

Válido  
para cualquier  
versión  
de **Delphi**  
32 bits

# Los Tomos de Delphi: *Api Gráfico* **Win32**



John Ayres,  
David Bowden, Larry Diehl,  
Phil Dorcas, Kenneth Harrison,  
Rod Mathes, Ovais Reza, Mike Tobin

 **Danysoft**  
INTERNACIONAL



# ***Los Tomos de Delphi: El API gráfico Win32***

***John Ayres, David Bowden, Larry Diehl,  
Phil Dorcas, Ken Harrison, Rod Mathes,  
Ovais Reza, Mike Tobin***

***[grupo Danysoft]  
[www.danypress.com](http://www.danypress.com)***

Ayres, John y otros

**Los Tomos de Delphi: El API gráfico Win32**

No está permitida la reproducción total o parcial de esta obra,  
ni su tratamiento o transmisión por cualquier medio o método  
sin el consentimiento explícito de la editorial.

**DERECHOS RESERVADOS**

Copyright versión en castellano © 2000, Danysoft Internacional  
Avda. de España, 17  
28100 Alcobendas (Madrid)  
[www.danysoft.com](http://www.danysoft.com)

ISBN: 84-923926-3-0

Depósito Legal: M-

Traducido de:

The tomes of Delphi: Win32 Graphical API

© 1998 Wordware Publishing, Inc.

ISBN 1-55622-610-1

Delphi es una marca registrada de Inprise Corporation.

Windows es una marca registrada de Microsoft Corporation.

El resto de los productos se mencionan únicamente con fines de identificación y están registrados por sus respectivas compañías.

## Dedicatoria

### **John Ayres**

Quiero dedicar este libro a las siguientes personas y divinidades, que han ejercido una profunda influencia en mi vida. Ante todo, a Dios, de quien he estado mucho más cerca en los últimos años, por haberme dado la inteligencia necesaria para navegar a través de la confusa y a veces desquiciante documentación del API de Windows y sacar sentido de ella; en segundo lugar, a mi familia, que ha tenido que soportar mi ausencia de las actividades familiares durante el tiempo que ha durado este proyecto; y finalmente, pero de forma muy especial, a mi esposa y compañera del alma, Marci. Ella me dio aliento en los momentos en los que el peso del proyecto se hacía insostenible, y por eso es responsable directa de que este trabajo se haya finalizado. No merezco una esposa tan devota y amante, y agradezco a Dios cada día por haberme dado tan perfecta compañera. ¡Cariño, te lo dedico a tí!

### **David Bowden**

Las personas que se enumeran a continuación han dejado una profunda marca en mi vida. De todos ellos he recibido enseñanza, estímulo y retos interesantes. Puedo honestamente decir que sin el impacto y la influencia que ellos han causado en mi vida no habría sido capaz de llegar a ninguna parte. Por eso les estaré eternamente agradecidos: Don y Ethel AllRed, Richard y Susie Azar, Ken Blystone, Lori Bruns, Larry Bujanda, E. Davis Chauviere, Sandra Crow, Alfonso Díaz, Larry F. Diehl III, Wade Driver, Charles Evans, Jerry W. Francis II, Glen Hill, Patrick Smith, Ann Stewart, Dan Stiles, Volker Thein, Ryan y Desi. Finalmente, el más importante, Henry J. Dipietro de InterGraph Corporation. De no ser por su visión y deseo de ofrecerme una oportunidad, no estaría hoy donde estoy. Las palabras no pueden expresar mi gratitud por la oportunidad, el estímulo y el apoyo que me has dado. Nunca olvidaré lo que hiciste por mi carrera. ¡Gracias, Hank, eres un gran hombre!

### **Larry Diehl**

Dedico mi trabajo en este libro al pasado y al futuro. Desde el fondo de mi alma, agradezco a mi esposa, padres y hermano por su apoyo constante durante estos años. Ese apoyo me ha permitido llegar a un punto en mi vida en el que me siento todo lo feliz que un hombre puede serlo. En relación con el futuro, deseo que nuestro trabajo en este libro contribuya a hacer un mundo mejor, en el que mi primer hijo pronto nacerá. En cuanto al presente, es lo que uno obtiene de él...

## IV ■

### ***Phil Dorcas***

A mis hijas, Jennifer y Amanda, que son la luz de mi vida.

### ***Ken Harrison***

Quiero dedicar este libro a mis padres, Melvin y Judith Harrison, por todos los sacrificios que han tenido que hacer para darnos a mí y a mis hermanos un entorno seguro en el que crecer, e inculcarnos la idea de que podemos hacer cualquier cosa que nos propongamos. También deseo agradecer a Daniel Roberts y Stethman Grayson por todo el conocimiento sobre Windows que me han aportado y por ser tan buenos amigos.

### ***Rod Mathes***

Desearía agradecer a todos los coautores de este libro haberme ofrecido la oportunidad de contribuir a esta pequeña obra maestra, y muy en especial a John Ayres. También deseo dar las gracias a mi esposa, Sherry, y a mis tres hijos, Keith, Kimberly y Amber, que tuvieron que hacer su vida sin mí durante meses, pero al final lo hemos superado todo—¡ahora tenemos una vida!

### ***Ovais Reza***

Dedico este libro a mi familia y amigos.

### ***Mike Tobin***

A mi esposa, Lisa, por haber comprendido mi trabajo nocturno en el libro en fechas tan cercanas a nuestra boda, y a mis cuatro gatos, que me hicieron compañía durante esas noches.

# Contenido

Sobre los autores . . . . .	XIX
Agradecimientos . . . . .	XXI
Prólogo . . . . .	XXIII
<b>Introducción . . . . .</b>	<b>XXV</b>
Descripción de los capítulos . . . . .	XXVI
Convenios . . . . .	XXVIII
Descripciones de funciones . . . . .	XXIX
Código fuente de los ejemplos . . . . .	XXIX
A quién está orientado este libro . . . . .	XXIX
<b>Capítulo 1 Delphi y el API de Windows . . . . .</b>	<b>1</b>
El API de Windows vs. la VCL . . . . .	1
Tipos de Datos de Windows . . . . .	2
Manejadores . . . . .	4
Constantes . . . . .	4
Cadenas de caracteres . . . . .	4
Importación de funciones de Windows . . . . .	5
Funciones importadas incorrectamente . . . . .	5
Funciones de respuesta . . . . .	6
Parámetros de funciones . . . . .	7
Unicode . . . . .	7
<b>Capítulo 2 Funciones de la Interfaz del Dispositivo Gráfico . . . . .</b>	<b>9</b>
Independencia del dispositivo . . . . .	10
Contextos de Dispositivos . . . . .	10
Tipos de Contextos de Dispositivos . . . . .	11
Contextos de Dispositivos de pantalla, ventana y área cliente . . . . .	13
Sistemas de Coordenadas . . . . .	14
Mapeado de coordenadas lógicas a coordenadas de dispositivo . . . . .	15
Modos de mapeado . . . . .	16
Problemas con el mapeado de coordenadas lógicas . . . . .	20
Funciones de la interfaz gráfica . . . . .	20
ChangeDisplaySettings . . . . .	21
ClientToScreen . . . . .	26

CreateCompatibleDC . . . . .	28
DeleteDC . . . . .	31
DPTOLP . . . . .	32
EnumDisplaySettings . . . . .	32
GetDC . . . . .	35
GetDCOrgEx . . . . .	37
GetDeviceCaps . . . . .	37
GetMapMode . . . . .	46
GetSystemMetrics . . . . .	48
GetViewportExtEx . . . . .	53
GetViewportOrgEx . . . . .	54
GetWindowDC . . . . .	55
GetWindowExtEx . . . . .	56
GetWindowOrgEx . . . . .	57
LPtoDP . . . . .	58
MapWindowPoints . . . . .	59
OffsetViewportOrgEx . . . . .	61
OffsetWindowOrgEx . . . . .	62
ReleaseDC . . . . .	63
RestoreDC . . . . .	64
SaveDC . . . . .	64
ScaleViewportExtEx . . . . .	65
ScaleWindowExtEx . . . . .	70
ScreenToClient . . . . .	71
ScrollDC . . . . .	71
SetMapMode . . . . .	74
SetViewportExtEx . . . . .	76
SetViewportOrgEx . . . . .	77
SetWindowExtEx . . . . .	78
SetWindowOrgEx . . . . .	78
<b>Capítulo 3 Funciones de dibujo . . . . .</b>	<b>81</b>
Objetos gráficos . . . . .	81
Plumas y Brochas . . . . .	82
Funciones de dibujo . . . . .	83
Arc . . . . .	85
BeginPaint . . . . .	87
Chord . . . . .	89
CreateBrushIndirect . . . . .	91
CreateHatchBrush . . . . .	94
CreatePatternBrush . . . . .	96
CreatePen . . . . .	97
CreatePenIndirect . . . . .	99

CreateSolidBrush . . . . .	102
DeleteObject . . . . .	103
DrawCaption . . . . .	103
DrawEdge . . . . .	105
DrawFocusRect . . . . .	108
DrawFrameControl . . . . .	109
DrawState . . . . .	114
Ellipse . . . . .	117
EndPaint . . . . .	119
EnumObjects . . . . .	120
ExtCreatePen . . . . .	123
ExtFloodFill . . . . .	127
FillPath . . . . .	128
FillRect . . . . .	129
FillRgn . . . . .	131
FrameRect . . . . .	132
FrameRgn . . . . .	133
GetBkColor . . . . .	135
GetBkMode . . . . .	136
GetBoundsRect . . . . .	136
GetBrushOrgEx . . . . .	138
GetCurrentObject . . . . .	139
GetCurrentPositionEx . . . . .	140
GetMiterLimit . . . . .	141
GetObject . . . . .	142
GetObjectType . . . . .	147
GetPixel . . . . .	148
GetPolyFillMode . . . . .	148
GetROP2 . . . . .	150
GetStockObject . . . . .	152
GetUpdateRect . . . . .	154
GetUpdateRgn . . . . .	155
GrayString . . . . .	156
InvalidateRect . . . . .	159
InvalidateRgn . . . . .	161
LineDDA . . . . .	163
LineTo . . . . .	165
LockWindowUpdate . . . . .	166
MoveToEx . . . . .	167
PaintDesktop . . . . .	168
PaintRgn . . . . .	168
Pie . . . . .	170
PolyBezier . . . . .	172

## VIII ■

PolyBezierTo . . . . .	173
Polygon . . . . .	175
Polyline . . . . .	175
PolylineTo . . . . .	177
PolyPolygon . . . . .	178
PolyPolyline . . . . .	179
Rectangle . . . . .	181
RoundRect . . . . .	183
SelectObject . . . . .	185
SetBkColor . . . . .	186
SetBkMode . . . . .	187
SetBoundsRect . . . . .	187
SetBrushOrgEx . . . . .	189
SetMiterLimit . . . . .	190
SetPixel . . . . .	191
SetPixelV . . . . .	192
SetPolyFillMode . . . . .	193
SetROP2 . . . . .	195
StrokeAndFillPath . . . . .	196
StrokePath . . . . .	197

### **Capítulo 4 Funciones de regiones y rutas . . . . . 199**

Regiones y rutas . . . . .	199
Regiones . . . . .	199
Rutas . . . . .	202
Efectos especiales . . . . .	202
Funciones de regiones y rutas . . . . .	205
AbortPath . . . . .	206
BeginPath . . . . .	207
CloseFigure . . . . .	208
CombineRgn . . . . .	210
CopyRect . . . . .	213
CreateEllipticRgn . . . . .	214
CreateEllipticRgnIndirect . . . . .	215
CreatePolygonRgn . . . . .	216
CreatePolyPolygonRgn . . . . .	219
CreateRectRgn . . . . .	222
CreateRectRgnIndirect . . . . .	223
CreateRoundRectRgn . . . . .	224
EndPath . . . . .	226
EqualRect . . . . .	226
EqualRgn . . . . .	227
ExcludeClipRect . . . . .	228

ExtCreateRegion . . . . .	231
ExtSelectClipRgn . . . . .	234
FlattenPath . . . . .	235
GetClipBox . . . . .	236
GetClipRgn . . . . .	237
GetPath. . . . .	238
GetRegionData. . . . .	241
GetRgnBox. . . . .	242
InflateRect . . . . .	243
IntersectRect . . . . .	244
InvertRect . . . . .	244
InvertRgn. . . . .	246
IsRectEmpty . . . . .	246
OffsetClipRgn . . . . .	247
OffsetRect . . . . .	249
OffsetRgn . . . . .	253
PathToRegion . . . . .	255
PtInRect . . . . .	257
PtInRegion . . . . .	258
PtVisible . . . . .	259
RectInRegion. . . . .	260
RectVisible. . . . .	260
SelectClipPath . . . . .	261
SelectClipRgn . . . . .	266
SetRect . . . . .	269
SetRectEmpty . . . . .	270
SetRectRgn. . . . .	271
SetWindowRgn . . . . .	272
SubtractRect . . . . .	275
UnionRect . . . . .	276
WidenPath . . . . .	277

## **Capítulo 5 Funciones de mapas de bits y metaфicheros. . . . . 279**

Mapas de bits . . . . .	279
Mapas de bits dependientes del dispositivo . . . . .	280
Mapas de bits independientes del dispositivo . . . . .	280
Operaciones de mapa de bits . . . . .	280
Metaфicheros. . . . .	287
Metaфicheros mejorados. . . . .	288
Funciones de mapas de bits y metaфicheros . . . . .	288
BitBlt. . . . .	290
CloseEnhMetaFile . . . . .	292
CopyEnhMetaFile . . . . .	293

CopyImage . . . . .	293
CreateBitmap . . . . .	296
CreateBitmapIndirect . . . . .	299
CreateCompatibleBitmap . . . . .	302
CreateDIBitmap . . . . .	304
CreateDIBSection . . . . .	308
CreateEnhMetaFile . . . . .	316
DeleteEnhMetaFile . . . . .	319
EnumEnhMetaFile . . . . .	320
GetBitmapBits . . . . .	323
GetBitmapDimensionEx . . . . .	325
GetDIBits . . . . .	325
GetEnhMetaFile . . . . .	329
GetEnhMetaFileDescription . . . . .	332
GetEnhMetaFileHeader . . . . .	333
GetStretchBltMode . . . . .	335
LoadBitmap . . . . .	336
LoadImage . . . . .	340
PatBlt . . . . .	343
PlayEnhMetaFile . . . . .	345
PlayEnhMetaFileRecord . . . . .	346
SetBitmapBits . . . . .	347
SetBitmapDimensionEx . . . . .	350
SetDIBits . . . . .	351
SetDIBitsToDevice . . . . .	355
SetStretchBltMode . . . . .	358
StretchBlt . . . . .	359
StretchDIBits . . . . .	362

## **Capítulo 6 Funciones de iconos, cursores y cursores de edición . . . . . 367**

Cursores de edición . . . . .	367
Máscaras de iconos y cursores . . . . .	368
Conversión de iconos en mapas de bits . . . . .	369
Funciones de iconos, cursores y cursores de edición . . . . .	372
CopyIcon . . . . .	373
CreateCaret . . . . .	374
CreateCursor . . . . .	376
CreateIcon . . . . .	379
CreateIconFromResource . . . . .	382
CreateIconFromResourceEx . . . . .	384
CreateIconIndirect . . . . .	386
DestroyCaret . . . . .	388
DestroyCursor . . . . .	389

DestroyIcon . . . . .	390
DrawIcon. . . . .	390
DrawIconEx . . . . .	391
ExtractAssociatedIcon . . . . .	393
ExtractIcon . . . . .	394
ExtractIconEx . . . . .	396
GetCursor . . . . .	399
GetIconInfo . . . . .	399
HideCaret . . . . .	403
LoadCursor. . . . .	403
LoadCursorFromFile. . . . .	405
LoadIcon . . . . .	406
LookupIconIdFromDirectory . . . . .	407
LookupIconIdFromDirectoryEx . . . . .	408
SetCursor. . . . .	409
SetSystemCursor. . . . .	410
ShowCaret . . . . .	412
ShowCursor . . . . .	413

## **Capítulo 7 Funciones de paleta . . . . . 415**

El administrador de paleta de Windows . . . . .	415
Paletas de identidad . . . . .	416
Especificadores de colores . . . . .	416
Funciones de paleta . . . . .	419
AnimatePalette. . . . .	420
CreateHalftonePalette . . . . .	424
CreatePalette . . . . .	427
GetBValue . . . . .	432
GetDIBColorTable. . . . .	433
GetEnhMetaFilePaletteEntries. . . . .	437
GetGValue . . . . .	440
GetNearestColor . . . . .	441
GetNearestPaletteIndex . . . . .	442
GetPaletteEntries. . . . .	445
GetRValue . . . . .	447
GetSysColor . . . . .	447
GetSystemPaletteEntries. . . . .	450
GetSystemPaletteUse . . . . .	454
PaletteIndex . . . . .	455
PaletteRGB. . . . .	455
RealizePalette . . . . .	456
ResizePalette. . . . .	458
RGB . . . . .	461

SelectPalette . . . . .	461
SetDIBColorTable . . . . .	463
SetPaletteEntries . . . . .	465
SetSysColors . . . . .	466
SetSystemPaletteUse. . . . .	468
<b>Capítulo 8 Funciones de salida de texto. . . . .</b>	<b>473</b>
Fuentes. . . . .	473
Familias de fuentes . . . . .	473
Conjuntos de caracteres . . . . .	474
Dimensiones de los caracteres . . . . .	475
La tabla de fuentes de Windows . . . . .	476
Incrustación de fuentes . . . . .	477
Funciones de salida de texto . . . . .	482
AddFontResource . . . . .	484
CreateFont . . . . .	484
CreateFontIndirect . . . . .	492
CreateScalableFontResource . . . . .	499
DrawText. . . . .	502
DrawTextEx . . . . .	506
EnumFontFamilies . . . . .	511
EnumFontFamiliesEx . . . . .	516
GetCharABCWidths . . . . .	523
GetCharWidth . . . . .	525
GetFontData . . . . .	526
GetGlyphOutline. . . . .	527
GetKerningPairs . . . . .	532
GetOutlineTextMetrics. . . . .	534
GetRasterizerCaps . . . . .	551
GetTabbedTextExtent . . . . .	552
GetTextAlign. . . . .	553
GetTextCharacterExtra. . . . .	555
GetTextColor. . . . .	555
GetTextExtentExPoint . . . . .	556
GetTextExtentPoint32 . . . . .	559
GetTextFace . . . . .	560
GetTextMetrics. . . . .	561
RemoveFontResource . . . . .	567
SetTextAlign . . . . .	568
SetTextCharacterExtra . . . . .	572
SetTextColor . . . . .	572
SetTextJustification . . . . .	573
TabbedTextOut. . . . .	574

TextOut. . . . .	577
<b>Capítulo 9 Funciones de recursos . . . . .</b>	<b>579</b>
Creando recursos . . . . .	579
Nombres y tipos de recursos . . . . .	580
Uniendo recursos . . . . .	580
Usando recursos . . . . .	581
Recursos definidos por el usuario . . . . .	582
Funciones de recursos. . . . .	584
EnumResourceLanguages . . . . .	584
EnumResourceNames . . . . .	589
EnumResourceTypes. . . . .	591
FindResource. . . . .	595
FindResourceEx . . . . .	597
LoadResource . . . . .	599
LoadString . . . . .	599
LockResource . . . . .	601
MakeIntResource . . . . .	602
SizeofResource. . . . .	604
<b>Capítulo 10 Funciones de movimiento de ventanas . . . . .</b>	<b>607</b>
Orden Z . . . . .	607
Efectos especiales . . . . .	608
Funciones de movimiento de ventanas . . . . .	610
AdjustWindowRect . . . . .	610
AdjustWindowRectEx . . . . .	612
BeginDeferWindowPos . . . . .	614
BringWindowToTop . . . . .	616
CascadeWindows . . . . .	617
CloseWindow . . . . .	618
DeferWindowPos . . . . .	619
EndDeferWindowPos . . . . .	622
GetWindowPlacement . . . . .	623
MoveWindow . . . . .	624
OpenIcon. . . . .	625
SetWindowPlacement . . . . .	626
SetWindowPos . . . . .	629
ShowOwnedPopups . . . . .	632
ShowWindow . . . . .	634
ShowWindowAsync . . . . .	636
TileWindows. . . . .	638
<b>Capítulo 11 Funciones del Shell . . . . .</b>	<b>641</b>

## XIV ■

Aplicaciones basadas en archivos . . . . .	641
Lista de identificadores de elementos . . . . .	645
La barra de aplicación. . . . .	646
Funciones del Shell . . . . .	649
DragAcceptFiles . . . . .	650
DragFinish . . . . .	652
DragQueryFile . . . . .	653
DragQueryPoint . . . . .	654
FindExecutable. . . . .	655
SHAddToRecentDocs . . . . .	658
SHAppBarMessage . . . . .	659
SHBrowseForFolder . . . . .	665
ShellAbout . . . . .	670
ShellExecute . . . . .	671
ShellExecuteEx . . . . .	675
Shell_NotifyIcon. . . . .	680
SHFileOperation . . . . .	683
SHFreeNameMappings . . . . .	686
SHGetFileInfo . . . . .	687
SHGetPathFromIDList. . . . .	691
SHGetSpecialFolderLocation . . . . .	691
<b>Capítulo 12 Funciones de menú. . . . .</b>	<b>695</b>
Información general sobre los menús. . . . .	695
El menú de sistema . . . . .	696
Menús emergentes. . . . .	696
Menús dibujados por el propietario. . . . .	697
Funciones de menú . . . . .	700
AppendMenu. . . . .	701
CheckMenuItem . . . . .	704
CheckMenuRadioItem . . . . .	705
CreateMenu . . . . .	707
CreatePopupMenu . . . . .	709
DeleteMenu . . . . .	712
DestroyMenu. . . . .	713
EnableMenuItem. . . . .	714
GetMenu . . . . .	716
GetMenuDefaultItem . . . . .	718
GetMenuItemCount . . . . .	719
GetMenuItemID . . . . .	719
GetMenuItemInfo . . . . .	720
GetMenuItemRect . . . . .	724
GetMenuState . . . . .	725

GetMenuString . . . . .	727
GetSubMenu . . . . .	728
GetSystemMenu . . . . .	729
HiliteMenuItem . . . . .	730
InsertMenu . . . . .	731
InsertMenuItem . . . . .	734
IsMenu . . . . .	735
ModifyMenu . . . . .	736
RemoveMenu . . . . .	739
SetMenu . . . . .	740
SetMenuDefaultItem . . . . .	741
SetMenuItemBitmaps . . . . .	743
SetMenuItemInfo . . . . .	745
TrackPopupMenu . . . . .	747
TrackPopupMenuEx . . . . .	748

## **Apéndice A Mensajes . . . . . 753**

WM_ACTIVATE . . . . .	753
WM_ACTIVATEAPP . . . . .	754
WM_ASKCBFORMATNAME . . . . .	755
WM_CANCELMODE . . . . .	756
WM_CHANGECHAIN . . . . .	756
WM_CHAR . . . . .	757
WM_CHARTOITEM . . . . .	758
WM_CHILDACTIVATE . . . . .	759
WM_CLEAR . . . . .	760
WM_CLOSE . . . . .	760
WM_COMMAND . . . . .	761
WM_COMPACTING . . . . .	762
WM_COMPAREITEM . . . . .	762
WM_COPY . . . . .	764
WM_COPYDATA . . . . .	765
WM_CREATE . . . . .	766
WM_CTLCOLORBTN . . . . .	767
WM_CTLCOLOREDIT . . . . .	767
WM_CTLCOLORLISTBOX . . . . .	768
WM_CTLCOLORMSGBOX . . . . .	769
WM_CTLCOLORSCROLLBAR . . . . .	770
WM_CTLCOLORSTATIC . . . . .	770
WM_CUT . . . . .	771
WM_DEADCHAR . . . . .	772
WM_DELETEITEM . . . . .	773
WM_DESTROY . . . . .	774

WM_DESTROYCLIPBOARD . . . . .	775
WM_DEVMODECHANGE . . . . .	775
WM_DISPLAYCHANGE . . . . .	776
WM_DRAWCLIPBOARD . . . . .	777
WM_DRAWITEM . . . . .	777
WM_DROPFILES . . . . .	780
WM_ENABLE . . . . .	780
WM_ENDSESSION . . . . .	781
WM_ENTERIDLE . . . . .	782
WM_ENTERMENULOOP . . . . .	783
WM_ERASEBKGD . . . . .	783
WM_EXITMENULOOP . . . . .	784
WM_FONTCHANGE . . . . .	784
WM_GETFONT . . . . .	785
WM_GETHOTKEY . . . . .	786
WM_GETICON . . . . .	786
WM_GETMINMAXINFO . . . . .	787
WM_GETTEXT . . . . .	788
WM_GETTEXTLENGTH . . . . .	789
WM_HELP . . . . .	790
WM_HSCROLL . . . . .	791
WM_HSCROLLCLIPBOARD . . . . .	792
WM_ICONERASEBKGD . . . . .	794
WM_INITMENU . . . . .	794
WM_INITMENUPOPUP . . . . .	795
WM_KEYDOWN . . . . .	796
WM_KEYUP . . . . .	797
WM_KILLFOCUS . . . . .	798
WM_LBUTTONDOWNCLK . . . . .	799
WM_LBUTTONDOWN . . . . .	800
WM_LBUTTONUP . . . . .	801
WM_MBUTTONDOWNCLK . . . . .	802
WM_MBUTTONDOWN . . . . .	803
WM_MBUTTONUP . . . . .	804
WM_MDIACTIVATE . . . . .	805
WM_MDICASCADE . . . . .	806
WM_MDICREATE . . . . .	807
WM_MDIDESTROY . . . . .	809
WM_MDIGETACTIVE . . . . .	809
WM_MDIICONARRANGE . . . . .	810
WM_MDIMAXIMIZE . . . . .	810
WM_MDINEXT . . . . .	811
WM_MDIREFRESHMENU . . . . .	812

WM_MDIRESTORE . . . . .	812
WM_MDISETMENU . . . . .	813
WM_MDITILE . . . . .	814
WM_MEASUREITEM . . . . .	815
WM_MENUCHAR . . . . .	816
WM_MENUSELECT . . . . .	817
WM_MOUSEACTIVATE . . . . .	818
WM_MOUSEMOVE . . . . .	819
WM_MOVE . . . . .	820
WM_NCACTIVATE . . . . .	821
WM_NCCALCSIZE . . . . .	822
WM_NCCREATE . . . . .	826
WM_NCDESTROY . . . . .	827
WM_NCHITTEST . . . . .	827
WM_NCLBUTTONDBLCLK . . . . .	829
WM_NCLBUTTONDOWN . . . . .	830
WM_NCLBUTTONUP . . . . .	830
WM_NCMBUTTONDBLCLK . . . . .	831
WM_NCMBUTTONDOWN . . . . .	832
WM_NCMBUTTONUP . . . . .	833
WM_NCMOUSEMOVE . . . . .	834
WM_NCPAINT . . . . .	834
WM_NCRBUTTONDBLCLK . . . . .	835
WM_NCRBUTTONDOWN . . . . .	836
WM_NCRBUTTONUP . . . . .	837
WM_NEXTDLGCTRL . . . . .	838
WM_NOTIFY . . . . .	838
WM_NOTIFYFORMAT . . . . .	840
WM_PAINT . . . . .	841
WM_PAINTCLIPBOARD . . . . .	842
WM_PALETTECHANGED . . . . .	843
WM_PALETTEISCHANGING . . . . .	844
WM_PARENTNOTIFY . . . . .	844
WM_PASTE . . . . .	846
WM_QUERYDRAGICON . . . . .	847
WM_QUERYENDSESSION . . . . .	847
WM_QUERYNEWPALETTE . . . . .	848
WM_QUERYOPEN . . . . .	849
WM_QUIT . . . . .	849
WM_RBUTTONDBLCLK . . . . .	850
WM_RBUTTONDOWN . . . . .	851
WM_RBUTTONUP . . . . .	852
WM_RENDERALLFORMATS . . . . .	853

## XVIII ■

WM_RENDERFORMAT . . . . .	854
WM_SETCURSOR . . . . .	855
WM_SETFOCUS . . . . .	856
WM_SETFONT . . . . .	856
WM_SETHOTKEY . . . . .	857
WM_SETICON . . . . .	858
WM_SETREDRAW . . . . .	859
WM_SETTEXT . . . . .	860
WM_SHOWWINDOW . . . . .	861
WM_SIZE . . . . .	862
WM_SIZECLIPBOARD . . . . .	863
WM_SPOOLERSTATUS . . . . .	863
WM_STYLECHANGED . . . . .	864
WM_STYLECHANGING . . . . .	865
WM_SYSCHAR . . . . .	866
WM_SYSCOLORCHANGE . . . . .	868
WM_SYSCOMMAND . . . . .	868
WM_SYSDEADCHAR . . . . .	870
WM_SYSKEYDOWN . . . . .	872
WM_SYSKEYUP . . . . .	873
WM_TIMECHANGE . . . . .	874
WM_TIMER . . . . .	875
WM_UNDO . . . . .	875
WM_VKEYTOITEM . . . . .	876
WM_VSCROLL . . . . .	877
WM_VSCROLLCLIPBOARD . . . . .	878
WM_WINDOWPOSCHANGED . . . . .	879
WM_WINDOWPOSCHANGING . . . . .	880
<b>Apéndice B Códigos de operaciones de barrido terciarias . . . . .</b>	<b>883</b>
<b>Apéndice C Conjunto de caracteres ASCII . . . . .</b>	<b>891</b>
<b>Apéndice D Tabla de códigos de teclas virtuales . . . . .</b>	<b>895</b>
<b>Apéndice E Bibliografía . . . . .</b>	<b>899</b>
<b>Apéndice F Código fuente y servicios en la web . . . . .</b>	<b>901</b>
<b>Indice Indice . . . . .</b>	<b>902</b>

## Sobre los autores

### **John Ayres**

John Ayres desarrolló su amor por la programación utilizando predecesores domésticos del PC como el Texas Instruments TI 99/4A y el Commodore 64. Entró en el mundo real al recibir clases de C y Pascal en la University of North Texas, donde participó en un proyecto experimental sobre programación de juegos dirigido por el Dr. Ian Parberry. Al terminar los estudios, comenzó a trabajar como técnico de soporte en Stream International, donde creó varios manuales de formación técnica y junto a Ken Harrison su primera aplicación profesional en Delphi 1, un sistema de seguimiento de llamadas telefónicas. Luego pasó a ser jefe de proyectos en Puzzled Software, y también trabajó para 7th Level, donde participó en el desarrollo de la serie de juegos educativos para niños. Actualmente John trabaja en Ensemble, Inc., donde utiliza Delphi para crear aplicaciones cliente-servidor corporativas para clientes Fortune 500. Forma parte de la Junta de Directores del Grupo de Usuarios Delphi de Dallas y contribuye a las publicaciones del grupo. Ha desarrollado componentes y utilidades *shareware* para Delphi, y escribe artículos para la revista *Delphi Informant*.

### **David Bowden**

Dave Bowden comenzó a escribir programas en 1982, en un ordenador Sinclair. Después de graduarse en Arquitectura en la Universidad Texas Tech, comenzó a practicar su profesión y a desarrollar paralelamente sistemas CAD. Eventualmente decide abandonar la arquitectura y dedicarse a lo que más le apasionaba, el desarrollo de *software*. Esto le llevó a aceptar una posición en InterGraph Corporation. Como desarrollador profesional, comenzó creando Puzzled Software. De ahí Dave pasó a ser Director de DemoShield Corporation (una división de InstallShield) donde dirigía el diseño, desarrollo, planificación y estrategia comercial de la empresa. Bowden tiene más de diez años de experiencia profesional en el diseño y desarrollo de *software*, y 12 en el tratamiento digital de imágenes y el diseño de interfaces de usuario. Después de utilizar varios lenguajes de programación durante años, Delphi se ha convertido en el único lenguaje que utiliza en la actualidad.

### **Larry Diehl**

Larry Diehl se ha incorporado recientemente a American Airlines como ingeniero *Web*, donde formará parte de un equipo altamente cualificado de desarrolladores que

utiliza Delphi para producir aplicaciones que serán usadas en todo el mundo. Anteriormente fue ingeniero de control de calidad en InstallShield Corporation. En el transcurso de quince años, Diehl ha utilizado varios lenguajes de programación, pero actualmente utiliza exclusivamente Delphi para desarrollar tanto las utilidades que su trabajo le exige como las aplicaciones *shareware* que desarrolla en su tiempo libre.

### **Phil Dorcas**

Phil Dorcas obtuvo el grado de Master en Ciencias Físicas de Texas Tech en 1969, y desde entonces ha estado relacionado con la ingeniería eléctrica y la informática. Fue co-propietario y director de la segunda tienda de productos informáticos de Texas. Ha trabajado como consultor y diseñador de numerosos proyectos de *hardware* y *software*. Es autor de diversos textos de informática que se utilizan en centros de formación de Texas, y artículos para las revistas del sector. Phil ha utilizado Turbo Pascal desde su primera versión para CP/M, y ahora Delphi es su herramienta de desarrollo favorita.

### **Ken Harrison**

Kenneth Harrison es Desarrollador Certificado de Delphi especializado en soluciones cliente/servidor. Actualmente trabaja para Rent Roll como miembro del Grupo de Tecnología Avanzada, desarrollando productos de contabilidad y análisis financiero. Kenneth ha trabajado con un amplio rango de sistemas de desarrollo, entre ellos dBase, Pascal, Delphi, Visual Basic y SQL. Kenneth es el *webmaster* del Grupo de Usuarios Delphi de Dallas.

### **Rod Mathes**

Rod Mathes ha programado en lenguajes tan diversos como Cobol, RPGII, dBase, RBase, Pascal y C++ durante los últimos doce años. Actualmente es programador de Delphi para The Thomas Group Interactive Technologies, creando una amplia gama de productos *software*. Ha sido miembro de la Junta de Directores del Grupo de Usuarios Delphi de Dallas durante los últimos dos años.

