.

Ejemplo El ejemplo siguiente supone que una ficha tiene un cuadro llamado *CajaUno*. Cuando el usuario pulsa Alt+C, el método **keyPhysical** de la ficha cambia el color de *CajaUno*. Este código se anexa al método **keyPhysical** de una ficha:

```
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    if eventInfo.isAltKeyDown() AND     ; si el usuario pulsa
Alt+C
      eventInfo.vChar() = "C" then
      disableDefault                    ; bloquea los procesos
habituales
      ; asignar el color de CajaUno entre el rojo y el azul
      CajaUno.color = iif(CajaUno.color = Red, Blue, Red)
    endif
  else
    ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif

endmethod
```

Para simular que se pulsa Alt+C, el código de este método crea una variable KeyEvent y define su carácter de tecla virtual como "C" y la pulsación de la tecla Alt.

```
; enviarAltC::pushButton
method pushButton(var eventInfo Event)
var
  ke KeyEvent
endVar
ke.setVChar("C")                  ; asigna el carácter C
ke.setAltKeyDown(Yes)             ; asigna el estado de la tecla
Alt como pulsada
estaFicha.keyPhysical(ke)         ; envia el suceso

endmethod
```

Vea también isAltKeyDown
setControlKeyDown

setShiftKeyDown

setChar

Método Especifica un carácter ANSI para un KeyEvent.

Tipo KeyEvent

Sintaxis **setChar (** const ***carácter*** String **)**

Descripción Define que un KeyEvent tenga un carácter ANSI en función del valor de *carácter*, donde *carácter* se evalúa en una cadena de un solo carácter (por ejemplo, a).

Ejemplo Este código se anexa al método estándar **keyChar** de un campo. El método **keyChar** de *CampoUno* convierte cada espacio en un carácter de subrayado conforme el usuario teclea caracteres en el campo:

```
; esteCampo::keyChar
method keyChar(var eventInfo KeyEvent)
if eventInfo.Char() = " " then    ; cuando el usuario escriba un
espacio
  eventInfo.setChar("_")          ; lo transformamos en signo de
subrayar
endif                             ; procesa el resto de las
pulsaciones de teclas
    ; normalmente
endmethod
```

Vea también setVChar
setVCharCode
String::toOEM
String::chr
String::chrOEM
String::keyNameToChr
String::toANSI

setControlKeyDown

Método Simula que se pulsa y se mantiene pulsada la tecla *Ctrl* durante un KeyEvent.

Tipo KeyEvent

Sintaxis **setControlKeyDown (** const ***yesNo*** Logical **)**

Descripción Añade información sobre el estado de *Ctrl* a un KeyEvent. Es necesario especificar Yes o No. Yes implica que la tecla *Ctrl* estaba pulsada durante un KeyEvent; No significa que no estaba pulsada.

Ejemplo El ejemplo siguiente supone que una ficha tiene un cuadro llamado *CajaUno*. Cuando el usuario pulsa Ctrl+C, el método **keyPhysical** de la ficha cambia el color de *CajaUno*. Este código se anexa al método **keyPhysical** de una ficha:

```
; estaFicha::keyPhysical
method keyPhysical(var eventInfo Event)
if eventInfo.isPreFilter() then
  ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
  if eventInfo.isControlKeyDown() and   ; si el usuario pulsa
Ctrl+C
    eventInfo.vChar() = "C" then
    disableDefault                      ; bloquea los procesos
habituales
    ; asignar el color de CajaUno entreel rojo y el azul
    CajaUno.color=iif(CajaUno.color = Red, Blue, Red)
  endif
else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif
endmethod
```

Para simular que se pulsa Ctrl+C, el código de este método crea una variable KeyEvent y define su carácter de tecla virtual como "C" y la pulsación de la tecla Ctrl.

```
; enviarCtrlC::pushButton
method pushButton(var eventInfo Event)
var
 ke KeyEvent
endVar
ke.setChar("C")                  ; asigna el carácter C
ke.setControlKeyDown(Yes)        ; asigna el estado de la tecla
Ctrl como pulsada
estaFicha.keyPhysical(ke)        ; envia el evento
endmethod
```

Vea también isControlKeyDown
setAltKeyDown
setShiftKeyDown

setShiftKeyDown

Método Simula que se pulsa y se mantiene pulsada la tecla *Mayús* durante un KeyEvent.

Tipo KeyEvent

Sintaxis **setShiftKeyDown (** const ***yesNo*** Logical **)**

Descripción Añade información sobre el estado de *Mayús* a un KeyEvent. Es necesario especificar Yes o No. Yes implica que la tecla *Mayús* estaba pulsada durante un KeyEvent; No significa que no estaba pulsada.

Ejemplo El ejemplo siguiente supone que una ficha tiene un cuadro llamado *CajaUno*. Cuando el usuario pulsa Mayús+C, el método **keyPhysical** de la ficha cambia el color de *CajaUno*. Este código se anexa al método **keyPhysical** de una ficha:

```
; estaFicha::keyPhysical
method keyPhysical(var eventInfo Event)
if eventInfo.isPreFilter() then
  ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
  if eventInfo.isShiftKeyDown() and     ; si el usuario pulsa
Mayúsculas+C
    eventInfo.vChar() = "C" then
    disableDefault                      ; bloquea los procesos
habituales
    ; asignar el color de CajaUno entre el rojo y el azul
    CajaUno.color=iif(CajaUno.color = Red, Blue, Red)
  endif
else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif
endmethod
```

Para simular que se pulsa Mayús+C, el código de este método crea una variable KeyEvent y define su carácter de tecla virtual como "C" y la pulsación de la tecla Mayús.

```
; enviarShiftC::pushButton
method pushButton(var eventInfo Event)
var
  ke KeyEvent
endVar
ke.setChar("C")                   ; asigna el carácter C
ke.setShiftKeyDown(Yes)           ; asigna el estado de la tecla
Mayúsculas como
    ; pulsada
estaFicha.keyPhysical(ke)         ; envia el suceso
endmethod
```

Vea también isShiftKeyDown
setAltKeyDown
setControlKeyDown

setVChar

Método Especifica un carácter virtual de Windows para un KeyEvent.

Tipo KeyEvent

Sintaxis **setVChar (** const ***carácter*** String **)**

Descripción Especifica en *carácter* una cadena de un carácter para un KeyEvent. Utilice **setVChar** para definir una cadena de código de carácter virtual para una sola letra. La cadena de código de carácter virtual para cualquier letra es la letra mayúscula. Por ejemplo, la cadena de código de carácter virtual para la letra k es "K" (sólo en mayúscula).

Ejemplo Consulte el ejemplo de setAltKeyDown.

Vea también setChar
setVCharCode
String::chr
String::chrOEM
String::chrToKeyName

setVCharCode

Método Especifica un carácter virtual de Windows para un KeyEvent.

Tipo KeyEvent

Sintaxis **setVCharCode (** const ***constanteVK_*** SmallInt **)**

Descripción Especifica en *constante_VK* una constante de carácter virtual de Windows para un KeyEvent. ObjectPAL proporciona constantes para *constante_VK*; consulte Keyboard en el cuadro de diálogo Constantes.

Ejemplo Este código se anexa al método estándar **keyPhysical** de una ficha. Cuando el usuario teclea **?**, este código activa el sistema de ayuda de Paradox.

```
; estaFicha::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    if eventInfo.char() = "?" then      ; si el usuario pulsa la
tecla [?]
      eventInfo.setVCharCode(VK_HELP)   ; llarmar al sistema de
ayuda estándar
    endif
  else
    ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif

endmethod
```

Vea también setChar
setVChar
String**::**keyNameToVKCode

vChar

Método Devuelve un carácter virtual de Windows.

Tipo KeyEvent

Sintaxis **vChar ( )** String

Descripción Devuelve el nombre de una tecla virtual de Windows en forma de cadena. En el caso de las letras, el nombre de tecla virtual es igual que la letra, pero siempre en mayúsculas. Así, es posible emplear **vChar** para comprobar si el usuario tecleó una **i** minúscula o una **I** mayúscula. ObjectPAL proporciona constantes para los nombres de tecla virtual; consulte Keyboard en el cuadro de diálogo Constantes.

Ejemplo El ejemplo siguiente supone que una ficha contiene un cuadro llamado *CajaUno*. Cuando el usuario pulsa una de las teclas del cursor, este código desplaza *CajaUno* en incrementos de 100 twips. Si se mantiene pulsada la tecla *Mayús* en combinación con una tecla del cursor, *CajaUno* se desplaza 1000 twips. Puesto que **vChar** devuelve el nombre de una tecla virtual como una cadena, este código debe comparar los nombres de tecla con valores de cadena como "VK_LEFT". Este código se anexa al método estándar **keyPhysical** de una ficha.

```
; estaFicha::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  Tecla String              ; nombre de la tecla que se pulsa
  PosXY Point               ; posiciones X e Y de la Caja
  Salto SmallInt            ; número de puntos (Point) a mover
la caja
  x, y  LongInt             ; coordenadas de la Caja
endVar

if eventInfo.isPreFilter()
    then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    disableDefault                      ; no ejecutamos el
código estándar

    Tecla = eventInfo.vChar()           ; almacenar en Tecla la
cadena vChar
    PosXY = CajaUno.position            ; PosXY almacena la
posición actual
                ; de la caja
    x = posPt.x()                       ; x almacena la posición
horizontal
    y = posPt.y()                       ; y almacena la posición
vertical

    ; si se mantiene pulsada la tecla Mayúsculas mientras se
pulsa la tecla
    ; de movimiento asignamos un número amplio a Salto, si no
un número
    ; más pequeño
    Salto = iif(eventInfo.isShiftKeyDown(), 1000, 100)
```

```
    ; este bloque asigna valores a las variables X o Y de
acuerdo con
    ; la combinación de teclas que el usuario pulsa
    switch
      case Tecla = "VK_LEFT"  : x = x  Salto
      case Tecla = "VK_RIGHT" : x = x + Salto
      case Tecla = "VK_UP"    : y = y  Salto
      case Tecla = "VK_DOWN"  : y = y + Salto
      otherwise               : enableDefault ; permitir que se
ejecute
                              ; el código estándar
    endswitch

    ; ahora movemos la caja a la posición especificada por las
variables X e Y,
    ; y mostramos el nombre de la tecla virtual asociada con la
pulsación
    CajaUno.position = Point(x,y)
    message("El valor de vChar() era " + Tecla)

  else
    ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha
endif

endmethod
```

Vea también

char
String::ansiCode
String::chrToKeyName
String::toANSI
String::toOEM

vCharCode

Método Devuelve el valor entero de un carácter virtual de Windows.

Tipo KeyEvent

Sintaxis **vCharCode ( )** SmallInt

Descripción Devuelve el valor entero de un carácter virtual de Windows.

ObjectPAL proporciona constantes para los caracteres virtuales; consulte Keyboard en el cuadro de diálogo Constantes.

Ejemplo En este ejemplo, supóngase que una ficha tiene un campo llamado *esteCampo*. Cuando el usuario teclea un valor en *esteCampo* y pulsa *Intro*, el código crea y ejecuta una consulta en función del valor del campo. Este código se anexa al método estándar **keyPhysical** de *esteCampo*.

```
;esteCampo::keyPhysical
method keyPhysical(var eventInfo KeyEvent)
var
  Nombre   String     ; utilizada como variable de tilde
  Consulta Query      ; la sentencia de la consulta
  tv       TableView        ; apuntador a tableView
endVar


if eventInfo.vCharCode() = VK_RETURN then     ; si el usuario
pulsa Intro
  Nombre = self.value                         ; almacena el
valor del campo
  Consulta = Query

          c:\pdoxwin\sample\Vidamar.db | Nombre Común  | Nombre
de la Especie |
                                          | check ~Nombre | check
|

            endQuery

executeQBE(Consulta, "Peces.db")  ; ejecuta la consulta.
Escribimos el contenido
          ; en la tabla Peces
tv.open("Peces")                  ; vemos Peces.db
endif

endmethod
```

Vea también vChar

data

Método Devuelve información sobre un MenuEvent.