### **Ovais Reza**

Ovais Reza es un analista/programador de First American. Tiene mucha experiencia con C/C++ y Delphi, graduado de la Western Michigan University y miembro activo del Grupo de Usuarios Delphi de Dallas. Además de la programación y la filosofía oriental, Ovais es aficionado a “Los Simpsons” y los episodios viejos de “Star Trek”.

### **Mike Tobin**

Mike Tobin ha estado programando en Delphi desde la versión 1.0. Mike comenzó estudios de criminología, pero al darse cuenta de que prefería ser un *hacker*, los abandonó para entrar en la industria del *software*. Actualmente es ingeniero de sistemas para American General en Dallas y es Director de Comunicaciones del Grupo de Usuarios Delphi de Dallas. Le gusta disfrutar de su tiempo libre en compañía de su novia Lisa y de sus cuatro gatos, Elisabeth, Alexander, Einstein y Copernicus.

## Agradecimientos

El trabajo en equipo. Este concepto nos lleva a pensar en otros conceptos abstractos como victoria, logro y conquista. El trabajo en equipo es el ingrediente secreto detrás de innumerables triunfos a través de la historia, y así ha sido en el caso de este libro. Los ocho autores hemos empleado muchas horas largas y duras en este proyecto, pero éste no podría haber llegado a buen final sin la ayuda y generosidad de muchas otras personas. Para dar mayor importancia a aquellos que merecen mucho más, los autores desean colectivamente agradecer a las siguientes personas su contribución al libro:

Marian Broussard, nuestra redactora. Incansable, nos indicó todos los errores gramaticales y ayudó a corregir diversas inconsistencias en el libro. Sin escatimar su tiempo, contribuyó voluntariamente a ayudar a un grupo de escritores noveles a llevar de forma clara y correcta sus ideas al papel.

Joe Hecht, nuestro mentor e ídolo. Joe siempre respondió amablemente a todas las preguntas que le hicimos, hurgó en nuestro código y señaló nuestros errores cuando teníamos problemas, y nos indicó la dirección correcta cuando la documentación del API de Microsoft era algo confusa.

Jim Hill y todos los amigos de Wordware Publishing, que se arriesgaron con un grupo de escritores entusiastas aunque sin experiencia. Jim nos puso al tanto de todos los detalles, nos indicó el camino a seguir, y hasta nos llevó a comer a todos periódicamente.

Marci Ayres, que verificó gran cantidad de código, conversión de formato de imágenes, formateo de documentos y otras labores de soporte.

Lisa Tobin, que realizó tareas adicionales de verificación.

Damon Tomlinson, de quien tomamos numerosas ideas para el libro, y que además nos ayudó en la revisión del texto cuando nadie más estaba disponible.

Marco Cocco, de D3K Artisan of Ware, que ideó el ejemplo de utilización de la función *GetGlyphOutline* que se incluye en el capítulo “Funciones de salida de texto”.

Y por supuesto, los agradecimientos no estarían completos sin mencionar al equipo de desarrollo de Delphi de Inprise Corp., por ofrecernos tan maravilloso entorno de desarrollo.

Adicionalmente, John Ayres quisiera agradecer a todo el equipo editorial el esfuerzo dedicado a producir algo de lo que podemos estar orgullosos. A Rusty Cornet, por engancharme a este maravilloso entorno de desarrollo que es Delphi; Debbie Vilbig y

Darla Corley, por haberme permitido aprender Delphi y desarrollar una aplicación de seguimiento de llamadas cuando debía haber estado trabajando; Sarah Miles, por prestarme el dinero para comprar el equipo en el que se ha escrito este libro; y Suzie Weaver y Brian Donahoo, que confiaron en un antiguo empleado y me ofrecieron un lugar tranquilo en el que trabajar los fines de semana.

## Prólogo

El API de Windows es la base sobre la cual se implementa la mayoría de las aplicaciones modernas. Es el corazón y el alma de las aplicaciones de bases de datos, multimedia, e incluso de muchas aplicaciones para redes. Todas las aplicaciones Windows se apoyan en el API para llevar a cabo desde la tarea más simple hasta la más esotérica.

Todos los buenos programadores que conozco tienen un conocimiento sólido del API de Windows. Es el lenguaje en el que se expresa más elocuentemente la arquitectura de Windows como sistema operativo, y que guarda los secretos que los programadores necesitan conocer si desean construir aplicaciones potentes y eficientes.

Puedo mencionar al menos tres razones por las que cualquier programador serio debe conocer el API de Windows:

1. Es ocasionalmente posible crear una aplicación robusta sin tener un buen dominio del API de Windows. Sin embargo, llega el momento en el transcurso del desarrollo de la mayoría de los proyectos en el que se hace sencillamente imprescindible hacer uso de los recursos del API de Windows para resolver un problema específico. Normalmente éso se produce porque la herramienta que se está utilizando no ofrece cierta característica o posibilidad que se necesita, o porque la característica no está implementada adecuadamente. En tales casos, se hace necesario utilizar el API de Windows para implementar por sí mismo la funcionalidad necesaria.
2. Otra razón para utilizar el API de Windows se hace evidente cuando se desea crear un componente o utilidad que otros puedan luego utilizar. Si Ud. desea crear un componente, control ActiveX o una simple utilidad que pueda ser necesaria a otros desarrolladores, probablemente necesitará utilizar los recursos del API de Windows. Sin recurrir al API de Windows, tales proyectos, por lo general, no pueden ser llevados a cabo.
3. Por último, la tercera y más convincente razón para aprender el API de Windows es que ese conocimiento le ayudará a comprender cómo debe Ud. diseñar sus aplicaciones. Hoy por hoy disponemos de muchas herramientas de alto nivel que nos permiten crear proyectos a un nivel de abstracción muy alto. Sin embargo, todas esas herramientas están implementadas encima del API de Windows, y es difícil, si no imposible, entender cómo utilizarlas eficazmente sin comprender la arquitectura en la que ellas se apoyan. Si Ud. domina el API de Windows, entonces sabe lo que el sistema operativo puede hacer por Ud., y bajo qué líneas generales se ejecutan esos servicios.

Armado de tales conocimientos, Ud. puede utilizar las herramientas de alto nivel a su disposición de una manera más juiciosa y efectiva.

Me produce especial placer escribir el prólogo para esta serie de libros sobre el API de Windows porque está orientada a su utilización desde la mejor herramienta de desarrollo disponible en el mundo: Delphi. Delphi ofrece al programador acceso total al API de Windows. Es una herramienta diseñada para permitir el uso de todas aquellas posibilidades que han hecho de Windows el sistema operativo predominante en el mundo.

Equipado con estos libros sobre el API de Windows y una copia de Delphi, Ud. podrá desarrollar cualquier tipo de aplicación que desee, y estar seguro de que la está desarrollando de una manera óptima. Ningún otro compilador puede llevarle más cerca del sistema operativo, ni permitirle explotar al máximo sus posibilidades. Por su parte, estos libros constituyen el puente entre Delphi y el API de Windows. Los lectores podrán utilizarlos para crear las aplicaciones más potentes soportadas por el sistema operativo. Me quito el sombrero ante los autores, que al crear estos libros han ofrecido un gran servicio a la comunidad de programadores.

Charlie Calvert

Director de Relaciones con los Desarrolladores

Inprise Corporation

## Introducción

Ningún otro sistema operativo en la historia ha causado tanta controversia o confusión en la industria del *software* como Windows. Por supuesto, tampoco ningún otro sistema operativo en la historia ha hecho millonarios a tantos. Nos guste o no, Windows es hoy el sistema operativo predominante. Es difícil ignorar una base de usuarios tan amplia, y cada vez son menos las ofertas de empleo en las que no se exige que el programador tenga conocimientos de la programación para Windows.

En los comienzos, la única opción que tenía el programador para desarrollar aplicaciones para Windows era C/C++. La era del predominio de este lenguaje nos dejó toneladas de documentación sobre el API de Windows, llenas de información abstracta e incompleta, y de ejemplos tan arcaicos y esotéricos como el lenguaje C en sí mismo. Y entonces apareció Delphi. Con él nació una nueva era de la programación para Windows, que hizo posible crear aplicaciones complejas en plazos inconcebibles hasta ese momento. Aunque Delphi intenta aislar al programador de la arquitectura de Windows subyacente, los programadores de Delphi se han dado cuenta de que algunos obstáculos de programación no pueden ser salvados sin acudir a las funciones de bajo nivel del API de Windows. Y aunque algunos libros han tocado el tema de la utilización del API de Windows desde Delphi, ninguno se ha dedicado a presentar el tema en profundidad. Por eso surgió la idea de escribir esta serie.

Este libro (junto a los restantes tomos de la serie) constituye un manual de referencia sobre cómo utilizar las funciones del API Win32 desde Delphi. Como tal, no es un libro introductorio acerca de la programación en Windows o en Delphi, ni tampoco una colección de trucos para resolver problemas específicos en Delphi. Hasta la fecha, este libro es la referencia más completa y exacta del API de Windows para el programador de Delphi. No es una referencia completa, ya que el API de Windows incluye miles de funciones que llenarían varios volúmenes de mayores dimensiones que el que Ud. tiene ahora en sus manos. Sin embargo, cubre las áreas más comunes e importantes del API de Windows. Adicionalmente, todas las funciones que se describen en este libro están disponibles bajo Windows 95, 98 y NT 4.0, y lo estarán en futuras versiones de Windows.

## Descripción de los capítulos

### *Capítulo 1: Delphi y el API de Windows*

Este capítulo introduce al lector a *Los Tomos de Delphi: El API gráfico Win32*. Presenta temas y técnicas generales de programación para Windows, y explica los detalles relacionados con la utilización del API Win32 desde Delphi.

### *Capítulo 2: Funciones de la interfaz del dispositivo gráfico*

Las funciones básicas de la interfaz del dispositivo gráfico constituyen la base de cualquier programación gráfica en Windows. Este capítulo describe las funciones que se utilizan para crear y manipular contextos de dispositivos. Los ejemplos incluyen la creación de varios tipos de contextos de dispositivos, la recuperación de sus capacidades y el cambio de los modos de visualización.

### *Capítulo 3: Funciones de dibujo*

Las salidas gráficas básicas se basan en el dibujo de líneas, círculos, cuadrados y otras primitivas geométricas. Este capítulo cubre funciones para dibujar y rellenar todo tipo de figuras geométricas. Los ejemplos incluyen el dibujo de líneas y figuras, la creación de plumas y brochas, y técnicas sencillas de disolución de mapas de bits.

### *Capítulo 4: Funciones de regiones y rutas*

Las funciones de regiones y rutas son casi ignoradas por la mayoría de las referencias sobre programación gráfica. Sin embargo, estas funciones permiten al desarrollador realizar una serie de efectos especiales sorprendentes. Este capítulo cubre las funciones que pueden utilizarse para crear y manipular regiones y rutas. Los ejemplos incluyen el recorte de una salida gráfica a una región o ruta y el uso de rutas para producir efectos especiales de texto.

### *Capítulo 5: Funciones de mapas de bits y metaфicheros*

Los mapas de bits y los metaфicheros son los dos formatos gráficos originalmente soportados por Windows. Las funciones de mapas de bits son esenciales a casi toda la programación gráfica de Windows y en este capítulo se cubren las funciones usadas para crear y manipular mapas de bits y metaфicheros gráficos. Los ejemplos incluyen la creación de mapas de bits dependientes e independientes del dispositivo, la creación de metaфicheros y el análisis del contenido de metaфicheros.

### *Capítulo 6: Funciones de iconos, cursores y cursores de edición*

Los iconos, cursores y cursores de edición son también objetos gráficos fundamentales. Este capítulo describe las funciones usadas para crear y manipular iconos, cursores y cursores de edición. En los ejemplos se muestra cómo extraer iconos de фicheros externos, la manipulación del cursor de edición y la creación de nuevos cursores.

### ***Capítulo 7: Funciones de paleta***

Bajo controladores de pantalla de 256 colores las paletas determinan los colores accesibles a una aplicación. Este capítulo describe las funciones que se utilizan para crear nuevas paletas, manipular la paleta del sistema y las paletas almacenadas en mapas de bits y metaarchivos. Los ejemplos incluyen la extracción y selección de la paleta de un mapa de bits, la creación de nuevas paletas y la animación de paletas.

### ***Capítulo 8: Funciones de salida de texto***

Mostrar texto en la pantalla es la operación gráfica que más frecuentemente realiza cualquier aplicación de Windows. Ningún programa puede funcionar bien si no muestra algún tipo de texto y este capítulo describe las funciones que se utilizan para manipular fuentes y mostrar texto en pantalla. Los ejemplos incluyen la enumeración de fuentes, la recuperación de información sobre fuentes, la incrustación de fuentes y varios métodos de salida de texto.

### ***Capítulo 9: Funciones de recursos***

Si bien Delphi hace que en la mayoría de los casos la manipulación directa de recursos sea innecesaria, un desarrollador de Delphi puede aprovecharse de posibilidades tales como la incrustación en archivos ejecutables de gráficos y otro tipo de información definida por el usuario. Este capítulo cubre las funciones que permiten cargar y utilizar varios tipos de recursos. Los ejemplos incluyen la carga de mapas de bits de 256 colores sin perder la información de su paleta y el uso de varios tipos de recursos definidos por el usuario.

### ***Capítulo 10: Funciones de movimiento de ventanas***

Controlar la posición de una ventana puede ser una parte importante de una interfaz de usuario compleja. Este capítulo trata aquellas funciones que permiten controlar la posición y el tamaño de una ventana. Los ejemplos incluyen el movimiento de múltiples ventanas simultáneamente y la recuperación de información sobre su tamaño y posición.

### ***Capítulo 11: Funciones del shell***

Windows ofrece numerosas funciones a nivel del sistema que pueden ser útiles cuando se manipulan archivos. Este capítulo cubre aquellas funciones que se utilizan para copiar y mover archivos, y recuperar información sobre archivos. En los ejemplos se describe cómo crear una barra de aplicaciones, copiar y mover archivos y solicitar información sobre los mismos.

### ***Capítulo 12: Funciones de menú***

Los menús son una herramienta de navegación muy común en la interfaz de usuario tradicional de Windows. Si bien Delphi encapsula de manera muy efectiva las funciones de menú del API, estas funciones pueden ser usadas por el programador de Delphi para producir algunos efectos especiales sorprendentes con los menús. Este capítulo describe las funciones usadas para crear y manipular menús. Los ejemplos

## XXVIII ■

incluyen la adición de elementos al menú de sistema, la creación de menús de mapas de bits, la especificación de mapas de bits asociados a las marcas de menú y el uso de menús dibujados por el usuario.

### *Apéndice A: Mensajes*

Los mensajes son el flujo sanguíneo de Windows, y la arquitectura basada en mensajes ofrece la base necesaria para la programación dirigida por eventos utilizada comúnmente hoy en día. Aunque el conocimiento de muchos mensajes de Windows no es imprescindible para aprender a programar en Delphi, es difícil realizar tareas complicadas sin conocer al menos algunos de los mensajes de Windows. Este apéndice documenta muchos de los mensajes de Windows para los cuales Delphi ofrece estructuras de datos específicas.

### *Apéndice B: Códigos de operaciones de barrido terciarias*

Una tabla que lista los 256 códigos de operaciones terciarias que pueden utilizarse para combinar mapas de bits y plumas.

### *Apéndice C: Conjunto de caracteres ASCII*

Una tabla que muestra los caracteres imprimibles correspondientes a los códigos del conjunto de caracteres ASCII.

### *Apéndice D: Tabla de códigos de teclas virtuales*

Una tabla que lista el valor de las teclas virtuales de Windows.

## Convenios

A lo largo de todo el libro se hace uso de ciertos convenios de escritura. Por ejemplo, todo el código de los ejemplos aparece en un tipo de letra más pequeño y de ancho fijo:

```
function HelloThere(Info: String): Integer;  
begin  
    ShowMessage(Info);  
end;
```

Para ser consistentes con otras publicaciones sobre Delphi, los ejemplos utilizan los convenios de escritura de Inprise, que incluye el uso de mayúsculas y minúsculas mezclados en los identificadores, minúsculas y negritas para las palabras claves, y dos espacios de indentación por cada nivel. Las constantes que aparecen en el código están en mayúsculas, como por ejemplo TRUE y FALSE. Nótese además que en la misma línea en que se describe el nombre de una función se ofrece el nombre de la unidad en la que la función está situada. El nombre de esta unidad deberá incluirse en la cláusula **uses** de cualquier módulo en el que se desee utilizar esta función. Sin embargo, la mayoría de las funciones de este libro están declaradas en el fichero **Windows.Pas**, que es añadido automáticamente a las cláusulas **uses** por Delphi.

## Descripciones de funciones

Las descripciones de las funciones del API de Windows han sido colocadas de modo que se ofrezca una cantidad de información creciente al lector. Esto debe permitir al lector dar un vistazo rápido para ver la descripción de la función y el significado de sus parámetros, o continuar leyendo la explicación detallada de la función, un ejemplo de su utilización, y los valores aceptables para los parámetros.

Cada descripción de función incluye la sintaxis exacta incluida en el código fuente de Delphi, una descripción de qué hace la función, una descripción detallada de sus parámetros, el valor que devuelve, una lista de funciones relacionadas con ésta, y un ejemplo de su utilización. Las constantes que se utilizan en los parámetros de la función se describen en tablas situadas a continuación del ejemplo.

## Código fuente de los ejemplos

Aunque todo libro llega al punto en el que los autores escriben código sin probar para cumplir con los plazos de entrega, los autores no querían que los ejemplos sufrieran por las limitaciones de tiempo. Los autores han hecho un gran esfuerzo para asegurarse de que todos los ejemplos de código que acompañan al libro funcionan, y de que el código que se muestra en el libro es copia fiel del código final del ejemplo. Sin embargo, tenga en cuenta que la mayoría de los ejemplos se apoyan en botones, cuadros de edición u otros componentes visuales residentes en formularios, lo cual puede no ser evidente en algunos listados. Cuando tenga dudas, consulte el código fuente.

Además en Danysoft hemos puesto en el siguiente área de nuestra Web : <http://www.danypress.com>, información de los libros publicados y un enlace para que capture el código fuente completo, totalmente actualizado y corregido ante posibles cambios (más información en el Apéndice F).

## A quién va dirigido este libro

Debido al estilo de manual de referencia, y la ausencia de explicaciones para principiantes acerca de la programación para Windows o Delphi, este libro está indicado para programadores con cierta experiencia de trabajo en Delphi y conocimientos básicos sobre la programación para Windows. Ello no quiere decir que un programador principiante en Delphi no obtendrá beneficios de la lectura de este libro; los ejemplos ampliamente documentados deben ofrecer suficientes explicaciones incluso al más neófito programador de Delphi. Como cualquier manual de referencia, este libro no está hecho para ser leído secuencialmente. Sin embargo, los capítulos han sido dispuestos en orden creciente de complejidad, comenzando por las funciones más básicas del API de Windows y avanzando progresivamente hacia las más complejas.

Si está Ud. buscando una introducción a la programación en Delphi, o un tutorial para aprender paso a paso la programación bajo Windows, hay otros buenos libros que pueden ayudarle a empezar. Si Ud. debe resolver un problema difícil y su única

**XXX ■**

esperanza es el API de Windows, si desea extender la funcionalidad de los componentes y objetos de Delphi, o si desea acceder a una buena colección de ejemplos de programación con el API Win32 desde Delphi, entonces este libro es para Ud. No encontrará una guía más completa y exacta acerca de la utilización del API de Windows para el programador de Delphi.

## Capítulo I

# Delphi y el API de Windows

Delphi ha traído una nueva era a la programación en Windows. Nunca antes ha sido tan sencillo crear aplicaciones potentes y robustas para el entorno Windows en tan cortos plazos de tiempo. Ahora en su quinta versión, Delphi se conoce mundialmente como el entorno de desarrollo visual por excelencia para Windows. Ninguna otra herramienta de programación puede siquiera acercarse a la potencia, facilidad de uso y calidad de los ejecutables de Delphi.

Uno de los puntos fuertes de Delphi es la Librería de Componentes Visuales (Visual Component Library), el modelo de objetos de Borland. Este modelo de objetos ha permitido al equipo de desarrolladores de Delphi encapsular la casi totalidad del tedioso proceso de programación para Windows en componentes de fácil utilización. Los anteriores lenguajes de programación para Windows exigían al desarrollador escribir grandes cantidades de código sólo para exprimir de Windows un mínimo de funcionalidad. El simple acto de crear una ventana e interceptar acciones de menú ocupaba páginas enteras de código. La excelente encapsulación por parte de Delphi de las tediosas exigencias de la programación para Windows ha convertido lo que una vez fue una tarea monótona en una experiencia divertida y excitante.

## El API de Windows vs. la VCL

El equipo de desarrollo de Delphi hizo un trabajo excelente al encapsular en la VCL la amplia mayoría de la funcionalidad del vasto API de Windows. Sin embargo, debido a las dimensiones de este API, sería imposible y poco práctico encapsular cada una de sus funciones en un objeto de Object Pascal. Para alcanzar ciertos objetivos o resolver un problema específico, un programador puede verse obligado a utilizar funciones de más bajo nivel del API de Windows que sencillamente no están encapsuladas en ningún objeto Delphi. Puede ser necesario, por otra parte, extender la funcionalidad de un objeto Delphi, y si este objeto encapsula alguna parte concreta del API de Windows, la extensión deberá realizarse, probablemente, utilizando también en gran medida los recursos del API.

## Tipos de datos de Windows

Las funciones del API de Windows utilizan ciertos tipos de datos que pueden no ser familiares para los programadores ocasionales de Delphi. Estos tipos se importan de los ficheros de cabecera originales en C que definen la estructura de las funciones del API de Windows. En su gran mayoría, estos nuevos tipos de datos son sencillamente tipos de datos de Pascal que han sido renombrados para hacerlos similares a los tipos de datos originalmente utilizados en lenguajes de programación anteriores. Esto se hizo así para facilitar que los programadores experimentados comprendieran los tipos de parámetros y valores de retorno de las funciones del API, y que los prototipos de las funciones fuesen muy parecidos a los mostrados en la documentación del API de Windows, para evitar posibles confusiones. La siguiente tabla muestra los tipos de datos más comunes de Windows y sus equivalentes en Object Pascal.

**Tabla I-1: Tipos de datos de Windows**

Tipo Windows	Tipo Object Pascal	Descripción
LPSTR	PAnsiChar	Puntero a cadena de caracteres.
LPCSTR	PAnsiChar	Puntero a cadena de caracteres.
DWORD	LongWord	Número entero sin signo.
BOOL	LongBool	Valor booleano.
PBOOL	^BOOL	Puntero a valor booleano.
PByte	^Byte	Puntero a valor byte.
PINT	^Integer	Puntero a valor entero.
PSingle	^Single	Puntero a número de coma flotante de precisión simple.
PWORD	^Word	Puntero a valor de 16 bits.
PDWORD	^DWORD	Puntero a valor de 32 bits.
LPDWORD	PDWORD	Puntero a valor de 32 bits.
UCHAR	Byte	Valor de 8 bits (puede representar un carácter).
PUCHAR	^Byte	Puntero a valor de 8 bits.
SHORT	Smallint	Número entero de 16 bits.
UINT	LongWord	Número entero de 32 bits (sin signo).
PUINT	^UINT	Puntero a entero de 32 bits.
ULONG	Cardinal	Número entero de 32 bits (sin signo).
PULONG	^ULONG	Puntero a entero de 32 bits.
PLongint	^Longint	Puntero a valor de 32 bits.
PInteger	^Integer	Puntero a valor de 32 bits.
PSmallInt	^Smallint	Puntero a valor de 16 bits.
PDouble	^Double	Puntero a número de coma flotante de doble precisión.
LCID	DWORD	Identificador local.
LANGID	Word	Identificador de lenguaje.

Tipo Windows	Tipo Object Pascal	Descripción
THandle	Integer	Identificador (manejador) de objeto. Muchas funciones del API de Windows devuelven un valor de este tipo, que identifica a un objeto dentro de las tablas internas de objetos de Windows.
PHandle	^THandle	Puntero a manejador.
WPARAM	Longint	Parámetro de mensaje de 32 bits. En versiones anteriores de Windows era un tipo de datos de 16 bits.
LPARAM	Longint	Parámetro de mensaje de 32 bits.
LRESULT	Longint	Valor de retorno de función de 32 bits.
HWND	Integer	Manejador de ventana. Todos los controles, ventanas principales o hijas, etc. tienen su manejador de ventana correspondiente, que les identifica dentro de las tablas internas de Windows.
HHOOK	Integer	Manejador de un “gancho” ( <i>hook</i> ) de sistema instalado.
ATOM	Word	Índice de una cadena de caracteres dentro de la tabla de átomos locales o globales.
HGLOBAL	THandle	Manejador que identifica un bloque de memoria global reservado dinámicamente. En las versiones de Windows de 32 bits, no hay distinción entre la memoria dinámica reservada global o localmente.
HLOCAL	THandle	Manejador que identifica un bloque de memoria local reservado dinámicamente. En las versiones de Windows de 32 bits, no hay distinción entre la memoria dinámica reservada global o localmente.
FARPROC	Pointer	Puntero a procedimiento. Se utiliza normalmente como tipo de parámetro en funciones que requieren que se suministre la dirección de una función de respuesta ( <i>callback function</i> ).
HGDIOBJ	Integer	Manejador de objeto gráfico. Los contextos de dispositivo, plumas, brochas, etc. tienen un manejador de este tipo que los identifica dentro de las tablas internas de Windows.
HBITMAP	Integer	Manejador de objeto de mapa de bits de Windows.
HBRUSH	Integer	Manejador de objeto de brocha de Windows.
HPEN	Integer	Manejador de objeto de pluma de Windows.
HDC	Integer	Manejador de contexto de dispositivo de Windows.
HPALETTE	Integer	Manejador de objeto de paleta de colores de Windows.
HFONT	Integer	Manejador de objeto de fuente lógica de Windows.
HICON	Integer	Manejador de objeto de icono de Windows.
HMENU	Integer	Manejador de objeto de menú de Windows.
HMETAFILE	Integer	Manejador de objeto de metaarchivo de Windows.
HENHMETAFILE	Integer	Manejador de objeto de metaarchivo mejorado de Windows.
HRGN	Integer	Manejador de objeto de región de Windows.

Tipo Windows	Tipo Object Pascal	Descripción
HINST	Integer	Manejador de objeto de instancia.
HMODULE	HINST	Manejador de objeto de un módulo.
HRSRC	Integer	Manejador de objeto de recurso de Windows.
HKL	Integer	Manejador de configuración de teclado.
HFILE	Integer	Manejador de fichero abierto.
HCURSOR	HICON	Manejador de objeto de cursor de Windows.
COLORREF	DWORD	Valor de referencia de color de Windows, que contiene valores para los componentes rojo, verde y azul de un color.

### Manejadores

Un concepto fundamental de la programación Windows es el de *manejador* de objeto. Muchas funciones devuelven un manejador de objeto que la función crea o carga desde un recurso. Funciones como *CreateWindow* y *CreateWindowEx* devuelven un manejador de ventana. Otras funciones devuelven un manejador de fichero abierto, como *CreateFile*, o un manejador de un *heap* recién reservado en memoria, como *HeapCreate*. Internamente, Windows mantiene la información necesaria sobre todos esos objetos, y los manejadores sirven como enlace a través del sistema operativo entre el objeto y la aplicación. Éste es el mecanismo que permite que una aplicación se comunique con el sistema operativo. A través de los manejadores, una aplicación puede fácilmente referirse a cualquiera de esos objetos, y el sistema operativo sabrá instantáneamente qué objeto desea manipular la aplicación.

### Constantes

Las funciones del API de Windows declaran miles de constantes diferentes para que sean utilizadas como valores de parámetros. En el fichero **Windows.PAS** se definen constantes para todo tipo de usos, desde valores de colores hasta valores de retorno. Las constantes definidas para cada función del API se listan junto con la función en cuestión dentro de ese fichero. Sin embargo, **Windows.PAS** puede ofrecer más información con relación a las constantes asociadas a cualquier función particular; por ello, una buena regla a tener siempre en cuenta es la de comprobar este fichero al utilizar funciones complejas.

### Cadenas de caracteres

Todas las funciones del API de Windows que utilizan cadenas requieren un puntero a un *array* de caracteres terminados en nulo. Windows ha sido escrito en C, que no ofrece el tipo de cadena de Pascal. Las versiones iniciales de Delphi exigían que la aplicación reservara un buffer para la cadena y convirtiera la variable de tipo **String** a *PChar*. Sin embargo, a partir de Delphi 3 el formato interno de las cadenas y un nuevo mecanismo de conversión permiten utilizar una cadena como un *PChar* mediante una simple conversión de tipo (por ejemplo, *PChar(MiCadena)*, donde *MiCadena* es una variable declarada como *MiCadena: String*). En la mayoría de los casos, esta

conversión podrá utilizarse al llamar a una función del API de Windows que requiera un parámetro de tipo cadena de caracteres.

## Importación de funciones de Windows

El API de Windows es enorme. Define funciones para casi cualquier utilidad o acción que un programador podría imaginar. Debido al generoso volumen del API de Windows, algunas funciones simplemente han sido “olvidadas” y no han sido importadas por las librerías de Delphi. Dado que todas las funciones del API de Windows son sencillamente funciones exportadas de DLLs, importar una nueva función del API de Windows es un proceso relativamente simple, siempre que se conozcan los parámetros de la misma.

Importar una nueva función del API de Windows es exactamente igual a importar cualquier otra función de una DLL. Por ejemplo, supongamos que la función *BroadcastSystemMessage* descrita en el capítulo “Funciones de Gestión de Mensajes” (del libro “Los Tomos de Delphi: Núcleo del API Win32”), no estuviese importada en el código fuente incluido en Delphi<sup>1</sup>. Para importar esa función y poder utilizarla en una aplicación, bastaría con declararla como:

```
function BroadcastSystemMessage(Flags: DWORD; Recipients: PDWORD;
    uiMessage: UINT; wParam: WPARAM; lParam: LPARAM): Longint; stdcall;
```

### implementation

```
function BroadcastSystemMessage; external user32 name 'BroadcastSystemMessage';
```

Siempre que se conozcan los tipos de los parámetros exigidos por la función y el nombre de la DLL que contiene la función, cualquier función del API de Windows puede ser importada y utilizada por una aplicación Delphi. Es importante destacar que la directiva **stdcall** debe añadirse siempre al prototipo de la función, ya que éste es el mecanismo estándar mediante el cual Windows pasa los parámetros a la función a través de la pila.

### Funciones importadas incorrectamente

Algunas funciones han sido importadas incorrectamente en el código fuente de Delphi. Esas excepciones se señalan en las descripciones individuales de las funciones. En la mayoría de los casos, las funciones que han sido incorrectamente importadas tienen relación con la posibilidad de pasar el valor **nil** como valor a un parámetro de tipo puntero, generalmente para recuperar el tamaño necesario para un *buffer* que la aplicación debe reservar dinámicamente una vez conozca la longitud necesaria. En

1 De hecho, no lo estaba en la versión de Windows.PAS incluida en Delphi 3. Esta omisión fue corregida en Delphi 4.

Delphi, algunas funciones de este tipo han sido importadas con parámetros definidos como **var** o **const**. Estos tipos de parámetros aceptan un puntero a un *buffer*, pero nunca pueden recibir **nil**, limitando de este modo la utilización de la función dentro de Delphi. Como ocurre casi siempre con Delphi, el problema es muy fácil de resolver. Simplemente vuelva a importar la función, según hemos descrito antes. Las funciones que han sido importadas incorrectamente se identifican en el libro en las descripciones individuales de cada función.

## Funciones de respuesta

Otro concepto muy importante de la programación Windows es el de *función de respuesta* (*callback function*). Una función de respuesta es una función dentro de la aplicación que no es llamada directamente por ninguna otra función o procedimiento de la aplicación, sino que es llamada por el sistema operativo. Esto le permite a Windows comunicarse directamente con la aplicación, pasándole los parámetros requeridos por la función de respuesta en cuestión. La mayoría de las funciones de enumeración requieren algún tipo de función de respuesta definida por la aplicación que reciba la información que se enumera.

Las funciones de respuesta individuales tienen parámetros específicos que deben ser declarados exactamente por la aplicación. Esto es necesario para que Windows pase a la función la información correcta en el orden correcto. Un buen ejemplo de función que utiliza una función de respuesta es *EnumWindows*. Esta función recorre todas las ventanas de nivel superior en la pantalla, pasando el manejador de cada ventana a la función de respuesta definida por la aplicación. El proceso continúa hasta que todas las ventanas hayan sido enumeradas o la función de respuesta devuelva FALSE. La función de respuesta utilizada por *EnumWindows* se define como:

```
EnumWindowsProc(
    hWnd: HWND;           {manejador de ventana de alto nivel}
    lParam: LPARAM         {datos definidos por la aplicación}
): BOOL;                 {devuelve TRUE o FALSE}
```

Una función con este prototipo deberá definirse dentro de la aplicación, y un puntero a ella deberá pasarse como parámetro a la función *EnumWindows*. El sistema operativo llamará a la función de respuesta una vez por cada ventana de nivel superior en la pantalla, pasando cada vez el manejador de una de ellas como parámetro. Es importante destacar que la directiva **stdcall** debe ser añadida al prototipo de la función de respuesta, ya que éste es el mecanismo estándar de traspaso de parámetros que utiliza Windows. Por ejemplo, la función de respuesta anterior deberá tener el siguiente prototipo:

```
function EnumWindowsProc(hWnd: HWND; lParam: LPARAM): BOOL; stdcall;
```

Sin la directiva **stdcall**, Windows no será capaz de acceder a la función de respuesta. Este potente mecanismo de *software* permite en muchos casos que una aplicación recupere información sobre el sistema que es almacenada sólo internamente por

Windows y que de otro modo sería totalmente inalcanzable. Para ver un ejemplo completo de utilización de funciones de respuesta, consulte la función *EnumWindows*, y muchas otras funciones a lo largo del libro.