Tipo MenuEvent

Sintaxis **data ( )** LongInt

Descripción Este método sólo deberían utilizarlo los programadores de Windows. **data** devuelve el argumento *lParam* (normalmente cero) de mensajes específicos de Windows, como WM_SYSCOMMAND y WM_COMMAND. Para más información, consulte la documentación de programación de Windows.

Vea también id
setData
setId

id

Principiante

Tipo MenuEvent

Método Devuelve el ID (identificador) de un MenuEvent.

Sintaxis **id ( )** SmallInt

Descripción Devuelve el número ID de un MenuEvent. ObjectPAL proporciona constantes (como MenuFileOpen) para muchas opciones de menú habituales; consulte MenuCommands en el cuadro de diálogo Constantes.

Ejemplo Este código se anexa al método estándar **menuAction** de una ficha. Cuando el usuario selecciona Cerrar en el menú de control, intenta activar o desactivar una ventana de diseño o elige Archivo|Salir, el método pide al usuario que confirme si se sale o no de la ficha.

```
;estaFicha::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Archivo String
  tv TableView
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
  if eventInfo.id() = MenuControlClose OR
    eventInfo.id() = MenuFileExit OR
    eventInfo.id() = MenuFormDesign then
    disableDefault                      ; bloquear la salida
    Resp = msgQuestion("Confirme", "¿Está seguro de que quiere
salir?")
    if Resp = "Yes" then
      dodefault
    endif
  endif
endif
endmethod
```

El ejemplo siguiente demuestra cómo puede utilizarse el argumento de ID de menú con **addText** para referenciar a las opciones de menú por su número (idealmente, constantes definidas por el usuario), en lugar de por su nombre. Este código establece constantes definidas por el usuario para que se recuerden con mayor facilidad las asignaciones de ID de menú. El código siguiente define constantes globales a *páginaUno*.

```
;páginaUno::Const
Const
  ; definimos constantes para los identificadores de los menús
  ; los valores reales (1, 2 y 3) son arbitrarios
  MenuDeHora  = 1
  MenuDeFecha = 2
```

```
  MenuDeAyuda = 3
endConst
```

El código siguiente se anexa al método **open** de *páginaUno*. Para controlar los atributos de visualización de menú, este código emplea constantes estándar como MenuEnabled. Para identificar cada opción de menú por su número, el código emplea constantes (TimeMenu, DateMenu y HelpMenu) definidas en la ventana Var de *páginaUno*.

```
;páginaUno::open
method open(var eventInfo Event)
var
  MenúPrincipal Menu
  UtilDespleg PopUpMenu
endVar

 ; construimos un menú desplegable y utlizamos las constantes
 ; (p.ej.: MenúdeHora) definidas en la ventana de constantes
 : (Const) para esta Página
UtilDespleg.addText("&Hora", MenuEnabled, MenúDeHora)
UtilDespleg.addText("&Fecha", MenuEnabled, MenúDeFecha)
 ; asociamos el menú desplegable al ítem Utilidades del menú
principal
MenúPrincipal.addPopUp("&Utilidades", UtilDespleg)

 ; añadimos "Ayuda" a la barra de menú y justificamos
 :"Ayuda" por la derecha con \008
MenúPrincipal.addText("\008&Ayuda", MenuEnabled, MenuDeAyuda)

MenúPrincipal.show()                ; mostramos el menú

endmethod
```

El código siguiente se anexa al método **menuAction** de *páginaUno*. Este método evalúa las selecciones en el menú por su número ID, y no por el nombre especificado en *nombreMenú*.

```
;páginaUno::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Selección SmallInt
endVar

Selección = eventInfo.id()      ; asignamos un valor constante
a Selección
  ; Ahora utilizamos las constantes para determinar qué menú se
seleccionó     switch
  case Selección = MenúDeHora :
    msgInfo("Hora actual", time())
  case Selección = MenúDeFecha :
    msgInfo("Fecha de hoy", today())
  case Selección = MenúDeAyuda :
    ; cambiamos el identificador del menú por una constante
estándar
    ; (MenuHelpContents)-que efectivamente abre el sistema de
ayuda estándar.
    eventInfo.setId(MenuHelpContents)
```

```
        eventInfo.setReason(MenuDesktop)
    endSwitch

    endmethod
```

Vea también setId

isFromUI

Método Informa de si un MenuEvent fue generado por la interacción del usuario con Paradox.

Tipo MenuEvent

Sintaxis **isFromUI ( )** logical

Descripción Informa de si un suceso fue generado por la interacción del usuario con Paradox o internamente (por ejemplo, por una sentencia de ObjectPAL). Este método devuelve True si el suceso fue generado por el usuario; en caso contrario, devuelve False.

Vea también Event::isPreFilter

menuChoice

Principiante

Tipo MenuEvent

Método Devuelve una cadena que contiene una opción elegida en un menú.

Sintaxis **menuChoice ( )** String

Descripción Devuelve una cadena que contiene una opción elegida en un menú. Utilice **menuChoice** para modificar el método estándar **menuAction** de un objeto y especificar cómo responde ese objeto a las opciones de menú.

Nota: Si la definición de una opción de menú incluye una letra subrayada (por ejemplo, "&Imprimir"), recuerde incluir el ampersand en la cadena de comparación; por ejemplo, el código siguiente compara el valor devuelto por **menuChoice** con la cadena "&Imprimir":

```
if eventInfo.menuChoice() = "&Imprimir" then
  ; imprimimos el informe
endif
```

Ejemplo Este ejemplo supone que una ficha contiene, al menos, un campo memo llamado *esteCampoMemo*. Cuando el usuario accede a *esteCampoMemo*, el método estándar **arrive** muestra un menú que permite al usuario realizar operaciones básicas de corte y pegado. El método estándar **menuAction** anexado a *esteCampoMemo* utiliza **menuChoice** para evaluar la selección del usuario y realizar la acción pertinente. Aunque este ejemplo repite el comportamiento de los menús por defecto, esta técnica es necesaria cuando los menús por defecto se sustituyen por menús personalizados.

Este código se anexa al método estándar **arrive** de *esteCampoMemo*:

```
;esteCampoMemo::arrive
method arrive(var eventInfo MoveEvent)
Var
  Desplegable PopUpMenu
  Principal   Menu
endVar

  Desplegable.addText("&Cortar")           ; creamos un menú
desplegable
  Desplegable.addText("&Copiar")
  Desplegable.addText("&Pegar")

Principal.addPopUp("&Editar", EditPopUp) ; añadimos el menú
desplegable
                                        : al ítem de la barra de
menú     Principal.show()                        ; mostramos el
menú
endmethod
```

Este código se anexa al método estándar **menuAction** de *esteCampoMemo*. Obsérvese que las comparaciones de la sentencia **switch...endSwitch** debe incluir el ampersand, como "&Cortar".

```
;esteCampoMemo::menuAction
```

```
method menuAction(var eventInfo MenuEvent)     v
ar
  Selección String
endVar
Selección = eventInfo.menuChoice()      ; almacenamos la
selección del
                                     :   menú  en Selección

; ahora respondemos a la selección apropiadamente
switch
  case Selección = "&Cortar" : self.action(EditCutSelection)
  case Selección = "&Copiar" : self.action(EditCopySelection)
  case Selección = "&Pegar"  : self.action(EditPaste)
endSwitch
endmethod
```

Este código se anexa al método estándar **depart** de *esteCampoMemo*. Cuando el usuario sale de *esteCampoMemo*, este código elimina el menú. En este ejemplo, los menús por defecto reaparecen cuando el usuario sale del campo. En una situación similar, tal vez desee mostrar otra estructura de menús personalizados.

```
;esteCampoMemo::depart
method depart(var eventInfo MoveEvent)
removeMenu()          ; elmiinamos el menú Editar
endmethod
```

Vea también id
setId

reason

Método Informa del tipo de menú elegido.

Tipo MenuEvent

Sintaxis **reason ( )** SmallInt

Descripción Devuelve un valor entero que informa de por qué se ha producido un MenuEvent. Las constantes de Reason para MenuEvent se producen cuando se ejecuta un método estándar **menuAction**. ObjectPAL proporciona las constantes siguientes para comprobar el valor devuelto por **reason** (también enumeradas en el cuadro de diálogo Constantes bajo MenuReason):

MenuNormal implica que el MenuEvent fue causado por un comando de menú o un botón de la barra rápida que cambia dependiendo de qué ventana se utiliza.

MenuControl implica que el MenuEvent fue causado por un comando del menú de control o por el botón de maximizar o minimizar.

MenuDesktop significa que el MenuEvent fue causado por uno de los comandos de menú básicos del Escritorio.

Ejemplo En este ejemplo, el método **menuAction** de la ficha examina cada MenuEvent para determinar la constante Reason del MenuEvent. La constante Reason se muestra, entonces, en el objeto de campo *CampoRazón*.

```
;estaFicha::menuAction
method menuAction(var eventInfo MenuEvent)
var
  Razón String
endVar
  if eventInfo.isPreFilter() then
    ; seleccionamos la razón y asignamos la cadena equivalente
en Razón
    Razón = iif(eventInfo.reason() = MenuNormal, "MenuNormal",
           iif(eventInfo.reason() = MenuControl, "MenuControl",
            MenuDesktop"))
    IdRazón = eventInfo.reason()
    CampoRazón = String(IdRazón) + " " + Razón
     ; el código fuente que hay aquí se ejecuta antes de cada
objeto
   else
     ; el código fuente que haya aquí se ejecuta después (o
para la Ficha)

  endif
endmethod
```

Vea también setReason

setData

Método Especifica información sobre un MenuEvent.

Tipo MenuEvent

Sintaxis **setData (** const ***datosMenú*** LongInt)

Descripción Este método sólo deberían utilizarlo los programadores de Windows. **setData** especifica el argumento *lParam* (normalmente cero) de mensajes específicos de Windows, como WM_SYSCOMMAND y WM_COMMAND. Para más información, consulte la documentación de programación de Windows.

Vea también data
id
setId

setId

Método Especifica el ID de un MenuEvent.

Tipo MenuEvent

Sintaxis **setId (** const ***idAcción*** SmallInt)

Descripción Especifica en *idAcción* una acción que se realizará como resultado de una opción de menú. ObjectPAL define constantes para *idAcción*; consulte MenuCommands en el cuadro de diálogo Constantes.

Si cambia el ID de un MenuEvent con **setId**, tal vez deba cambiar también la razón de ese MenuEvent con **setReason**.

Nota: En muchas circunstancias, **menuAction** debería utilizarse desde el tipo Form o el tipo UIObject para invocar un comando de menú. Aunque es posible cambiar el Reason e ID de un MenuEvent existente (*eventInfo*), y también es posible crear un nuevo MenuEvent y definir el Reason e ID de ese suceso (sólo deberían intentarlo los programadores avanzados), está técnica no siempre es aconsejable.

Ejemplo Consulte el ejemplo de id.

Vea también
id
Event::errorCode
Event::setErrorCode
Form::menuAction
UIObject::menuAction

setReason

Método Especifica una razón para la generación de un MenuEvent.

Tipo MenuEvent

Sintaxis **setReason (** const ***idRazón*** SmallInt)

Descripción Especifica una razón para la generación de un MenuEvent. Este método toma una de las constantes siguientes como argumento (mostradas en el cuadro de diálogo Constantes bajo MenuReasons):

MenuNormal implica que el MenuEvent fue causado por un comando de menú o un botón de la barra rápida que cambia dependiendo de qué ventana se utiliza.

MenuControl implica que el MenuEvent fue causado por un comando del menú de control o por el botón de maximizar o minimizar.

MenuDesktop significa que el MenuEvent fue causado por uno de los comandos de menú básicos del Escritorio.