## Parámetros de funciones

La gran mayoría de las funciones del API de Windows simplemente reciben los parámetros estáticos que se le envían y realizan cierta tarea en base a los valores de los parámetros. Sin embargo, ciertas funciones devuelven valores que deberán ser almacenados en un *buffer*, y este *buffer* deberá pasarse a la función en forma de puntero. En la mayoría de los casos en que la descripción de una función especifica que ésta devuelve algún valor a través de un *buffer* de cadena terminada en carácter nulo o de una estructura de datos, estos *buffers* y estructuras deberán ser reservados por la aplicación antes de llamar a la función.

En muchos casos, la documentación de un parámetro puede especificar que éste puede contener uno o más valores de una tabla. Esos valores se definen en forma de constantes, y pueden combinarse utilizando el operador **or**. Un valor concreto de ese parámetro generalmente identifica una máscara de bits, donde el estado de cada bit tiene un significado concreto para la función. Es por eso que las constantes pueden combinarse mediante operaciones de manipulación de bits. Por ejemplo, la función *CreateWindow* tiene un parámetro llamado *dwStyle* que puede aceptar un número variable de constantes combinadas mediante el operador **or**. Para pasar más de una constante a la función, al parámetro deberá asignársele un valor como “WS\_CAPTION **or** WS\_CHILD **or** WS\_CLIPCHILDREN”. Esto hará que se cree una ventana hija que tendrá una barra de título y no dibujará en el área ocupada por sus ventanas hijas al redibujarse.

A la inversa, cuando la documentación de una función especifica que ésta devuelve uno o más valores definidos como constantes, el valor devuelto puede combinarse con cualquiera de las constantes utilizando el operador **and** para determinar si la constante está incluida en el valor de retorno. Si el resultado de la combinación es igual a la constante (por ejemplo, **if** (Result **and** WS\_CHILD) = WS\_CHILD **then** ...), la constante está incluida en el valor devuelto por la función.

## Unicode

Originalmente, el software necesitaba únicamente un byte para definir un carácter de un conjunto de caracteres. Esto permitía hasta 256 caracteres diferentes, lo que era más que suficiente para el alfabeto, los dígitos, los signos de puntuación y los símbolos matemáticos comunes. Sin embargo, debido al desarrollo de la comunidad global y la subsiguiente internacionalización de Windows y el software para Windows, se necesitaba un nuevo método para identificar caracteres. Muchos idiomas utilizan más de 256 caracteres diferentes en su escritura, mucho más de lo que puede describirse con un byte. Por todo ello surgió Unicode. Un carácter Unicode ocupa 16 bits, lo que

## 8 ■ Capítulo I

permite identificar 65.535 caracteres diferentes. Para acomodar el nuevo conjunto de caracteres, muchas funciones del API de Windows se ofrecen en dos versiones diferentes: ANSI y Unicode. Cuando revise el fichero **Windows.PAS**, verá funciones definidas con una 'A' o 'W' añadidas al final del nombre de la función, identificando así las versiones ANSI o *Wide* (Unicode) de la función. En este libro nos centraremos en la descripción de las versiones ANSI de las funciones del API. Sin embargo, las funciones Unicode generalmente difieren únicamente en el tipo de las cadenas pasadas a la función, por lo que el texto de este libro describirá de forma adecuada el comportamiento de la versión Unicode de las funciones.

## Capítulo 2

# Funciones de la interfaz del dispositivo gráfico

## 2

## Capítulo

Las funciones de la Interfaz del Dispositivo Gráfico (*Graphics Device Interface - GDI*) forman el corazón del sistema de presentación gráfica de Windows 95/98 y Windows NT. Este sistema gráfico ofrece acceso a la pantalla, la impresora y el plotter a través de un amplio grupo de funciones del API.

Cualquier programa que interactúa con un dispositivo de salida tiene que hacerlo a través de dos niveles de abstracción: el núcleo del GDI de Windows y el controlador del dispositivo (*device driver*) suministrado por el fabricante. El *controlador de dispositivo* es una interfaz de *software* específica para un dispositivo concreto, y el GDI ofrece a las aplicaciones Windows la interfaz de acceso al controlador. El GDI está capacitado para conectarse a una variedad de dispositivos, incluso aquellos que no ofrecen las posibilidades sofisticadas de los dispositivos más avanzados. En algunas áreas el GDI debe tomar el control cuando el controlador no ofrece soporte de alto nivel para cierta operación.

El GDI puede soportar varios tipos de dispositivos simultáneamente, al tiempo que mantiene una interfaz consistente con las aplicaciones. El GDI, por lo tanto, tiene que ser capaz de manejar un grupo de controladores de dispositivos con diferentes capacidades, y relacionarse con ellos de acuerdo a la funcionalidad que ofrece cada uno. Y tiene que hacer esto al mismo tiempo que presenta a la aplicación un conjunto consistente de funciones del API, lo que libera al programador de la necesidad de tratar directamente con el dispositivo.

Las funciones del GDI operan a diferentes niveles, dependiendo de lo específica que necesite ser la aplicación con respecto a la salida gráfica. A bajo nivel, una aplicación puede manipular una imagen píxel a píxel. A un nivel más alto, la aplicación puede emitir comandos tales como dibujar elipses u otras primitivas gráficas, de forma totalmente independiente del dispositivo. Los comandos del GDI son inicialmente procesados por el componente del núcleo de Windows *GDI.EXE* a nivel de sistema y luego pasados al controlador del dispositivo correspondiente, para que éste produzca la salida gráfica de acuerdo con sus capacidades. Delphi encapsula la mayoría de las

funciones del GDI en el objeto *TCanvas*. No obstante, a veces es necesario hacer uso de las funciones de bajo nivel del GDI para llevar a cabo tareas tales como dibujar en áreas no clientes de ventanas o cambiar el modo de mapeado de una ventana. Este capítulo describe las funciones que se utilizan para manipular contextos de dispositivos y modificar coordenadas del sistema.

## Independencia del dispositivo

Windows soporta una amplia variedad de dispositivos de salida con diferentes capacidades. La tarea del GDI consiste en ser capaz de entender el nivel de soporte que un controlador de dispositivo ofrece y emitir comandos al controlador basándose en esas capacidades.

Si una aplicación emite un comando para dibujar una elipse en un dispositivo, el GDI determinará si ese dispositivo está en condiciones de recibir un comando de alto nivel para dibujar una elipse. Si lo está, el GDI suministrará al controlador el conjunto de comandos más eficientes para que la imagen pueda ser dibujada bajo su control. Sin embargo, si el controlador no posee tales capacidades, el GDI tiene que asumir la responsabilidad del dibujo y emitir comandos de bajo nivel, quizás píxel por píxel, para lograr el mismo resultado.

Independientemente del controlador que soporta el dispositivo de salida, el GDI brinda las capacidades de alto nivel a la aplicación. El programador generalmente no tiene que preocuparse de escribir código para formatear una imagen para diversos dispositivos de salida. Esa tarea corresponde al GDI. El API Win32 ofrece una gran cantidad de comandos de alto y bajo nivel para presentar salidas gráficas.

El programador puede generalmente decidir lo independiente que será una aplicación de los dispositivos. Las funciones del API que referencian píxeles de *hardware* no son independientes del dispositivo, porque dispositivos de diferente resolución mostrarán las imágenes de acuerdo a sus propias capacidades. No habría escalamiento automático para las diferentes resoluciones. Las funciones que operan en términos de medidas lógicas en lugar de píxeles son más independientes de los dispositivos. El sistema Win32 GDI ejecuta muchas tareas internas que mapean las llamadas al API en comandos específicos del dispositivo, lo que garantiza a la aplicación un cierto nivel de independencia del *hardware*.

## Contextos de dispositivo

Windows almacena un registro interno asociado a las imágenes que son visualizables o imprimibles. Este registro interno es conocido como *contexto de dispositivo* (*device context*) y contiene información acerca de cómo la imagen será mostrada. Las funciones del GDI necesitan un manejador de contexto de dispositivo porque este registro contiene la información sobre los atributos de presentación, el sistema de coordenadas, el recorte, objetos gráficos y modos de visualización. Los objetos gráficos pueden incluir plumas, brochas, mapas de bits, paletas, regiones y rutas. Una aplicación

no accede a la información del registro de contexto de dispositivo directamente. La información de un contexto de dispositivo es obtenida y manipulada usando llamadas del API. Existen funciones para obtener o crear un contexto de dispositivo, para asignar u obtener los valores de sus atributos y para liberar un contexto de dispositivo.

Las funciones *GetDC* y *GetWindowDC* recuperan manejadores de contextos de dispositivo que representan la superficie visualizable de una ventana. Estos contextos de dispositivo pueden ser usados en subsiguientes funciones de dibujo para producir salidas directamente en la ventana o el formulario. La función *CreateCompatibleDC* permite crear un contexto de dispositivo asociado a la memoria interna del ordenador. Esto le permite a la aplicación preparar imágenes fuera de la pantalla para luego copiarlas rápidamente a la superficie de una ventana.

### Tipos de Contextos de Dispositivo

Un contexto de dispositivo puede ser de tres tipos: común, de clase, o privado. Cuando el contexto de dispositivo se refiere a la visualización, puede llamarse contexto de visualización (*display context*) y permitirá referirse a un área específica de la pantalla, como por ejemplo una ventana, al área cliente de una ventana o a la pantalla completa.

Los tipos de contextos de visualización son creados sobre la base de las opciones de clase que se especifican para la ventana, cuando la ventana es registrada. La siguiente tabla describe el tipo de contexto de visualización devuelto por una llamada a la función *GetDC*, que depende del estilo de la clase bajo el que ha sido registrada la ventana.

**Tabla 2-1: Opciones de clase y contextos de visualización**

Valor	Descripción
ninguno	Un nuevo contexto de dispositivo tiene que obtenerse cada vez que se necesite. El nuevo contexto es creado con valores por defecto. El área de recorte por defecto es el área cliente de la ventana. Este es un contexto de dispositivo común y es reservado en el espacio de la pila de la aplicación. No hay límites prácticos al número de contextos de dispositivo comunes que pueden ser creados, salvo las limitaciones de la memoria disponible, pero es una buena práctica liberar la memoria asociada a un contexto de dispositivo común llamando a la función <i>ReleaseDC</i> cuando éste deje de ser necesario.
CS_CLASSDC	Un único contexto de visualización es compartido por todas las ventanas de la misma clase. Los cambios hechos a un contexto de visualización afectarán a todas las ventanas que pertenezcan a esa clase. El uso de contextos de dispositivo de clase no se recomienda.

## 12 ■ Capítulo 2

### CS\_OWNDNC

Un contexto de dispositivo es creado para cada ventana de la clase, por lo que cada una tendrá su propio contexto de dispositivo. Los atributos de ese contexto de dispositivo son persistentes. Después que la función `GetDC` sea llamada por primera vez, cualquier cambio hecho al contexto de dispositivo estará presente cuando sea recuperado de nuevo. Es innecesario llamar a la función `ReleaseDC` para los contextos de dispositivo privados. Esto produce un aumento de la velocidad de ejecución con un coste de 800 bytes adicionales de memoria por cada ventana.

---

En general, si la aplicación ejecuta pocas operaciones de salida gráfica, los contextos de dispositivo comunes ofrecerán toda la funcionalidad necesaria. Sin embargo, para aplicaciones que generen abundantes gráficos y escriban continuamente en pantalla, es aconsejable crear una ventana con un contexto de dispositivo privado. El siguiente ejemplo ilustra cómo usar el estilo de clase `CS_OWNDNC` para crear una ventana con un contexto de dispositivo privado.

#### Listado 2-1: Creación de una ventana con un contexto de dispositivo privado

##### var

```
Form1: TForm1;  
NewBrush,           // manejador de una nueva brocha  
OldBrush: HBRUSH;   // manejador de la brocha anterior
```

##### implementation

```
{ $R *.DFM }
```

```
procedure TForm1.CreateParams(var Params: TCreateParams);
```

##### begin

```
{Inicializar el parámetro con los valores por defecto}
```

```
inherited CreateParams(Params);
```

```
{Indicar que esta ventana debe tener su propio contexto de dispositivo.
```

```
Comente esta línea para observar los efectos}
```

```
Params.WindowClass.Style := Params.WindowClass.Style or CS_OWNDNC;
```

##### end;

```
procedure TForm1.FormActivate(Sender: TObject);
```

##### var

```
TempDC: HDC;           // manejador temporal de un contexto de dispositivo
```

##### begin

```
{Recuperar un manejador del contexto de dispositivo privado para esta ventana}
```

```
TempDC := GetDC(Form1.Handle);
```

```
{Crear una nueva brocha y seleccionarla dentro del contexto de dispositivo}
```

```
NewBrush := CreateHatchBrush(HS_DIAGCROSS, clRed);
```

```
OldBrush := SelectObject(TempDC, NewBrush);
```

```
{Liberar el contexto de dispositivo. Observe que como estamos tratando con un
```

```

    contexto de dispositivo privado, la nueva brocha permanecerá dentro del contexto
    de dispositivo.}
    ReleaseDC(Form1.Handle, TempDC);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    TempDC: HDC;           // un manejador temporal de un contexto de dispositivo
begin
    {Recuperar un manejador al contexto de dispositivo privado}
    TempDC := GetDC(Form1.Handle);

    {Dibujar un rectángulo. Observe que no estamos creando una nueva brocha, por lo
    que el rectángulo será rellenado con la brocha por defecto seleccionada en el
    contexto de dispositivo. Como se trata de un contexto de dispositivo privado,
    usará la brocha anteriormente seleccionada para rellenar el rectángulo}
    Rectangle(TempDC, 0, 0, ClientWidth, ClientHeight);

    {Liberar el contexto de dispositivo}
    ReleaseDC(Form1.Handle, TempDC);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Borrar la brocha}
    SelectObject(GetDC(Form1.Handle), OldBrush);
    DeleteObject(NewBrush);
end;

```

## Contextos de dispositivo de pantalla, ventana y área cliente

Por defecto todo contexto de visualización se asocia con el área cliente de una ventana. Es ahí donde normalmente se dibuja. Es también posible obtener un contexto de dispositivo para la ventana completa o para la pantalla.

Un contexto de dispositivo para una ventana se obtiene usando la función *GetWindowDC*. Esta función recupera un contexto de dispositivo para la ventana que incluye el área no cliente (bordes, barra de título, etc). El contexto de dispositivo es siempre un contexto de dispositivo común y no comparte propiedades con otros contextos de dispositivo, independientemente de su tipo. Cualquier cambio hecho al contexto de dispositivo recuperado desde la función *GetWindowDC* no afectará a contextos de dispositivo privados.

Un contexto de dispositivo para la pantalla completa puede recuperarse llamando a la función *GetDC*, pasando un valor cero como manejador de la ventana. Esto permite a una aplicación dibujar directamente en la pantalla, escribiendo sobre ventanas y otros gráficos. El uso de esta técnica viola las reglas generales de programación para Windows y no es recomendable.

## Sistemas de Coordenadas

Windows ofrece varios sistemas de coordenadas diferentes para producir salida gráfica en la pantalla o impresoras. El sistema de coordenadas puede estar basado en unidades del dispositivo, como los píxeles, o puede estarlo en un sistema de medidas lógicas, donde una unidad lógica puede ser convertida en uno o más píxeles según el modo en que estas unidades sean mapeadas. Un sistema *lógico* o *relativo* de coordenadas permitirá a una aplicación tener un conjunto común de comandos que producirán los mismos efectos en diferentes dispositivos, aún cuando éstos tengan diferentes propiedades.

La salida gráfica es ejecutada sobre una pantalla o impresora que tiene sus propias unidades de medida. Los dispositivos de salida operan en píxeles. El píxel es la unidad de medida del dispositivo de salida. Por eso, a veces se le llama la unidad del dispositivo (*device unit*). El sistema de coordenadas en el dispositivo mide las distancias en píxeles (unidades del dispositivo). Muchas de las funciones gráficas del API usan las unidades del dispositivo como referencia.

La ubicación de los píxeles en la pantalla o la impresora es generalmente relativa al origen en la esquina superior izquierda del papel, la pantalla, la ventana o el área cliente de una ventana. En la mayoría de los casos, los valores se incrementan en la medida en que el punto se desplaza a la derecha o hacia abajo a partir del origen. El sistema de coordenadas del dispositivo mide las distancias en píxeles reales, de manera que si el dispositivo tiene 100 píxeles por pulgada, un punto que se encuentre una pulgada debajo del origen tendrá 100 como coordenada vertical.

El GDI ofrece algunas funciones de alto nivel que utilizan el sistema de coordenadas lógicas. Tal sistema puede ofrecer a la aplicación una representación lógica del área de dibujo que posee unas medidas independientes de la resolución del dispositivo. Las funciones del GDI que usan coordenadas lógicas efectuarán una conversión de dichas coordenadas y emitirán comandos al controlador del dispositivo en el sistema de coordenadas de éste. Esta conversión de medidas entre coordenadas lógicas y coordenadas del dispositivo es soportado por un amplio conjunto de funciones del API.

Algunas funciones del GDI soportan coordenadas lógicas de alto nivel y otras sólo operaciones con píxeles. Las funciones del GDI que utilizan coordenadas lógicas son generalmente comandos de dibujo que no ofrecen posibilidades de mapeado, y que parecen, en efecto, ser aplicados directamente a medidas en píxeles. Para obtener un mapeado de coordenadas lógicas independiente del dispositivo, la aplicación tiene que establecer un modo de mapeado y utilizar las capacidades de desplazamiento y escalamiento que se describen a continuación.

Cada contexto de dispositivo mantiene un registro que contiene la información necesaria para mapear las coordenadas lógicas a coordenadas del dispositivo. El contexto de dispositivo para una pantalla conoce las características del *hardware* lo suficientemente bien como para soportar las llamadas del GDI que ejecutan el mapeado de coordenadas. La conversión es ejecutada al nivel de contexto de dispositivo, y cada

creación de un contexto de dispositivo deberá establecer su modo de mapeado para ser capaz de convertir coordenadas lógicas a coordenadas de visualización.

### **Mapeado de coordenadas lógicas a coordenadas de dispositivo**

El mapeado de coordenadas lógicas a coordenadas de visualización abarca varias funciones del GDI. Hay un desplazamiento a partir del origen que puede ser aplicado a cada referencia en coordenadas lógicas, al que se denomina ventana (*window*) y un desplazamiento que puede ser aplicado a las referencias en coordenadas del dispositivo que se conoce como *puerto de visualización* (*viewport*). Igualmente existen factores de escala que pueden ser aplicados a la ventana y al puerto de visualización. A cualquier punto que se especifique en coordenadas lógicas puede aplicársele una transformación matemática si el modo de mapeado que está en uso en el contexto de pantalla lo permite. El modo *MM\_TEXT* es el modo de mapeado por defecto, y ofrece un mapeado 1:1 de las coordenadas lógicas a las del dispositivo.

Los cálculos para la transformación de las coordenadas tienen lugar de manera independiente para los componentes horizontales y verticales, y consisten en las siguientes operaciones matemáticas:

$$xViewport = (xWindow - xWindowOrg) * (xViewportExt/xWindowExt) + xViewportOrg$$

$$yViewport = (yWindow - yWindowOrg) * (yViewportExt/yWindowExt) + yViewportOrg$$

A las coordenadas lógicas se les aplica el desplazamiento lógico, luego se multiplican por la relación entre los factores de escala físico y lógico, y por último se les aplica el desplazamiento del dispositivo. Sería más simple aplicar el factor de escala y desplazamiento sólo al sistema lógico o al sistema de dispositivo, pero éstos son números enteros, no valores de coma flotante. Asimismo, estas transformaciones han sido diseñadas pensando en el soporte simultáneo para varios dispositivos, donde la aplicación del desplazamiento y escalamiento a las coordenadas lógicas ("ventana") sean aplicables a todos los dispositivos, mientras que las aplicaciones al dispositivo ("puerto de visualización") puedan ser programadas para que se apliquen a diferentes dispositivos de una manera individualizada.

Las funciones "*org*" tienen que ver con la adición o sustracción de valores a las coordenadas cuando éstas cambian del contexto lógico al de dispositivo. Las funciones "*ext*" tienen que ver con el escalamiento de las coordenadas, aumentadas o reducidas, cuando cambian del contexto lógico al de dispositivo. Cada uno de estos conceptos tiene una propiedad tanto en el contexto lógico como en el de dispositivo. Por ejemplo, la función *SetWindowOrgEx* establece el valor que es sustraído a una coordenada cuando ésta abandona el contexto de coordenadas lógicas. Por su parte, la función *SetViewportOrgEx* establece el valor que se añade a la coordenada cuando ésta llega al sistema de coordenadas de dispositivo. *SetWindowExtEx* establece un factor entre el cual es dividida la coordenada cuando ésta deja el sistema de coordenadas lógicas, y

*SetViewportExtEx* establece el valor que se multiplica por la coordenada cuando ésta llega al sistema de coordenadas del dispositivo. Este es el comportamiento expresado en las fórmulas matemáticas mostradas anteriormente.

## Modos de mapeado

El modo de mapeado por defecto en el contexto de un dispositivo es *MM\_TEXT*, que consiste en una conversión 1:1, razón por la cual ninguna transformación tiene lugar. Las funciones del GDI cuyos parámetros se expresan en unidades lógicas están también en unidades del dispositivo cuando el modo *MM\_TEXT* está activo. Para que una aplicación utilice verdaderamente coordenadas lógicas hay que cambiar el modo de mapeado. La función *SetMapMode* se usa para este propósito. El modo *MM\_ANISOTROPIC* ofrece una completa flexibilidad para programar el origen del desplazamiento y escalamiento. Con este modo (y otros, excepto *MM\_TEXT* y *MM\_ISOTROPIC*) es posible escalar factores horizontales y verticales de forma diferenciada, lo que permite generar imágenes oblicuas. La unidad básica de medida parte del píxel a nivel de dispositivo, con el escalamiento y desplazamiento definidos por el programador. *MM\_ISOTROPIC* es similar a *MM\_ANISOTROPIC*, con la excepción de que se asegura que el escalamiento horizontal y vertical sean los mismos. El resto de modos de mapeado tienen factores de escala iniciales implícitos, basados en la resolución del dispositivo, y son inicializados para unidades de medida específicas de las coordenadas lógicas. Una aplicación puede ubicar coordenadas en un área de dibujo basándose en las medidas físicas actuales, dejándole al GDI el trabajo de calcular cuántos píxeles serán necesarios para producir tal medida.

El Listado 2-2 ilustra cómo mover el origen y aplicar un factor de escala a la transformación de coordenadas. Permite al usuario seleccionar un modo de mapeado, aplicar un desplazamiento al sistema lógico (ventana) y al sistema de dispositivo (puerto de visualización) y factores de escala a los sistemas lógico y del dispositivo. El ejemplo muestra los valores “org” y “ext” que se encuentran en uso. El valor *org* puede ser modificado con las funciones *SetWindowOrgEx*, *SetViewportOrgEx*, *OffsetWindowOrgEx* y *OffsetViewportOrgEx*. El valor de escalado y la extensión “ext” pueden ser modificados mediante las funciones *SetWindowExtEx*, *SetViewportExtEx*, *ScaleWindowExtEx* y *ScaleViewportExtEx*.

**Listado 2-2: Modificando las dimensiones y origen del puerto de visualización y la ventana**

```
var
  Form1: TForm1;
  WOrigin: TPoint;      // almacena el origen de la ventana
  VOrigin: TPoint;      // almacena el origen del puerto de visualización
  WExt: TPoint;         // almacena la dimensión de la ventana
  VExt: TPoint;         // almacena la dimensión del puerto de visualización
  MyDisplayDC: HDC;     // almacena el contexto de dispositivo
  MyMapMode: Integer;   // almacena el modo de mapeado
implementation
{$R *.DFM}
```

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Seleccionar la escala del origen del selector}
    TrackbarSWOX.Max := Panel1.Width;
    TrackbarSWOY.Max := Panel1.Height;
    TrackbarSVOX.Max := Panel1.Width;
    TrackbarSVOY.Max := Panel1.Height;

    {Inicializar el selector a su punto medio}
    TrackbarSWOX.Position := TrackbarSWOY.Max div 2;
    TrackbarSWOY.Position := TrackbarSWOX.Max div 2;
    TrackbarSVOX.Position := TrackbarSVOY.Max div 2;
    TrackbarSVOY.Position := TrackbarSVOX.Max div 2;
    TrackbarSWEX.Position := TrackbarSWEY.Max div 2;
    TrackbarSWEY.Position := TrackbarSWEX.Max div 2;
    TrackbarSVEX.Position := TrackbarSVEY.Max div 2;
    TrackbarSVEY.Position := TrackbarSVEX.Max div 2;
end;

procedure TForm1.ReportPosition;
var
    ReturnValue: Tpoint;           // almacena orígenes de la ventana y del puerto de
                                // visualización
    ReturnSize: Tsize;           // almacena dimensiones de la ventana y del puerto
                                // de visualización
    ReadMapMode: Integer;        // almacena el modo de mapeado
    ReadFinalOrigin: TPoint;     // almacena el origen del dispositivo
begin
    {Mostrar el origen de la ventana}
    GetWindowOrgEx(MyDisplayDC, ReturnValue);
    Label9.Caption := IntToStr(ReturnValue.x)
        + ', ' + IntToStr(ReturnValue.y);

    {Mostrar el origen del puerto de visualización}
    GetViewportOrgEx(MyDisplayDC, ReturnValue);
    Label10.Caption := IntToStr(ReturnValue.x)
        + ', ' + IntToStr(ReturnValue.y);

    {Mostrar la dimensión de la ventana}
    GetWindowExtEx(MyDisplayDC, ReturnSize);
    Label11.Caption := IntToStr(ReturnSize.cx)
        + ', ' + IntToStr(ReturnSize.cy);

    {Mostrar la dimensión del puerto de visualización}
    GetViewportExtEx(MyDisplayDC, ReturnSize);
    Label12.Caption := IntToStr(ReturnSize.cx)
        + ', ' + IntToStr(ReturnSize.cy);

    {Mostrar el modo de mapeado actual}
    ReadMapMode := GetMapMode(MyDisplayDC);
case ReadMapMode of
        MM_TEXT:      LabelGMMresult.Caption := 'MM_TEXT';
        MM_ANISOTROPIC: LabelGMMresult.Caption := 'MM_ANISOTROPIC';
        MM_ISOTROPIC:  LabelGMMresult.Caption := 'MM_ISOTROPIC';
    
```

## 18 ■ Capítulo 2

```
MM_HIENGLISH: LabelGMMresult.Caption := 'MM_HIENGLISH';
MM_HIMETRIC:  LabelGMMresult.Caption := 'MM_HIMETRIC';
MM_LOENGLISH: LabelGMMresult.Caption := 'MM_LOENGLISH';
MM_LOMETRIC:  LabelGMMresult.Caption := 'MM_LOMETRIC';
MM_TWIPS:     LabelGMMresult.Caption := 'MM_TWIPS';
end;

{Mostrar la conversión final del origen para el contexto de dispositivo}
GetDCOrgEx(MyDisplayDC, ReadFinalOrigin);
LabelGetDCOrgExResult.Caption := IntToStr(ReadFinalOrigin.X) + ', ' +
                                IntToStr(ReadFinalOrigin.Y);
end;

procedure TForm1.ReadUserRequest;
begin
    {Recuperar el modo de mapeado actual}
    case RadioGroup1.ItemIndex of
        0: MyMapMode := MM_TEXT;
        1: MyMapMode := MM_ANISOTROPIC;
        2: MyMapMode := MM_ISOTROPIC;
        3: MyMapMode := MM_HIENGLISH;
        4: MyMapMode := MM_HIMETRIC;
        5: MyMapMode := MM_LOENGLISH;
        6: MyMapMode := MM_LOMETRIC;
        7: MyMapMode := MM_TWIPS;
    end;

    {Seleccionar los orígenes y dimensiones de acuerdo a la posición del selector}
    WOrigin.x := TrackBarSWOX.Position;
    WOrigin.y := TrackBarSWOY.Position;
    VOrigin.x := TrackBarSVOX.Position;
    VOrigin.y := TrackBarSVOY.Position;
    WExt.x    := TrackBarSWEX.Position;
    WExt.y    := TrackBarSWEY.Position;
    VExt.x    := TrackBarSVEX.Position;
    VExt.y    := TrackBarSVEY.Position;
end;

procedure TForm1.PaintImage;
begin
    {Recuperar un contexto de dispositivo para el panel}
    MyDisplayDC := GetDC(Pane11.Handle);

    {Borrar la imagen actual}
    Pane11.Repaint;

    {Recuperar los valores definidos por el usuario}
    ReadUserRequest;

    {Asignar los valores seleccionados para el modo de mapeado}
    SetMapMode(MyDisplayDC, MyMapMode);
    if Checkbox1.Checked
        then SetWindowOrgEx(MyDisplayDC, WOrigin.x, WOrigin.y, nil);
    if Checkbox2.Checked
```

```

    then SetViewportOrgEx(MyDisplayDC, VOrigin.x, VOrigin.y, nil);
if Checkbox3.Checked
    then SetWindowExtEx(MyDisplayDC, WExt.x, WExt.y, nil);
if Checkbox4.Checked
    then SetViewportExtEx(MyDisplayDC, VExt.x, VExt.y, nil);

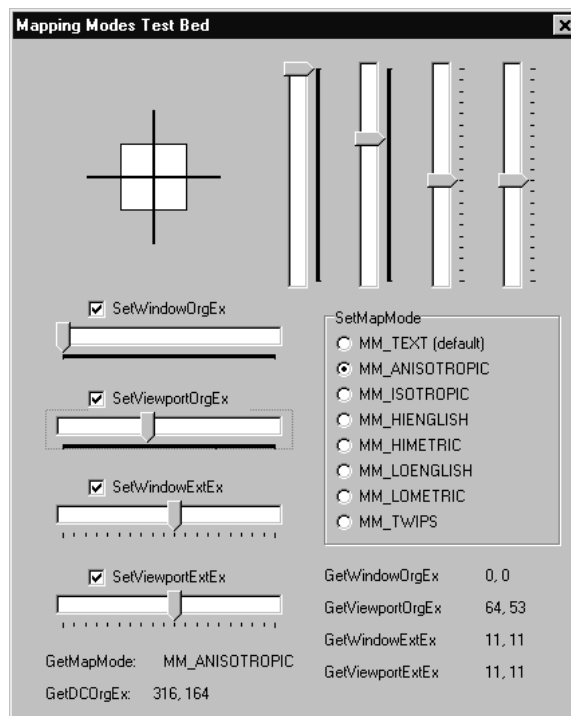
{Dibujar la imagen. Observe que la imagen es dibujada en las mismas coordenadas
fijas. Esto demuestra cómo puede afectar el cambiar el origen, la dimensión de
la ventana y el puerto de visualización al objeto dibujado}
Windows.Rectangle(MyDisplayDC, 0, 0, 50, 50);
Windows.Rectangle(MyDisplayDC, -25, 24, 75, 26);
Windows.Rectangle(MyDisplayDC, 24, -25, 26, 75);
{Mostrar los valores actuales}
ReportPosition;

{Liberar el contexto de dispositivo}
ReleaseDC(Pane11.Handle, MyDisplayDC);
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Mostrar la imagen}
    PaintImage;
end;

```

Figura 2-1: El programa de prueba de la ventana y el puerto de visualización en acción



### Problemas con el mapeado de coordenadas lógicas

Una aplicación que no esté ejecutando de manera adecuada la conversión de coordenadas lógicas a las de dispositivo puede presentar alguno de los siguientes problemas:

1. El contexto de dispositivo puede tener un modo de mapeado que no soporte la conversión necesaria. El modo de mapeado por defecto *MM\_TEXT* debe ser cambiado si alguna transformación va a realizarse.
2. Las coordenadas pueden estar fuera de rango. En la medida de lo posible mantenga las coordenadas lógicas y del dispositivo dentro de valores de 16 bits. La transformación soporta hasta un tamaño de 27 bits, pero algunas funciones de visualización sólo soportan coordenadas de 16 bits.
3. La imagen pudo ser recortada (del área de visualización) o no ser visible por ser demasiado pequeña o grande.
4. El escalamiento puede no ser el esperado porque la aplicación esté asignando el mismo valor para las dimensiones de la ventana y el puerto de visualización, lo que produce un factor de escala igual a uno (sin efecto).
5. El factor de escala puede no estar produciendo efecto porque la aplicación está multiplicando y dividiendo por el mismo número. Para ampliar el factor de escala efectivo, trate de asignar sólo el parámetro de multiplicación o el de división, o asegúrese de que sean diferentes.
6. El contexto de dispositivo puede no ser válido. Compruebe si los valores que devuelven las funciones del GDI indican códigos de error.

## Funciones de la interfaz gráfica

Este capítulo describe las siguientes funciones de la interfaz gráfica:

**Tabla 2-2: Funciones de la interfaz del dispositivo gráfico**

Función	Descripción
ChangeDisplaySettings	Cambia el modo de visualización.
ClientToScreen	Convierte coordenadas del cliente a coordenadas de pantalla.
CreateCompatibleDC	Crea un contexto de dispositivo en memoria.
DeleteDC	Elimina un contexto de dispositivo.
DPTtoLP	Convierte puntos del dispositivo a puntos lógicos.
EnumDisplaySettings	Enumera los modos disponibles de visualización.
GetDC	Recupera un manejador de un contexto de dispositivo.
GetDCOrgEx	Recupera la conversión final del origen del contexto de dispositivo indicado.
GetDeviceCaps	Recupera las capacidades del dispositivo.
GetMapMode	Recupera el modo de mapeado actual.

Función	Descripción
GetSystemMetrics	Recupera las medidas de los elementos del sistema.
GetViewportExtEx	Recupera la extensión del puerto de visualización.
GetViewportOrgEx	Recupera el origen del puerto de visualización.
GetWindowDC	Recupera un manejador de un contexto de dispositivo de ventana.
GetWindowExtEx	Recupera la extensión de una ventana.
GetWindowOrgEx	Recupera el origen de una ventana.
LPtoDP	Convierte puntos lógicos a puntos del dispositivo.
MapWindowPoints	Convierte múltiples coordenadas del sistema de coordenadas de una ventana a otro.
OffsetViewportOrgEx	Desplaza el origen del puerto de visualización.
OffsetWindowOrgEx	Desplaza el origen de una ventana.
ReleaseDC	Libera un contexto de dispositivo.
RestoreDC	Restaura el estado almacenado de un contexto de dispositivo.
SaveDC	Almacena el estado de un contexto de dispositivo.
ScaleViewportExtEx	Escala la extensión del puerto de visualización.
ScaleWindowExtEx	Escala la extensión de la ventana.
ScreenToClient	Convierte coordenadas de pantalla en coordenadas de área cliente.
ScrollDC	Desplaza un área de un contexto de dispositivo.
SetMapMode	Establece el modo de mapeado.
SetViewportExtEx	Establece la extensión del puerto de visualización.
SetViewportOrgEx	Establece el origen del puerto de visualización.
SetWindowExtEx	Establece la extensión de la ventana.
SetWindowOrgEx	Establece el origen de la ventana.

## ChangeDisplaySettings Windows.Pas

### Sintaxis