Nota: En muchas circunstancias, **menuAction** debería utilizarse desde el tipo Form o el tipo UIObject para invocar un comando de menú. Aunque es posible cambiar la razón e ID de un MenuEvent existente (eventInfo), y también es posible crear un nuevo MenuEvent y definir el Reason e ID de ese suceso (sólo deberían intentarlo los programadores avanzados), está técnica no siempre es aconsejable.

Ejemplo Consulte el ejemplo de id.

Vea también reason
id
Form::menuAction
UIObject::menuAction

getMousePosition

Método Informa de la posición del ratón.

Tipo MouseEvent

Sintaxis **1. getMousePosition** (var ***p*** Point)

2. getMousePosition (var ***posiciónX*** LongInt, var ***posiciónY*** LongInt)

Descripción Escribe la posición del ratón en una variable. La sintaxis 1 almacena el valor en una variable Point, p, mientras que la sintaxis 2 lo almacena en posiciónX e *posiciónY*, dos variables LongInt que representan las coordenadas *x* e *y* del puntero del ratón. Cuando se emplea la sintaxis 1, es posible utilizar los métodos del tipo Point (por ejemplo, **isLeft** e **isRight**) para obtener más información.

Este método obtiene la posición del ratón en el momento del MouseEvent; su posición actual puede ser diferente.

Ejemplo Este ejemplo obtiene la posición del último suceso **mouseUp** y dibuja un pequeño círculo en esa posición. El método comprueba, en primer lugar, si el suceso procede del interfaz de usuario (este caso, del usuario) y, si el destino del suceso es la página misma (en oposición a que fue lanzado hacia arriba a la página desde algún otro objeto). Este método dibuja el círculo sólo cuando el usuario hace clic sobre la página.

```
;páginaUno::mouseUp method mouseUp(var eventInfo MouseEvent)
var
  Objeto UIObject
  x, y  LongInt    ; coordenadas del punto
endVar
if eventInfo.isFromUI() AND eventInfo.isTargetSelf() then
  ; creamos un pequeño círculo azul en la posición del ratón
  eventInfo.getMousePosition(x, y)
  Objeto.create(ellipseTool, x, y, 100, 100)
  Objeto.Color = DarkBlue
  Objeto.Visible = True
endif endmethod
```

Vea también setMousePosition
getObjectHit
Event::getTarget

getObjectHit

Método Crea un gestor para el UIObject que recibió el suceso.

Tipo MouseEvent

Sintaxis **getObjectHit** (var **destino** UIObject) Logical

Descripción Devuelve en destino un gestor al destino del suceso. Este método es útil para los MouseEvent internos que ejecutan **mouseExit** y **mouseEnter**. **getObjectHit** puede devolver un objeto distinto al devuelto por **getTarget** durante un método **mouseExit** o **mouseEnter**.

Ejemplo El código siguiente se anexa al método **mouseExit** de una ficha. Cuando el ratón sale de un objeto, aparece un mensaje en la ventana de estado mostrando el nombre del objeto destino (**getTarget**) frente al nombre del objeto tocado (**getObjectHit**).

```
;estaFicha::mouseExit method mouseExit(var eventInfo
MouseEvent)
var
  ObjetoDestino,
  Objeto     UIObject
endVar
if eventInfo.isPreFilter()
  then    ; el código fuente que haya aquí se ejecuta para cada

          ; objeto de la ficha
  eventInfo.getTarget(ObjetoDestino)
  eventInfo.getObjectHit(Objeto)
  message(ObjetoDestino.Name +   contra   + Objeto.Name)
else
  ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha              endif
endmethod
```

Vea también getMousePosition
Event::getTarget

isControlKeyDown

Método Informa de si el usuario ha mantenido pulsada (o está pulsando) la tecla Ctrl durante un MouseEvent.

Tipo MouseEvent

Sintaxis **isControlKeyDown** ( ) Logical

Descripción Devuelve True si la tecla Ctrl está pulsada durante un MouseEvent; en caso contrario, devuelve False.

Ejemplo Este ejemplo examina el estado del teclado durante un clic del ratón para averiguar si debe insertar automáticamente el mayor valor del rango, el valor menor o el valor por defecto. Las constantes siguientes se declaran en la ventana Const de *campoUno*:

```
;campoUno::Const
Const
  ValorAlto = Number(10000)
  ValorBajo = Number(100000)
  ValorPorDefecto = Number(50000)
endConst
```

Este es el código del método **mouseUp** de *campoUno*:

```
;campoUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
; insertamos los valores alto, bajo y por defecto dependiendo
de cómo se
; utilizó el ratón
switch
  case eventInfo.isControlKeyDown() : self.Value = ValorBajo
                                          message( Ctrl+clic )

  case eventInfo.isShiftKeyDown()   : self.Value = ValorAlto
                                          message( Mays+clic )

  otherwise                         : self.Value =
ValorPorDefecto
                                          message( Clic )
endswitch
endmethod
```

Vea también setControlKeyDown
isShiftKeyDown

isFromUI

Método Informa de si un MouseEvent fue generado por la interacción del usuario con Paradox.

Tipo MouseEvent

Sintaxis **isFromUI ( )** logical

Descripción Informa de si un suceso fue generado por la interacción del usuario con Paradox o internamente (por ejemplo, por una sentencia de ObjectPAL). Este método devuelve True si el suceso fue generado por el usuario; en caso contrario, devuelve False.

Vea también Event**::**isPreFilter

isInside

Método Informa de si el ratón se halla dentro del borde del objeto destino.

Tipo MouseEvent

Sintaxis **isInside** ( ) Logical

Descripción Informa de si el ratón está dentro del borde del objeto destino en el momento del suceso.

Ejemplo En este ejemplo, el método **mouseUp** de *BotónUno* informa de si el último suceso está dentro de los bordes del objeto destino. Si el usuario hace clic sobre *BotónUno*, el MouseEvent **mouseUp** se envía a *BotónUno* e **isInside** devuelve True. Si arrastra el ratón desde el interior del botón hasta el exterior, de forma que el **mouseUp** se produzca fuera de los bordes de *BotónUno*, el MouseEvent se produce para *BotónUno* y activa el método **mouseUp**, pero **isInside** devuelve False para ese MouseEvent.

```
;BotónUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
msgInfo( ¿ Está dentro el último suceso ? ,
eventInfo.isInside())
endmethod
```

Vea también getObjectHit
setInside

isLeftDown

Método Informa de si el botón izquierdo (o principal) del ratón está pulsado durante un MouseEvent.

Tipo MouseEvent

Sintaxis **isLeftDown** ( ) Logical

Descripción Devuelve True si el botón izquierdo del ratón se mantiene pulsado durante un MouseEvent (por ejemplo, mientras se arrastra el ratón); en caso contrario, devuelve False.

Ejemplo En el ejemplo siguiente, supóngase que el campo *Notas Lugar* de la tabla *Lugares* está situado en una ficha. Este método, anexado al método **mouseMove** de *Notas Lugar*, comprueba si el botón izquierdo o derecho del ratón está pulsado en el momento del desplazamiento. Si está pulsado el botón izquierdo, se selecciona el campo desde el punto del clic hasta el comienzo del campo. Si está pulsado el botón derecho, se selecciona el campo desde el punto del clic hasta el final del campo.

```
;Notas Lugar::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)         ; selecciona desde el punto
hasta el principio
else
  if eventInfo.isRightDown() then
    self.action(SelectBottom)      ; selecciona desde el punto
hasta el final
  endif
endif
endmethod
```

Vea también isRightDown
isMiddleDown
setLeftDown

isMiddleDown

Método Informa de si el botón central del ratón está pulsado durante un MouseEvent.

Tipo MouseEvent

Sintaxis **isMiddleDown** ( ) Logical

Descripción Devuelve True si el botón central del ratón se mantiene pulsado durante un MouseEvent; en caso contrario (aunque el ratón no tenga botón central), devuelve False.

Ejemplo Este ejemplo supone que una ficha contiene un botón llamado *enviarMovimiento* y un campo de la tabla *Lugares* llamado *Notas Lugar*. El método **pushButton** de *enviarMovimiento* construye un MouseEvent con el botón central del ratón y envía el MouseEvent fuera del campo *Notas Lugar*.

```
;enviarMovimiento::pushButton
method pushButton(var eventInfo Event)
var
  Ratón MouseEvent        ; declaramos un MouseEvent para
enviarlo
  ui    UIObject
endVar
ui.attach( Notas Lugar )  ; asociamos con Notas Lugar
Ratón.setMiddleDown(Yes)  ; asignamos botón del medio pulsado
con MouseEvent     ui.mouseMove(Ratón)       ; servimos el
suceso a mouseMove para Notas Lugar endmethod
```

Este método se anexa al método **mouseMove** de *Notas Lugar*. Si el botón central está pulsado durante el MouseEvent, el método se desplaza al comienzo de la palabra actual y selecciona toda la palabra.

```
;Notas Lugar::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isMiddleDown() then
  self.action(MoveLeftWord)    ; vamos al principio de la
palabra
  self.action(SelectRightWord) ; seleccionamos la palabra
completa
endif
endmethod
```

Vea también setMiddleDown
isLeftDown
isRightDown

isRightDown

Método Informa de si el botón derecho del ratón está pulsado durante un MouseEvent.

Tipo MouseEvent

Sintaxis **isRightDown** ( ) Logical

Descripción Devuelve True si el botón derecho del ratón se mantiene pulsado durante un MouseEvent (por ejemplo, al realizar un arrastre con el botón derecho); en caso contrario, devuelve False.

Ejemplo En el ejemplo siguiente, supóngase que el campo *Notas Lugar* de la tabla *Lugares* está situado en una ficha. El método **mouseMove** de *Notas Lugar* comprueba si está pulsado el botón izquierdo o derecho en el momento del desplazamiento. Si está pulsado el botón izquierdo, se selecciona el campo desde el punto del clic hasta el comienzo del campo; si está pulsado el botón derecho, se selecciona el campo desde el punto del clic hasta el final del campo.

```
;Notas Lugar::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)         ; selecciona desde el punto
hasta el principio
else
   if eventInfo.isRightDown() then
     self.action(SelectBottom)   ; selecciona desde el punto
hasta el final
   endif
endif
endmethod
```

Vea también setRightDown
isLeftDown
isMiddleDown

isShiftKeyDown

Método Informa de si la tecla Mayúsestá pulsada durante un MouseEvent.

Tipo MouseEvent

Sintaxis **isShiftKeyDown** ( ) Logical

Descripción Devuelve True si la tecla Mayússe mantiene pulsada durante un MouseEvent; en caso contrario, devuelve False.

Ejemplo El ejemplo siguiente se anexa al método **mouseUp** del campo *Notas Lugar*. Cuando el usuario pulsa Mayúsy hace clic simultáneamente, se selecciona la palabra situada a la derecha del punto de inserción.

```
;Notas Lugar::mouseUp
method mouseUp(var eventInfo MouseEvent)
;si la tecla de Mayús está pulsada, seleccionamos la palabra a
la derecha
if eventInfo.isShiftKeyDown() then
  self.action(SelectRightWord)
endif
endmethod
```

Vea también isControlKeyDown
setShiftKeyDown

setControlKeyDown

Método Simula que se pulsa y se mantiene pulsada la tecla Ctrldurante un MouseEvent.

Tipo MouseEvent

Sintaxis **setControlKeyDown** (const yesNo Logical)

Descripción Añade información sobre el estado de Ctrl para un MouseEvent. El programador debe especificar Yes o No. Yes implica que la tecla Ctrl se mantuvo pulsada durante un KeyEvent; No significa que no estaba pulsada.