```
ChangeDisplaySettings(
    lpDevMode: PDeviceMode;    {puntero a registro TDeviceMode}
    dwFlags: DWORD              {opciones de cambio}
): Longint;                    {devuelve el código resultante}
```

### Descripción

Esta función cambia el modo gráfico del sistema de visualización. Los nuevos valores del dispositivo están contenidos en el registro *TDeviceMode* pasado como primer parámetro a la función. Es común hacer una llamada a *EnumDisplaySettings* antes de llamar a *ChangeDisplaySettings* para obtener un registro *TDeviceMode* válido. Esto

ayuda a asegurar que la función *ChangeDisplaySettings* tendrá parámetros que son compatibles con el controlador de visualización instalado.

Como consecuencia de la llamada a esta función, un mensaje *WM\_DISPLAYCHANGE* es enviado a todas las aplicaciones como notificación de que los valores de visualización fueron modificados. Esto es ejecutado automáticamente por Windows.

### Parámetros

*lpDevMode*: Un puntero al registro *TDeviceMode* que contiene la información a utilizar para inicializar el nuevo modo gráfico. Si a este parámetro se le asigna **nil**, el valor del modo de visualización actualmente almacenado en el registro será usado para el nuevo modo de visualización. De todos los campos del registro *TDeviceMode* sólo *dmSize*, *dmBitsPerPel*, *dmFields*, *dmPelsWidth*, *dmPelsHeight*, *dmDisplayFlags*, y *dmDisplayFrequency* son usados por esta función. El registro *TDeviceMode* se define de la siguiente forma:

*TDeviceMode* = **packed record**

<i>dmDeviceName</i> : <b>array</b> [0.. <i>CCHDEVICENAME</i> - 1]	
<b>of</b> AnsiChar;	{no se utiliza}
<i>dmSpecVersion</i> : Word;	{no se utiliza}
<i>dmDriverVersion</i> : Word;	{no se utiliza}
<i>dmSize</i> : Word;	{tamaño del registro}
<i>dmDriverExtra</i> : Word;	{no se utiliza}
<i>dmFields</i> : DWORD;	{campos válidos}
<i>dmOrientation</i> : SHORT;	{no se utiliza}
<i>dmPaperSize</i> : SHORT;	{no se utiliza}
<i>dmPaperLength</i> : SHORT;	{no se utiliza}
<i>dmPaperWidth</i> : SHORT;	{no se utiliza}
<i>dmScale</i> : SHORT;	{no se utiliza}
<i>dmCopies</i> : SHORT;	{no se utiliza}
<i>dmDefaultSource</i> : SHORT;	{no se utiliza}
<i>dmPrintQuality</i> : SHORT;	{no se utiliza}
<i>dmColor</i> : SHORT;	{no se utiliza}
<i>dmDuplex</i> : SHORT;	{no se utiliza}
<i>dmYResolution</i> : SHORT;	{no se utiliza}
<i>dmTTOption</i> : SHORT;	{no se utiliza}
<i>dmCollate</i> : SHORT;	{no se utiliza}
<i>dmFormName</i> : <b>array</b> [0.. <i>CCHFORMNAME</i> - 1]	
<b>of</b> AnsiChar;	{no se utiliza}
<i>dmLogPixels</i> : Word;	{no se utiliza}
<i>dmBitsPerPel</i> : DWORD;	{profundidad de color}
<i>dmPelsWidth</i> : DWORD;	{ancho de pantalla}
<i>dmPelsHeight</i> : DWORD;	{alto de pantalla}
<i>dmDisplayFlags</i> : DWORD;	{modo de visualización}
<i>dmDisplayFrequency</i> : DWORD;	{frecuencia}

dmICMMethod: DWORD;	{no se utiliza}
dmICMIntent: DWORD;	{no se utiliza}
dmMediaType: DWORD;	{no se utiliza}
dmDitherType: DWORD;	{no se utiliza}
dmICCManufacturer: DWORD;	{no se utiliza}
dmICCModel: DWORD;	{no se utiliza}
dmPanningWidth: DWORD;	{no se utiliza}
dmPanningHeight: DWORD;	{no se utiliza}

**end;**

Sólo los siguientes campos del registro son usados por esta función:

*dmSize*: Especifica el tamaño del registro *TDeviceMode*. Debe asignársele *SizeOf(TDeviceMode)*.

*dmFields*: Un conjunto de opciones que indican qué otros campos del registro contienen información válida. A este campo puede asignársele uno o más valores de la Tabla 2-3.

*dmBitsPerPel*: Indica el número de bits requeridos para describir el color de un píxel (por ejemplo, 4 bits para visualizar 16 colores, 8 bits para 256, etc.).

*dmPelsWidth*: Especifica el ancho de la pantalla, en píxeles.

*dmPelsHeight*: Especifica la altura de la pantalla, en píxeles.

*dmDisplayFlags*: Una opción que indica el modo de visualización. A este campo se le puede asignar uno de los valores de la Tabla 2-4.

*dmDisplayFrequency*: Especifica la frecuencia de refresco vertical, en Hertzios. Un valor cero o uno representa la frecuencia de refresco por defecto del *hardware*.

*dwFlags*: Una opción que especifica cómo será modificado el modo gráfico. A este parámetro puede asignársele un valor de la Tabla 2-5. Si la opción *CDS\_UPDATEREGISTRY* es especificada, el sistema trata de hacer un cambio dinámico del modo gráfico y actualizar el registro sin reiniciar. Si es necesario reiniciar, la función devuelve el valor *DISP\_CHANGE\_RESTART* y la aplicación es responsable de reiniciar Windows. El modo *CDS\_TEST* puede ser usado para determinar qué modos gráficos están disponibles sin ejecutar los cambios.

#### Valor que devuelve

Esta función devuelve una opción que indica el éxito o fracaso y puede ser uno de los valores de la Tabla 2-6.

#### Véase además

*EnumDisplaySettings*, *WM\_DISPLAYCHANGE*

*Ejemplo***Listado 2-3: Cambio del modo de visualización**

```
{¡OJO! Delphi importa esta función incorrectamente, debemos hacerlo manualmente}
function ChangeDisplaySettings(lpDevMode: PDeviceMode;
                               dwFlags: DWORD): Longint; stdcall;

var
    Form1: TForm1;
    DevModeArray: TList;    // almacena una lista de registros con
                           información de los modos de los dispositivos

implementation

uses Math;

{$R *.DFM}

{La función importada}
function ChangeDisplaySettings; external user32 name 'ChangeDisplaySettingsA';

procedure TForm1.FormCreate(Sender: TObject);
var
    DevModeCount: Integer;           // cantidad de modos gráficos
    DevModeInfo: ^TDevMode;         // puntero a info del modo gráfico
begin
    {Crear la lista que almacenará registros con la información de los modos}
    DevModeArray := TList.Create;

    {Inicializar el contador}
    DevModeCount := 0;

    {Reservar memoria dinámica para almacenar la información sobre un modo}
    GetMem(DevModeInfo, SizeOf(TDevMode));

    {Enumerar los modos de visualización}
    while EnumDisplaySettings(nil, DevModeCount, DevModeInfo^) do
    begin
        {Añadir la información a la lista}
        DevModeArray.Add(DevModeInfo);

        {Incrementar el contador}
        Inc(DevModeCount);

        {Mostrar la resolución de los modos de visualización enumerados}
        ListBox1.Items.Add(IntToStr(DevModeInfo^.dmPelsWidth) + 'x' +
                             IntToStr(DevModeInfo^.dmPelsHeight) + ', ' +
                             IntToStr(Trunc(IntPower(2, DevModeInfo^.dmBitsPerPel))) +
                             ' colors');

        {Reservar otro espacio para información sobre el siguiente modo}
        GetMem(DevModeInfo, SizeOf(TDevMode));
    end;
```

```

{El bucle anterior siempre reserva un bloque de memoria extra innecesario, por lo
tanto lo borramos}
FreeMem(DevModeInfo, SizeOf(TDevMode));

{Seleccionar el primer elemento en la lista}
ListBox1.ItemIndex := 0;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
    iCount: Integer;          // un contador general para bucle
begin
    {Liberar la memoria apuntada por cada elemento de la lista}
    for iCount := 0 to DevModeArray.Count-1 do
        FreeMem(DevModeArray.Items[iCount], SizeOf(TDevMode));

    {Liberar la lista}
    DevModeArray.Free;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    ModeChange: Longint;      // indica si es necesario reiniciar Windows
begin
    {Cambiar el modo de visualización}
    ModeChange := ChangeDisplaySettings(DevModeArray[ListBox1.ItemIndex],
                                        CDS_UPDATEREGISTRY);

    {Indicar si el cambio dinámico tuvo éxito o si es necesario reiniciar Windows}
    if ModeChange = DISP_CHANGE_SUCCESSFUL then
        ShowMessage('Dynamic display mode change successful.');
```

if ModeChange = DISP\_CHANGE\_RESTART then  
 ShowMessage('Change successful; Windows must be restarted for the changes ' +  
 'to take effect');

```

end;

```

**Tabla 2-3: Valores del campo lpDevMode.dmFields de ChangeDisplaySettings**

Valor	Descripción
DM_BITSPERPEL	El campo dmBitsPerPel contiene nuevos datos.
DM_PELSWIDTH	El campo dmPelsWidth contiene nuevos datos.
DM_PELSHEIGHT	El campo dmPelsHeight contiene nuevos datos.
DM_DISPLAYFLAGS	El campo dmDisplayFlags contiene nuevos datos.
DM_DISPLAYFREQUENCY	El campo dmDisplayFrequency contiene nuevos datos.

**Tabla 2-4: Valores del campo lpDevMode.dmDisplayFlags de ChangeDisplaySettings**

Valor	Descripción
DM_GRAYSCALE	Indica un monitor en blanco y negro.
DM_INTERLACED	Indica un monitor entrelazado.

Tabla 2-5: Valores del parámetro `dwFlags` de `ChangeDisplaySettings`

Valor	Descripción
0	El cambio será hecho dinámicamente.
<code>CDS_UPDATEREGISTRY</code>	El cambio es hecho dinámicamente y el registro será actualizado para reflejar el nuevo modo gráfico bajo la entrada USER.
<code>CDS_TEST</code>	El cambio no es realizado, pero se verifica el sistema para comprobar si soporta ese modo. La función devuelve el mismo valor que devolvería si el cambio fuera hecho.

Tabla 2-6: Valores que devuelve `ChangeDisplaySettings`

Valor	Descripción
<code>DISP_CHANGE_SUCCESSFUL</code>	La función tuvo éxito.
<code>DISP_CHANGE_RESTART</code>	Windows tiene que ser reiniciado para que los cambios tengan efecto.
<code>DISP_CHANGE_BADFLAGS</code>	Se pasaron valores de opción no válidos a la función.
<code>DISP_CHANGE_FAILED</code>	El controlador de visualización no acepta el nuevo modo gráfico.
<code>DISP_CHANGE_BADMODE</code>	El modo gráfico especificado no es soportado.
<code>DISP_CHANGE_NOTUPDATED</code>	Sólo Windows NT: Imposible escribir los nuevos valores al registro.

**ClientToScreen****Windows.Pas****Sintaxis**

```
ClientToScreen(
  hWnd: HWND;           {manejador de ventana}
  var lpPoint: TPoint    {puntero a un registro TPoint}
): BOOL;                {devuelve TRUE o FALSE}
```

**Descripción**

Esta función convierte las coordenadas de un punto (dadas en coordenadas de área cliente) a coordenadas de pantalla. El punto a ser convertido se recibe a través de un registro *TPoint* al que apunta el parámetro *lpPoint*. La función toma las coordenadas a las que apunta el parámetro *lpPoint* y las convierte en coordenadas de pantalla. El resultado es depositado en el mismo registro *TPoint*. Las coordenadas del punto toman como origen la esquina superior izquierda del área cliente de la ventana especificada. Las coordenadas resultantes toman como origen la esquina superior izquierda de la pantalla.

### Parámetros

*hWnd*: Manejador de la ventana que contiene el punto. La esquina superior izquierda del área cliente de esta ventana es el origen del sistema de coordenadas que define las coordenadas del punto a ser convertido.

*lpPoint*: Un puntero al registro *TPoint* que contiene el punto a ser convertido. Este registro *TPoint* recibe el punto convertido cuando la función retorna.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a *GetLastError*.

### Véase además

*MapWindowPoints*, *ScreenToClient*

### Ejemplo

#### Listado 2-4: Convirtiendo Coordenadas entre Sistemas de Coordenadas

```
procedure TForm1.Memo1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
var
    Coords: TPoint;    // almacena el punto que será convertido
begin
    {Mostrar las coordenadas señaladas relativas a la ventana hija}
    Label1.Caption := 'Memo Coordinates: ' + IntToStr(X) + ', ' + IntToStr(Y);

    {Convertir estas coordenadas a coordenadas de pantalla}
    Coords := Point(X, Y);
    Windows.ClientToScreen(Memo1.Handle, Coords);

    {Mostrar las coordenadas señaladas relativas a la pantalla}
    Label2.Caption := 'Screen Coordinates: ' + IntToStr(Coords.X) + ', ' +
        IntToStr(Coords.Y);

    {Convertir las coordenadas a coordenadas de la ventana cliente}
    Windows.ScreenToClient(Form1.Handle, Coords);

    {Mostrar las coordenadas señaladas relativas al área cliente de la ventana}
    Label3.Caption := 'Form Coordinates: ' + IntToStr(Coords.X) + ', ' +
        IntToStr(Coords.Y);
end;
```

Figura 2-2:  
Las  
coordenadas  
convertidas



## **CreateCompatibleDC**      **Windows.Pas**

### **Sintaxis**

```
CreateCompatibleDC(
    DC: HDC                                {manejador de contexto de dispositivo}
); HDC;                                   {devuelve un manejador de un contexto de dispositivo
                                         en memoria}
```

### **Descripción**

Esta función crea un contexto de dispositivo en memoria que es compatible con el contexto de dispositivo especificado. Esto puede utilizarse para preparar imágenes que luego serán copiadas a la pantalla o a la impresora. Un mapa de bits tiene que ser seleccionado en el contexto de dispositivo devuelto por esta función, antes de que el contexto de dispositivo pueda ser usado en operaciones de dibujo. Cuando una aplicación termina de utilizar un contexto de dispositivo en memoria deberá eliminarlo llamando a la función *DeleteDC*.

### **Parámetros**

*DC*: Especifica un manejador de contexto de dispositivo con el cual el nuevo contexto de dispositivo será compatible. Este contexto deberá soportar operaciones sobre mapas de bits. La aplicación puede llamar a la función *GetDeviceCaps* para determinar si el contexto de dispositivo cumple este requisito. Si a este parámetro se le asigna cero, la función crea un contexto de dispositivo compatible con la pantalla.

### **Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del nuevo contexto de dispositivo de memoria. Si falla, devuelve cero.

### **Véase además**

*CreateCompatibleBitmap*, *DeleteDC*, *GetDeviceCaps*

## Ejemplo

### Listado 2-5: Usando un contexto de dispositivo en memoria para animación

```
var
  Form1: TForm1;
  OffscreenDC: HDC;           // contexto de dispositivo fuera de pantalla
  ANDMaskBitmap,             // usado para almacenar las diferentes partes del
  ORMaskBitmap,              // círculo
  BackgroundBitmap,
  OldBitmap: HBITMAP;
  BallXCoord: Integer;        // las coordenadas horizontales actuales del círculo
```

#### implementation

```
{ $R *.DFM }

procedure TForm1.Timer1Timer(Sender: TObject);
var
  ScreenDC,                 // manejador de contexto de dispositivo de pantalla
  WorkDC: HDC;              // manejador de contexto de dispositivo temporal
  OldBitmap: HBITMAP;       // almacena el mapa de bits anterior
begin
  {Recuperar manejador al contexto de dispositivo para la pantalla}
  ScreenDC := GetDC(0);

  {Crear contexto de dispositivo en memoria}
  WorkDC := CreateCompatibleDC(Canvas.Handle);

  {Restaurar el fondo de pantalla anterior}
  BitBlt(ScreenDC, BallXCoord, Form1.Top, 40, 40, OffscreenDC, 0, 0, SRCCOPY);

  {Incrementar la coordenada horizontal del círculo}
  Inc(BallXCoord);

  {Rotar el círculo alrededor de la pantalla si se va más allá de los límites}
  if BallXCoord > GetSystemMetrics(SM_CXSCREEN) then
    BallXCoord := -40;

  {Almacenar el fondo de la pantalla donde está ubicado el círculo}
  BitBlt(OffscreenDC, 0, 0, 40, 40, ScreenDC, BallXCoord, Form1.Top, SRCCOPY);

  {Seleccionar la máscara AND del círculo en el contexto de dispositivo en
  memoria y copiar éste en la pantalla}
  OldBitmap := SelectObject(WorkDC, ANDMaskBitmap);
  BitBlt(ScreenDC, BallXCoord, Form1.Top, 40, 40, WorkDC, 0, 0, SRCAND);

  {Seleccionar la máscara OR del círculo en el contexto de dispositivo en
  memoria y copiar éste en la pantalla}
  SelectObject(WorkDC, ORMaskBitmap);
  BitBlt(ScreenDC, BallXCoord, Form1.Top, 40, 40, WorkDC, 0, 0, SRCPAINT);

  {Seleccionar el antiguo mapa de bits en el contexto de dispositivo en memoria
  y borrar o liberar los objetos innecesarios}
  SelectObject(WorkDC, OldBitmap);
```

```

    ReleaseDC(0, ScreenDC);
    DeleteDC(WorkDC);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    TempBrush: HBRUSH;    // un manejador de una brocha
begin
    {Crear un contexto de dispositivo en memoria}
    OffscreenDC := CreateCompatibleDC(Canvas.Handle);

    {Una gran cantidad de atributos del contexto de dispositivo cambiarán, por lo
    tanto se debe almacenar su estado original para no tener que reseleccionar de
    nuevo el objeto original dentro del contexto de dispositivo}
    SaveDC(OffscreenDC);

    {Crear un mapa de bits para la máscara AND del círculo}
    AndMaskBitmap := CreateCompatibleBitmap(Canvas.Handle, 40, 40);

    {Seleccionar el mapa de bits en el contexto de dispositivo en memoria
    y dibujar un círculo negro sobre un fondo blanco}
    SelectObject(OffscreenDC, AndMaskBitmap);
    SelectObject(OffscreenDC, GetStockObject(WHITE_BRUSH));
    SelectObject(OffscreenDC, GetStockObject(NULL_PEN));
    Rectangle(OffscreenDC, 0, 0, 41, 41);
    SelectObject(OffscreenDC, GetStockObject(BLACK_BRUSH));
    Ellipse(OffscreenDC, 0, 0, 40, 40);

    {Crear un mapa de bits para la máscara or del círculo}
    ORMaskBitmap := CreateCompatibleBitmap(Canvas.Handle, 40, 40);

    {Seleccionar el mapa de bits en el contexto de dispositivo en memoria
    y dibujar un círculo sombreado sobre un fondo negro}
    SelectObject(OffscreenDC, ORMaskBitmap);
    SelectObject(OffscreenDC, GetStockObject(BLACK_BRUSH));
    Rectangle(OffscreenDC, 0, 0, 41, 41);
    TempBrush := CreateHatchBrush(HS_DIAGCROSS, clRed);
    SelectObject(OffscreenDC, GetStockObject(BLACK_PEN));
    SelectObject(OffscreenDC, TempBrush);
    Ellipse(OffscreenDC, 0, 0, 40, 40);

    {Restaurar los valores originales del contexto de dispositivo. Esto elimina la
    necesidad de reseleccionar objetos cuando hayamos avanzado}
    RestoreDC(OffscreenDC, -1);

    {Eliminar la brocha}
    DeleteObject(TempBrush);

    {Finalmente, crear un mapa de bits que almacena el fondo de la pantalla. Esto
    evita que el círculo deje un rastro tras de sí}
    BackgroundBitmap := CreateCompatibleBitmap(Canvas.Handle, 40, 40);

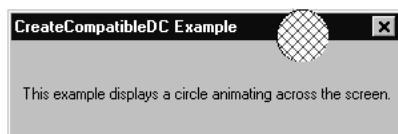
    {Seleccionar el mapa de bits del fondo en el contexto en memoria}
    SelectObject(OffscreenDC, BackgroundBitmap);

```

```
{Inicializar las coordenadas del círculo de manera que comiencen fuera de la
 pantalla por la izquierda}
BallXCoord := -40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Eliminar todos los mapas de bits y contextos de dispositivos innecesarios}
    SelectObject(OffscreenDC, OldBitmap);
    DeleteObject(BackgroundBitmap);
    DeleteObject(ANDMaskBitmap);
    DeleteObject(ORMaskBitmap);
    DeleteDC(OffscreenDC);
end;
```

Figura 2-3: El círculo animado



## DeleteDC

## Windows.Pas

### Sintaxis

```
DeleteDC(
    DC: HDC                {manejador de un contexto de dispositivo}
); BOOL;                  {devuelve TRUE o FALSE}
```

### Descripción

La función *DeleteDC* elimina el contexto de dispositivo especificado. Cuando una aplicación utiliza *CreateCompatibleDC*, deberá siempre llamar a *DeleteDC* cuando termine de utilizar el manejador.

### Parámetros

*DC*: Manejador del contexto de dispositivo a eliminar.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### Véase además

*CreateCompatibleDC*, *GetDC*, *ReleaseDC*

### Ejemplo

Vea el Listado 2-5 bajo *CreateCompatibleDC*.

**DPTOLP****Windows.Pas****Sintaxis**

```

DPTOLP(
    DC: HDC;                {manejador de contexto de dispositivo}
    var Points;              {puntero a un array de registros TPoint}
    Count: Integer           {el número de entradas en el array}
): BOOL;                   {devuelve TRUE o FALSE}

```

**Descripción**

La función *DPTOLP* convierte uno o más pares de coordenadas de dispositivo a coordenadas lógicas. El parámetro *Points* apunta a un *array* de registros *TPoint* que contiene los pares de coordenadas que serán convertidas. Estos registros *TPoint* recibirán las coordenadas convertidas cuando la función retorne. La transformación de las coordenadas se basa en los valores asignados por las funciones *SetWindowOrgEx*, *SetViewportOrgEx*, *SetWindowExtEx* y *SetViewportExtEx*. La función *DPTOLP* fallará si alguno de los puntos en el registro *TPoint* especifica un valor mayor de 27 bits. También esta función fallará si la transformación de alguno de los puntos produce un valor mayor de 32 bits. En el caso de que se produzca un error, todos los valores del *array* quedarán indefinidos.

**Parámetros**

*DC*: Manejador del contexto de dispositivo para el cual será realizada la conversión de las coordenadas.

*Points*: Puntero a un *array* de registros *TPoint* que contiene los pares de coordenadas que serán convertidas.

*Count*: Especifica el número de entradas en el *array* al que apunta el parámetro *Points*.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*LPtoDP*, *SetWindowOrgEx*, *SetViewportOrgEx*, *SetWindowExtEx*, *SetViewportExtEx*

**Ejemplo**

Vea el Listado 2-12 bajo *ScaleViewportExtEx*.

**EnumDisplaySettings****Windows.Pas****Sintaxis**

```

EnumDisplaySettings(
    lpzDeviceName: PChar;    {el dispositivo de visualización}

```

```
iModeNum: DWORD;           {el modo gráfico}
var lpDevMode: TDeviceMode {puntero a un registro que recibe los valores
                             del dispositivo}
): BOOL;                    {devuelve TRUE o FALSE}
```

### Descripción

La función *EnumDisplaySettings* recupera información acerca de los modos gráficos del dispositivo de visualización especificado. Para recuperar todos los modos accesibles se debe comenzar asignando cero a *iModeNum*, e incrementarlo en uno en cada una de las subsiguientes llamadas a la función. El proceso debe continuar hasta que la función devuelva FALSE.

### Parámetros

*lpDeviceName*: El nombre del dispositivo acerca del cual se recupera la información. Si a este parámetro se le asigna **nil**, la función enumerará los modos de visualización para la pantalla actual. La cadena a la que apunta este parámetro debe ser de la forma *\\.\Display1*, *\\.\Display2*, ó *\\.\Display3*. Bajo Windows 95/98 a este parámetro debe asignársele siempre **nil**.

*iModeNum*: El valor del índice del modo gráfico para el que se recuperará la información. Este valor tiene que ser menor que el índice del último modo gráfico de la pantalla. Si el parámetro *iModeNum* está fuera de rango, la función devolverá un error.

*lpDevMode*: Puntero a un registro *TDeviceMode* que recibe información sobre el modo específico de pantalla. De todos los campos del registro *TDeviceMode* solo *dmSize*, *dmBitsPerPel*, *dmFields*, *dmPelsWidth*, *dmPelsHeight*, *dmDisplayFlags* y *dmDisplayFrequency* son usados por esta función. El registro *TDeviceMode* se define de la siguiente forma:

TDeviceMode = **packed record**

```
  dmDeviceName: array[0..CCHDEVICENAME - 1]
    of AnsiChar;           {no se utiliza}
  dmSpecVersion: Word;     {no se utiliza}
  dmDriverVersion: Word;   {no se utiliza}
  dmSize: Word;            {tamaño del registro}
  dmDriverExtra: Word;     {no se utiliza}
  dmFields: DWORD;         {campos válidos}
  dmOrientation: SHORT;    {no se utiliza}
  dmPaperSize: SHORT;      {no se utiliza}
  dmPaperLength: SHORT;    {no se utiliza}
  dmPaperWidth: SHORT;     {no se utiliza}
  dmScale: SHORT;          {no se utiliza}
  dmCopies: SHORT;         {no se utiliza}
  dmDefaultSource: SHORT;  {no se utiliza}
  dmPrintQuality: SHORT;   {no se utiliza}
  dmColor: SHORT;          {no se utiliza}
```

dmDuplex: SHORT;	{no se utiliza}
dmYResolution: SHORT;	{no se utiliza}
dmTTOption: SHORT;	{no se utiliza}
dmCollate: SHORT;	{no se utiliza}
dmFormName: <b>array</b> [0..CCHFORMNAME - 1]	
<b>of</b> AnsiChar;	{no se utiliza}
dmLogPixels: Word;	{no se utiliza}
dmBitsPerPel: DWORD;	{profundidad de color}
dmPelsWidth: DWORD;	{ancho de pantalla}
dmPelsHeight: DWORD;	{alto de pantalla}
dmDisplayFlags: DWORD;	{modo de pantalla}
dmDisplayFrequency: DWORD;	{frecuencia}
dmICMMethod: DWORD;	{no se utiliza}
dmICMIntent: DWORD;	{no se utiliza}
dmMediaType: DWORD;	{no se utiliza}
dmDitherType: DWORD;	{no se utiliza}
dmICCManufacturer: DWORD;	{no se utiliza}
dmICCModel: DWORD;	{no se utiliza}
dmPanningWidth: DWORD;	{no se utiliza}
dmPanningHeight: DWORD;	{no se utiliza}

**end;**

Consulte la función *ChangeDisplaySettings* para ver una descripción de esta estructura de datos.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*ChangeDisplaySettings*

#### Ejemplo

##### Listado 2-6: Enumerando todos los modos de visualización disponibles para la pantalla actual

```
procedure TForm1.Button1Click(Sender: TObject);
var
    DeviceInfo: TDevMode;           // almacena información del dispositivo
    DeviceCount: Integer;          // cantidad de modos de visualización
begin
    {Inicializar contador}
    DeviceCount := 0;

    {Enumerar los modos de visualización del dispositivo de visualización en uso}
    while EnumDisplaySettings(nil, DeviceCount, DeviceInfo) do
        begin
```

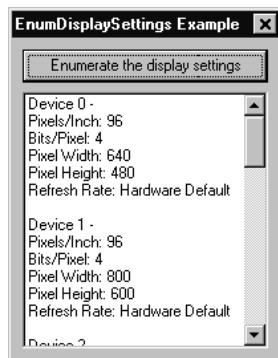
```
{Mostrar la información relevante del modo de visualización}
ListBox1.Items.Add('Device ' + IntToStr(DeviceCount) + ' -');
ListBox1.Items.Add('Pixels/Inch: ' + IntToStr(DeviceInfo.dmLogPixels));
ListBox1.Items.Add('Bits/Pixel: ' + IntToStr(DeviceInfo.dmBitsPerPel));
ListBox1.Items.Add('Pixel Width: ' + IntToStr(DeviceInfo.dmPelsWidth));
ListBox1.Items.Add('Pixel Height: ' + IntToStr(DeviceInfo.dmPelsHeight));

{Indicar el tipo del modo de visualización}
case DeviceInfo.dmDisplayFlags of
  DM_GRAYSCALE: ListBox1.Items.Add('Display Mode: Grayscale');
  DM_INTERLACED: ListBox1.Items.Add('Display Mode: Interlaced');
end;

{Indicar la frecuencia de refresco}
if (DeviceInfo.dmDisplayFrequency=0) or (DeviceInfo.dmDisplayFrequency=1) then
  ListBox1.Items.Add('Refresh Rate: Hardware Default')
else
  ListBox1.Items.Add('Refresh Rate: ' +
    IntToStr(DeviceInfo.dmDisplayFrequency) + ' Hz');

{Añadir una línea en blanco e incrementar contador}
ListBox1.Items.Add('');
Inc(DeviceCount);
end;
end;
```

Figura 2-4: La información sobre el modo de visualización accesible



## GetDC Windows.Pas

### Sintaxis

```
GetDC(
  hWnd: HWND           {manejador de ventana}
): HDC;                {devuelve contexto de dispositivo}
```

### Descripción

La función *GetDC* recupera un contexto de dispositivo para el área cliente de la ventana especificada en el parámetro *hWnd*. El contexto de dispositivo recuperado será

un contexto de dispositivo común, de clase, o privado, dependiendo del estilo de la clase de la ventana especificada. Para contextos de dispositivo comunes, la función *GetDC* inicializa el contexto de dispositivo con atributos por defecto cada vez que es recuperado. Los contextos de clase y privados mantendrán los últimos valores de sus atributos. Cuando el contexto de dispositivo deje de ser necesario, deberá ser liberado llamando la función *ReleaseDC*.

#### Parámetros

*hWnd*: Manejador de la ventana para la cual el contexto de dispositivo es recuperado. Si se le asigna cero a este parámetro, la función recupera un contexto de dispositivo para la pantalla.

#### Valor que devuelve

Si la función tiene éxito, devuelve un contexto de dispositivo para el área cliente de la ventana especificada. Si falla, devuelve cero.

#### Véase además

*ReleaseDC*, *GetWindowDC*

#### Ejemplo

##### Listado 2-7: Recuperando un contexto de dispositivo común para una ventana

```
procedure TForm1.FormPaint(Sender: TObject);
var
    FormDC: HDC;      // almacena el contexto de dispositivo
    OldFont: HFONT;   // almacena las fuentes originales
begin
    {Recuperar contexto de dispositivo común para el formulario}
    FormDC := GetDC(Form1.Handle);

    {Seleccionar la fuente del formulario en el contexto de dispositivo}
    OldFont := SelectObject(FormDC, Form1.Font.Handle);

    {Mostrar algún texto en el contexto de dispositivo}
    SetBkMode(FormDC, TRANSPARENT);
    TextOut(FormDC, 10, 10, 'Delphi Rocks!', Length('Delphi Rocks!'));

    {Seleccionar la fuente original y liberar el contexto de dispositivo}
    SelectObject(FormDC, OldFont);
    ReleaseDC(Form1.Handle, FormDC);
end;
```

Figura 2-5:  
Dibujando en  
el dispositivo



**GetDCOrgEx      Windows.Pas****Sintaxis**

```
GetDCOrgEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    var Origin: TPoint      {puntero a registro TPoint}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

La función *GetDCOrgEx* recupera la conversión final del origen para el contexto de dispositivo especificado. Esta ubicación es el desplazamiento final que Windows utilizará cuando convierta las coordenadas del dispositivo en coordenadas de cliente.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo origen es recuperado.

*Origin*: Puntero al registro *TPoint* que recibirá el origen de coordenadas. Las coordenadas son relativas al origen físico de la pantalla y son dadas en unidades del dispositivo.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*GetWindowOrgEx*, *GetViewportOrgEx*

**Ejemplo**

Vea el Listado 2-2 en la introducción de este capítulo.