Ejemplo El ejemplo siguiente crea un MouseEvent y Ctrl como Yes. El suceso se envía entonces al método estándar **mouseUp** de un campo llamado *Campo*. Este método se anexa al método **pushButton** de un botón llamado *EnviarCtrl*.

```
;EnviarCtrl::pushButton
method pushButton(var eventInfo Event)
var
  ctrlMsEvent MouseEvent             ; declarar el suceso
endVar
ctrlMsEvent.setControlKeyDown(Yes)   ; definimos la tecla
Control     Campo.mouseUp(ctrlMsEvent)           ; enviar el
suceso
endmethod
```

Este código se anexa al método **mouseUp** de *Campo*. Este método comprueba si la tecla Ctrl estaba pulsada cuando se hizo clic. Si es así, todas las letras del valor del campo se cambia a minúsculas.

```
;Campo::mouseUp
method mouseUp(var eventInfo MouseEvent)
if eventInfo.isControlKeyDown() then ; comprobamos el estado de
la tecla Control
  self.Value = lower(self.Value)     ; cambiamos a minúsculas

endif endmethod
```

Vea también isControlKeyDown
setShiftKeyDown

setInside

Método Define que el ratón se sitúe dentro del objeto actual.

Tipo MouseEvent

Sintaxis **setInside** (const trueFalse Logical) Logical

Descripción Define que el MouseEvent se sitúe dentro del objeto actual.

Ejemplo En este ejemplo, el método **mouseUp** de *EnviarUnsuceso* utiliza **setInside** para cambiar la variable *eventInfo* y envía el suceso a *botónUno*.

```
;EnviarUnsuceso::mouseUp
method mouseUp(var eventInfo MouseEvent)
eventInfo.setInside(Yes)
botónUno.mouseUp(eventInfo)
endmethod
```

Vea también isInside
getObjectHit

setLeftDown

Método Simula la pulsación del botón izquierdo del ratón.

Tipo MouseEvent

Sintaxis **setLeftDown** (const yesNo Logical)

Descripción Añade información sobre el estado del botón izquierdo del ratón para un MouseEvent. Es necesario especificar Yes o No. Yes implica que se hizo clic con el botón izquierdo; No significa que no estaba pulsado.

Ejemplo Este ejemplo construye un MouseEvent con el botón izquierdo pulsado. El MouseEvent se envía entonces al método **mouseMove** de *Notas_Lugar*. Este código se anexa al método **pushButton** de *EnviarBotónIzquierdo*:

```
;EnviarBotónIzquierdo::pushButton
method pushButton(var eventInfo Event)
var
  MoverRatón MouseEvent          ; creamos un suceso de ratón
  ui          UIObject
endVar
leftMoveMouse.setLeftDown(Yes) ; asignamos Yes al Botón Izquierdo
ui.attach( Notas_Lugar )
ui.mouseMove(leftMoveMouse)      ; enviamos el suceso a Notas_Lugar
endmethod
```

Este código se anexa al método **mouseMove** de *Notas_Lugar*:

```
;Notas_Lugar::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)        ; selecciona desde el punto
hasta el principio     else
  if eventInfo.isRightDown() then
    self.action(SelectBottom)       ; selecciona desde el punto
hasta el final
  endif
endif
endmethod
```

Vea también isLeftDown
setMiddleDown
setRightDown

setMiddleDown

Método Simula la pulsación del botón central del ratón.

Tipo MouseEvent

Sintaxis **setMiddleDown** (const yesNo Logical)

Descripción Añade información sobre el estado del botón central del ratón para un MouseEvent. Es necesario especificar Yes o No. Yes implica que se hizo clic con el botón central; No significa que no estaba pulsado.

Ejemplo Este ejemplo supone que una ficha contiene un botón llamado *enviarMovimiento* y un objeto de campo de la tabla *Lugares* llamado *Notas_Lugar*. El método **pushButton** de *enviarMovimiento* construye un MouseEvent con el botón central pulsado y envía el MouseEvent al objeto de campo *Notas_Lugar*.

```
;EnviarMovimiento::pushButton
method pushButton(var eventInfo Event)
var
  Ratón MouseEvent          ; declaramos un MouseEvent para enviarlo
  ui    UIObject
endVar
ui.attach( Notas_Lugar )   ; asociamos con Notas_Lugar
Ratón.setMiddleDown(Yes)   ; asignamos botón del medio pulsado con
MouseEvent
ui.mouseMove(Ratón)        ; servimos el suceso a mouseMove para
Notas_Lugar
endmethod
```

Este método se anexa al método **mouseMove** de *Notas_Lugar*. Si el botón central está pulsado para el MouseEvent, el método se desplaza al principio de la palabra actual y selecciona toda la palabra.

```
;Notas_Lugar::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isMiddleDown() then
  self.action(MoveLeftWord)    ; vamos al principio de la
palabra
  self.action(SelectRightWord) ; seleccionamos la palabra
completa
endif
endmethod
```

Vea también isMiddleDown
setLeftDown
setRightDown

setMousePosition

Método Define la posición del ratón para un suceso.

Tipo MouseEvent

Sintaxis **1. setMousePosition** (const ***posiciónX*** LongInt, const ***posiciónY*** LongInt)
2. setMousePosition (const ***p*** Point)

Descripción Añade información sobre la posición del ratón para un MouseEvent. posiciónX y posiciónY especifican las coordenadas x e y en twips, respecto al ángulo superior izquierdo del contenedor del objeto destino.

Ejemplo El ejemplo siguiente crea un nuevo suceso, define la posición del ratón 500 twips a la derecha y por debajo de la posición actual y envía un suceso al método **mouseRightUp** del mismo objeto. Este código se anexa al método **mouseUp** de un objeto llamado *CajaUno*:

```
;CajaUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  Suceso MouseEvent
endVar     ; asignamos a la nueva posición la actual más 500,
500
Suceso.setMousePosition(eventInfo.x() + 500, eventInfo.y() +
500)
mouseRightUp(Suceso)            ; enviamos el nuevo suceso
endmethod
```

Vea también getMousePosition

setRightDown

Método Simula la pulsación del botón derecho del ratón.

Tipo MouseEvent

Sintaxis **setRightDown** (const ***yesNo*** Logical)

Descripción Añade información sobre el estado del botón derecho del ratón para un MouseEvent. Es necesario especificar Yes o No. Yes implica que se hizo clic con el botón derecho; No significa que no estaba pulsado.

Ejemplo Este ejemplo construye un MouseEvent con el botón derecho pulsado. El MouseEvent se envía entonces al método **mouseMove** de *Notas_Lugar*. Este código se anexa al método **pushButton** de *enviarBotónDerecho*:

```
;EnviarBotónDerecho::pushButton
method pushButton(var eventInfo Event)
var
  MoverRatón MouseEvent      ; declaramos el suceso
  ui         UIObject
endVar
MoverRatón.setRightDown(Yes) ; asignamos como pulsado el botón
derecho     ui.attach( Notas_Lugar )
ui.mouseMove(MoverRatón)     ; enviamos el suceso a Notas_Lugar

endmethodç
```

Este código se anexa al método **mouseMove** de *Notas_Lugar*:

```
;Notas_Lugar::mouseMove
method mouseMove(var eventInfo MouseEvent)
if eventInfo.isLeftDown() then
  self.action(SelectTop)       ; selecciona desde el punto
hasta el principio     else
  if eventInfo.isRightDown() then
    self.action(SelectBottom)  ; selecciona desde el punto
hasta el final
  endif
endif
endmethod
```

Vea también isRightDown
setMiddleDown
setLeftDown

setShiftKeyDown

Método Simula que se pulsa y se mantiene pulsada la tecla Mayús.

Tipo MouseEvent

Sintaxis **setShiftKeyDown** (const ***yesNo*** Logical)

Descripción Añade información sobre el estado de Mayúspara un MouseEvent. Es necesario especificar Yes o No. Yes implica que la tecla Mayús estaba pulsada; No significa que no estaba pulsada.

Ejemplo El código siguiente crea un MouseEvent y define Mayús como Yes. El suceso se envía entonces al método estándar **mouseUp** de un campo llamado *CampoUc*. Este método se anexa al método **pushButton** de un botón llamado *EnviarMayúsculas*.

```
;EnviarMayúsculas::pushButton
method pushButton(var eventInfo Event)
var
sucesoRatón MouseEvent               ; declaramos el suceso

endVar
sucesoRatón.setShiftKeyDown(Yes)     ; ajustar la tecla de
Mayúsculas
CampoUc.mouseUp(shiftMsEvent)        ; enviamos el suceso
endmethod
```

Este código se anexa al método **mouseUp** de *CampoUc*. Este método comprueba si la tecla Mayús está pulsado cuando se hace clic con el ratón. Si es así, las letras del valor del campo se cambian a mayúsculas.

```
;CampoUc::mouseUp
method mouseUp(var eventInfo MouseEvent)
if eventInfo.isShiftKeyDown() then  ; Comprobar el estado de la
tecla Mayús
   self.Value = upper(self.Value)   ; cambiamos a mayúsculas

endif
endmethod
```

Vea también isShiftKeyDown
setControlKeyDown

setX

Método Especifica la coordenada horizontal de la posición del puntero. Las coordenadas deben especificarse respecto al ángulo superior izquierdo del objeto actual.

Tipo MouseEvent

Sintaxis **setX** (const ***posiciónX*** LongInt)

Descripción Define como posiciónX la coordenada horizontal (en twips) de la posición del puntero.

Ejemplo Este ejemplo implica a dos métodos del mismo objeto, *CajaUno*. El método **mouseUp** crea un MouseEvent, definiendo las coordenadas 500 twips mayor que el punto del clic. El método **mouseUp** envía entonces el suceso a **mouseRightUp**. El método **mouseRightUp** obtiene las coordenadas, las convierte para que se sitúen correctamente sobre *CajaUno* y dibuja un cuadro en el punto indicado por el MouseEvent. Si el MouseEvent es el resultado de una interacción del usuario (**isFromUI** devuelve True), el nuevo cuadro cambia su color a Red. Si el MouseEvent no es el resultado de una interacción del usuario, como cuando el suceso se pasó desde el método **mouseUp**, el nuevo cuadro cambia su color a Green. El método **mouseUp** de *CajaUno* es:

```
;CajaUno::mouseUp
method mouseUp(var eventInfo MouseEvent)
var
  suceso MouseEvent
endVar     ; asignar a la nueva posición la actual más 500, 500
suceso.setX(eventInfo.x() + 500)     suceso.setY(eventInfo.y()
+ 500)     mouseRightUp(suceso)            ; enviamos el nuevo
suceso
endmethod
```

Este código se anexa al método **mouseRightUp** de *CajaUno*:

```
;CajaUno::mouseRightUp
method mouseRightUp(var eventInfo MouseEvent)
var
  ui     UIObject     ; para crear un objeto en el punto en que
se hace el clic
  Puntos Point        ; los puntos x e y del clic
endVar                ; obtenemos las coordenadas x e y
Puntos = Point(eventInfo.x(), eventInfo.y())
     ; transformar el punto de la página al cuadro
self.convertPointWithRespectTo(páginaUno, Puntos, Puntos)
; creamos la caja, le damos color, y la hacemos visible
ui.create(boxTool, Puntos.x(), Puntos.y(), 200, 200)

ui.Visible = True
if eventInfo.isFromUI() then
  ui.Color = Red         ; el suceso original
else
  ui.Color = Green        ; el suceso del ratón que pasó
mouseUp
endif
endmethod
```

Vea también x
y
setY
UIObject**::**convertPointWithRespectTo

setY

Método	Especifica la coordenada vertical de la posición del puntero. Las coordenadas deben especificarse respecto al ángulo superior izquierdo del objeto actual.
Tipo	MouseEvent
Sintaxis	**setY** (const ***posiciónY*** LongInt)
Descripción	Define como posiciónY la coordenada vertical (en twips) de la posición del puntero.
Ejemplo	Consulte el ejemplo de setX.
Vea también	x y setX UIObject**::**convertPointWithRespectTo

x

Método	Devuelve la coordenada horizontal de la posición del puntero.
Tipo	MouseEvent
Sintaxis	**x** ( ) LongInt
Descripción	Devuelve (en twips) la coordenada horizontal de la posición del puntero del ratón.
Ejemplo	Consulte el ejemplo de setX.
Vea también	y setX UIObject**::**convertPointWithRespectTo

y

Método	Devuelve la coordenada vertical de la posición del puntero del ratón.
Tipo	MouseEvent
Sintaxis	**y** ( ) LongInt
Descripción	Devuelve (en twips) la coordenada vertical de la posición del puntero.
Ejemplo	Consulte el ejemplo de setX.
Vea también	x setY setX UIObject**::**convertPointWithRespectTo

getDestination

Método Informa de qué objeto es el destino de un MoveEvent.