**GetDeviceCaps      Windows.Pas****Sintaxis**

```
GetDeviceCaps(
    DC: HDC;                {manejador de contexto de dispositivo}
    Index: Integer          {índice de capacidades}
): Integer;                {devuelve el valor de la capacidad}
```

**Descripción**

La función *GetDeviceCaps* obtiene información de una capacidad particular del contexto de dispositivo especificado. Se puede solicitar una amplia variedad de capacidades, como se muestra en la Tabla 2-7.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuya capacidad se desea obtener.

*Index*: Una opción indicando la capacidad específica que se solicita. A este parámetro se le puede asignar un valor de la Tabla 2-7.

**Valor que devuelve**

Si la función tiene éxito, devuelve el valor específico de la capacidad. Esta función no indica errores.

**Véase además**

*CreateEnhMetaFile, GetDIBits, GetObjectType, GetSystemMetrics, SetDIBits, SetDIBitsToDevice, StretchBlt, StretchDIBits*

**Ejemplo****Listado 2-8: Recuperando las capacidades del dispositivo**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    with ListBox1.Items do
        begin
            {Mostrar la versión del controlador}
            Add('Display Driver Version: ' + IntToStr(GetDeviceCaps(Canvas.Handle,
                DRIVERVERSION)));

            {Mostrar las características tecnológicas}
            case GetDeviceCaps(Canvas.Handle, TECHNOLOGY) of
                DT_PLOTTER:    Add('Driver Type: Vector Plotter');
                DT_RASDISPLAY: Add('Driver Type: Raster Display');
                DT_RASPRINTER: Add('Driver Type: Raster Printer');
                DT_RASCAMERA:  Add('Driver Type: Raster Camera');
                DT_CHARSTREAM: Add('Driver Type: Character Stream');
                DT_METAFILE:   Add('Driver Type: Metafile');
                DT_DISPFILE:   Add('Driver Type: Display File');
            end;

            {Mostrar el tamaño de la pantalla}
            Add('Screen Size: ' + IntToStr(GetDeviceCaps(Canvas.Handle, HORZSIZE)) + ' X '
                + IntToStr(GetDeviceCaps(Canvas.Handle, VERTSIZE)) + ' millimeters');
            Add('Screen Resolution: ' + IntToStr(GetDeviceCaps(Canvas.Handle, HORZRES))
                + ' X ' + IntToStr(GetDeviceCaps(Canvas.Handle, VERTRES)) + ' pixels');

            {Mostrar los píxeles por pulgada lógica}
            Add('Pixels/Logical Inch - Horizontal: ' +
                IntToStr(GetDeviceCaps(Canvas.Handle, LOGPIXELSX)));
            Add('Pixels/Logical Inch - Vertical: ' +
                IntToStr(GetDeviceCaps(Canvas.Handle, LOGPIXELSY)));

            {Mostrar la profundidad de color y el número de objetos gráficos comunes}
            Add('Bits/Pixel: ' + IntToStr(GetDeviceCaps(Canvas.Handle, BITSPIXEL)));
            Add('Brushes: ' + IntToStr(GetDeviceCaps(Canvas.Handle, NUMBRUSHES)));
        end
    end;
```

```

Add('Pens: ' + IntToStr(GetDeviceCaps(Canvas.Handle, NUMPENS)));
Add('Fonts: ' + IntToStr(GetDeviceCaps(Canvas.Handle, NUMFONTS)));

{Mostrar el número de entradas en la tabla de colores}
if GetDeviceCaps(Canvas.Handle, NUMCOLORS) > -1 then
    Add('Entries in color table: ' +
        IntToStr(GetDeviceCaps(Canvas.Handle, NUMCOLORS)));
{Mostrar las dimensiones de los píxeles}
Add('Pixel Width: ' + IntToStr(GetDeviceCaps(Canvas.Handle, ASPECTX)));
Add('Pixel Height: ' + IntToStr(GetDeviceCaps(Canvas.Handle, ASPECTY)));
Add('Pixel Diagonal: ' + IntToStr(GetDeviceCaps(Canvas.Handle, ASPECTXY)));

{Indicar si el dispositivo puede recortar un rectángulo}
if GetDeviceCaps(Canvas.Handle, CLIPCAPS) = 1 then
    Add('Device can clip to a rectangle')
else
    Add('Device cannot clip to a rectangle');

{Mostrar tamaño de la paleta, colores reservados y profundidad de color}
Add('Palette Size: ' + IntToStr(GetDeviceCaps(Canvas.Handle, SIZEPALETTE)));
Add('Number of Reserved Colors: ' +
    IntToStr(GetDeviceCaps(Canvas.Handle, NUMRESERVED)));
Add('Color Resolution: ' +
    IntToStr(Trunc(IntPower(2, GetDeviceCaps(Canvas.Handle, COLORRES)))) +
    ' colors');

{Mostrar las operaciones sobre mapas de bits soportadas}
Add('Raster Capabilities -');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_BANDING)=RC_BANDING then
    Add('    Requires Banding');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_BITBLT)=RC_BITBLT then
    Add('    Can Transfer Bitmaps');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_BITMAP64)=RC_BITMAP64 then
    Add('    Supports Bitmaps > 64K');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_DI_BITMAP)=RC_DI_BITMAP
    then Add('    Supports SetDIBits and GetDIBits');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_DIBTODEV)=RC_DIBTODEV then
    Add('    Supports SetDIBitsToDevice');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_FLOODFILL)=RC_FLOODFILL
    then Add('    Can Perform Floodfills');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS)
    and RC_GDI20_OUTPUT)=RC_GDI20_OUTPUT then
    Add('    Supports Windows 2.0 Features');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_PALETTE)=RC_PALETTE then
    Add('    Palette Based');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS) and RC_SCALING)=RC_SCALING then
    Add('    Supports Scaling');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS)
    and RC_STRETCHBLT)=RC_STRETCHBLT then
    Add('    Supports StretchBlt');
if (GetDeviceCaps(Canvas.Handle, RASTERCAPS)
    and RC_STRETCHDIB)=RC_STRETCHDIB then
    Add('    Supports StretchDIBits');

```

```

{Mostrar las capacidades para curvas}
Add('Curve Capabilities -');
if GetDeviceCaps(Canvas.Handle, CURVECAPS)=CC_NONE then
    Add('    Device Does Not Support Curves')
else begin
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_CIRCLES)=CC_CIRCLES then
        Add('    Supports Circles');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_PIE)=CC_PIE then
        Add('    Supports Pie Wedges');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_CHORD)=CC_CHORD then
        Add('    Supports Chords');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_ELLIPSES)=CC_ELLIPSES then
        Add('    Supports Ellipses');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_WIDE)=CC_WIDE then
        Add('    Supports Wide Borders');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_STYLED)=CC_STYLED then
        Add('    Supports Styled Borders');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_WIDESTYLED)=CC_WIDESTYLED
        then Add('    Supports Wide And Styled Borders');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_INTERIORS)=CC_INTERIORS
        then Add('    Supports Interiors');
    if (GetDeviceCaps(Canvas.Handle, CURVECAPS) and CC_ROUNDRECT)=CC_ROUNDRECT
        then Add('    Supports Rounded Rectangles');
end;

{Mostrar las capacidades para líneas}
Add('Line Capabilities -');
if GetDeviceCaps(Canvas.Handle, LINECAPS)=LC_NONE then
    Add('    Device Does Not Support Lines')
else begin
    if (GetDeviceCaps(Canvas.Handle, LINECAPS)
        and LC_POLYLINE)=LC_POLYLINE then Add('    Supports Polylines');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_MARKER)=LC_MARKER then
        Add('    Supports Markers');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_POLYMARKER)=LC_POLYMARKER
        then Add('    Supports Multiple Markers');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_WIDE)=LC_WIDE then
        Add('    Supports Wide Lines');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_STYLED)=LC_STYLED then
        Add('    Supports Styled Lines');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS)
        and LC_WIDESTYLED)=LC_WIDESTYLED then
        Add('    Supports Wide And Styled Lines');
    if (GetDeviceCaps(Canvas.Handle, LINECAPS) and LC_INTERIORS)=LC_INTERIORS
        then Add('    Supports Interiors');
end;

{Mostrar las capacidades para polígonos}
Add('Polygonal Capabilities -');
if GetDeviceCaps(Canvas.Handle, POLYGONALCAPS)=PC_NONE then
    Add('    Device Does Not Support Polygons')
else begin
    if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and PC_POLYGON)=PC_POLYGON

```

```

    then Add('    Supports Alternate Fill Polygons');
if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS)
    and PC_RECTANGLE)=PC_RECTANGLE then
    Add('    Supports Rectangles');
if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS)
    and PC_WINDPOLYGON)=PC_WINDPOLYGON then
    Add('    Supports Winding Fill Polygons');
if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and PC_SCANLINE)=PC_SCANLINE
    then Add('    Supports Single Scanlines');
if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and PC_WIDE)=PC_WIDE then
    Add('    Supports Wide Borders');
if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS) and PC_STYLED)=PC_STYLED then
    Add('    Supports Styled Borders');
if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS)
    and PC_WIDESTYLED)=PC_WIDESTYLED then
    Add('    Supports Wide And Styled Borders');
if (GetDeviceCaps(Canvas.Handle, POLYGONALCAPS)
    and PC_INTERIORS)=PC_INTERIORS then
    Add('    Supports Interiors');
end;

{Mostrar las capacidades para texto}
Add('Text Capabilities -');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_OP_CHARACTER)=TC_OP_CHARACTER then
    Add('    Capable of Character Output Precision');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_OP_STROKE)=TC_OP_STROKE then
    Add('    Capable of Stroke Output Precision');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_CP_STROKE)=TC_CP_STROKE then
    Add('    Capable of Stroke Clip Precision');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_CR_90)=TC_CR_90 then
    Add('    Supports 90 Degree Character Rotation');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_CR_ANY)=TC_CR_ANY then
    Add('    Supports Character Rotation to Any Angle');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_SF_X_YINDEP)=TC_SF_X_YINDEP then
    Add('    X And Y Scale Independent');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_SA_DOUBLE)=TC_SA_DOUBLE then
    Add('    Supports Doubled Character Scaling');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_SA_INTEGER)=TC_SA_INTEGER then
    Add('    Supports Integer Multiples Only When Scaling');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_SA_CONTIN)=TC_SA_CONTIN then
    Add('    Supports Any Multiples For Exact Character Scaling');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS)
    and TC_EA_DOUBLE)=TC_EA_DOUBLE then
    Add('    Supports Double Weight Characters');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_IA_ABLE)=TC_IA_ABLE then
    Add('    Supports Italics');

```

```

if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_UA_ABLE)=TC_UA_ABLE then
    Add('    Supports Underlines');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_SO_ABLE)=TC_SO_ABLE then
    Add('    Supports Strikeouts');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_RA_ABLE)=TC_RA_ABLE then
    Add('    Supports Raster Fonts');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_VA_ABLE)=TC_VA_ABLE then
    Add('    Supports Vector Fonts');
if (GetDeviceCaps(Canvas.Handle, TEXTCAPS) and TC_SCROLLBLT)=TC_SCROLLBLT then
    Add('    Cannot Scroll Using Blits');
end;
end;

```

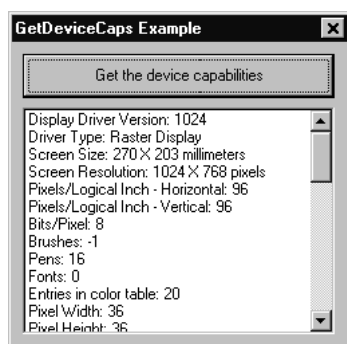


Figura 2-6:  
Las  
capacidades  
del dispositivo

Tabla 2-7: Valores del parámetro Index de GetDeviceCaps

Valor	Descripción
DRIVERVERSION	La versión del controlador.
TECHNOLOGY	Tipo de tecnología del dispositivo. Esta opción devuelve un valor de la Tabla 2-8. El parámetro DC puede referirse a un metaarchivo mejorado, en cuyo caso la tecnología devuelta será la del dispositivo referido en el metaarchivo. Utilice la función GetObjectType para determinar si el contexto de dispositivo se refiere a un dispositivo en un metaarchivo mejorado.
HORZSIZE	Ancho de la pantalla física en milímetros.
VERTSIZE	Altura de la pantalla física en milímetros.
HORZRES	Ancho de la pantalla en píxeles.
VERTRES	Altura de la pantalla en líneas de barrido.
LOGPIXELSX	Número de píxeles horizontales por pulgada lógica.
LOGPIXELSY	Número de píxeles verticales por pulgada lógica.
BITSPIXEL	Número de bits de colores adyacentes por píxel.
PLANES	Número de planos de colores.
NUMBRUSHES	Número de brochas específicas del dispositivo.
NUMPENS	Número de plumas específicas del dispositivo.

Valor	Descripción
NUMFONTS	Número de fuentes específicas del dispositivo.
NUMCOLORS	Número de entradas en la tabla de colores del dispositivo, si el dispositivo tiene una profundidad de color de 8 bits o menos. Devuelve -1 para profundidades de color superiores.
ASPECTX	Ancho relativo del píxel del dispositivo para dibujar líneas.
ASPECTY	Altura relativa del píxel del dispositivo para dibujar líneas.
ASPECTXY	Ancho de la diagonal del píxel del dispositivo para dibujar líneas.
CLIPCAPS	Indicador de las capacidades de recorte del dispositivo. Si el dispositivo puede recortar un rectángulo, este valor es 1; en caso contrario, es 0.
SIZEPALETTE	Número de entradas en la paleta del sistema. Este resultado es válido solo en Windows 3.0 ó controladores posteriores y solo si el controlador del dispositivo selecciona el bit RC_PALETTE en el índice RASTERCAPS.
NUMRESERVED	Número de entradas reservadas en la paleta del sistema. Este índice es válido solamente en controladores para Windows 3.0 ó superiores, y sólo si el controlador del dispositivo tiene seleccionado el bit RC_PALETTE en el índice RASTERCAPS.
COLORRES	Resolución de color del dispositivo en bits por píxel. Este índice es válido solamente en controladores para Windows 3.0 ó superiores, y sólo si el controlador del dispositivo tiene seleccionado el bit RC_PALETTE en el índice RASTERCAPS.
PHYSICALWIDTH	Ancho físico de una página impresa para dispositivos de impresión, en unidades del dispositivo. Este número es generalmente más grande que el ancho imprimible en píxeles, debido a los márgenes no imprimibles.
PHYSICALHEIGHT	Altura física de una página impresa para dispositivos de impresión, en unidades del dispositivo. Este número es generalmente más grande que la altura imprimible en píxeles, debido a los márgenes no imprimibles.
PHYSICALOFFSETX	Margen izquierdo de la impresora. Esta es la distancia desde el borde izquierdo de la página física al borde izquierdo del área de impresión en unidades del dispositivo.
PHYSICALOFFSETY	Margen superior de la impresora. Esta es la distancia del borde superior de la página física al borde superior del área de impresión en unidades del dispositivo.
VREFRESH	Sólo Windows NT: La frecuencia de refresco vertical del monitor en Hz. Un valor de 0 ó 1 representa la frecuencia por defecto del monitor, generalmente seleccionable mediante interruptores en la tarjeta de vídeo o en la placa base del ordenador, o mediante un programa de configuración, que no es compatible con las funciones de Win32 tales como ChangeDisplaySettings.

Valor	Descripción
DESKTOPHORZRES	Sólo Windows NT: ancho del escritorio virtual en píxeles. Este valor puede ser mayor que HORZRES si el dispositivo soporta un escritorio virtual o múltiples monitores.
DESKTOPVERTRES	Sólo Windows NT: Altura del escritorio virtual en píxeles. Este valor puede ser mayor que VERTRES si el dispositivo soporta un escritorio virtual o múltiples monitores.
BLTALIGNMENT	Sólo Windows NT: Alineamiento preferido para el dibujo horizontal, expresado como un múltiplo de píxeles. Para mejor ejecución del dibujo, las ventanas deben ser alineadas horizontalmente a un múltiplo de este valor. Un valor cero indica que el dispositivo ofrece aceleración por hardware y cualquier tipo de alineamiento puede ser usado.
RASTERCAPS	Indica las capacidades de barrido del dispositivo. Devuelve uno o más valores de la Tabla 2-9.
CURVECAPS	Indica las capacidades para curvas del dispositivo. Devuelve uno o más valores de la Tabla 2-10.
LINECAPS	Indica las capacidades para líneas del dispositivo. Devuelve uno o más valores de la Tabla 2-11.
POLYGONALCAPS	Indica las capacidades para polígonos del dispositivo. Devuelve uno o más valores de la Tabla 2-12.
TEXTCAPS	Indica las capacidades para texto del dispositivo. Devuelve uno o más valores de la Tabla 2-13.

Tabla 2-8: Valores que devuelve GetDeviceCaps para el índice TECHNOLOGY

Valor	Descripción
DT_PLOTTER	Un plotter vectorial.
DT_RASDISPLAY	Un monitor de barrido.
DT_RASPRINTER	Una impresora de barrido.
DT_RASCAMERA	Una cámara de barrido.
DT_CHARSTREAM	Un flujo de caracteres.
DT_METAFILE	Un metaarchivo.
DT_DISPFILE	Un archivo de monitor.

Tabla 2-9: Valores que devuelve GetDeviceCaps para el índice RASTERCAPS

Valor	Descripción
RC_BANDING	El dispositivo requiere soporte de bandas.
RC_BITBLT	El dispositivo es capaz de transferir mapas de bits.
RC_BITMAP64	El dispositivo soporta mapas de bits mayores de 64K.
RC_DI_BITMAP	El dispositivo soporta las funciones SetDIBits y GetDIBits.
RC_DIBTODEV	El dispositivo soporta la función SetDIBitsToDevice.
RC_FLOODFILL	El dispositivo puede realizar relleno de interiores.

Valor	Descripción
RC_GDI20_OUTPUT	El dispositivo soporta características de Windows 2.0.
RC_PALETTE	El dispositivo está basado en paleta.
RC_SCALING	El dispositivo es escalable.
RC_STRETCHBLT	El dispositivo soporta la función <i>StretchBlt</i> .
RC_STRETCHDIB	El dispositivo soporta la función <i>StretchDIBits</i> .

Tabla 2-10: Valores que devuelve *GetDeviceCaps* para el índice **CURVECAPS**

Valor	Descripción
CC_NONE	El dispositivo no soporta el dibujo de curvas.
CC_CIRCLES	El dispositivo puede dibujar círculos.
CC_PIE	El dispositivo puede dibujar trozos de tarta.
CC_CHORD	El dispositivo puede dibujar arcos de cuerda.
CC_ELLIPSES	El dispositivo puede dibujar elipses.
CC_WIDE	El dispositivo puede dibujar bordes anchos.
CC_STYLED	El dispositivo puede dibujar bordes con estilo.
CC_WIDESTYLED	El dispositivo puede dibujar bordes anchos y con estilo.
CC_INTERIORS	El dispositivo puede dibujar interiores.
CC_ROUNDRECT	El dispositivo puede dibujar rectángulos redondeados.

Tabla 2-11: Valores que devuelve *GetDeviceCaps* para el índice **LINECAPS**

Valor	Descripción
LC_NONE	El dispositivo no soporta el dibujo de líneas.
LC_POLYLINE	El dispositivo puede dibujar polígonos.
LC_MARKER	El dispositivo puede dibujar marcadores.
LC_POLYMARKER	El dispositivo puede dibujar múltiples marcadores.
LC_WIDE	El dispositivo puede dibujar líneas anchas.
LC_STYLED	El dispositivo puede dibujar líneas con estilo.
LC_WIDESTYLED	El dispositivo puede dibujar líneas anchas y con estilo.
LC_INTERIORS	El dispositivo puede dibujar interiores.

Tabla 2-12: Valores que devuelve *GetDeviceCaps* para el índice **POLYGONALCPAS**

Valor	Descripción
PC_NONE	El dispositivo no soporta el dibujo de polígonos.
PC_POLYGON	El dispositivo puede dibujar polígonos de relleno alternativo.
PC_RECTANGLE	El dispositivo puede dibujar rectángulos.
PC_WINDPOLYGON	El dispositivo puede dibujar polígonos con relleno Winding.
PC_SCANLINE	El dispositivo puede dibujar una línea de barrido sencilla.

Valor	Descripción
PC_WIDE	El dispositivo puede dibujar bordes anchos.
PC_STYLED	El dispositivo puede dibujar bordes con estilo.
PC_WIDESTYLED	El dispositivo puede dibujar bordes anchos y con estilo.
PC_INTERIORS	El dispositivo puede dibujar interiores.

Tabla 2-13: Valores que devuelve `GetDeviceCaps` para el índice `TEXTCAPS`

Valor	Descripción
TC_OP_CHARACTER	El dispositivo soporta la impresión precisa de caracteres.
TC_OP_STROKE	El dispositivo soporta la impresión precisa de trazos.
TC_CP_STROKE	El dispositivo soporta recorte de precisión.
TC_CR_90	El dispositivo puede hacer rotación de caracteres de 90 grados.
TC_CR_ANY	El dispositivo puede hacer cualquier rotación de caracteres.
TC_SF_X_YINDEP	El dispositivo puede escalar tanto en la dirección horizontal como en la vertical.
TC_SA_DOUBLE	El dispositivo soporta carácter de doble tamaño para escalar.
TC_SA_INTEGER	El dispositivo utiliza sólo múltiplos enteros para escalar caracteres.
TC_SA_CONTIN	El dispositivo soporta cualquier múltiplo para un exacto escalamiento de caracteres.
TC_EA_DOUBLE	El dispositivo puede dibujar caracteres en negrita.
TC_IA_ABLE	El dispositivo soporta itálica.
TC_UA_ABLE	El dispositivo soporta subrayado.
TC_SO_ABLE	El dispositivo puede dibujar tachado.
TC_RA_ABLE	El dispositivo puede dibujar fuentes de barrido.
TC_VA_ABLE	El dispositivo puede dibujar fuentes vectoriales.
TC_SCROLLBLT	El dispositivo no puede hacer <i>scroll</i> usando transferencias de bloques de bits.

**GetMapMode      Windows.Pas****Sintaxis**

```
GetMapMode(
    DC: HDC                {manejador de contexto de dispositivo}
); Integer;               {devuelve el modo de mapeado}
```

### Descripción

La función *GetMapMode* recupera el modo actual de mapeado del contexto de dispositivo especificado.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuyo modo de mapeado se desea obtener.

### Valor que devuelve

Si la función tiene éxito, devuelve una opción indicando el modo actual de mapeado, que puede ser un valor de la Tabla 2-14; en caso contrario, devuelve cero.

### Véase además

*SetMapMode*, *SetWindowExtEx*, *SetViewportExtEx*

### Ejemplo

Vea el Listado 2-2 en la introducción.

**Tabla 2-14: Valores que devuelve *GetMapMode***

Valor	Descripción
MM_ANISOTROPIC	Las unidades, el escalamiento y la orientación son asignados por <i>SetWindowExtEx</i> y <i>SetViewportExtEx</i> . Los ejes de escalamiento <i>x</i> e <i>y</i> son tratados de forma independiente y los valores correspondientes no tienen que coincidir.
MM_HIENGLISH	Mapeado de alta resolución en unidades inglesas. Cada unidad tiene 0,001 pulgadas. Las <i>x</i> crecen hacia la derecha y las <i>y</i> hacia arriba.
MM_HIMETRIC	Mapeado de alta resolución en unidades métricas decimales. Cada unidad tiene 0,01 milímetros. Las <i>x</i> crecen hacia la derecha y las <i>y</i> hacia arriba.
MM_ISOTROPIC	Las unidades, el escalamiento y la orientación son asignados por <i>SetWindowExtEx</i> y <i>SetViewportExtEx</i> con las unidades verticales y horizontales iguales. Las unidades y orientación pueden ser elegidos por el programador, pero las unidades para los ejes <i>x</i> e <i>y</i> están obligadas por el GDI a ser las mismas. Esto asegura un relación de aspecto 1:1.
MM_LOENGLISH	Mapeado de baja resolución en unidades inglesas. Cada unidad tiene 0,01 de pulgada. Las <i>x</i> crecen hacia la derecha y las <i>y</i> hacia arriba.
MM_LOMETRIC	Mapeado de baja resolución en unidades métricas decimales. Cada unidad tiene 0,1 de pulgada. Las <i>x</i> crecen hacia la derecha y las <i>y</i> hacia arriba.

Valor	Descripción
MM_TEXT	Cada unidad es mapeada a un píxel del dispositivo. Este no es un mapeado independiente del dispositivo. Dispositivos con diferentes resoluciones o escalamientos producirán diferentes resultados de las funciones gráficas. Las <i>x</i> crecen hacia la derecha y las <i>y</i> hacia abajo. Este es el modo de mapeado por defecto.
MM_TWIPS	Cada unidad es mapeada a 1/1440 de pulgada, que corresponde a un veinteavo de un punto de impresora. Las <i>x</i> crecen hacia la derecha y las <i>y</i> hacia arriba.

**GetSystemMetrics****Windows.Pas***Sintaxis*

```
GetSystemMetrics(
  nIndex: Integer           {índice del elemento}
): Integer;                {devuelve la medida del elemento}
```

*Descripción*

La función *GetSystemMetrics* recupera las dimensiones, en píxeles, de un elemento específico de visualización de Windows. Una gran variedad de parámetros pueden ser solicitados dependiendo del valor del parámetro *nIndex*. Todos los resultados se expresan en valores numéricos o píxeles, excepto para la opción *SM\_ARRANGE*, que devuelve una combinación de valores de la Tabla 2-15.

*Parámetros*

*nIndex*: Una opción que indica el elemento de visualización de Windows cuya medida será recuperada. A este parámetro se le puede asignar un valor de la Tabla 2-16.

*Valores que devuelve*

Si la función tiene éxito, devuelve la medida del elemento solicitado. Si falla, devuelve cero.

*Véase además*

*GetDeviceCaps*

*Ejemplo***Listado 2-9: Recuperando las dimensiones específicas de un elemento**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with ListBox1.Items do
  begin
    {Mostrar la distribución de la ventana minimizada}
    Add('Minimized Window Arrangement -');
```

```

if (GetSystemMetrics(SM_ARRANGE) and ARW_BOTTOMLEFT)=ARW_BOTTOMLEFT then
  Add('  Starts in the lower left corner');
if (GetSystemMetrics(SM_ARRANGE) and ARW_BOTTOMRIGHT)=ARW_BOTTOMRIGHT then
  Add('  Starts in the lower right corner');
if (GetSystemMetrics(SM_ARRANGE) and ARW_HIDE)=ARW_HIDE then
  Add('  Minimized windows are hidden');
if (GetSystemMetrics(SM_ARRANGE) and ARW_TOPLEFT)=ARW_TOPLEFT then
  Add('  Starts in the top left corner');
if (GetSystemMetrics(SM_ARRANGE) and ARW_TOPRIGHT)=ARW_TOPRIGHT then
  Add('  Starts in the top right corner');

if (GetSystemMetrics(SM_ARRANGE) and ARW_DOWN)=ARW_DOWN then
  Add('  Arranged vertically, top to bottom');
if (GetSystemMetrics(SM_ARRANGE) and ARW_LEFT)=ARW_LEFT then
  Add('  Arranged horizontally, left to right');
if (GetSystemMetrics(SM_ARRANGE) and ARW_RIGHT)=ARW_RIGHT then
  Add('  Arranged horizontally, right to left');
if (GetSystemMetrics(SM_ARRANGE) and ARW_UP)=ARW_UP then
  Add('  Arrange vertically, bottom to top');

{Mostrar las dimensiones del borde de la ventana}
Add('Window border width: ' + IntToStr(GetSystemMetrics(SM_CXEDGE)));
Add('Window border height: ' + IntToStr(GetSystemMetrics(SM_CYEDGE)));
{Mostrar las dimensiones del cursor}
Add('Cursor width: ' + IntToStr(GetSystemMetrics(SM_CXCURSOR)));
Add('Cursor height: ' + IntToStr(GetSystemMetrics(SM_CYCURSOR)));
{Mostrar las dimensiones del icono}
Add('Icon width: ' + IntToStr(GetSystemMetrics(SM_CXICON)));
Add('Icon height: ' + IntToStr(GetSystemMetrics(SM_CYICON)));
{Mostrar las dimensiones de la ventana maximizada}
Add('Maximized window width: ' + IntToStr(GetSystemMetrics(SM_CXMAXIMIZED)));
Add('Maximized window height: ' + IntToStr(GetSystemMetrics(SM_CYMAXIMIZED)));
{Mostrar las dimensiones de la pantalla}
Add('Screen width: ' + IntToStr(GetSystemMetrics(SM_CXSCREEN)));
Add('Screen height: ' + IntToStr(GetSystemMetrics(SM_CYSCREEN)));
{Mostrar la altura de la barra de título}
Add('Caption height: ' + IntToStr(GetSystemMetrics(SM_CYCAPTION)));

end;
end;

```

Figura 2-7:  
Dimensiones  
específicas del  
elemento

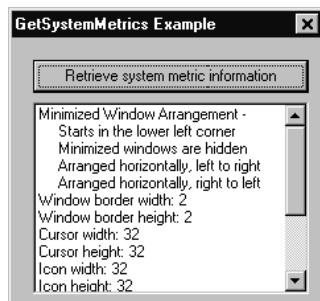


Tabla 2-15: Valores que devuelve `GetSystemMetrics` para el índice `SM_ARRANGE`

Valor	Descripción
<code>ARW_BOTTOMLEFT</code>	La posición inicial por defecto para situar las ventanas minimizadas es la esquina inferior izquierda de la pantalla.
<code>ARW_BOTTOMRIGHT</code>	La posición inicial por defecto para situar las ventanas minimizadas es la esquina inferior derecha de la pantalla.
<code>ARW_HIDE</code>	Colocar las ventanas minimizadas fuera de la pantalla, en una zona invisible.
<code>ARW_TOPLEFT</code>	La posición inicial por defecto para situar las ventanas minimizadas es la esquina superior izquierda de la pantalla.
<code>ARW_TOPRIGHT</code>	La posición inicial por defecto para situar las ventanas minimizadas es la esquina superior derecha de la pantalla.
<code>ARW_DOWN</code>	Coloca verticalmente las ventanas minimizadas, de arriba a abajo.
<code>ARW_LEFT</code>	Coloca horizontalmente las ventanas minimizadas, de izquierda a derecha.
<code>ARW_RIGHT</code>	Coloca horizontalmente las ventanas minimizadas, de derecha a izquierda.
<code>ARW_UP</code>	Coloca verticalmente las ventanas minimizadas, de abajo a arriba.

Tabla 2-16: Valores del parámetro `nIndex` de `GetSystemMetrics`

Valor	Descripción
<code>SM_ARRANGE</code>	Devuelve una combinación de valores de la Tabla 2-15 que especifica cómo el sistema distribuye las ventanas minimizadas.
<code>SM_CLEANBOOT</code>	Devuelve un valor que especifica cómo el sistema ha sido iniciado: 0 = Inicio normal. 1 = Inicio en modo seguro. 2 = Inicio en modo seguro con soporte para redes.
<code>SM_CMOUSEBUTTONS</code>	Devuelve el número de botones del ratón o cero si no hay ratón instalado.
<code>SM_CXBORDER</code> , <code>SM_CYBORDER</code>	Ancho y altura del borde de la ventana. Estos son los mismos valores de <code>SM_CXEDGE</code> y <code>SM_CYEDGE</code> para ventanas con apariencia 3-D.
<code>SM_CXCURSOR</code> , <code>SM_CYCURSOR</code>	Ancho y altura del cursor. Estas son las dimensiones soportadas por el controlador de pantalla en uso. Debido a los requerimientos del controlador de pantalla, el sistema no puede crear cursores de otras dimensiones.

Valor	Descripción
SM_CXDOUBLECLK, SM_CYDOUBLECLK	Ancho y altura del rectángulo alrededor de la zona del primer clic en una operación de doble clic. El segundo clic debe ocurrir dentro de ese rectángulo para que el sistema considere los dos clics como un doble clic. Para que se genere un doble clic, el segundo clic debe ocurrir además en un tiempo determinado.
SM_CXDRAG, SM_CYDRAG	Ancho y altura de un rectángulo centrado en un punto de arrastre que permite movimientos limitados del puntero del ratón antes de que el arrastre comience. Esto permite al usuario ciertos movimientos del ratón sin comenzar inadvertidamente la operación de arrastre.
SM_CXEDGE, SM_CYEDGE	Dimensiones de un borde 3-D. Equivalentes a SM_CXBORDER y SM_CYBORDER.
SM_CXFIXEDFRAME, SM_CYFIXEDFRAME	Grosor del marco que rodea una ventana que tiene una barra de título, pero que no es redimensionable. SM_CXFIXEDFRAME es el ancho del borde horizontal y SM_CYFIXEDFRAME es la altura del borde vertical.
SM_CXFULLSCREEN, SM_CYFULLSCREEN	Ancho y altura del área cliente de una ventana a pantalla completa. El tamaño de la ventana máxima que no se solape con la barra de tareas puede obtenerse llamando a la función SystemParametersInfo con el valor SPI_GETWORKAREA.
SM_CXHSCROLL, SM_CYHSCROLL	Ancho del mapa de bits de la flecha en una barra de desplazamiento horizontal y altura de la barra de desplazamiento horizontal.
SM_CXHTHUMB	Ancho del botón en una barra de desplazamiento horizontal.
SM_CXICON, SM_CYICON	Ancho y altura por defecto de un icono. Normalmente es 32x32, pero puede depender del hardware instalado. La función LoadIcon está limitada a cargar solamente iconos de esas dimensiones.
SM_CXICONSPACING, SM_CYICONSPACING	Dimensiones de una celda de la rejilla para elementos en una vista de iconos grandes. La pantalla es mapeada en rectángulos de este tamaño, con cada elemento llenando un rectángulo. Estos valores son siempre mayores o iguales que los de SM_CXICON y SM_CYICON.
SM_CXMAXIMIZED, SM_CYMAXIMIZED	Tamaño por defecto de una ventana de nivel superior maximizada.
SM_CXMAXTRACK, SM_CYMAXTRACK	Tamaño máximo por defecto de una ventana que tiene una barra de título y bordes redimensionables. El sistema no permitirá al usuario arrastrar el borde de la ventana a un tamaño superior. Una aplicación puede superar estos límites procesando el mensaje WM_GETMINMAXINFO.
SM_CXMENUCHECK, SM_CYMENUCHECK	Tamaño por defecto del mapa de bits de la marca de verificación de elementos de un menú.