Tipo MoveEvent

Sintaxis **getDestination** (var dest UIObject)

Descripción Devuelve en *dest* el objeto al que el usuario está intentando desplazarse en una ficha.

Ejemplo En este ejemplo, supóngase que la ficha contiene un objeto multirregistro asociado con la tabla *Pedidos*. El método **canDepart** de la ficha se ejecuta siempre que el usuario intenta salir de un campo u otro objeto de la ficha. El método **canDepart** mostrado en este ejemplo emplea **getDestination** para averiguar el destino previsto del MoveEvent. Este método emplea **getTarget** para averiguar el origen del desplazamiento y compararlo con el destino.

Si los contenedores de los dos objetos son los mismos, como cuando el usuario se desplaza de un campo al siguiente en un objeto multirregistro, el método muestra un cuadro de diálogo solicitando confirmación. Cuando el usuario responde, se produce el desplazamiento y se define como amarillo el campo del que ha salido el usuario. Si el contenedor del origen y el del destino son diferentes, como cuando el usuario intenta salir de la ficha por completo, el método no muestra el cuadro de diálogo. El código siguiente se anexa al método canDepart de una ficha:

```
; estaFicha::canDepart
method canDepart(var eventInfo MoveEvent)
var
  Objeto1   UIObject
  Objeto2   UIObject
  Respuesta String
endVar
if eventInfo.isPreFilter()
  then
    ;el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    eventInfo.getTarget(Objeto1)
    eventInfo.getDestination(Objeto2)
    if Objeto1.ContainerName = Objeto1.ContainerName then
      ; el manejador sólo mueve campo-a-campo dentro del Objeto
Multirregistro
      Respuesta = msgQuestion( ¿Desplazar? ,  ¿Nos movemos a
+
                               Objeto2.name +  ? )
      if Respuesta =  No  then
        eventInfo.setErrorCode(CanNotDepart)
      else
        Objeto1.Color = Yellow         ; dejamos un rastro de
campos amarillos
      endIf
    endif
  else
    ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha

endif
endmethod
```

Vea también reason
Event::getTarget

reason

Método Informa de por qué se ha producido un desplazamiento.

Tipo MoveEvent

Sintaxis **reason** ( ) SmallInt

Descripción Devuelve un valor entero que indica por qué se produjo un MoveEvent. Las constantes Reason de MoveEvent se producen cuando se ejecuta un método estándar **arrive**, **depart**, **canArrive** o **canDepart**. ObjectPAL proporciona las constantes siguientes para comprobar el valor devuelto por **reason** (enumeradas en el cuadro de diálogo Constantes bajo MoveReasons):

PalMove significa que el desplazamiento fue generado por una sentencia de ObjectPAL.

RefreshMove significa que fue generado por los valores de tabla que se están actualizando en una red.

ShutDownMove significa que fue generado al cerrarse la ficha.

StartupMove significa que fue generado al abrirse la ficha.

UserMove significa que fue provocado por la interacción del usuario con la ficha.

Ejemplo En este ejemplo, supóngase que una ficha contiene un objeto de campo llamado *campoUno* y un botón llamado *IrAlCampoUno*, así como al menos otro objeto de campo. La salida de *campoUno* se trata como normal; sin embargo, para volver a *campoUno*, el usuario debe pulsar el botón *IrAlCampoUno*. El método **canArrive** de *campoUno* comprueba la razón del desplazamiento e impide la llegada al campo si la razón no es UserMove. El código siguiente se anexa al método **canArrive** de *campoUno*:

```
; campoUno::canArrive
method canArrive(var eventInfo MoveEvent)
; no permitimos al usuario mover el campo con la tecla Tab ni
mediante clics
if eventInfo.reason() = UserMove then
  eventInfo.setErrorCode(CanNotArrive)
  beep()
  message( Pulse el botón [Mover el campo uno] para moverlo. )

endif
endmethod
```

El código siguiente se anexa al método **pushButton** de *IrAlCampoUno*:

```
; IrAlCampoUno::pushButton
method pushButton(var eventInfo Event)
; nos desplazamos al CampoUno si no tiene el foco
if campoUno.Focus = False then
  CampoUno.moveTo()
else
  CampoDos.moveTo()
endif
endmethod
```

Vea también setReason

setReason

Método Especifica una razón para un MoveEvent.

Tipo MoveEvent

Sintaxis **setReason** (const idRazón SmallInt)

Descripción Especifica una razón para la generación de un MoveEvent. Este método toma una de las siguientes constantes como argumento (enumeradas en el cuadro de diálogo Constantes bajo MoveReasons):

PalMove significa que el desplazamiento fue generado por una sentencia de ObjectPAL.

RefreshMove significa que el desplazamiento fue generado por los valores de tabla que se están actualizando en una red.

ShutDownMove significa que el desplazamiento fue generado al cerrarse la ficha.

StartupMove significa que el desplazamiento fue generado al abrirse la ficha.

UserMove significa que el desplazamiento fue provocado por la interacción del usuario con la ficha.

Ejemplo En este ejemplo, el método **canArrive** de *campoUno* impide la llegada a un campo si la razón para el desplazamiento es UserMove. Para burlar temporalmente esta restricción, el método **canArrive** de la ficha cambia la razón para sucesos UserMove a sucesos PalMove. Este código se anexa al método **canArrive** de *campoUno*:

```
; campoUno::canArrive
method canArrive(var eventInfo MoveEvent)
; no permitimos al usuario mover el campo con la tecla Tab ni
mediante clics
if eventInfo.reason() = UserMove then
  eventInfo.setErrorCode(CanNotArrive)
  beep()
  message( Pulse el botón [Mover el campo uno] para moverlo. )

endif
endmethod
```

Este código se anexa al método **canArrive** de la ficha:

```
; estaFicha::canArrive
method canArrive(var eventInfo MoveEvent)
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    ; cambiamos los sucesos por una razón de UserMove a PalMove

    if eventInfo.reason() = UserMove then
      eventInfo.setReason(PalMove)
    endif
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
endif
endmethod
```

Vea también reason

reason

Método Informa de por qué se ha producido un StatusEvent.

Tipo StatusEvent

Sintaxis **reason** ( ) SmallInt

Descripción Devuelve un valor entero que informa de por qué se ha producido un StatusEvent. Las razones de StatusEvent se producen cuando se ejecuta un método estándar **status**. ObjectPAL proporciona las constantes siguientes para comprobar el valor devuelto por **reason** (enumeradas en el cuadro de diálogo Constantes bajo StatusReasons):

StatusWindow implica que el mensaje se envió al área de estado (el área mayor de la barra de estado, que ocupa aproximadamente los dos tercios izquierdos de la barra de estado).

ModeWindow1 implica que el mensaje se envió a la primera ventana pequeña a la derecha del área de estado.

ModeWindow2 significa que el mensaje se envió a la segunda ventana pequeña a la derecha del área de estado.

ModeWindow3 indica que el mensaje se envió a la tercera ventana pequeña a la derecha del área de estado (la ventana situada más a la derecha).

Ejemplo En el ejemplo siguiente, supóngase que una ficha contiene dos campos, *campoUno* y *campoDos*. El método **status** de *campoDos* busca en el paquete de suceso un código de error; si encuentra uno, el método muestra un mensaje en la línea de estado. El método **status** de la ficha define un código de error si el destino del suceso es *campoDos*. El código siguiente se anexa al método **status** de *campoDos*.

```
; campoDos::status
method status(var eventInfo StatusEvent)
if eventInfo.errorCode() 0 then
  ; mostramos un mensaje diferente si hay un error
  eventInfo.setStatusValue("Aquí hay un error.")
endif
endmethod
```

Este código se anexa al método **status** de la ficha:

```
; estaFicha::status
method status(var eventInfo StatusEvent)
var
  Objeto UIObject
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
     eventInfo.getTarget(Objeto)
    ; forzamos un error arbitrario para CampoDos
    if Objeto.Name = "CampoDos" AND
      eventInfo.reason() = StatusWindow then
       eventInfo.setErrorCode(1)
     endif
   else
     ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha

endif
endmethod
```

Vea también setReason

setReason

Método Especifica una razón para la generación de un StatusEvent.

Tipo StatusEvent

Sintaxis **setReason** (const idRazón SmallInt)

Descripción Especifica una constante Reason para la generación de un StatusEvent. Las razones de StatusEvent indican a qué ventana de la barra de estado se envió el mensaje. ObjectPAL proporciona las siguientes constantes para definir la razón de un StatusEvent (enumeradas en el cuadro de diálogo Constantes bajo StatusReasons):

StatusWindow implica que el mensaje se envió al área de estado (el área mayor de la barra de estado, que ocupa aproximadamente los dos tercios izquierdos de la barra de estado).

ModeWindow1 implica que el mensaje se envió a la primera ventana pequeña a la derecha del área de estado.

ModeWindow2 significa que el mensaje se envió a la segunda ventana pequeña a la derecha del área de estado.

ModeWindow3 indica que el mensaje se envió a la tercera ventana pequeña a la derecha del área de estado (la ventana situada más a la derecha).

Ejemplo En este ejemplo, para un StatusEvent lanzado a la ficha desde un campo, el método **status** de la ficha cambia la razón y contenido del mensaje. El método cambia la razón a ModeWindow1, y define el valor del mensaje como el nombre del objeto que originó el suceso original.

```
; estaFicha::status
method status(var eventInfo StatusEvent)
var
  Objeto UIObject
  Nombre String
endVar
if eventInfo.isPreFilter()
  then
    ; el código fuente que haya aquí se ejecuta para cada
objeto de la Ficha
  else
    ; el código fuente que hay aquí se ejecuta sólo para la
propia Ficha
    ; una vez que se muestra el mensaje regular, también
mostramos
    ; el nombre del campo en ModeWindow1
     eventInfo.getTarget(Objeto)
     if Objeto.Class = "Campo" then     ; si es un campo
       Nombre = Objeto.Name             ; obtenemos su nombre
       eventInfo.setReason(ModeWindow1) ; asignamos la ventana
      eventInfo.setStatusValue(Nombre) ; enviamos la cadena
    endif
endif
endmethod
```

Vea también reason
Event::errorCode
Event::setErrorCode

setStatusValue

Método	Especifica el texto de un mensaje de estado.
Tipo	StatusEvent
Sintaxis	**setStatusValue** (const valorEstado AnyType)
Descripción	Especifica el texto de un mensaje de estado en *valorEstado*.
Ejemplo	Consulte el ejemplo de setReason.
Vea también	setReason statusValue

statusValue

Método Devuelve el texto de un mensaje de estado.

Tipo StatusEvent

Sintaxis **statusValue** ( ) AnyType

Descripción Devuelve el texto de un mensaje de estado.

Ejemplo Este ejemplo resalta más los mensajes de estado por defecto copiando cada mensaje en un campo de la ficha. Esta función se controla mediante el botón *resaltarMensaje*, que también se halla en la misma ficha. El código siguiente se anexa al método **pushButton** del botón *resaltarMensaje*:

```
; resaltarMensaje::pushButton
method pushButton(var eventInfo Event)
; cambiamos statusMessageField a visible o invisible y
; cambiamos la etiqueta a "Mensajes resaltados" o "Mensajes
normales"
if self.LabelText = "Mensajes resaltados" then
  statusMessageField.Visible = True
  self.LabelText = "Mensajes normales"
else
  statusMessageField.Visible = False
  self.LabelText = "Mensajes Resaltados"
endif
endmethod
```

Este código se anexa al método **status** de la ficha:

```
; estaFicha::status
method status(var eventInfo StatusEvent)
if eventInfo.isPreFilter()
  then
    ; el código fuente que hay aquí se ejecuta para cada objeto
de la Ficha
    ; escribimos cada evento de estado en un campo de la Ficha
    if statusMessageField.Visible = True then
      if eventInfo.reason() = StatusWindow then
        statusMessageField = eventInfo.statusValue()
      endif
    endif
 else
   ; el código fuente que haya aquí se ejecuta sólo para la
propia Ficha

endif
endmethod
```

Vea también reason
setStatusValue

newValue

Principiante

Método Devuelve el nuevo valor mostrado como resultado de un ValueEvent.