Valor	Descripción
SM_CXMENUSIZE, SM_CYMENUSIZE	Tamaño de los botones de la barra de menú.
SM_CXMIN, SM_CYMIN	Ancho y altura mínimos de una ventana.
SM_CXMINIMIZED, SM_CYMINIMIZED	Tamaño de una ventana normal minimizada.
SM_CXMINSPACING, SM_CYMINSPACING	Tamaño de una celda de la rejilla para ventanas minimizadas. Consulte SM_CXICONSPACING y SM_CYICONSPACING. Las ventanas minimizadas son distribuidas dentro de rectángulos de este tamaño. Estos valores son siempre mayores o iguales que SM_CXMINIMIZED y SM_CYMINIMIZED.
SM_CXMINTRACK, SM_CYMINTRACK	Ancho y altura mínimos de una ventana. El sistema no permitirá arrastrar el borde de la ventana a un tamaño inferior a éste. Una aplicación puede superar estos límites procesando el mensaje WM_GETMINMAXINFO.
SM_CXSCREEN, SM_CYSCREEN	Ancho y altura de la pantalla.
SM_CXSIZE, SM_CYSIZE	Ancho y altura de los botones de la barra de título de una ventana.
SM_CXSIZEFRAME, SM_CYSIZEFRAME	Grosor del borde de una ventana redimensionable. SM_CXSIZEFRAME es el ancho del borde horizontal y SM_CYSIZEFRAME la altura del borde vertical. Estos valores son los mismos que SM_CXFRAME y SM_CYFRAME.
SM_CXSMICON, SM_CYSMICON	Tamaño recomendado de un icono pequeño. Generalmente los iconos pequeños se utilizan en la barra de título de las ventanas y en vistas de iconos pequeños.
SM_CXSMSIZE, SM_CYSMSIZE	Tamaño de los botones pequeños de la barra de título.
SM_CXVSCROLL, SM_CYVSCROLL	Ancho de la barra de desplazamiento vertical y altura del mapa de bits de la flecha de la barra de desplazamiento vertical.
SM_CYCAPTION	Altura del área de la barra de título normal.
SM_CYKANJIWINDOW	La altura de la ventana Kanji en la parte inferior de la pantalla para sistemas que usan el conjunto de caracteres de doble byte.
SM_CYMENU	Altura de la barra de menú de una sola línea.
SM_CYSMCAPTION	Altura de una barra de título pequeña.
SM_CYVTHUMB	Altura del botón en una barra de desplazamiento vertical.
SM_DBCSENABLED	Valor distinto de cero si la versión de doble byte del conjunto de caracteres de USER.EXE está instalada; cero si DBCS no está instalada.
SM_DEBUG	Distinto de cero si la versión de depuración de USER.EXE está instalada; cero en caso contrario.

Valor	Descripción
SM_MENUDROPALIGNMENT	Distinto de cero si los menús desplegables se alinean a la derecha respecto a la opción en la barra de menú; cero si se alinean a la izquierda.
SM_MIDEASTENABLED	Distinto de cero si el sistema tiene habilitados los idiomas árabe y hebreo; cero si no los tiene.
SM_MOUSEPRESENT	Distinto de cero si hay un ratón instalado; cero si no lo hay.
SM_MOUSEWHEELPRESENT	Sólo Windows NT: Distinto de cero si hay instalado un ratón con rueda; cero en caso contrario.
SM_NETWORK	Si existe una red se activa el bit menos significativo; en caso contrario, se deja en cero. Los otros bits están reservados.
SM_PENWINDOWS	Distinto de cero si la extensión de Windows para pantallas táctiles ( <i>pen computing</i> ) está instalada; cero en caso contrario.
SM_SECURE	Distinto de cero si hay seguridad instalada; cero si no la hay.
SM_SHOWSOUNDS	Distinto de cero si el usuario especifica que las presentaciones de audio tienen también una representación visual; cero si no la tienen.
SM_SLOWMACHINE	Distinto de cero si la computadora tiene un procesador de bajas prestaciones; cero en caso contrario.
SM_SWAPBUTTON	Distinto de cero si los botones izquierdo y derecho del ratón han sido configurados para intercambiarse; cero en caso contrario.

**GetViewportExtEx****Windows.Pas****Sintaxis**

```
GetViewportExtEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    var Size: TSize          {dimensiones x e y del puerto de visualización}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

La función *GetViewportExtEx* recupera las dimensiones horizontal y vertical del puerto de visualización asociadas con el manejador del contexto de dispositivo especificado.

**Parámetros**

*DC*: El manejador del contexto de dispositivo del cual van a ser recuperadas las dimensiones del puerto de visualización.

*Size*: Puntero a un registro *TSize* que recibe las dimensiones horizontal y vertical del puerto de visualización asociado con el contexto de dispositivo especificado.

#### Valores que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*GetViewportOrgEx*, *GetWindowExtEx*, *GetWindowOrgEx*, *SetViewportExtEx*, *SetViewportOrgEx*, *SetWindowExtEx*, *SetWindowOrgEx*

#### Ejemplo

Vea el Listado 2-2 en la introducción.

### **GetViewportOrgEx**      **Windows.Pas**

#### Sintaxis

```
GetViewportOrgEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    var Point: TPoint        {origen de coordenadas del puerto de
                             visualización}
): BOOL;                   {devuelve TRUE o FALSE}
```

#### Descripción

La función *GetViewportOrgEx* recupera el origen del sistema de coordenadas del puerto de visualización asociado con el contexto de dispositivo especificado.

#### Parámetros

*DC*: El manejador del contexto de dispositivo cuyo origen del sistema de coordenadas del puerto de visualización será recuperado.

*Point*: Puntero a un registro *TPoint* que recibe los valores *x* e *y* del origen del sistema de coordenadas del puerto de visualización.

#### Valores que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*GetViewportExtEx*, *GetWindowExtEx*, *GetWindowOrgEx*, *SetViewportExtEx*, *SetViewportOrgEx*, *SetWindowExtEx*, *SetWindowOrgEx*

#### Ejemplo

Vea el Listado 2-2 en la introducción.

**GetWindowDC**    **Windows.Pas****Sintaxis**

```
GetWindowDC(
    hWnd: HWND           {manejador de ventana}
): HDC;                 {devuelve un contexto de dispositivo de la ventana}
```

**Descripción**

La función *GetWindowDC* devuelve un contexto de dispositivo para la ventana especificada por el parámetro *hWnd*. El contexto de dispositivo recuperado se refiere a toda la ventana, incluyendo áreas no cliente, tales como la barra de título, el menú, las barras de desplazamiento y los marcos. Esto le permite a una aplicación implementar gráficos a la medida en áreas no cliente como la barra de título o los bordes. Cuando el contexto de dispositivo deje de ser necesario deberá ser liberado llamando a la función *ReleaseDC*. Observe que esta función recupera sólo un contexto de dispositivo común y cualquier atributo modificado en este contexto de dispositivo no se reflejará en ningún contexto del dispositivo de clase o privado de la ventana, si los hubiera.

**Parámetros**

*hWnd*: Manejador de la ventana de la que se recuperará el contexto de dispositivo.

**Valores que devuelve**

Si la función tiene éxito, devuelve un manejador del contexto de dispositivo para la ventana seleccionada. Si falla, devuelve cero.

**Véase además**

*BeginPaint*, *GetDC*, *GetSystemMetrics*, *ReleaseDC*

**Ejemplo****Listado 2-10: Pintando una barra de título a la medida**

```
procedure TForm1.WMNCPaint(var Msg: TMessage);
var
    WinDC: HDC;           // almacena el contexto de dispositivo de la ventana
    OldFont: HFONT;       // almacena la fuente anterior
begin
    {Llamar al manejador heredado}
    inherited;

    {Recuperar un manejador para el contexto de dispositivo de la ventana}
    WinDC := GetWindowDC(Form1.Handle);

    {Inicializar la fuente}
    Canvas.Font.Height := GetSystemMetrics(SM_CYCAPTION) - 4;
    Canvas.Font.Name := 'Times New Roman';
    Canvas.Font.Style := [fsBold, fsItalic];
```

```

{Seleccionar la fuente en el contexto de dispositivo de la ventana}
OldFont := SelectObject(WinDC, Canvas.Font.Handle);

{Si la ventana está activa}
if GetActiveWindow = 0 then begin
    {Dibujar los colores activos}
    SetBkColor(WinDC, GetSysColor(COLOR_INACTIVECAPTION));
    SetTextColor(WinDC, GetSysColor(COLOR_INACTIVECAPTIONTEXT));
end
else begin
    {En caso contrario, dibujar los colores inactivos}
    SetBkColor(WinDC, GetSysColor(COLOR_ACTIVECAPTION));
    SetTextColor(WinDC, GetSysColor(COLOR_CAPTIONTEXT));
end;

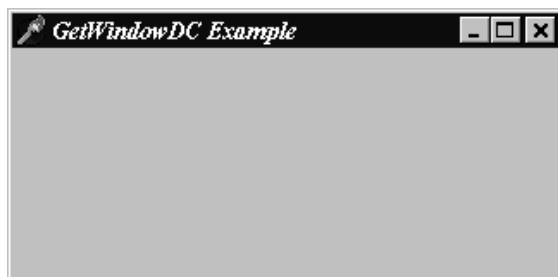
{Dibujar el texto de la barra de título en negrita itálica}
SetBkMode(WinDC, OPAQUE);
TextOut(WinDC, GetSystemMetrics(SM_CXEDGE) + GetSystemMetrics(SM_CXSMICON) + 6,
        GetSystemMetrics(SM_CYEDGE) + 3, 'GetWindowDC Example',
        Length('GetWindowDC Example'));

{Reemplazar la fuente original y liberar contexto de dispositivo de la ventana}
SelectObject(WinDC, OldFont);
ReleaseDC(Form1.Handle, WinDC);
end;

procedure TForm1.WMActivate(var Msg: TWMActivate);
begin
    {Llamar al manejador de mensaje heredado y redibujar la barra de título}
    inherited;
    PostMessage(Form1.Handle, WM_NCPAINT, 0, 0);
end;

```

Figura 2-8:  
Una barra de  
título a la  
medida



### GetWindowExtEx

### Windows.Pas

#### Sintaxis

GetWindowExtEx(	
DC: HDC;	{manejador de contexto de dispositivo}
var Size: TSize	{dimensiones x e y de la ventana}
); BOOL;	{devuelve TRUE o FALSE}

### Descripción

La función *GetWindowExtEx* recupera las dimensiones horizontal y vertical de la ventana asociada con el contexto de dispositivo.

### Parámetros

*DC*: Manejador del contexto de dispositivo para el cual se recuperan las dimensiones horizontal y vertical de la ventana.

*Size*: Puntero al registro *TSize* que recibe las dimensiones horizontal y vertical de la ventana asociada con el contexto de dispositivo especificado.

### Valores que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### Véase además

*GetViewportExtEx*, *GetViewportOrgEx*, *GetWindowOrgEx*, *SetViewportExtEx*, *SetViewportOrgEx*, *SetWindowExtEx*, *SetWindowOrgEx*

### Ejemplo

Vea el Listado 2-2 en la introducción.

## GetWindowOrgEx

## Windows.Pas

### Sintaxis

```
GetWindowOrgEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    var Point: TPoint        {origen de las coordenadas de la ventana}
); BOOL;                    {devuelve TRUE o FALSE}
```

### Descripción

La función *GetWindowOrgEx* recupera el origen de la ventana asociada con el contexto de dispositivo especificado.

### Parámetros

*DC*: Manejador del contexto de dispositivo para el cual se recuperan las coordenadas *x* e *y* del origen del sistema de coordenadas de la ventana asociada.

*Point*: Puntero al registro *TPoint* que recibe los valores *x* e *y* del origen del sistema de coordenadas de la ventana.

### Valores que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

Véase además

*GetViewportExtEx, GetViewportOrgEx, GetWindowExtEx, SetViewportExtEx, SetViewportOrgEx, SetWindowExtEx, SetWindowOrgEx*

**Ejemplo**

Vea el Listado 2-2 en la introducción.

## **LPTODP**

## **Windows.Pas**

**Sintaxis**

```
LPTODP(
    DC: HDC;                {manejador de contexto de dispositivo}
    var Points;              {puntero a un array de registros TPoint}
    Count: Integer           {número de elementos en el array}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

La función *LPTODP* convierte puntos en coordenadas lógicas a sus respectivos valores en coordenadas de dispositivo. El parámetro *Points* apunta a un *array* de registros *TPoint* que contiene las coordenadas que serán convertidas. Estos mismos registros *TPoint* contendrán las coordenadas convertidas cuando la función retorne. La transformación de las coordenadas es ejecutada basándose en los valores asignados por las funciones *SetWindowOrgEx, SetViewportOrgEx, SetWindowExtEx* y *SetViewportExtEx*. La función *LPTODP* fallará si alguno de los puntos en el *array* especifica un valor mayor de 27 bits. También fallará si alguna coordenada de alguno de los puntos transformados es superior a 32 bits en tamaño. En caso de fallo, los valores de todo el *array Points* serán indefinidos.

**Parámetros**

*DC*: El contexto de dispositivo para el cual se hará la transformación de coordenadas.

*Points*: Puntero a un *array* de registros *TPoint* que contiene las coordenadas de puntos que serán convertidas.

*Count*: Especifica el número de elementos que contiene el *array* al que apunta el parámetro *Points*.

**Valores que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, FALSE.

Véase además

*DPTOLP, SetWindowOrgEx, SetViewportOrgEx, SetWindowExtEx, SetViewportExtEx*

### Ejemplo

Vea el Listado 2-12 bajo *ScaleViewportExtEx*.

## MapWindowPoints Windows.Pas

### Sintaxis

```
MapWindowPoints(
    hWndFrom: HWND;           {manejador de la ventana fuente}
    hWndTo: HWND;             {manejador de la ventana destino}
    var lpPoints: UINT;        {puntero a un array de puntos}
    cPoints: UINT              {el tamaño del array}
): Integer;                   {devuelve desplazamientos de píxeles}
```

### Descripción

La función *MapWindowPoints* convierte un grupo de puntos de un sistema de coordenadas relativo a una ventana al sistema de coordenadas de otra ventana. Cualquier cantidad de puntos puede ser transformada con una sola llamada a esta función.

### Parámetros

*hWndFrom*: Manejador de la ventana de la cual los puntos serán traducidos. Los puntos listados en el parámetro *lpPoints* tienen coordenadas relativas a esta ventana. Si a este parámetro se le asigna **nil** o `HWND_DESKTOP`, los puntos serán relativos a la pantalla.

*hWndTo*: Manejador de la ventana para la cual los puntos serán traducidos. Si a este parámetro se le asigna **nil** o `HWND_DESKTOP`, los puntos serán relativos a la pantalla.

*lpPoints*: Un puntero a un *array* de registros *TPoint* que contiene las coordenadas de puntos a convertir. Cada registro *TPoint* recibe las coordenadas convertidas cuando la función retorna.

*cPoints*: Especifica el número de elementos en el *array* al que apunta el parámetro *lpPoints*.

### Valores que devuelve

Si la función tiene éxito, la palabra menos significativa del valor devuelto especifica el número de píxeles que son añadidos a la dimensión horizontal de las coordenadas, y la palabra más significativa el número de píxeles añadidos a la dimensión vertical. Si la función falla, devuelve cero.

### Véase además

*ClientToScreen*, *ScreenToClient*

*Ejemplo***Listado 2-II: Convirtiendo múltiples coordenadas de un sistema a otro**

```

var
  Form1: TForm1;
  DrawnRect: TRect;      // almacena las coordenadas del rectángulo
  Drawing: Boolean;      // indica si un rectángulo está siendo dibujado

implementation

{$R *.DFM}

procedure TForm1.PaintBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {Indicar que una operación de dibujo ha comenzado e inicializar las coordenadas
  del rectángulo}
  Drawing := TRUE;
  DrawnRect := Rect(X, Y, X, Y);
end;

procedure TForm1.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  {Si estamos redibujando...}
  if Drawing then
    with PaintBox1.Canvas do begin
      {Inicializar la brocha y la pluma del área de dibujo}
      Pen.Mode := pmNot;
      Pen.Width := 2;
      Brush.Style := bsClear;

      {Dibujar un rectángulo sobre el anterior para borrarlo}
      Rectangle(DrawnRect.Left, DrawnRect.Top, DrawnRect.Right, DrawnRect.Bottom);

      {Colocar el rectángulo en las coordenadas actuales}
      DrawnRect := Rect(DrawnRect.Left, DrawnRect.Top, X, Y);

      {Dibujar el nuevo rectángulo}
      Rectangle(DrawnRect.Left, DrawnRect.Top, DrawnRect.Right, DrawnRect.Bottom);
    end;
end;

procedure TForm1.PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {No dibujaremos más}
  Drawing := FALSE;

  {Mostrar las coordenadas relativas al panel}
  Label2.Caption := 'Panel coordinates - L: ' + IntToStr(DrawnRect.Left) + ', T: ' +
    IntToStr(DrawnRect.Top) + ', R: ' + IntToStr(DrawnRect.Right) +
    ', B: ' + IntToStr(DrawnRect.Bottom);

```

```
{Convertir las coordenadas del rectángulo relativas al formulario}
MapWindowPoints(Pane1.Handle, Form1.Handle, DrawnRect, 2);

{Mostrar las coordenadas relativas al formulario}
Label3.Caption := 'Form coordinates - L: ' + IntToStr(DrawnRect.Left) + ', T: ' +
  IntToStr(DrawnRect.Top) + ', R: ' + IntToStr(DrawnRect.Right) +
  ', B: ' + IntToStr(DrawnRect.Bottom);

end;
```

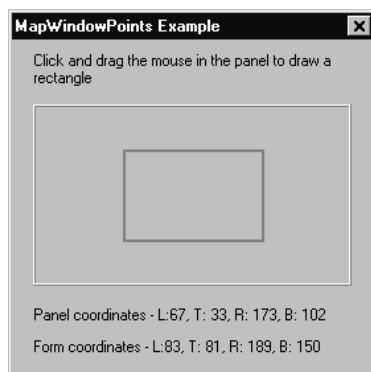


Figura 2-9:  
Los puntos  
trasladados

## OffsetViewportOrgEx Windows.Pas

### Sintaxis

```
OffsetViewportOrgEx(
  DC: HDC;           {manejador de contexto de dispositivo}
  X: Integer;         {desplazamiento horizontal}
  Y: Integer;         {desplazamiento vertical}
  Points: Pointer     {origen anterior}
): BOOL;             {devuelve TRUE o FALSE}
```

### Descripción

La función *OffsetViewportOrgEx* modifica el origen del puerto de visualización añadiéndole un valor a su ubicación actual. Los parámetros pueden especificar desplazamientos positivos o negativos en las direcciones horizontal o vertical. La ubicación del origen anterior es devuelto a través del parámetro *Points*. *OffsetViewportOrgEx* desplaza el origen del puerto de visualización a una nueva ubicación relativa a las coordenadas actuales. Para ubicar el origen en una posición absoluta independiente de la actual utilice *SetViewportOrgEx*.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuyo origen del puerto de visualización será modificado.

*X*: El desplazamiento horizontal que se añadirá o sustraerá del valor actual de *x* en el origen.

*Y*: El desplazamiento vertical que se añadirá o sustraerá del valor actual de *y* en el origen.

*Points*: Puntero a un registro *TPoint* que recibirá la ubicación anterior del origen. A este parámetro se le puede asignar **nil** si las coordenadas anteriores no son necesarias.

#### Valores que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, FALSE.

#### Véase además

*GetViewportOrgEx*, *OffsetWindowOrgEx*, *ScaleViewportExtEx*, *SetViewportOrgEx*

#### Ejemplo

Vea el Listado 2-12 bajo *ScaleViewportExtEx*.

### **OffsetWindowOrgEx**      **Windows.Pas**

#### Sintaxis

```
OffsetWindowOrgEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    X: Integer;              {desplazamiento horizontal}
    Y: Integer;              {desplazamiento vertical}
    Points: Pointer          {origen anterior}
): BOOL;                   {devuelve TRUE o FALSE}
```

#### Descripción

La función *OffsetWindowOrgEx* modifica el origen de la ventana añadiéndole un valor a su ubicación actual. Los parámetros pueden especificar desplazamientos positivos o negativos en las direcciones horizontal o vertical. La ubicación del origen anterior es devuelta a través del parámetro *Points*. *OffsetWindowOrgEx* mueve el origen a un nuevo valor relativo a las coordenadas existentes. Para ubicar el origen en una posición absoluta, independiente de la actual, utilice *SetWindowOrgEx*.

#### Parámetros

*DC*: Manejador del contexto de dispositivo asociado a la ventana cuyo origen será modificado.

*X*: El desplazamiento horizontal que se añadirá o sustraerá del valor actual de *x* en el origen.

*Y*: El desplazamiento vertical que se añadirá o sustraerá del valor actual de *y* en el origen.

*Points*: Puntero a un registro *TPoint* que recibirá la ubicación anterior del origen. A este parámetro se le puede asignar **nil** si el origen de coordenadas anterior se necesita.

#### Valores que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, FALSE.

#### Véase además

*GetViewportOrgEx*, *OffsetViewportOrgEx*, *ScaleWindowExtEx*, *SetViewportOrgEx*

#### Ejemplo

Vea el Listado 2-12 bajo *ScaleViewportExtEx*.

### ReleaseDC

### Windows.Pas

#### Sintaxis

```
ReleaseDC(
    hWnd: HWND;           {manejador de ventana}
    hDC: HDC              {contexto de dispositivo}
): Integer;              {devuelve cero o uno}
```

#### Descripción

La función *ReleaseDC* libera el contexto de dispositivo recuperado por las funciones *GetDC* o *GetWindowDC*, devolviendo esos recursos a Windows. La función *ReleaseDC* afecta sólo a contextos de dispositivo comunes. *ReleaseDC* no tiene efecto alguno en caso de contextos de dispositivo de clase o privados.

#### Parámetros

*hWnd*: Manejador de la ventana cuyo contexto de dispositivo será liberado.

*hDC*: Manejador del contexto de dispositivo que será liberado.

#### Valor que devuelve

Si la función tiene éxito, devuelve uno; en caso contrario, devuelve cero.

#### Véase además

*CreateCompatibleDC*, *DeleteDC*, *GetDC*, *GetWindowDC*

#### Ejemplo

Vea el Listado 2-5 bajo *CreateCompatibleDC* y otras funciones en este capítulo.

**RestoreDC      Windows.Pas****Sintaxis**

```
RestoreDC(
    DC: HDC;                {manejador de contexto de dispositivo}
    SavedDC: Integer         {estado que debe ser restaurado}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

La función *RestoreDC* restaura el estado, anteriormente almacenado, de un contexto de dispositivo. El estado de un contexto puede almacenarse llamando a la función *SaveDC*. La función *SaveDC* devuelve un valor que identifica al contexto de dispositivo almacenado; este valor debe utilizarse en el parámetro *SavedDC* para restaurar un estado requerido.

**Parámetros**

*DC*: Manejador de un contexto de dispositivo cuyo estado será restaurado. El contexto de dispositivo debe existir previamente y tener estados anteriormente almacenados con la función *SaveDC*.

*SavedDC*: Especifica el número de instancia del estado que será restaurado. Este valor es devuelto por la función *SaveDC* cuando el estado es originalmente almacenado. Puede utilizarse un valor negativo para restaurar un estado relativo al actual (por ejemplo, -1 recupera el estado más recientemente almacenado). Si el estado restaurado no es el más recientemente almacenado, todos los otros estados entre el más recientemente almacenado y el especificado son descartados.

**Valores que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*CreateCompatibleDC*, *GetDC*, *GetWindowDC*, *SaveDC*

**Ejemplo**

Vea el Listado 2-5 bajo *CreateCompatibleDC*.

**SaveDC      Windows.Pas****Sintaxis**

```
SaveDC(
    DC: HDC                {manejador de contexto de dispositivo}
): Integer;               {devuelve índice del estado almacenado}
```

### Descripción

Esta función almacena el estado actual del contexto de dispositivo especificado en una pila interna mantenida por Windows. Este estado incluye la información y los objetos gráficos asociados con el contexto de dispositivo, tales como mapas de bits, brochas, paletas, fuentes, el modo de dibujo, el modo de mapeado, etc. El estado puede ser recuperado utilizando la función *RestoreDC*.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuyo estado será almacenado.

### Valores que devuelve

Si la función tiene éxito, devuelve un valor que identifica el estado almacenado, que podrá ser usado en llamadas posteriores a la función *RestoreDC*. Si falla, devuelve cero.

### Véase además

*CreateCompatibleDC*, *GetDC*, *GetWindowDC*, *RestoreDC*

### Ejemplo

Vea el Listado 2-5 bajo *CreateCompatibleDC*.

## ScaleViewportExtEx

## Windows.Pas

### Sintaxis

```
ScaleViewportExtEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    XM: Integer;             {multiplicador horizontal}
    XD: Integer;             {divisor horizontal}
    YM: Integer;             {multiplicador vertical}
    YD: Integer;             {divisor vertical}
    Size: PSize              {puntero a la dimensión anterior}
): BOOL;                   {devuelve TRUE o FALSE}
```

### Descripción

La función *ScaleViewportExtEx* modifica las dimensiones del puerto de visualización asociado con el contexto de dispositivo especificado en correspondencia con los factores de escala especificados. Para las dimensiones horizontal y vertical, se puede especificar un parámetro divisor y otro multiplicador para hacer las dimensiones más grandes o más pequeñas en esas direcciones. A los parámetros que no se utilicen se les deberá asignar el valor 1. Por ejemplo, para hacer la dimensión horizontal la mitad de su valor actual es necesario asignar 2 al parámetro *XD*. Todos los demás parámetros deberán ser 1. Para hacer la dimensión horizontal tres cuartos de su valor actual, el parámetro *XM* debe valer 3 y el parámetro *XD* debe valer 4.

**Parámetros**

*DC*: Manejador del contexto de dispositivo para cuyo puerto de visualización asociado serán escaladas las dimensiones.

*XM*: El multiplicador de la dimensión horizontal.

*XD*: El divisor de la dimensión horizontal.

*YM*: El multiplicador de la dimensión vertical.

*YD*: El divisor de la dimensión vertical.

*Size*: Puntero a un registro *TSize* que recibe los valores previos de las dimensiones.

**Valores que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*GetViewportExtEx*, *ScaleWindowExtEx*

**Ejemplo****Listado 2-12: Escalando puertos de visualización y ventanas**

```
{iOJO! Delphi importa estas funciones incorrectamente; las importamos manualmente}
function OffsetViewportOrgEx(DC: HDC; X, Y: Integer;
    Points: Pointer): BOOL; stdcall;
function OffsetWindowOrgEx(DC: HDC; X, Y: Integer;
    Points: Pointer): BOOL; stdcall;
```

**var**

```
Form1: TForm1;
MyDisplayDC: HDC;           // manejador del contexto de dispositivo
MyMapMode: Integer;         // almacena el modo de mapeado
PrevWindowSize: TSize;      // almacena el tamaño anterior de la ventana
PrevViewportSize: TSize;    // almacena el tamaño anterior del puerto
PrevWindowPoint: TPoint;    // almacena el origen anterior de la ventana
PrevViewportPoint: TPoint;  // almacena el origen anterior del puerto
```

**implementation**

```
{$R *.DFM}
```

```
{Re-importar las funciones correctamente}
```

```
function OffsetViewportOrgEx; external gdi32 name 'OffsetViewportOrgEx';
```

```
function OffsetWindowOrgEx; external gdi32 name 'OffsetWindowOrgEx';
```

```
procedure TForm1.ReportPosition;
```

**var**

```
ReturnValue: TPoint;        // almacena el origen de la ventana y el puerto
ReturnSize: TSize;         // almacena dimensiones de la ventana y el puerto
```

```
begin
```

```

{Mostrar el origen de la ventana}
GetWindowOrgEx(MyDisplayDC, ReturnValue);
Label9.Caption := IntToStr(ReturnValue.x)
                + ', ' + IntToStr(ReturnValue.y);

{Mostrar el origen del puerto de visualización}
GetViewportOrgEx(MyDisplayDC, ReturnValue);
Label10.Caption := IntToStr(ReturnValue.x)
                 + ', ' + IntToStr(ReturnValue.y);

{Mostrar las dimensiones de la ventana}
GetWindowExtEx(MyDisplayDC, ReturnSize);
Label11.Caption := IntToStr(ReturnSize.cx)
                  + ', ' + IntToStr(ReturnSize.cy);

{Mostrar las dimensiones del puerto de visualización}
GetViewportExtEx(MyDisplayDC, ReturnSize);
Label12.Caption := IntToStr(ReturnSize.cx)
                  + ', ' + IntToStr(ReturnSize.cy);
end;

procedure TForm1.ReadUserRequest;
begin
    {Recuperar el modo de mapeado seleccionado}
    case RadioGroup1.ItemIndex of
        0: MyMapMode := MM_TEXT;
        1: MyMapMode := MM_ANISOTROPIC;
        2: MyMapMode := MM_ISOTROPIC;
        3: MyMapMode := MM_HIENGLISH;
        4: MyMapMode := MM_HIMETRIC;
        5: MyMapMode := MM_LOENGLISH;
        6: MyMapMode := MM_LOMETRIC;
        7: MyMapMode := MM_TWIPS;
    end;
end;

procedure TForm1.PaintImage;
begin
    {Borrar la imagen previa}
    Panel1.Repaint;

    {Atrapar los valores de los controles del usuario}
    ReadUserRequest;

    {Asignar el modo de mapeado según el grupo de botones de radio}
    SetMapMode(MyDisplayDC, MyMapMode);

    {Desplazar el origen de la ventana en la cantidad especificada}
    OffsetWindowOrgEx(MyDisplayDC,
        StrToInt(EditOWX.text), StrToInt(EditOWY.text), @PrevWindowPoint);

    {Desplazar el origen del puerto de visualización en la cantidad especificada}
    OffsetViewportOrgEx(MyDisplayDC,
        StrToInt(EditOVX.text), StrToInt(EditOVY.text), @PrevViewportPoint);

```

```

    {Escalar las dimensiones de la ventana en la cantidad especificada}
    ScaleWindowExtEx(MyDisplayDC,
        StrToInt(EditSWEXM.text), StrToInt(EditSWEXD.text),
        StrToInt(EditSWEYM.text), StrToInt(EditSWEYD.text), @PrevWindowSize);

    {Escalar las dimensiones del puerto de visualización en la cantidad especificada}
    ScaleViewportExtEx(MyDisplayDC,
        StrToInt(EditSVEXM.text), StrToInt(EditSVEXD.text),
        StrToInt(EditSVEYM.text), StrToInt(EditSVEYD.text), @PrevViewportSize);

    {Dibujar la imagen. Observe que las coordenadas usadas son fijas y no cambian,
    demostrando como el origen y las dimensiones de la ventana afectan a las
    operaciones de dibujo}
    Windows.Rectangle(MyDisplayDC, 0, 0, 50, 50);
    Windows.Rectangle(MyDisplayDC, -25, 24, 75, 26);
    Windows.Rectangle(MyDisplayDC, 24, -25, 26, 75);

    {Mostrar los valores del nuevo origen y la dimensión del mismo}
    ReportPosition;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Dibujar la imagen}
    PaintImage;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Recuperar un manejador del contexto de dispositivo del panel}
    MyDisplayDC := GetDC(Pane11.handle);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Liberar el contexto de dispositivo}
    ReleaseDC(Pane11.handle, MyDisplayDC);
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
var
    MyPoint: TPoint; // almacena los puntos convertidos
begin
    {Convierte unidades de dispositivo a unidades lógicas con DPtoLP}
    MyPoint.X := StrToInt(EditDevX.text);
    MyPoint.Y := StrToInt(EditDevY.text);

    {Chequeo de errores}
    if not DPtoLP(MyDisplayDC, MyPoint, 1) then
        ShowMessage('Error in device coordinates')
    else begin
        {MyPoint ahora contiene coordenadas lógicas convertidas}
        EditLogX.text := IntToStr(MyPoint.X);
        EditLogY.text := IntToStr(MyPoint.Y);
    end;

```

```

end;
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
var
  MyPoint: TPoint;    // almacena puntos convertidos
begin
  {Convertir unidades de dispositivo a unidades lógicas con DPTOLP}
  MyPoint.X := StrToInt(EditLogX.Text);
  MyPoint.Y := StrToInt(EditLogY.Text);

  {Chequeo de errores}
  if not LPtoDP(MyDisplayDC, MyPoint, 1) then
    ShowMessage('Error in logical coordinates')
  else begin
    {MyPoint ahora contiene coordenadas de dispositivo convertidas}
    EditDevX.Text := IntToStr(MyPoint.X);
    EditDevY.Text := IntToStr(MyPoint.Y);
  end;
end;
end;

```

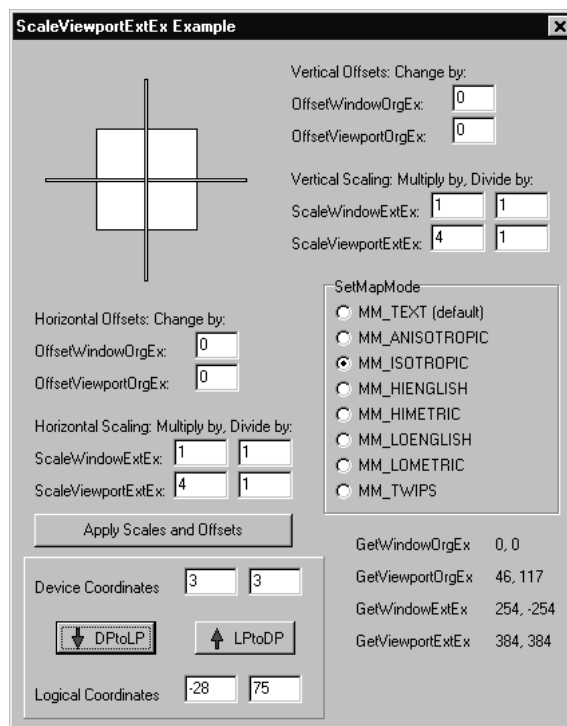


Figura 2-10:  
La imagen  
escalada

**ScaleWindowExtEx****Windows.Pas****Sintaxis**

```
ScaleWindowExtEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    XM: Integer;             {el multiplicador horizontal}
    XD: Integer;             {el divisor horizontal}
    YM: Integer;             {el multiplicador vertical}
    YD: Integer;             {el divisor vertical}
    Size: PSize              {puntero a la dimensión anterior}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

La función *ScaleWindowExtEx* modifica las dimensiones de la ventana asociada con el contexto de dispositivo especificado, de acuerdo a los factores de escala especificados. Para las dimensiones horizontal y vertical, se puede especificar un parámetro multiplicador y uno divisor para hacer la dimensión más grande o más pequeña en esas direcciones. A los parámetros que no sean utilizados se les debe asignar el valor 1. Por ejemplo, para hacer la dimensión horizontal la mitad de su valor actual es necesario asignar 2 al parámetro *XD*. Todos los demás parámetros deberán valer 1. Para hacer la dimensión horizontal tres cuartos de su valor actual, al parámetro *XM* se le debe asignar 3 y al parámetro *XD* 4.