Tipo ValueEvent

Sintaxis **newValue** ( ) AnyType

Descripción Devuelve el nuevo valor que desea asignar a un campo como resultado de un ValueEvent. El nuevo valor no se ha asignado aún al campo, por lo que las dos sentencias siguientes pueden devolver valores distintos:

```
campo.Value
eventInfo.newValue()
```

Ejemplo En este ejemplo, el método **changeValue** del campo *límiteCrédito* comprueba el valor antiguo y el nuevo para averiguar si ha cambiado más del 25%. Si la diferencia entre los dos valores es demasiado grande, el método impide el cambio. Supóngase que *límiteCrédito* es un campo no asociado de una ficha y que hay al menos otro campo al que es posible desplazarse.

```
; límitecrédito::changeValue
method changeValue(var eventInfo ValueEvent)
var
  AntiguoValor,
 NuevoValor    Number
endVar

AntiguoValor = self.Value                              ; la
propiedad puede ser distinta
NuevoValor = eventInfo.NewValue()             ; del nuevo valor

if NuevoValor >> AntiguoValor AND AntiguoValor <<>> 0 then
  if (NuevoValor - AntiguoValor)/AntiguoValor >> 0.25 then
    msgStop( íAtención! ,  No se te permite incrementar el   +

                             Crédito límite más de un 25%. )
    self.action(EditUndoField)    ; lo utilizamos para restaurar
el valor antiguo
   eventInfo.setErrorCode(CanNotDepart)             ; salida
bloqueada
  endif     <127>
endif

endMethod
```

Vea también setNewValue

setNewValue

Método Especifica un valor que se establece para un ValueEvent.

Tipo ValueEvent

Sintaxis **setNewValue** (const nuevoValor AnyType)

Descripción Especifica en nuevoValor un valor que se establece para un ValueEvent. Los datos suministrados en *nuevoValor* deberían ser coherentes con el tipo del campo.

Ejemplo En este ejemplo, suponga que una ficha contiene el campo *authorAbbrToName*, así como al menos otro campo más. Cuando el usuario introduce una abreviatura de autor y se desplaza fuera del campo, el m todo **changeValue** rellena el nombre completo del autor.

```
; autorAlNombre::changeValue
method changeValue(var eventInfo ValueEvent)
var
  ValorAbreviado String
  ValorCompleto String
endVar

ValorAbreviado = upper(eventInfo.newValue())  ; obtenemos el
ValorAbreviado
                              ; y lo convertimos a mayúsculas
; el usuario introduce una abreviatura  la cambiamos por el
nombre completo
switch
  case ValorAbreviado =  CJC  : ValorCompleto =  Camilo José
Cela
  case ValorAbreviado =  JJB  : ValorCompleto =  Juan José
Benítez
  case ValorAbreviado =  MC     : ValorCompleto =  Miguel de
Cervantes
  case ValorAbreviado    =  FVC  : ValorCompleto =  Fernando
Vizcaíno Casas
  case ValorAbreviado =  BPG  : ValorCompleto =  Benito Pérez
Galdós
  case ValorAbreviado =  GAB  : ValorCompleto =  Gustavo Adolfo
Becquer
otherwise : ValorCompleto =  Autor Desconocido
endswitch

eventInfo.setNewValue(ValorCompleto)
endMethod
```

Vea también newValue

j$# Glosario de ObjectPAL

Términos de ObjectPAL

active
Variable de ObjectPAL que representa al objeto que tiene foco.

ANSI
Acrónimo de American National Standards Institute; una secuencia de códigos de ocho bits que define 256 caracteres, letras, números y símbolos estándar. El juego de caracteres ASCII coincide con los 128 primeros caracteres ANSI.

aplicación
Grupo de fichas, métodos, consultas y procedimientos que forman una sola unidad en la que los usuarios pueden introducir, visualizar y mantener datos, además de obtener información acerca de ellos.

argumento
Información que se pasa a un método o procedimiento.

matriz
Tipo especial de objeto formado por varios elementos independientes. Los elementos de una matriz se designan con subíndices entre corchetes, por lo que ar[1] y ar[2] serán los dos primeros elementos de la matriz denominada *ar*.

elemento de matriz
Componente de una matriz. Por ejemplo, la matriz creada con la declaración siguiente tiene siete elementos de cadena, de ar[1] a ar[7].

```
ar Array [7] String
```

Los elementos son también llamados ítems.

ASCII
Acrónimo de American Standard Code for Information Interchange; una secuencia de códigos de siete bits que definen 128 caracteres, letras, números y símbolos estándar. El juego de caracteres ampliado de IBM amplía este código a 8 bits para incluir algunos caracteres gráficos especiales. Los productos Windows utilizan el juego de caracteres ANSI.

blanco
Campo o variable que no tiene ningún valor.

llaves
Los símbolos { y }. Se utilizan para señalar los comentarios dentro del código.

comandos de bifurcación
Estructuras de control que llevan a cabo comandos específicos si se cumplen ciertas condiciones. Ejemplos: if, while

punto de ruptura
Marca incluida en el código fuente que hace que se suspenda la ejecución. Se utiliza a la hora de depurar el código.

lanzamiento
Proceso por el que los sucesos se elevan desde el objeto destino por la jerarquía de contenedores.

método estándar
Código predefinido incorporado a todos los objetos que pueden colocarse en una ficha; define la respuesta por defecto de los objetos a los sucesos.

columna
Componente vertical de las tablas de Paradox que contiene un campo. En los informes de Paradox, área vertical que contiene uno o más campos.

objeto compuesto
Objeto formado por dos o más objetos. Por ejemplo, un marco de tabla es un objeto compuesto formado por objetos de campo y de registro.

concatenación
Unión de dos o más cadenas para formar una sola. El operador de concatenación es el signo más (+).

constante
Las constantes representan valores que no pueden cambiarse. Por ejemplo, DataNextRecord es una constante de ObjectPAL que especifica el desplazamiento al siguiente registro de una tabla.

contenedor
Un objeto es contenedor de otro si el segundo se encuentra totalmente dentro de los bordes del primero. La situación de un objeto en la jerarquía de contenedores determina si las variables, los métodos y los procedimientos están disponibles para él o no.

estructura de control
Secuencia de sentencias de bifurcación, como if…then…endIf o while…endWhile, que afecta al orden en que se ejecutan las sentencias.

Ctrl+Inter
Combinación de teclas que detiene la ejecución de un programa. También puede configurarse Paradox para responder a Ctrl+Inter abriendo el Depurador.

datos
Información que Paradox almacena en una tabla.

Database
Tipo de objeto que contiene información acerca de las relaciones entre las tablas.

tipo de datos
El tipo de los datos contenidos en un campo, una variable o un elemento de matriz.

DDE
Acrónimo de Dynamic Data Exchange (intercambio dinámico de datos). Forma de compartir datos entre aplicaciones Windows.

punto muerto
Situación creada en entornos multiusuario cuando se envían de forma simultánea dos comandos de bloqueo incompatibles.

Depurador
Parte del entorno integrado de desarrollo (IDE) de ObjectPAL que permite comprobar y realizar el seguimiento de los comandos de los métodos de forma interactiva.

Escritorio
Ventana principal de Paradox.

Gestor de visualización
Categoría de tipos de objeto que incluye Application, Form, Report y TableView.

DLL
Acrónimo de Dynamic Link Library (biblioteca de vínculo dinámico). Programa que permite a los programas Windows compartir el código que realiza tareas más comunes.

matriz dinámica
Tipo especial de matriz en la que cada elemento tiene una cadena para el índice. Por ejemplo,

[Producto], [Paradox para Windows], [Tipo] [Base de datos], [Versión][1.0]

Editor
Componente del IDE de Paradox utilizado para crear y editar métodos de ObjectPAL.

cifrar
Traducir una tabla o una macro a código que no puede leerse sin la contraseña adecuada.

suceso
Acción que activa un método (es decir, que hace que se ejecute código). También un tipo de objeto (Event).

aplicación guiada por sucesos
Aplicación donde el código se ejecuta como respuesta a sucesos, al contrario que en aplicaciones basadas en procedimientos, donde el código se ejecuta en secuencia lineal.

modelo de sucesos
Normas que especifican la forma en que los objetos procesan los sucesos en una ficha.

elemento ejemplo
En una sentencia de consulta, una secuencia arbitraria de caracteres que representa cualquier valor de un campo. En Paradox, los elementos ejemplo se indican pulsando F5 y escribiendo los caracteres en la plantilla de la consulta. En los métodos, se indican anteponiendo un carácter de subrayado a los caracteres.

expresión
Grupo de caracteres que puede incluir valores de datos, variables, matrices, operadores o funciones que representan una cantidad o un valor. Una expresión puede dar como resultado un tipo de datos específico o, en algunos casos, puede convertirse primero en valores de cadena para calcular después el resultado.

campo
Elemento de información en una tabla. Un grupo de campos relacionado forma un registro.

asignación de campo
Uso de la notación de puntos para asignar el valor de una expresión a un campo.

objeto de campo
Objeto del interfaz de usuario (UIObject) que puede o no estar asociado a un campo de una tabla.

tipo de campo
Tipo de información que puede introducirse en un campo de una tabla. Consulte también tipo de datos.

valor de campo
Datos contenidos en un campo de un registro. Si no existen datos, el campo se considera vacío. Los objetos de campo tienen la propiedad Value.

modo campo
Permite desplazar el punto de inserción por un campo, de carácter en carácter. Se utiliza para visualizar los valores de un campo que son demasiado extensos para aparecer en el campo con el ancho actual, o para editar el valor de un campo.

archivo
Información almacenada bajo un solo nombre en un disco. Por ejemplo, las tablas de Paradox se almacenan en archivos.

FileSystem
Tipo de ObjectPAL. Las variables FileSystem contienen información acerca de los archivos en disco.

foco
Atributo de un objeto. El objeto que tiene foco (también llamado objeto activo) está preparado para

recibir las entradas de teclado. Normalmente suele estar resaltado.

especificación de formato
Forma en que el valor de un campo aparece en pantalla o se imprime.

ficha
Ventana para visualizar datos y objetos. También, un tipo de ObjectPAL (Form). La ficha es el objeto contenedor de nivel máximo.

teclas de función
Las 12 teclas de la parte superior del teclado (algunos teclados tienen 10 en la parte izquierda con rótulos de *F1* a *F10*).

variable global
Variable disponible para todos los objetos de una ficha. Consulte también variable local.

gestor
Variable que puede utilizarse en código para manipular objetos.

Ayuda
Sistema de Ayuda en línea de Paradox. Pulsando F1 en cualquier punto de Paradox se obtiene información en pantalla acerca de la operación actual.

jerarquía
Relación entre los objetos de una ficha derivada de su relación visual y espacial. Consulte también contenedor.

códigos ampliados IBM
Teclas o combinaciones de teclas que no corresponden a ningún código de carácter estándar ASCII y tienen un código ampliado especial con números entre 1 y 132.

desarrollo incremental
Proceso del desarrollo de aplicaciones en el que se designan y comprueban de forma interactiva partes pequeñas o la estructura general de la aplicación.

índice
Archivo que determina el orden en que Paradox puede acceder a los registros de una tabla. El campo clave de una tabla de Paradox establece el índice primario. Consulte también clave e índice secundario.

punto de inserción
Lugar en que se inserta el texto al escribir. El punto de inserción suele representarse por una barra vertical parpadeante.

inspeccionar
Visualizar o cambiar las propiedades de un objeto. Para ello, puede hacer clic con el botón derecho del ratón, o seleccionar el objeto con el teclado y pulsar F6. Aparece el menú del objeto, donde puede seleccionarse la propiedad que se desea modificar.

clave
Campo o grupo de campos utilizado para ordenar registros. Consulte también campo clave.

código de tecla
Código que representa un carácter del teclado en los métodos de ObjectPAL. Puede ser un número ANSI o una cadena que representa el nombre de la tecla que Paradox conoce.

campo clave
Campo designado como la totalidad o parte de un identificador de los registros de una tabla de Paradox. Una clave tiene tres funciones: impedir que existan registros duplicados, mantener los registros en el orden fijado por los campos clave y crear un índice primario para la tabla. Consulte también índice.

palabra clave
Palabra reservada para ObjectPAL. Las palabras clave no deben utilizarse como nombres de variables, matrices, métodos o procedimientos.

biblioteca
Código de ObjectPAL que puede ser utilizado por los objetos de una o más fichas.

vida
Período de tiempo durante el cual un elemento permanece activo o disponible.

clave de enlace
En fichas multitabla enlazadas, parte de la clave de la tabla subordinada que está enlazada o cuyos campos coinciden con la tabla principal.

variable local
Variable disponible sólo para el método o procedimiento en que se declara. Consulte también variable global.

operador lógico
Uno de los tres operadores (AND, OR o NOT) que pueden utilizarse con datos lógicos. Por ejemplo, AND entre dos valores lógicos da como resultado el valor lógico True si los dos valores originales son True. También se conocen como operadores booleanos.

valor lógico
Valor (True o False) asignado a una expresión cuando se evalúa. Conocido también como valor booleano.

bucles
Estructuras de control que repiten una serie de comandos hasta que se cumple una condición determinada. Consulte también estructuras de control.

menú
Conjunto de opciones disponibles. Con ObjectPAL pueden crearse y editarse los menús emergentes y de aplicación.

opción de menú
Comando seleccionado en un menú.

mensaje
Expresión de cadena que se muestra en la línea de estado.

método
Código de ObjectPAL anexado a un objeto que define la respuesta de éste ante un suceso.

estructura de datos normalizada
Organización de los datos en tablas en las que cada registro incluye el menor número de campos necesario para establecer categorías únicas. En lugar de usar pocos registros para proporcionar toda la información posible, una tabla normalizada extiende la información a muchos registros y utiliza menos campos. Las tablas normalizadas proporcionan mayor flexibilidad en cuanto a análisis.

objeto
Encapsulación de código y datos. Todas las entidades que pueden manipularse en Paradox son objetos.

arbol de objetos
Diagrama que muestra las relaciones de contenedores entre los objetos.

parámetro
La variable a la que se pasa un argumento. Se utiliza en la definición de procedimientos.

plantilla
Modelo de caracteres que define lo que un usuario puede introducir en un campo durante la edición

o introducción de datos o como respuesta a una petición del programa.

píxel
Uno de los puntos de la pantalla. El nombre viene del inglés *pic*ture *el*ement (elemento de imagen).

punto
Par de números ordenados que representan una posición en la pantalla.

puntero
Marca visual que indica la posición del ratón en la pantalla.

almacenar
Aceptar los cambios realizados en un registro y colocar los datos en la tabla. También llamado consignar.

índice primario
Indice de los campos clave de una tabla de Paradox. Los índices principales determinan la posición de los registros, permiten utilizar la tabla como detalle de un enlace, mantienen los registros ordenados y aceleran las operaciones. Consulte también clave, clave secundaria.

procedimiento
Código incluido entre las palabras clave PROC y ENDPROC. A diferencia de los métodos, no tiene contexto determinado por un objeto.

mensajes
Instrucciones que se muestran en pantalla, normalmente en la barra de estado. Los mensajes piden información o guían al usuario durante una operación.

propiedades
Atributos de un objeto. Haciendo clic con el botón derecho del ratón sobre un objeto pueden verse o cambiarse sus propiedades. Consulte también inspeccionar.

QBE
Consulte consulta mediante ejemplo.

consulta
Pregunta formulada acerca de la información contenida en una tabla de Paradox en una ficha de consulta. También un tipo de ObjectPAL (Query).

consulta mediante ejemplo (QBE)
Método para formular preguntas acerca de los datos proporcionando ejemplos sobre las respuestas que se buscan.

cadena entrecomillada
Texto encerrado entre comillas.

exploración
Operación que especifica la forma en que se mezclan los colores en pantalla.

registro
Fila horizontal de una tabla de Paradox que contiene un grupo de campos de datos relacionados. También un tipo de ObjectPAL (Record).

número de registro
Número único que identifica cada registro de una tabla.

base de datos relacional
Diseño de base de datos que concuerda con un grupo de principios denominados modelo relacional. Los datos de las bases de datos relacionales deben estar organizados en tablas.

palabras reservadas
Nombres de comandos, palabras claves, funciones, variables del sistema y operadores. Estas

palabras no pueden utilizarse como variables de ObjectPAL ni nombres de matrices. Consulte también palabra clave.

tabla restringida
Tabla detallada de una ficha multitabla, enlazada con la tabla principal, donde los registros siguen los principios unouno o unovarios, y sólo se muestran los registros que coinciden con el registro principal actual.

fila
Elemento horizontal de una tabla, llamado registro en Paradox.

error de ejecución
Error que se produce cuando una sentencia cuya sintaxis es válida no puede llevarse a cabo en el contexto actual.

biblioteca run time
Conjunto de métodos y procedimientos predefinidos que funcionan en objetos de tipos concretos.

ámbito
Accesibilidad o disponibilidad de una variable, método o procedimiento para otros objetos.

macro
Código de ObjectPAL que se ejecuta sin abrir ninguna ventana.

índice secundario
Indice utilizado para enlaces, consultas y cambios del orden de visualización de las tablas.

Self
Variable de ObjectPAL que hace referencia al objeto al que está anexado el código que se está ejecutando en ese momento.

sesión
Canal que conduce a la maquinaria de la base de datos. También, tipo de ObjectPAL (Session).

secuencia de barra invertida
Barra invertida seguida de uno o más caracteres que representa un carácter ASCII, por ejemplo, \" o \018. Las secuencias de barra invertida se utilizan para colocar comillas dentro de cadenas y para incluir otros caracteres que tienen un significado especial para Paradox.

cadena
Valor alfanumérico o expresión formada por caracteres alfanuméricos. También, un tipo de ObjectPAL (String).

estructura
Organización de los campos en una tabla.

sujeto
Objeto utilizado para llamar a un método personalizado. Por ejemplo, en la sentencia siguiente, el sujeto es elCuadro.

```
elCuadro.hazAlgo()
```

subcadena
Cualquier parte de una cadena.

error de sintaxis
Error que se produce a causa de una sentencia mal expresada.

Tableview
Representación de una tabla en formato de tabla de Paradox en filas y columnas. También, un tipo de ObjectPAL.

destino

Objeto al que va dirigido el suceso. Por ejemplo, cuando se hace clic en un botón, el destino es el botón.

TCursor
Tipo de ObjectPAL que constituye un puntero hacia los datos de una tabla. Los TCursors permiten manipular los datos sin necesidad de visualizar la tabla.

variable de tilde
Variable utilizada en una ficha de consulta que debe ir precedida de una tilde (~).

transacción
Grupo de cambios relacionados realizados en una base de datos.

twip
Unidad de medida igual a 1/1440 de una pulgada (1/20 de un punto de impresora).

tipo
Forma de clasificar los objetos que tienen atributos similares. Por ejemplo, todas las tablas tienen atributos en común y todas las fichas tienen atributos en común, pero los atributos de las fichas son distintos de los de las tablas, por lo que pertenecen a tipos diferentes.

control de validación
Restricción de los valores que pueden introducirse en un campo.

variable
Lugar de la memoria donde se almacenan datos de forma temporal.

active

Variable de ObjectPAL que representa al objeto que tiene foco.

ANSI

Acrónimo de American National Standards Institute; una secuencia de códigos de ocho bits que define 256 caracteres, letras, números y símbolos estándar. El juego de caracteres ASCII coincide con los 128 primeros caracteres ANSI.

aplicación

Grupo de fichas, métodos, consultas y procedimientos que forman una sola unidad en la que los usuarios pueden introducir, visualizar y mantener datos, además de obtener información acerca de ellos.

argumento

Información que se pasa a un método o procedimiento.

matriz

Tipo especial de objeto formado por varios elementos independientes. Los elementos de una matriz se designan con subíndices entre corchetes, por lo que ar[1] y ar[2] serán los dos primeros elementos de la matriz denominada *ar*.

elemento de matriz

Componente de una matriz. Por ejemplo, la matriz creada con la declaración siguiente tiene siete elementos de cadena, de ar[1] a ar[7].

```
ar Array [7] String
```

Los elementos son también llamados ítems.

ASCII

Acrónimo de American Standard Code for Information Interchange; una secuencia de códigos de siete bits que definen 128 caracteres, letras, números y símbolos estándar. El juego de caracteres ampliado de IBM amplía este código a 8 bits para incluir algunos caracteres gráficos especiales. Los productos Windows utilizan el juego de caracteres ANSI.

blanco

Campo o variable que no tiene ningún valor.

llaves

Los símbolos { y }. Se utilizan para señalar los comentarios dentro del código.

comandos de bifurcación

Estructuras de control que llevan a cabo comandos específicos si se cumplen ciertas condiciones. Ejemplos: if, while.

punto de ruptura

Marca incluida en el código fuente que hace que se suspenda la ejecución. Se utiliza a la hora de depurar el código.

lanzamiento

Proceso por el que los sucesos se elevan desde el objeto destino por la jerarquía de contenedores.

método estándar

Código predefinido incorporado a todos los objetos que pueden colocarse en una ficha; define la respuesta por defecto de los objetos a los sucesos.

columna

Componente vertical de las tablas de Paradox que contiene un campo. En los informes de Paradox, área vertical que contiene uno o más campos.

objeto compuesto

Objeto formado por dos o más objetos. Por ejemplo, un marco de tabla es un objeto compuesto formado por objetos de campo y de registro.

constante

Las constantes representan valores que no pueden cambiarse. Por ejemplo, DataNextRecord es una constante de ObjectPAL que especifica el desplazamiento al siguiente registro de una tabla.

contenedor

Un objeto es contenedor de otro si el segundo se encuentra totalmente dentro de los bordes del primero. La situación de un objeto en la jerarquía de contenedores determina si las variables, los métodos y los procedimientos están disponibles para él o no.

estructura de control

Secuencia de sentencias de bifurcación, como if…then…endIf o while…endWhile, que afecta al orden en que se ejecutan las sentencias.

Ctrl+Inter

Combinación de teclas que detiene la ejecución de un programa. También puede configurarse Paradox para responder a Ctrl+Inter abriendo el Depurador.

datos

Información que Paradox almacena en una tabla.

Database

Tipo de objeto que contiene información acerca de las relaciones entre las tablas.

data type

El tipo de los datos contenidos en un campo, una variable o un elemento de matriz.

DDE

Acrónimo de Dynamic Data Exchange (intercambio dinámico de datos). Forma de compartir datos entre aplicaciones Windows.

punto muerto

Situación creada en entornos multiusuario cuando se envían de forma simultánea dos comandos de bloqueo incompatibles.

Depurador

Parte del entorno integrado de desarrollo (IDE) de ObjectPAL que permite comprobar y realizar el seguimiento de los comandos de los métodos de forma interactiva.

Escritorio

Ventana principal de Paradox.

Gestor de visualización

Categoría de tipos de objeto que incluye Application, Form, Report y TableView.