**Parámetros**

*DC*: Manejador del contexto de dispositivo para cuya ventana asociada serán escaladas las dimensiones.

*XM*: El multiplicador de la dimensión horizontal.

*XD*: El divisor de la dimensión horizontal.

*YM*: El multiplicador de la dimensión vertical.

*YD*: El divisor de la dimensión vertical.

*Size*: Puntero al registro *TSize* que recibirá los valores anteriores de las dimensiones.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*GetWindowExtEx*, *ScaleViewportExtEx*

**Ejemplo**

Vea el Listado 2-12 bajo *ScaleViewportExtEx*.

**ScreenToClient**     **Windows.Pas****Sintaxis**

```
ScreenToClient(
    hWnd: HWND;           {manejador de ventana}
    var lpPoint: TPoint    {puntero a registro TPoint}
); BOOL;                 {devuelve TRUE o FALSE}
```

**Descripción**

Esta función convierte las coordenadas de un punto en coordenadas de pantalla a coordenadas de área cliente. El punto a convertir se suministra en un registro *TPoint* al que apunta el parámetro *lpPoint*. La función toma las coordenadas a las que apunta el parámetro *lpPoint* y las convierte en coordenadas relativas al área cliente de la ventana especificada. El resultado es ubicado en el mismo registro *TPoint*. Las coordenadas del punto original utilizan como origen la esquina superior izquierda de la pantalla. Las coordenadas del resultado utilizan como origen la esquina superior izquierda del área cliente de la ventana.

**Parámetros**

*hWnd*: Manejador de la ventana a cuyas coordenadas se convierte el punto. Cuando la función retorne, el punto será relativo a la esquina superior izquierda del área cliente de la ventana.

*lpPoint*: Puntero a un registro *TPoint* que contiene el punto que será convertido. Este mismo registro *TPoint* recibe el punto convertido cuando la función retorna.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*ClientToScreen*, *MapWindowPoints*

**Ejemplo**

Vea el Listado 2-4 bajo *ClientToScreen*.

**ScrollDC**     **Windows.Pas****Sintaxis**

```
ScrollDC(
    DC: HDC;               {manejador de contexto de dispositivo}
    DX: Integer;           {el incremento del desplazamiento horizontal}
    DY: Integer;           {el incremento del desplazamiento vertical}
    var Scroll: TRect;      {el rectángulo de desplazamiento}
    Clip: TRect;           {el rectángulo que se recorta}
    Rgn: HRGN;             {la región expuesta}
```

```
Update: PRect          {el rectángulo expuesto}
): BOOL;              {devuelve TRUE o FALSE}
```

### Descripción

La función *ScrollDC* desplaza (*scrolls*) un rectángulo de bits horizontal y verticalmente. El desplazamiento está dado en unidades del dispositivo.

### Parámetros

*DC*: Manejador del contexto de dispositivo que contiene el rectángulo cuyos bits serán desplazados.

*DX*: El número de unidades de dispositivo horizontales que serán desplazados. Este valor es positivo para el desplazamiento a la derecha y negativo para el desplazamiento a la izquierda.

*DY*: El número de unidades del dispositivo verticales que serán desplazadas. Este valor es positivo para el desplazamiento hacia abajo y negativo para el desplazamiento hacia arriba.

*Scroll*: Especifica un registro *TRect* que contiene la ubicación del rectángulo que será desplazado.

*Clip*: Especifica un registro *TRect* que contiene la ubicación del rectángulo de recorte.

*Rgn*: Especifica el manejador de la región que el desplazamiento descubrirá. Esta región no tiene necesariamente que ser un rectángulo. A este parámetro se le puede asignar cero si la región de actualización no es necesaria.

*Update*: Un puntero a un registro *TRect* que recibirá la ubicación del rectángulo descubierto por el área desplazada. Las coordenadas de este rectángulo están dadas en coordenadas de cliente, independientemente del modo de mapeado actual del contexto de dispositivo. A este parámetro se le puede asignar **nil** si el rectángulo de actualización no se necesita.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### Véase además

*InvalidateRect*, *InvalidateRgn*, *BitBlt*

### Ejemplo

#### Listado 2-13: Desplazando una imagen dentro de un área de visualización

```
var
  Form1: TForm1;
  PreviousX, PreviousY: Integer;    // desplazamiento anterior

implementation
```

```
{ $R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Inicializar la barra de desplazamiento}
    ScrollBar1.Max := Image1.Picture.Bitmap.Width - Image1.Width;
    ScrollBar2.Max := Image1.Picture.Bitmap.Height - Image1.Height;

    {Inicializar las variables de seguimiento del desplazamiento}
    PreviousX := 0;
    PreviousY := 0;
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
var
    ScrollRect,           // el rectángulo que será desplazado
    ClipRect,            // el rectángulo de recorte del área desplazada
    UpdateRect: TRect;    // el área descubierta por el desplazamiento
begin
    {Inicializar rectángulo a desplazar y el de recorte al área entera de la imagen}
    ScrollRect := Image1.BoundsRect;
    ClipRect := Image1.BoundsRect;

    {Desplazar el área horizontalmente en la cantidad especificada}
    ScrollDC(Canvas.Handle, PreviousX - ScrollBar1.Position, 0, ScrollRect,
        ClipRect, 0, @UpdateRect);

    {Copiar el área apropiada del mapa de bits original en el área descubierta}
    Canvas.CopyRect(UpdateRect, Image1.Picture.Bitmap.Canvas,
        Rect((UpdateRect.Left - Image1.Left) + ScrollBar1.Position,
        ScrollBar2.Position, (UpdateRect.Left - Image1.Left) +
        ScrollBar1.Position + (UpdateRect.Right - UpdateRect.Left),
        Image1.Height+ScrollBar2.Position));

    {Registrar la posición actual}
    PreviousX := ScrollBar1.Position;
end;

procedure TForm1.ScrollBar2Change(Sender: TObject);
var
    ScrollRect,           // el rectángulo que será desplazado
    ClipRect,            // el rectángulo de recorte del área desplazada
    UpdateRect: TRect;    // el área descubierta por el desplazamiento
begin
    {Inicializar el rectángulo a desplazar y el de recorte al área de la imagen}
    ScrollRect := Image1.BoundsRect;
    ClipRect := Image1.BoundsRect;

    {Desplazar el área verticalmente en la cantidad especificada}
    ScrollDC(Canvas.Handle, 0, PreviousY-ScrollBar2.Position, ScrollRect,
        ClipRect, 0, @UpdateRect);

    {Copiar el área apropiada del mapa de bits original en el área descubierta}
    Canvas.CopyRect(UpdateRect, Image1.Picture.Bitmap.Canvas,
```

```

Rect(ScrollBar1.Position, (UpdateRect.Top-Image1.Top) +
ScrollBar2.Position, Image1.Width + ScrollBar1.Position,
(UpdateRect.Top - Image1.Top) + ScrollBar2.Position +
(UpdateRect.Bottom - UpdateRect.Top));

{Registrar la posición actual}
PreviousY := ScrollBar2.Position;
end;

```

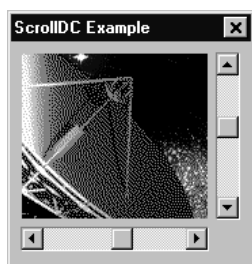


Figura 2-11:  
La imagen  
desplazada

## SetMapMode Windows.Pas

### Sintaxis

```

SetMapMode(
  DC: HDC;           {manejador de contexto de dispositivo}
  p2: Integer        {modo de mapeado}
): Integer;          {devuelve el modo de mapeado anterior}

```

### Descripción

Esta función asigna un nuevo modo de mapeado de unidades gráficas en el contexto de dispositivo especificado. Las unidades pueden ser medidas en píxeles, pulgadas, milímetros o unidades de impresora. La orientación de los ejes *x* e *y* también puede ser seleccionada. Esta función se utiliza para determinar cómo las unidades de medida definidas por *software* son mapeadas a dispositivos gráficos físicos.

### Parámetros

*DC*: Manejador del contexto de dispositivo al que se le asignará un nuevo modo de mapeado.

*p2*: Opción que indica el nuevo modo de mapeado. Este parámetro puede tomar un valor de la Tabla 2-17.

### Valor que devuelve

Si la función tiene éxito, devuelve el valor del modo de mapeado anterior, que será uno de los valores de la Tabla 2-17. Si falla, devuelve cero.

Véase además

*GetMapMode, SetViewportExtEx, SetViewportOrgEx, SetWindowExtEx, SetWindowOrgEx*

### Ejemplo

Vea el Listado 2-2 en la introducción.

**Tabla 2-17: Valores p2 de SetMapMode**

Valor	Descripción
MM_ANISOTROPIC	Las unidades, escala y orientación son seleccionados por SetWindowExtEx y SetViewportExtEx. Los ejes de escalamiento x e y son tratados de forma independiente y los valores correspondientes no tienen que coincidir.
MM_HIENGLISH	Mapeado de alta resolución en unidades inglesas. Cada unidad tiene 0,001 pulgadas. Las x crecen hacia la derecha y las y hacia arriba.
MM_HIMETRIC	Mapeado de alta resolución en unidades métricas decimales. Cada unidad tiene 0,01 milímetros. Las x crecen hacia la derecha y las y hacia arriba.
MM_ISOTROPIC	Las unidades, escala y orientación son asignados por SetWindowExtEx y SetViewportExtEx con las unidades verticales y horizontales iguales. Las unidades y orientación pueden ser elegidos por el programador, pero las unidades para los ejes x e y están obligadas por el GDI a ser las mismas. Esto asegura una relación de aspecto 1:1.
MM_LOENGLISH	Mapeado de baja resolución en unidades inglesas. Cada unidad tiene 0,01 de pulgada. Las x crecen hacia la derecha y las y hacia arriba.
MM_LOMETRIC	Mapeado de baja resolución en unidades métricas decimales. Cada unidad tiene 0,1 de pulgada. Las x crecen hacia la derecha y las y hacia arriba.
MM_TEXT	Cada unidad es mapeada a un píxel del dispositivo. Este no es un mapeado independiente del dispositivo. Dispositivos con diferentes resoluciones o escalamientos producirán diferentes resultados de las funciones gráficas. Las x crecen hacia la derecha y las y hacia abajo. Este es el modo de mapeado por defecto.
MM_TWIPS	Cada unidad es mapeada a 1/1440 de pulgada, que corresponde a un veinteavo de un punto de impresora. Las x crecen hacia la derecha y las y hacia arriba.

**SetViewportExtEx****Windows.Pas****Sintaxis**

```
SetViewportExtEx(
    DC: HDC;           {manejador de contexto de dispositivo}
    XExt: Integer;      {nueva dimensión horizontal}
    YExt: Integer;      {nueva dimensión vertical}
    Size: PSize         {puntero a dimensiones anteriores}
); BOOL;              {devuelve TRUE o FALSE}
```

**Descripción**

Esta función establece un nuevo tamaño para el puerto de visualización asociado con el contexto de dispositivo especificado. Las llamadas a esta función son válidas solamente cuando la función *SetMapMode* ha asignado el modo de mapeado del contexto de dispositivo a *MM\_ANISOTROPIC* o *MM\_ISOTROPIC*. Las llamadas a *SetViewportExtEx* son ignoradas para cualquier otro modo de mapeado. En el caso de *MM\_ISOTROPIC*, una llamada a la función *SetWindowExtEx* tiene que realizarse antes de que la función *SetViewportExtEx* sea usada.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo tamaño del puerto de visualización asociado será modificado.

*XExt*: El nuevo tamaño horizontal del puerto de visualización en unidades del dispositivo.

*YExt*: El nuevo tamaño vertical del puerto de visualización en unidades del dispositivo.

*Size*: Puntero a un registro *TSize* que recibirá el tamaño anterior del puerto de visualización. Si las dimensiones anteriores no se necesitan, a este parámetro se le puede asignar **nil**.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*GetMapMode*, *GetViewportExtEx*, *SetMapMode*, *SetWindowExtEx*

**Ejemplo**

Vea el Listado 2-2 en la introducción.

**SetViewportOrgEx****Windows.Pas****Sintaxis**

```
SetViewportOrgEx(
    DC: HDC;                {manejador del contexto de dispositivo}
    X: Integer;              {nuevo valor x del origen}
    Y: Integer;              {nuevo valor y del origen}
    Point: PPoint            {puntero a los valores anteriores del origen}
); BOOL;                    {devuelve TRUE o FALSE}
```

**Descripción**

Esta función establece un nuevo origen de coordenadas para el contexto de dispositivo especificado. El dispositivo normalmente asigna su origen a la esquina superior izquierda de la imagen mostrada en el monitor o impresa. La función *SetViewportOrgEx* puede ser útil al dibujar funciones con valores negativos. Localizando el origen del contexto de dispositivo en el punto donde se coloca el origen de un gráfico (generalmente en la esquina inferior izquierda), una aplicación puede evitar tener que realizar una transformación de coordenadas para cada uno de los puntos a dibujar. Debido a un pequeño gasto de recursos en que incurre el GDI al aceptar un nuevo origen de sistema de coordenadas, la velocidad de ejecución de la aplicación deberá ser probada cuando se decida utilizar esta función. Una aplicación que ya hace cálculos para situar puntos en un contexto de dispositivo, podría ejecutarse más rápidamente si el dispositivo origen permanece en la posición por defecto mientras la aplicación calcula la posición del dispositivo origen.

**Parámetros**

*DC*: Manejador del contexto de dispositivo a cuyo puerto de visualización le será modificado el origen.

*X*: La ubicación horizontal del nuevo origen en unidades del dispositivo.

*Y*: La ubicación vertical del nuevo origen en unidades del dispositivo.

*Point*: Puntero a un registro *TPoint* que recibirá la ubicación del origen previo en unidades del dispositivo. Si la ubicación anterior del origen no se necesita, a este parámetro se le puede asignar **nil**.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*GetViewportOrgEx*, *SetWindowOrgEx*

**Ejemplo**

Vea el Listado 2-2 en la introducción.

**SetWindowExtEx****Windows.Pas****Sintaxis**

```
SetWindowExtEx(
    DC: HDC;           {manejador de contexto de dispositivo}
    XExt: Integer;      {nueva dimensión horizontal}
    YExt: Integer;      {nueva dimensión vertical}
    Size: PSize         {puntero a dimensiones originales}
): BOOL;              {devuelve TRUE o FALSE}
```

**Descripción**

Esta función establece un nuevo tamaño para la ventana asociada con el contexto de dispositivo especificado. La llamada a esta función sólo es válida cuando *SetMapMode* ha asignado el modo de mapeado del dispositivo a *MM\_ANISOTROPIC* o *MM\_ISOTROPIC*. Las llamadas a *SetWindowExtEx* son ignoradas para otros modos de mapeado. En el caso de *MM\_ISOTROPIC*, una llamada a *SetWindowExtEx* tiene que ser efectuada antes de que la función *SetViewportExtEx* sea usada.

**Parámetros**

*DC*: Manejador del contexto de dispositivo a cuya ventana asociada se le modificará el tamaño.

*XExt*: El nuevo tamaño horizontal de la ventana, en unidades del dispositivo.

*YExt*: El nuevo tamaño vertical de la ventana, en unidades del dispositivo

*Size*: Puntero a un registro *TSize* que recibe el tamaño previo de la ventana. Si las dimensiones anteriores no se necesitan, a este parámetro se le puede asignar **nil**.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*GetViewportExtEx*, *GetViewportOrgEx*, *GetWindowExtEx*, *GetWindowOrgEx*, *SetViewportExtEx*, *SetViewportOrgEx*, *SetWindowOrgEx*

**Ejemplo**

Vea el Listado 2-2 en la introducción.

**SetWindowOrgEx****Windows.Pas****Sintaxis**

```
SetWindowOrgEx(
    DC: HDC;           {manejador de contexto de dispositivo}
    X: Integer;         {nueva ubicación horizontal del origen}
```

Y: Integer;	{nueva ubicación vertical del origen}
Point: PPoint	{puntero a los valores anteriores del origen}
): BOOL;	{devuelve TRUE o FALSE}

### Descripción

*SetWindowOrgEx* establece un nuevo origen del sistema de coordenadas para la ventana asociada al contexto de dispositivo especificado. Una ventana tiene normalmente su origen en su esquina superior izquierda. La función *SetWindowOrgEx* puede ser útil al dibujar funciones con valores negativos. Haciendo coincidir el origen de la ventana con el origen del gráfico, una aplicación puede evitar tener que realizar una transformación de coordenadas para cada uno de los puntos a dibujar. Debido a un pequeño gasto de recursos en que incurre el GDI al aceptar un nuevo origen de sistema de coordenadas, la velocidad de ejecución de la aplicación debe ser probada cuando se decida utilizar esta función. Una aplicación que ya hace cálculos para situar puntos en un contexto de dispositivo, podría ejecutarse más rápidamente si la ventana origen permanece en la posición por defecto mientras la aplicación calcula la posición de la ventana origen.

### Parámetros

*DC*: Manejador del contexto de dispositivo a cuya ventana asociada le será modificado el origen.

*X*: La ubicación horizontal del nuevo origen, en unidades del dispositivo.

*Y*: La ubicación vertical del nuevo origen, en unidades del dispositivo.

*Point*: Puntero a un registro *TPoint* que recibirá la ubicación anterior del origen, en unidades del dispositivo. Si las coordenadas de origen anteriores no se necesitan, a este parámetro se le puede asignar **nil**.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### Véase además

*GetViewportOrgEx*, *GetWindowOrgEx*, *SetViewportOrgEx*

### Ejemplo

Vea el Listado 2-2 en la introducción.



**Capítulo 3**

# Funciones de dibujo

Windows ofrece una amplia gama de funciones para dibujar gráficos simples y primitivas gráficas. Windows es muy potente a la hora de efectuar manipulación de gráficos de alto nivel, como resulta evidente dada la cantidad de programas para la manipulación de imágenes digitales que hay en el mercado. Sin embargo, dibujar un simple gráfico es a veces la forma más eficiente de comunicarse con el usuario. Por ejemplo, se puede trazar un simple rectángulo alrededor de una zona de una imagen para indicar que ésta ha sido seleccionada. El propio Windows hace un uso extensivo de las funciones que se describen en este capítulo a la hora de dibujar los elementos estándar de interfaz de usuario y los controles comunes. Delphi encapsula una buena parte de las funciones presentadas en este capítulo en forma de métodos y propiedades de la clase *TCanvas*. No obstante, la complejidad de la VCL a veces se interpone en el camino y en esos casos, descender hasta el nivel del API de Windows es el único modo de seguir adelante. Este capítulo describe las funciones más comunes que pueden utilizarse para dibujar gráficos simples sobre un contexto de dispositivo.

## Objetos gráficos

Con vistas a dibujar sobre un contexto de dispositivo, Windows necesita un método para saber qué se supone que debe dibujar: de qué color será el objeto, sus dimensiones, etc. Estos atributos están encapsulados en un *objeto gráfico*. Todas las funciones del tipo *CreateXXX*, tales como *CreateBrush*, *CreateBitmap* o *CreatePen*, devuelven un manejador de un objeto gráfico. Internamente, este manejador hace referencia a una estructura que contiene los valores actuales de los atributos del objeto.

Para utilizar la mayoría de los objetos gráficos, éstos necesitan ser seleccionados dentro de un contexto de dispositivo mediante la función *SelectObject*. Una vez que un objeto ha sido seleccionado en el contexto de dispositivo, es automáticamente utilizado por aquellas funciones que necesitan ese objeto específicamente. Por ejemplo, es posible crear una nueva pluma llamando a la función *CreatePen* y seleccionarla dentro de un contexto de dispositivo. De ahí en adelante, cualquier función de dibujo aplicada a ese contexto que necesite una pluma utilizará automáticamente la pluma seleccionada. Solamente un objeto de un tipo dado puede estar seleccionado en un contexto de dispositivo en un momento dado.

Es importante eliminar un objeto gráfico cuando éste ya no es necesario. Cada manejador de objeto gráfico representa una cierta cantidad de memoria y recursos que se sustraen de los recursos generales del sistema. Si bien la escasez de recursos es un problema de menor importancia en Windows 95/98 y Windows NT con relación a las versiones anteriores de Windows, la falta de memoria puede afectar al rendimiento del sistema si no se eliminan los objetos gráficos adecuadamente. Para eliminar un objeto gráfico, primeramente éste debe ser *deseleccionado* de su contexto de dispositivo. Esto se logra almacenando un manejador del objeto anterior cuando el nuevo objeto es seleccionado y volviendo a seleccionar el objeto original cuando el nuevo va a ser descartado. Una vez que el objeto no está seleccionado en ningún contexto de dispositivo, puede ser eliminado mediante una llamada a la función *DeleteObject*.

## Plumas y Brochas

Los objetos más utilizados en las funciones de dibujo de Windows son probablemente las plumas y brochas. Una *brocha* define un color y un patrón que se usarán al rellenar el interior de figuras cerradas, como polígonos, rectángulos, rutas y regiones. Una *pluma* define un color y patrón usado para delinear figuras, abiertas o cerradas. Estos dos objetos gráficos son encapsulados por Delphi como las propiedades *Pen* y *Brush* de la clase *TCanvas*. Delphi abarca toda la funcionalidad ofrecida por las funciones de brocha de Windows. Sin embargo, Windows ofrece dos estilos de pluma, cósmico y geométrico, y Delphi no encapsula actualmente toda la funcionalidad de estos objetos.

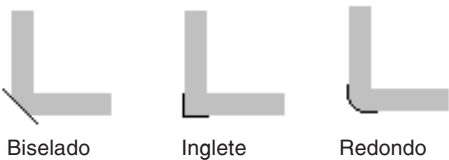
**Plumas cósmicas** Una pluma cósmica se mide en unidades del dispositivo y no puede ser escalada. Actualmente, Windows soporta plumas cósmicas de un solo píxel de ancho. El estilo de pluma se puede seleccionar entre varios patrones diferentes, que van desde el sólido hasta una amplia variedad de diferentes combinaciones de guiones y puntos. Las plumas cósmicas son mucho más rápidas que las plumas geométricas.

**Plumas Geométricas** Una pluma geométrica se mide en unidades lógicas y puede ser escalada. Soporta los mismos estilos de plumas disponibles para las plumas cósmicas, pero además permite estilos definidos por el usuario y estilos normalmente accesibles únicamente a las brochas. Adicionalmente, las plumas geométricas pueden aplicar un *estilo de punto final* a los finales de líneas y un *estilo de unión* donde dos líneas se unen. Los posibles estilos de punto final y de unión se ilustran en las figuras 3-1 y 3-2. Tenga en cuenta que bajo Windows 95/98, las plumas geométricas no soportan estilos de plumas definidos por el usuario, no pueden usar la mayoría de los estilos de las plumas cósmicas y pueden ser usadas solamente para dibujar rutas.

Figura 3-1:  
Estilo de  
terminación  
de líneas  
geométricas



Figura 3-2:  
Estilo de unión  
de líneas  
geométricas



### Funciones de dibujo

En el siguiente capítulo se describen las siguientes funciones de dibujo:

Tabla 3-I: Funciones de dibujo

Función	Descripción
Arc	Dibuja un arco.
BeginPaint	Comienza una operación de dibujo.
Chord	Dibuja una cuerda.
CreateBrushIndirect	Crea una brocha a partir de un registro de datos.
CreateHatchBrush	Crea una brocha de patrón basada en una trama.
CreatePatternBrush	Crea una brocha de patrón.
CreatePen	Crea una pluma.
CreatePenIndirect	Crea una pluma a partir de un registro de datos.
CreateSolidBrush	Crea una brocha de color sólido.
DeleteObject	Elimina un objeto gráfico.
DrawCaption	Dibuja una barra de título.
DrawEdge	Dibuja líneas tridimensionales.
DrawFocusRect	Dibuja un rectángulo de foco.
DrawFrameControl	Dibuja botones estándar de la interfaz de usuario.
DrawState	Dibuja texto o gráficos deshabilitados.
Ellipse	Dibuja una elipse.
EndPaint	Termina una operación de pintura.
EnumObjects	Enumera objetos.
ExtCreatePen	Crea plumas cosméticas o geométricas.
ExtFloodFill	Rellena un área con un color.
FillPath	Rellena una ruta con un color.
FillRect	Rellena un rectángulo con un color.
FillRgn	Rellena una región con un color.
FrameRect	Dibuja el perímetro de un rectángulo.
FrameRgn	Dibuja el perímetro de una región.
GetBkColor	Recupera el color de fondo de un contexto de dispositivo.
GetBkMode	Recupera el modo de fondo de un contexto de dispositivo.

Función	Descripción
GetBoundsRect	Recupera el rectángulo límite acumulado de un contexto de dispositivo.
GetBrushOrgEx	Recupera el origen de un patrón de brocha.
GetCurrentObject	Recupera el objeto seleccionado en un contexto de dispositivo.
GetCurrentPositionEx	Recupera la posición actual de dibujo de un contexto de dispositivo.
GetMiterLimit	Recupera el límite del inglete en líneas unidas mediante inglete.
GetObject	Recupera información sobre un objeto gráfico.
GetObjectType	Determina el tipo de un objeto gráfico.
GetPixel	Recupera el color de un píxel.
GetPolyFillMode	Recupera el modo de relleno actual de un polígono.
GetROP2	Recupera el modo de mezcla del primer plano de un contexto de dispositivo.
GetStockObject	Recupera un manejador de un objeto gráfico predefinido.
GetUpdateRect	Recupera el rectángulo límite de la región de actualización actual.
GetUpdateRgn	Recupera la región de actualización actual.
GrayString	Dibuja una cadena con un color convertido.
InvalidateRect	Invalida un área rectangular.
InvalidateRgn	Invalida una región.
LineDDA	Dibuja una línea a la medida.
LineTo	Dibuja una línea.
LockWindowUpdate	Deshabilita el redibujado de una ventana.
MoveToEx	Mueve la posición actual del contexto de dispositivo.
PaintDesktop	Pinta el papel tapiz del fondo del escritorio sobre un contexto de dispositivo.
PaintRgn	Rellena una región con la brocha en uso.
Pie	Dibuja una cuña de tarta.
PolyBezier	Dibuja una curva de Bézier.
PolyBezierTo	Dibuja múltiples curvas de Bézier.
Polygon	Dibuja un polígono relleno.
Polyline	Dibuja un polígono.
PolylineTo	Dibuja un polígono, actualizando la posición actual.
PolyPolygon	Dibuja múltiples polígonos rellenos.
PolyPolyline	Dibuja múltiples polígonos.
Rectangle	Dibuja un rectángulo.
RoundRect	Dibuja un rectángulo redondeado.
SelectObject	Selecciona un objeto gráfico en un contexto de dispositivo.

Función	Descripción
SetBkColor	Establece el color de fondo de un contexto de dispositivo.
SetBkMode	Establece el modo de fondo de un contexto de dispositivo.
SetBoundsRect	Establece el comportamiento del rectángulo límite.
SetBrushOrgEx	Establece el origen de un patrón de brocha.
SetMiterLimit	Establece el límite del inglete en líneas unidas mediante inglete.
SetPixel	Establece el color de un píxel en un contexto de dispositivo.
SetPixelV	Establece el color de un píxel en un contexto de dispositivo (generalmente más rápido que SetPixel).
SetPolyFillMode	Establece el modo de relleno de polígonos.
SetROP2	Establece el modo de mezcla del primer plano del contexto de dispositivo.
StrokeAndFillPath	Dibuja y rellena una ruta.
StrokePath	Dibuja una ruta.

## Arc Windows.Pas

### Sintaxis

```

Arc(
    hDC: HDC;           {manejador de contexto de dispositivo}
    left: Integer;       {coordenada x de la esquina superior izquierda}
    top: Integer;        {coordenada y de la esquina superior izquierda}
    right: Integer;      {coordenada x de la esquina inferior derecha}
    bottom: Integer;     {coordenada y de la esquina inferior derecha}
    startX: Integer;     {coordenada x del extremo del primer radio}
    startY: Integer;     {coordenada y del extremo del primer radio}
    endX: Integer;       {coordenada x del extremo del segundo radio}
    endY: Integer;       {coordenada y del extremo del segundo radio}
): BOOL;               {devuelve TRUE o FALSE}

```

### Descripción

Esta función dibuja un arco elíptico. El arco será dibujado utilizando la pluma seleccionada y no usará ni actualizará la posición actual. El rectángulo límite definido por los parámetros *left*, *top*, *right* y *bottom* define la curva del arco. Los parámetros *startX* y *startY* definen los puntos finales de una línea que comienza en el centro del rectángulo límite y que identifica la posición de inicio del arco. Los parámetros *endX* y *endY* definen los puntos finales de una línea que comienza en el centro del rectángulo límite y que identifica la posición final del arco.

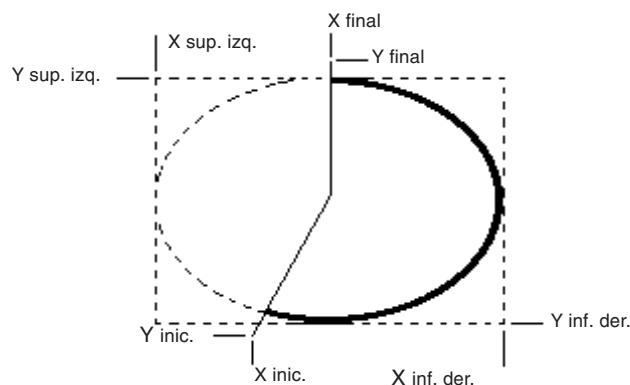


Figura 3-3:  
Coordenadas  
del arco

#### Parámetros

*hDC*: Especifica el contexto de dispositivo sobre el cual el arco es dibujado.

*left*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo límite, en unidades lógicas. Bajo Windows 95/98, la suma de los parámetros *left* y *right* tiene que ser menor que 32.767.

*top*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo límite, en unidades lógicas. Bajo Windows 95/98, la suma de los parámetros *top* y *bottom* tiene que ser menor que 32.767.

*right*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo límite, en unidades lógicas.

*bottom*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo límite, en unidades lógicas.

*startX*: Especifica la coordenada horizontal, en unidades lógicas, del punto final de la línea radial que define el punto de inicio del arco.

*startY*: Especifica la coordenada vertical, en unidades lógicas, del punto final de la línea radial que define el punto de inicio del arco.

*endX*: Especifica la coordenada horizontal, en unidades lógicas, del punto final de la línea radial que define el punto final del arco.