DLL

Acrónimo de Dynamic Link Library (biblioteca de vínculo dinámico). Programa que permite a los programas Windows compartir el código que realiza tareas más comunes.

matriz dinámica

Tipo especial de matriz en la que cada elemento tiene una cadena para el índice. Por ejemplo, [Producto], [Paradox para Windows], [Tipo] [Base de datos], [Versión][1.0]

Editor

Componente del IDE de Paradox utilizado para crear y editar métodos de ObjectPAL.

cifrar

Traducir una tabla o una macro a código que no puede leerse sin la contraseña adecuada.

suceso

Acción que activa un método (es decir, que hace que se ejecute código). También un tipo de objeto (Event).

aplicación guiada por sucesos

Aplicación donde el código se ejecuta como respuesta a sucesos, al contrario que en aplicaciones basadas en procedimientos, donde el código se ejecuta en secuencia lineal.

modelo de sucesos

Normas que especifican la forma en que los objetos procesan los sucesos en una ficha.

elemento ejemplo

En una sentencia de consulta, una secuencia arbitraria de caracteres que representa cualquier valor de un campo. En Paradox, los elementos ejemplo se indican pulsando F5 y escribiendo los caracteres en la plantilla de la consulta. En los métodos, se indican anteponiendo un carácter de subrayado a los caracteres.

expresión

Grupo de caracteres que puede incluir valores de datos, variables, matrices, operadores o funciones que representan una cantidad o un valor. Una expresión puede dar como resultado un tipo de datos específico o, en algunos casos, puede convertirse primero en valores de cadena para calcular después el resultado.

campo

Elemento de información en una tabla. Un grupo de campos relacionado forma un registro.

asignación de campo

Uso de la notación de puntos para asignar el valor de una expresión a un campo.

objeto de campo

Objeto del interfaz de usuario (UIObject) que puede o no estar asociado a un campo de una tabla.

tipo de campo

Tipo de información que puede introducirse en un campo de una tabla. Consulte también tipo de datos.

valor de campo

Datos contenidos en un campo de un registro. Si no existen datos, el campo se considera vacío. Los objetos de campo tienen la propiedad Value.

modo campo

Permite desplazar el punto de inserción por un campo, de carácter en carácter. Se utiliza para visualizar los valores de un campo que son demasiado extensos para aparecer en el campo con el ancho actual, o para editar el valor de un campo.

archivo

Información almacenada bajo un solo nombre en un disco. Por ejemplo, las tablas de Paradox se almacenan en archivos.

FileSystem

Tipo de ObjectPAL. Las variables FileSystem contienen información acerca de los archivos en disco.

foco

Atributo de un objeto. El objeto que tiene foco (también llamado objeto activo) está preparado para recibir las entradas de teclado. Normalmente suele estar resaltado.

especificación de formato

Forma en que el valor de un campo aparece en pantalla o se imprime.

ficha

Ventana para visualizar datos y objetos. También, un tipo de ObjectPAL (Form). La ficha es el objeto contenedor de nivel máximo.

teclas de función

Las 12 teclas de la parte superior del teclado (algunos teclados tienen 10 en la parte izquierda con rótulos de *F1* a *F10*).

variable global

Variable disponible para todos los objetos de una ficha. Consulte también variable local.

gestor

Variable que puede utilizarse en código para manipular objetos.

Ayuda

Sistema de Ayuda en línea de Paradox. Pulsando F1 en cualquier punto de Paradox se obtiene información en pantalla acerca de la operación actual.

jerarquía

Relación entre los objetos de una ficha derivada de su relación visual y espacial. Consulte también contenedor.

códigos ampliados IBM

Teclas o combinaciones de teclas que no corresponden a ningún código de carácter estándar ASCII y tienen un código ampliado especial con números entre 1 y 132.

desarrollo incremental

Proceso del desarrollo de aplicaciones en el que se designan y comprueban de forma interactiva partes pequeñas o la estructura general de la aplicación.

índice

Archivo que determina el orden en que Paradox puede acceder a los registros de una tabla. El campo clave de una tabla de Paradox establece el índice primario. Consulte también clave e índice secundario.

punto de inserción

Lugar en que se inserta el texto al escribir. El punto de inserción suele representarse por una barra vertical parpadeante.

inspeccionar

Visualizar o cambiar las propiedades de un objeto. Para ello, puede hacer clic con el botón derecho del ratón, o seleccionar el objeto con el teclado y pulsar F6. Aparece el menú del objeto, donde puede seleccionarse la propiedad que se desea modificar.

clave

Campo o grupo de campos utilizado para ordenar registros. Consulte también campo clave.

código de tecla

Código que representa un carácter del teclado en los métodos de ObjectPAL. Puede ser un número ANSI o una cadena que representa el nombre de la tecla que Paradox conoce.

campo clave

Campo designado como la totalidad o parte de un identificador de los registros de una tabla de Paradox. Una clave tiene tres funciones: impedir que existan registros duplicados, mantener los registros en el orden fijado por los campos clave y crear un índice primario para la tabla. Consulte también índice.

palabra clave

Palabra reservada para ObjectPAL. Las palabras clave no deben utilizarse como nombres de variables, matrices, métodos o procedimientos.

biblioteca

Código de ObjectPAL que puede ser utilizado por los objetos de una o más fichas.

vida

Período de tiempo durante el cual un elemento permanece activo o disponible.

clave de enlace

En fichas multitabla enlazadas, parte de la clave de la tabla subordinada que está enlazada o cuyos campos coinciden con la tabla principal.

variable local

Variable disponible sólo para el método o procedimiento en que se declara. Consulte también variable global.

operador lógico

Uno de los tres operadores (AND, OR o NOT) que pueden utilizarse con datos lógicos. Por ejemplo, AND entre dos valores lógicos da como resultado el valor lógico True si los dos valores originales son True. También se conocen como operadores booleanos.

valor lógico

Valor (True o False) asignado a una expresión cuando se evalúa. Conocido también como valor booleano.

bucles

Estructuras de control que repiten una serie de comandos hasta que se cumple una condición determinada. Consulte también estructuras de control.

menú

Conjunto de opciones disponibles. Con ObjectPAL pueden crearse y editarse los menús emergentes y de aplicación.

opción de menú

Comando seleccionado en un menú.

mensaje

Expresión de cadena que se muestra en la línea de estado.

método

Código de ObjectPAL anexado a un objeto que define la respuesta de éste ante un suceso.

estructura de datos normalizada

Organización de los datos en tablas en las que cada registro incluye el menor número de campos necesario para establecer categorías únicas. En lugar de usar pocos registros para proporcionar toda la información posible, una tabla normalizada extiende la información a muchos registros y utiliza menos campos. Las tablas normalizadas proporcionan mayor flexibilidad en cuanto a análisis.

objeto

Encapsulación de código y datos. Todas las entidades que pueden manipularse en Paradox son objetos.

árbol de objetos

Diagrama que muestra las relaciones de contenedores entre los objetos.

parámetro

La variable a la que se pasa un argumento. Se utiliza en la definición de procedimientos.

plantilla

Modelo de caracteres que define lo que un usuario puede introducir en un campo durante la edición o introducción de datos o como respuesta a una petición del programa.

píxel

Uno de los puntos de la pantalla. El nombre viene del inglés *pic*ture *el*ement (elemento de imagen).

punto

Par de números ordenados que representan una posición en la pantalla.

puntero

Marca visual que indica la posición del ratón en la pantalla.

almacenar

Aceptar los cambios realizados en un registro y colocar los datos en la tabla. También llamado consignar.

índice primario

Indice de los campos clave de una tabla de Paradox. Los índices principales determinan la posición de los registros, permiten utilizar la tabla como detalle de un enlace, mantienen los registros ordenados y aceleran las operaciones. Consulte también clave, clave secundaria.

procedimiento

Código incluido entre las palabras clave PROC y ENDPROC. A diferencia de los métodos, no tiene contexto determinado por un objeto.

mensajes

Instrucciones que se muestran en pantalla, normalmente en la barra de estado. Los mensajes piden información o guían al usuario durante una operación.

propiedades

Atributos de un objeto. Haciendo clic con el botón derecho del ratón sobre un objeto pueden verse o cambiarse sus propiedades. Consulte también inspeccionar.

QBE

Consulte consulta mediante ejemplo.

consulta

Pregunta formulada acerca de la información contenida en una tabla de Paradox en una ficha de consulta. También un tipo de ObjectPAL (Query).

consulta mediante ejemplo (QBE)

Método para formular preguntas acerca de los datos proporcionando ejemplos sobre las respuestas que se buscan.

cadena entrecomillada

Texto encerrado entre comillas.

exploración

Operación que especifica la forma en que se mezclan los colores en pantalla.

registro

Fila horizontal de una tabla de Paradox que contiene un grupo de campos de datos relacionados. También un tipo de ObjectPAL (Record).

número de registro

Número único que identifica cada registro de una tabla.

base de datos relacional

Diseño de base de datos que concuerda con un grupo de principios denominados modelo relacional. Los datos de las bases de datos relacionales deben estar organizados en tablas.

palabras reservadas

Nombres de comandos, palabras claves, funciones, variables del sistema y operadores. Estas palabras no pueden utilizarse como variables de ObjectPAL ni nombres de matrices. Consulte también palabra clave.

tabla restringida

Tabla detallada de una ficha multitabla, enlazada con la tabla principal, donde los registros siguen los principios unouno o unovarios, y sólo se muestran los registros que coinciden con el registro principal actual.

fila

Elemento horizontal de una tabla, llamado registro en Paradox.

error de ejecución

Error que se produce cuando una sentencia cuya sintaxis es válida no puede llevarse a cabo en el contexto actual.

biblioteca run time

Conjunto de métodos y procedimientos predefinidos que funcionan en objetos de tipos concretos.

ámbito

Accesibilidad o disponibilidad de una variable, método o procedimiento para otros objetos.

macro

Código de ObjectPAL que se ejecuta sin abrir ninguna ventana.

índice secundario

Indice utilizado para enlaces, consultas y cambios del orden de visualización de las tablas.

Self

Variable de ObjectPAL que hace referencia al objeto al que está anexado el código que se está ejecutando en ese momento.

sesión

Canal que conduce a la maquinaria de la base de datos. También, tipo de ObjectPAL (Session).

secuencia de barra invertida

Barra invertida seguida de uno o más caracteres que representa un carácter ASCII, por ejemplo, \" o \018. Las secuencias de barra invertida se utilizan para colocar comillas dentro de cadenas y para incluir otros caracteres que tienen un significado especial para Paradox.

cadena

Valor alfanumérico o expresión formada por caracteres alfanuméricos. También, un tipo de ObjectPAL (String).

estructura

Organización de los campos en una tabla.

sujeto

Objeto utilizado para llamar a un método personalizado. Por ejemplo, en la sentencia siguiente, el sujeto es elCuadro.

```
elCuadro.hazAlgo()
```

subcadena

Cualquier parte de una cadena.

error de sintaxis

Error que se produce a causa de una sentencia mal expresada.

Tableview

Representación de una tabla en formato de tabla de Paradox en filas y columnas. También, un tipo de ObjectPAL.

destino

Objeto al que va dirigido el suceso. Por ejemplo, cuando se hace clic en un botón, el destino es el botón.

TCursor

Tipo de ObjectPAL que constituye un puntero hacia los datos de una tabla. Los TCursors permiten manipular los datos sin necesidad de visualizar la tabla.

variable de tilde

Variable utilizada en una ficha de consulta que debe ir precedida de una tilde (~).

transacción

Grupo de cambios relacionados realizados en una base de datos.

twip

Unidad de medida igual a 1/1440 de una pulgada (1/20 de un punto de impresora).

tipo

Forma de clasificar los objetos que tienen atributos similares. Por ejemplo, todas las tablas tienen atributos en común y todas las fichas tienen atributos en común, pero los atributos de las fichas son distintos de los de las tablas, por lo que pertenecen a tipos diferentes.

control de validación

Restricción de los valores que pueden introducirse en un campo.

variable

Lugar de la memoria donde se almacenan datos de forma temporal.