*endY*: Especifica la coordenada vertical, en unidades lógicas, del punto final de la línea radial que define el punto final del arco.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*Chord*, *Ellipse*, *Pie*

*Ejemplo***Listado 3-1: Dibujando un arco iris**

```

procedure TForm1.Button1Click(Sender: TObject);
var
    iCount: Integer;    // variable de control para un bucle general
    ArcBounds: TRect;   // rectángulo límite del arco
begin
    {Inicializar el rectángulo límite}
    ArcBounds := PaintBox1.BoundsRect;

    {Inicializar la pluma usada para dibujar los arcos}
    PaintBox1.Canvas.Pen.Width := 2;

    {Dibujar 5 arcos}
    for iCount := 1 to 5 do begin
        {Dibujar el arco}
        Arc(PaintBox1.Canvas.Handle, ArcBounds.Left, ArcBounds.Top, ArcBounds.Right,
            ArcBounds.Bottom, ArcBounds.Right, (ArcBounds.Bottom-ArcBounds.Top) div 2,
            ArcBounds.Left, (ArcBounds.Bottom-ArcBounds.Top) div 2);

        {Reducir el tamaño del rectángulo límite para el próximo arco}
        InflateRect(ArcBounds, -2, -2);

        {Cambiar el color de la pluma usada para dibujar el próximo arco}
        PaintBox1.Canvas.Pen.Color := PaletteIndex(iCount + 10);
    end;
end;

```

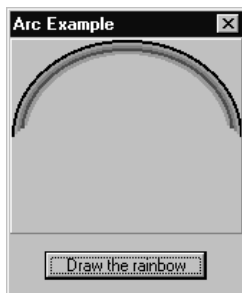


Figura 3-4:  
Un arco iris  
dibujado con  
arcos

**BeginPaint****Windows.Pas***Sintaxis*

```

BeginPaint(
    hWnd: HWND;                {manejador de ventana}
    var lpPaint: TPaintStruct    {puntero a un registro TPaintStruct}
): HDC                         {devuelve un manejador de contexto de dispositivo}

```

### Descripción

Esta función prepara para el redibujado (*painting*) a la ventana específica y rellena el registro *TPaintStruct*, al que apunta el parámetro *lpPaint*, con información relacionada con la operación de pintura. La función *BeginPaint* excluye cualquier área fuera de la región de actualización, estableciendo la región de recorte del contexto de dispositivo. La región de actualización se asigna llamando a las funciones *InvalidateRect* o *InvalidateRgn*, o mediante una acción que afecte al área cliente de la ventana, como por ejemplo redimensionarla, moverla, hacer *scroll* sobre ella, etc. *BeginPaint* envía un mensaje *WM\_ERASEBKGD* a la ventana si se indica que la región a actualizar debe ser borrada. La función *BeginPaint* debe ser llamada conjuntamente con *EndPaint* y únicamente en respuesta a un mensaje *WM\_PAINT*.

### Parámetros

*hWnd*: Especifica el manejador de la ventana que será redibujada.

*lpPaint*: Especifica un puntero a un registro *TPaintStruct* que recibe información sobre la operación de pintura. El registro *TPaintStruct* se define de la siguiente forma:

*TPaintStruct* = **packed record**

<i>hdc</i> : HDC;	{manejador de contexto de dispositivo}
<i>fErase</i> : BOOL;	{opción de borrar fondo}
<i>rcPaint</i> : TRect;	{coordenadas del rectángulo a pintar}
<i>fRestore</i> : BOOL;	{reservado}
<i>fIncUpdate</i> : BOOL;	{reservado}
<i>rgbReserved</i> : array[0..31] of Byte;	{reservado}

**end;**

*hdc*: Especifica el contexto de dispositivo sobre el cual la operación de pintura debe ocurrir.

*fErase*: Una opción que indica si el fondo debe ser borrado. Si a este campo se le asigna TRUE, el fondo del contexto deberá ser borrado antes de que cualquier operación de pintado se realice. La aplicación debe manejar el borrado del fondo si la clase de la ventana no tiene brocha de fondo.

*RcPaint*: Un registro *TRect* que define el área rectangular dentro del contexto de dispositivo donde deben ocurrir las operaciones de pintura.

*fRestore*: Este campo es reservado para uso interno y debe ignorarse.

*fIncUpdate*: Este campo es reservado para uso interno y debe ignorarse.

*rgbReserved*: Este campo es reservado para uso interno y debe ignorarse.

### Valor que devuelve

Si esta función tiene éxito, devuelve un manejador del contexto de dispositivo para la ventana especificada; en caso contrario, devuelve cero.

Véase además

*EndPaint, InvalidateRect, InvalidateRgn*

### Ejemplo

Vea el Listado 3-29 bajo *InvalidateRect* y el Listado 3-30 bajo *InvalidateRgn*.

## Chord Windows.Pas

### Sintaxis

```
Chord(
    DC: HDC;                {manejador de contexto de dispositivo}
    X1: Integer;             {coordenada x de la esquina superior izquierda}
    Y1: Integer;             {coordenada y de la esquina superior izquierda}
    X2: Integer;             {coordenada x de la esquina inferior derecha}
    Y2: Integer;             {coordenada y de la esquina inferior derecha}
    X3: Integer;             {coordenada x del extremo del primer radio}
    Y3: Integer;             {coordenada y del extremo del primer radio}
    X4: Integer;             {coordenada x del extremo del segundo radio}
    Y4: Integer;             {coordenada y del extremo del segundo radio}
): BOOL;                   {devuelve TRUE o FALSE}
```

### Descripción

Esta función dibuja una cuerda utilizando la pluma actual y rellena el interior de la cuerda utilizando la brocha actual. Una *cuerda* es una región limitada por una elipse y un segmento de línea. La dimensión de la cuerda está definida por el rectángulo límite. La curva está definida por una línea identificada por los parámetros *X3*, *Y3*, *X4* e *Y4*. Se extenderá, en sentido contrario a las manecillas del reloj, desde el primer punto de intersección de la línea con el rectángulo límite hasta el segundo punto de intersección de la línea con el rectángulo límite. Si estos dos puntos coinciden, una elipse completa es dibujada. Esta función no modifica la posición actual.

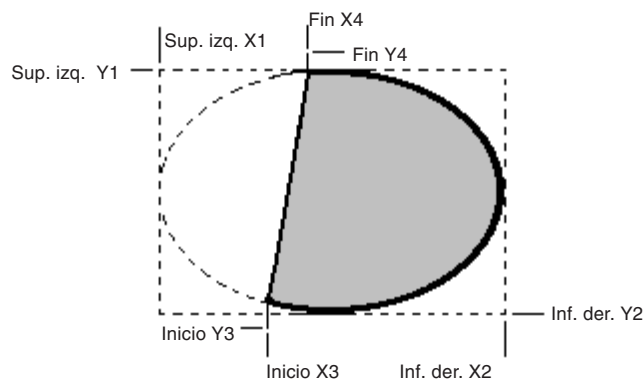


Figura 3-5:  
Coordenadas  
del arco

**Parámetros**

*DC*: Especifica el contexto de dispositivo sobre el cual se dibujará la cuerda.

*X1*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo límite, en unidades lógicas. Bajo Windows 95/98, la suma de los parámetros *X1* y *X2* tiene que ser menor que 32.767.

*Y1*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo límite, en unidades lógicas. Bajo Windows 95/98, la suma de los parámetros *Y1* e *Y2* tiene que ser menor que 32.767.

*X2*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo límite, en unidades lógicas.

*Y2*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo límite, en unidades lógicas.

*X3*: Especifica la coordenada horizontal, en unidades lógicas, del punto final de la línea que define el punto de inicio de la cuerda.

*Y3*: Especifica la coordenada vertical, en unidades lógicas, del punto final de la línea que define el punto de inicio de la cuerda.

*X4*: Especifica la coordenada horizontal, en unidades lógicas, del punto final de la línea que define el punto final de la cuerda.

*Y4*: Especifica la coordenada vertical, en unidades lógicas, del punto final de la línea que define el punto final de la cuerda.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información adicional sobre un error se debe llamar a la función *GetLastError*.

**Véase además**

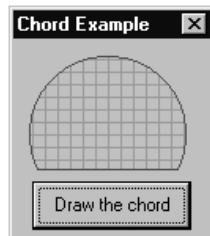
*Arc*, *Ellipse*, *Pie*

**Ejemplo****Listado 3-2: Dibujando una cuerda**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    {Inicializar la brocha y la pluma usadas para dibujar la cuerda}
    Canvas.Brush.Color := clLime;
    Canvas.Brush.Style := bsCross;
    Canvas.Pen.Color := clRed;

    {Dibujar una cuerda}
    Chord(Canvas.Handle, 10, 10, 110, 110, 110, 85, 10, 85);
end;
```

Figura 3-6: La cuerda



## CreateBrushIndirect Windows.Pas

### Sintaxis

```
CreateBrushIndirect(
    const p1: TLogBrush      {puntero a registro TLogBrush}
): HBRUSH;                  {devuelve un manejador de brocha}
```

### Descripción

Esta función crea una nueva brocha basada en los valores del registro *TLogBrush* al que apunta el parámetro *p1*. Si el patrón de la brocha es un mapa de bits monocromo, los píxeles negros son dibujados usando el color de texto actual y los píxeles blancos son dibujados usando el color de fondo actual. Cuando la brocha deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

### Parámetros

*p1*: Un puntero al registro *TLogBrush* que define la nueva brocha. El registro *TLogBrush* se define de la siguiente forma:

```
TLogBrush = packed record
    lbStyle: UINT;           {opción de estilo de la brocha}
    lbColor: COLORREF;       {indicador de color}
    lbHatch: Longint;        {opción de estilo de la trama}
end;
```

*lbStyle*: Una opción especificando el estilo de la brocha. Este campo puede tomar un valor de la Tabla 3-2.

*lbColor*: Especifica un indicador de color que define el color de la brocha. Este campo es ignorado si al campo *lbStyle* se le asigna *BS\_HOLLOW* o *BS\_PATTERN*. Si al campo *lbStyle* se le asigna *BS\_DIBPATTERN* o *BS\_DIBPATTERNBT*, la palabra menos significativa de este campo contendrá una opción que indica la paleta de colores usada por el mapa de bits independiente del dispositivo (DIB). Esta opción puede ser *DIB\_PAL\_COLORS*, que indica que la paleta del DIB es un *array* de índices dentro de la paleta lógica actualmente activa, o *DIB\_RGB\_COLORS*, que indica que la paleta es un *array* de valores literales RGB.

*lbHatch*: Especifica una opción que indica el tipo de trama utilizada por la brocha. Si al campo *lbStyle* se le asigna *BS\_HATCHED*, este campo contendrá una opción de la Tabla 3-3 que especifica la orientación de las líneas usadas para dibujar la trama. Si al campo *lbStyle* se le asigna *BS\_DIBPATTERN*, este campo contendrá un manejador de un DIB empaquetado. Si al campo *lbStyle* se le asigna *BS\_DIBPATTERNPT*, este campo contendrá un puntero a un DIB empaquetado. Si al campo *lbStyle* se le asigna *BS\_PATTERN*, este campo contendrá un manejador de un mapa de bits. Este manejador de mapa de bits no puede ser un manejador de un DIB. Si al campo *lbStyle* se le asigna *BS\_SOLID* o *BS\_HOLLOW*, este campo será ignorado.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de una nueva brocha; en caso contrario, devuelve cero.

#### Véase además

*CreateDIBSection*, *CreateHatchBrush*, *CreatePatternBrush*, *CreateSolidBrush*, *DeleteObject*, *GetBrushOrgEx*, *SelectObject*, *SetBrushOrgEx*

#### Ejemplo

##### Listado 3-3: Creando y usando una nueva brocha

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Region: HRGN;           // manejador de una región
  LogBrush: TLogBrush;    // almacena información de la brocha lógica
  NewBrush: HBrush;       // manejador de la nueva brocha
begin
  {Definir los atributos de la nueva brocha}
  with LogBrush do
  begin
    lbStyle := BS_HATCHED;
    lbColor := clBlue;
    lbHatch := HS_CROSS;
  end;

  {Crear la brocha}
  NewBrush := CreateBrushIndirect(LogBrush);

  {Crea la región a rellenar}
  Region := CreateEllipticRgnIndirect(PaintBox1.BoundsRect);

  {Rellenar la región con la nueva brocha}
  FillRgn(PaintBox1.Canvas.Handle, Region, NewBrush);

  {Borrar la región y la brocha}
  DeleteObject(NewBrush);
```

```
DeleteObject(Region);  
end;
```

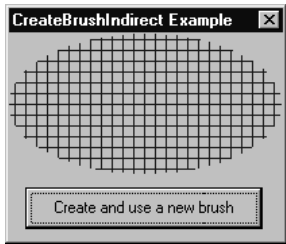


Figura 3-7: El nuevo patrón de la brocha

Tabla 3-2: Valores del campo pl.lbStyle de CreateBrushIndirect

Valor	Descripción
BS_DIBPATTERN	Indica que el patrón de la brocha está definido mediante un mapa de bits independiente del dispositivo. El campo lbHatch contendrá un manejador de DIB empaquetado que se utilizará como patrón de la brocha. Bajo Windows 95/98, un patrón de brocha DIB no puede ser superior en área a ocho píxeles cuadrados. Si se especifica un DIB más grande como patrón, sólo se utilizará una porción de ocho píxeles cuadrados.
BS_DIBPATTERNPT	Indica que el patrón de la brocha está definido mediante un mapa de bits independiente del dispositivo. El campo lbHatch contendrá un manejador de DIB empaquetado que se utilizará como patrón de la brocha. Bajo Windows 95/98, un patrón de brocha DIB no puede ser superior en área a ocho píxeles cuadrados. Si se especifica un DIB más grande como patrón, sólo se utilizará una porción de ocho píxeles cuadrados.
BS_HATCHED	Indica una brocha con trama.
BS_HOLLOW	Indica una brocha vacía.
BS_PATTERN	Indica que el patrón de la brocha está definido mediante un mapa de bits dependiente del dispositivo. El campo lbHatch contendrá un manejador del mapa de bits que será usado como patrón de la brocha. Bajo Windows 95/98, un patrón de brocha no puede ser superior en área a ocho píxeles cuadrados. Si se especifica un mapa de bits más grande como patrón, sólo se utilizará una porción de ocho píxeles cuadrados.
BS_SOLID	Indica una brocha sólida.

Tabla 3-3: Valores del campo pl.lbHatch de CreateBrushIndirect

Valor	Descripción
HS_BDIAGONAL	Trama compuesta por líneas a 45 grados hacia arriba, de izquierda a derecha.
HS_CROSS	Trama compuesta por líneas cruzadas horizontales y verticales.

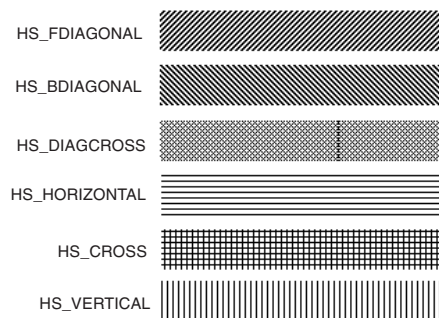
Valor	Descripción
HS_DIAGCROSS	Equivalente a HS_CROSS, pero rotado 45 grados.
HS_FDIAGONAL	Trama compuesta por líneas a 45 grados hacia abajo, de izquierda a derecha.
HS_HORIZONTAL	Trama compuesta por líneas horizontales.
HS_VERTICAL	Trama compuesta por líneas verticales.

**CreateHatchBrush****Windows.Pas****Sintaxis**

```
CreateHatchBrush(
    p1: Integer;           {estilo de trama}
    p2: COLORREF           {indicador de color}
); HBRUSH;               {devuelve un manejador de brocha}
```

**Descripción**

Esta función crea una nueva brocha con el color y patrón de trama especificados. Los patrones accesibles para esta función se ilustran en la figura 3-8. Si una brocha de trama con el mismo patrón y color es utilizada para pintar el fondo de una ventana hija y su ventana madre, podría ser necesario llamar a la función *SetBrushOrgEx* para alinear el patrón de la brocha antes de pintar el fondo de la ventana hija. Cuando la brocha deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.



**Figura 3-8:**  
**Patrones de**  
**tramas**

**Parámetros**

*p1*: Una opción que especifica el patrón de trama de la brocha. A este parámetro se le puede asignar un valor de la Tabla 3-4.

*p2*: Un indicador de color que indica el color de primer plano utilizado para dibujar las líneas de la trama.

Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la nueva brocha; en caso contrario, devuelve cero.

Véase además

*CreateBrushIndirect, CreatePatternBrush, CreateSolidBrush, DeleteObject, GetBrushOrgEx, SelectObject, SetBrushOrgEx*

Ejemplo

Listado 3-4: Creando una brocha sombreada

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TheBrush: HBRUSH; // almacena la nueva brocha
  HandleRgn: THandle; // manejador de una región
begin
  {Crear una brocha sombreada}
  TheBrush := CreateHatchBrush(HS_DIAGCROSS, clRed);

  {Crear una región}
  HandleRgn := CreateEllipticRgnIndirect(ClientRect);

  {Rellenar la región con la brocha}
  FillRgn(Canvas.Handle, HandleRgn, TheBrush);

  {Eliminar la brocha y la región}
  DeleteObject(TheBrush);
  DeleteObject(HandleRgn);
end;
```

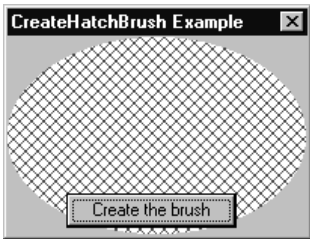


Figura 3-9: La brocha de sombreado

Tabla 3-4: Valores del parámetro pl de CreateHatchBrush

Valor	Descripción
HS_BDIAGONAL	Trama compuesta por líneas a 45 grados hacia arriba, de izquierda a derecha.
HS_CROSS	Trama compuesta por líneas cruzadas horizontales y verticales.
HS_DIAGCROSS	Equivalente a HS_CROSS, pero rotado 45 grados.
HS_FDIAGONAL	Trama compuesta por líneas a 45 grados hacia abajo, de izquierda a derecha.

Valor	Descripción
HS_HORIZONTAL	Trama compuesta por líneas horizontales.
HS_VERTICAL	Trama compuesta por líneas verticales.

**CreatePatternBrush****Windows.Pas****Sintaxis**

```
CreatePatternBrush(
    Bitmap: HBITMAP           {manejador del mapa de bits}
): HBRUSH;                   {devuelve un manejador de brocha}
```

**Descripción**

Esta función crea una nueva brocha con el patrón del mapa de bits especificado. Si el patrón de la brocha es un mapa de bits monocromo, los píxeles negros serán dibujados utilizando el color de texto actual y los píxeles blancos serán dibujados utilizando el color de fondo actual. Cuando la brocha deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*. Observe que al eliminar la brocha no se elimina el mapa de bits que define el patrón de la brocha.

**Parámetros**

*Bitmap*: Especifica el manejador del mapa de bits que define el patrón de la brocha. Este no puede ser un manejador de un DIB creado mediante una llamada a la función *CreateDIBSection*. Bajo Windows 95/98, un patrón de una brocha de mapa de bits no puede ser superior a ocho píxeles cuadrados. Si un mapa de bits más grande es especificado como patrón, sólo será utilizada una porción de ocho píxeles cuadrados de ese mapa de bits.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de una nueva brocha; en caso contrario, devuelve cero.

**Véase además**

*CreateBitmap*, *CreateBitmapIndirect*, *CreateCompatibleBitmap*, *CreateDIBSection*, *CreateHatchBrush*, *DeleteObject*, *GetBrushOrgEx*, *LoadBitmap*, *SelectObject*, *SetBrushOrgEx*

**Ejemplo****Listado 3-5: Usando un mapa de bits como patrón de una brocha****implementation**

```
{ $R *.DFM }
{ $R BrushPatterns.Res }
```

```

procedure TForm1.Button1Click(Sender: TObject);
var
    NewBrush: HBrush;      // manejador de la brocha
    BitmapHandle: THandle; // manejador del mapa de bits
begin
    {Obtener un mapa de bits que está almacenado en el .EXE}
    BitmapHandle := LoadBitmap(HInstance, 'BrushPattern');

    {Crear el patrón de la brocha con el mapa de bits como patrón}
    NewBrush := CreatePatternBrush(BitmapHandle);

    {Rellena la región con el patrón, usando la brocha}
    FillRect(Canvas.Handle, ClientRect, NewBrush);

    {Limpiar la memoria}
    DeleteObject(NewBrush);
    DeleteObject(BitmapHandle);
end;

```

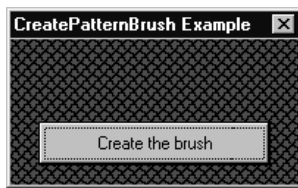


Figura 3-10:  
El patrón de la  
brocha

## CreatePen Windows.Pas

### Sintaxis

```

CreatePen(
    Style: Integer;           {opción de estilo de la pluma}
    Width: Integer;           {ancho de la pluma}
    Color: COLORREF           {color de la pluma}
): HPEN;                     {devuelve un manejador de una nueva pluma}

```

### Descripción

Esta función crea una nueva pluma con el estilo, ancho y color especificados. Cuando la pluma deje de ser necesaria, deberá ser eliminada mediante una llamada a la función *DeleteObject*.

### Parámetros

*Style*: Una opción que indica el estilo de la pluma. A este parámetro se le puede asignar un valor de la Tabla 3-5.

*Width*: Ancho de la pluma en unidades lógicas. Un ancho cero creará una pluma de un píxel de ancho, independientemente de cualquier transformación activa. Si a este parámetro se le asigna un valor mayor que uno, al parámetro *Style* se le tiene que asignar una de las opciones *PS\_NULL*, *PS\_SOLID* o *PS\_INSIDEFRAME*. Si este

parámetro es mayor que uno y al parámetro *Style* se le asigna *PS\_INSIDEFRAME*, las líneas dibujadas con esta pluma quedarán situadas dentro del marco para todas las primitivas gráficas, excepto aquellas creadas con funciones de polígonos y polilíneas.

*Color*: Un especificador de color que indica el color de la pluma.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la nueva pluma; en caso contrario, devuelve cero.

#### Véase además

*CreatePenIndirect*, *DeleteObject*, *ExtCreatePen*, *GetObject*, *SelectObject*

#### Ejemplo

##### Listado 3-6: Creando una nueva pluma

```

procedure TForm1.Button1Click(Sender: TObject);
var
    Style: Integer;      // almacena los estilos de la pluma
    PenHandle: HPen;     // manejador de la pluma
begin
    {Borrar cualquier imagen anterior}
    Canvas.Brush.Color := clBtnFace;
    Canvas.FillRect(Rect(10, 10, 111, 111));

    {Determinar el estilo de la pluma}
    case RadioGroup1.ItemIndex of
        0: Style := PS_SOLID;
        1: Style := PS_DASH;
        2: Style := PS_DOT;
        3: Style := PS_DASHDOT;
        4: Style := PS_DASHDOTDOT;
        5: Style := PS_NULL;
        6: Style := PS_INSIDEFRAME;
    end;

    {Crear la pluma}
    PenHandle := CreatePen(Style, 1, 0);

    {Indicar al área de dibujo que utilice la nueva pluma}
    Canvas.Pen.Handle := PenHandle;

    {Dibujar un cuadrado con la pluma}
    Canvas.MoveTo(10, 10);
    Canvas.LineTo(110, 10);
    Canvas.LineTo(110, 110);
    Canvas.LineTo(10, 110);
    Canvas.LineTo(10, 10);

    {Eliminar la pluma}
    DeleteObject(PenHandle);
end;

```

Figura 3-11:  
La nueva  
pluma

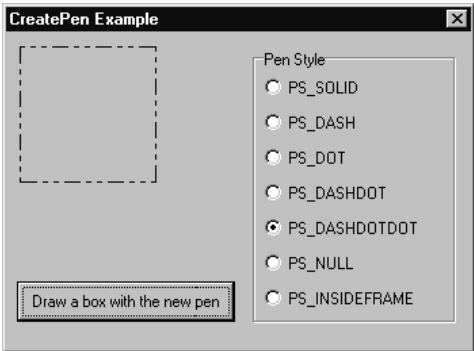


Tabla 3-5: Valores del parámetro Style de CreatePen

Valor	Descripción
PS_SOLID	Especifica una pluma sólida.
PS_DASH	Especifica una pluma de líneas discontinuas. Esta opción puede ser usada sólo cuando el ancho de la pluma es uno o menos.
PS_DOT	Especifica una pluma de línea de puntos. Esta opción puede ser usada solamente cuando el ancho de la pluma es uno o menos.
PS_DASHDOT	Especifica una pluma que alterna combinaciones de un guión y un punto. Esta opción puede ser usada solamente cuando el ancho de la pluma es uno o menos.
PS_DASHDOTDOT	Especifica una pluma que alterna combinaciones de un guión y dos puntos. Esta opción puede ser usada solamente cuando el ancho de la pluma es uno o menos.
PS_NULL	Especifica una pluma invisible.
PS_INSIDEFRAME	Especifica una pluma sólida. Cuando esta pluma es usada con funciones de dibujo que requieren un rectángulo límite, las dimensiones de la figura son comprimidas, de manera que llenen el rectángulo con respecto al ancho de la pluma.

CreatePenIndirect Windows.Pas

Sintaxis

```
CreatePenIndirect(  
    const LogPen: TLogPen      {puntero a registro TLogPen}  
): HPEN;                      {devuelve un manejador de la pluma nueva}
```

Descripción

Esta función crea una nueva pluma con el estilo, ancho y color especificados en el registro *TLogPen*, a la cual apunta el parámetro *LogPen*. Cuando la pluma deje de ser necesaria, deberá ser eliminada mediante una llamada a la función *DeleteObject*.

### Parámetros

*LogPen*: Puntero al registro *TLogPen* que define los atributos de la nueva pluma. El registro *TLogPen* se define de la siguiente forma:

```
TLogPen = packed record
    lopnStyle: UINT;           {estilo de la pluma}
    lopnWidth: TPoint;         {ancho de la pluma}
    lopnColor: COLORREF;       {color de la pluma}
end;
```

*lopnStyle*: Opción que indica el estilo de la pluma. A este campo se le puede asignar un valor de la Tabla 3-6.

*lopnWidth*: El campo *X* de este registro *TPoint* especifica el ancho de la pluma en unidades lógicas. El campo *Y* no se utiliza. Un ancho cero creará una pluma de un píxel de ancho, independientemente de cualquier transformación activa. Si a este campo se le asigna un valor mayor que uno, al campo *lopnStyle* deberá asignársele una de las opciones *PS\_NULL*, *PS\_SOLID* o *PS\_INSIDEFRAME*. Si este parámetro es mayor que uno y al parámetro *lopnStyle* se le asigna *PS\_INSIDEFRAME*, las líneas dibujadas con esta pluma quedarán situadas dentro del marco para todas las primitivas gráficas, excepto aquellas dibujadas con las funciones de polígonos y polilíneas.

*lopnColor*: Un especificador de color que indica el color de la pluma.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la nueva pluma; en caso contrario, devuelve cero.

### Véase además

*CreatePen*, *DeleteObject*, *ExtCreatePen*, *GetObject*, *SelectObject*

### Ejemplo

#### Listado 3-7: Creando una pluma indirectamente

```
procedure TForm1.Button1Click(Sender: TObject);
var
    Pen: TLogPen;           // registro lógico para la pluma
    PenHandle: HPen;        // manejador de una pluma
begin
    {Borrar cualquier imagen previa}
    Canvas.Brush.Color := clBtnFace;
    Canvas.FillRect(Rect(10, 10, 111, 111));

    {Inicializar el registro lógico de la pluma}
    with Pen do begin
        {Determinar el estilo de la pluma}
        case RadioGroup1.ItemIndex of
            0: lopnStyle := PS_SOLID;
            1: lopnStyle := PS_DASH;
```

```

2: lopnStyle := PS_DOT;
3: lopnStyle := PS_DASHDOT;
4: lopnStyle := PS_DASHDOTDOT;
5: lopnStyle := PS_NULL;
6: lopnStyle := PS_INSIDEFRAME;
end;

{Asignar el color y ancho de la pluma}
lopnWidth.X := 1;
lopnColor   := clRed;
end;

{Crear una nueva pluma}
PenHandle := CreatePenIndirect(Pen);

{Dibujar un cuadrado con la nueva pluma}
Canvas.Pen.Handle := PenHandle;
Canvas.MoveTo(10, 10);
Canvas.LineTo(110, 10);
Canvas.LineTo(110, 110);
Canvas.LineTo(10, 110);
Canvas.LineTo(10, 10);
{Borrar la nueva pluma}
DeleteObject(PenHandle);
end;

```

Tabla 3-6: Valores del campo LogPen.lopnStyle de CreatePenIndirect

Valor	Descripción
PS_SOLID	Especifica una pluma sólida.
PS_DASH	Especifica una pluma de líneas discontinuas. Esta opción puede ser usada solamente cuando el ancho de la pluma es uno o menos.
PS_DOT	Especifica una pluma de línea de puntos. Esta opción puede ser usada solamente cuando el ancho de la pluma es uno o menos.
PS_DASHDOT	Especifica una pluma que alterna combinaciones de un guión y un punto. Esta opción puede ser usada solamente cuando el ancho de la pluma es uno o menos.
PS_DASHDOTDOT	Especifica una pluma que alterna combinaciones de un guión y dos puntos. Esta opción puede ser usada solamente cuando el ancho de la pluma es uno o menos.
PS_NULL	Especifica una pluma invisible.
PS_INSIDEFRAME	Especifica una pluma sólida. Cuando esta pluma es usada con funciones de dibujo que requieren un rectángulo límite, las dimensiones de la figura son comprimidas, de manera que llenen el rectángulo con respecto al ancho de la pluma.

**CreateSolidBrush****Windows.Pas****Sintaxis**

```
CreateSolidBrush(
  p1: COLORREF           {color de la brocha}
): HBRUSH;               {devuelve un manejador de la nueva brocha}
```

**Descripción**

Esta función crea una nueva brocha sólida con el color especificado. Una vez que la brocha deje de ser necesaria, deberá ser eliminada mediante una llamada a la función *DeleteObject*.

**Parámetros**

*PI*: Un especificador de color que indica el color de una brocha.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de la nueva brocha; en caso contrario, devuelve cero.

**Véase además**

*CreateHatchBrush*, *CreatePatternBrush*, *DeleteObject*, *SelectObject*

**Ejemplo****Listado 3-8: Creando una brocha sólida**

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NewBrush: HBrush;    // manejador de brocha
  OldBrush: HBrush;    // manejador de la brocha original del contexto
  FormDC: HDC;         // manejador del contexto de dispositivo del formulario
begin
  {Crear la brocha}
  NewBrush := CreateSolidBrush(clGreen);

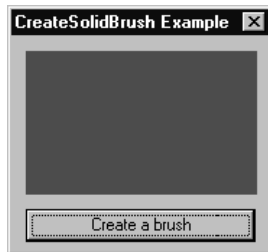
  {Obtener el contexto de dispositivo del formulario}
  FormDC := GetDC(Form1.Handle);

  {Obtener manejador de la brocha del contexto de dispositivo del formulario}
  OldBrush := SelectObject(FormDC, NewBrush);

  {Rellenar el rectángulo con la brocha}
  FillRect(FormDC, Rect(10, 10, 170, 110), NewBrush);

  {Limpiar la memoria}
  SelectObject(FormDC, OldBrush);
  DeleteObject(NewBrush);
end;
```

Figura 3-12:  
La brocha  
sólida



### DeleteObject      Windows.Pas

#### Sintaxis

```
DeleteObject(
    p1: HGDIOBJ           {manejador de objeto del GDI}
): BOOL;                 {devuelve TRUE o FALSE}
```

#### Descripción

Esta función elimina una pluma, una brocha, una fuente, un mapa de bits, una región, o una paleta lógicas, liberando sus recursos asociados. El manejador del objeto es invalidado cuando esta función retorna. Esta función fallará si se intenta eliminar un objeto que esté seleccionado dentro de algún contexto de dispositivo. Nota: Eliminar una brocha de patrón de mapa de bits no afecta a su mapa de bits. El mapa de bits asociado a la brocha deberá ser eliminado independientemente.

#### Parámetros

*PI*: Especifica el manejador del objeto que será eliminado.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE. Si la función falla, el manejador especificado no es válido, o el objeto está actualmente seleccionado en algún contexto del dispositivo, devuelve FALSE.

#### Véase además

*GetObject*, *SelectObject*

#### Ejemplo

Vea el Listado 3-3 bajo *CreateBrushIndirect* y otros ejemplos a lo largo del libro.

### DrawCaption      Windows.Pas

#### Sintaxis

```
DrawCaption(
    p1: HWND;              {manejador de ventana}
```

```

p2: HDC;                {manejador de contexto de dispositivo}
const p3: TRect;        {coordenadas del rectángulo}
p4: UINT                {opciones de dibujo}
): BOOL;                {devuelve TRUE o FALSE}

```

### Descripción

Esta función dibuja una barra de título en el área rectangular identificada por el parámetro *p3*. La barra de título recupera su texto y su icono desde la ventana identificada por el parámetro *p1*.

### Parámetros

*p1*: Manejador de la ventana que contiene el texto y el icono usados para dibujar la barra de título.

*p2*: Manejador de contexto de dispositivo sobre el cual se dibujará la barra de título.

*p3*: Especifica las coordenadas del rectángulo dentro de las cuales se dibujará la barra de título.

*p4*: Especifica una combinación de valores que definen opciones de dibujo. A este parámetro se le puede asignar un valor de la Tabla 3-7.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### Véase además

*DrawEdge*, *DrawFocusRect*, *DrawFrameControl*, *DrawState*, *SetWindowRgn*

### Ejemplo

#### Listado 3-9: Dibujando por programa una barra de título

```

procedure TForm1.FormPaint(Sender: TObject);
begin
    DrawCaption(Handle, Canvas.Handle, Rect(16, 40, 288, 60),
                DC_ACTIVE or DC_ICON or DC_TEXT);
end;

```

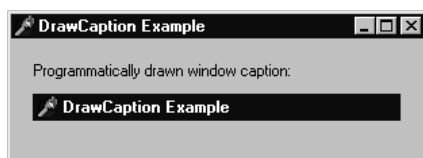


Figura 3-13:  
La barra de  
título

Tabla 3-7: Valores del parámetro p4 de DrawCaption

Valor	Descripción
DC_ACTIVE	La barra de título es dibujada utilizando el color activo para barras de título.
DC_ICON	El icono de la ventana es dibujado en la barra de título.
DC_INBUTTON	La barra de título es dibujada en estado “hundido”.
DC_SMALLCAP	El texto de la barra de título es dibujado usando la fuente pequeña para barras de título activa.
DC_TEXT	El texto de la ventana es dibujado en la barra de título.

**DrawEdge****Windows.Pas****Sintaxis**

```

DrawEdge(
    hdc: HDC;                {contexto de dispositivo}
    var qrc: TRect;          {coordenadas del rectángulo}
    edge: UINT;              {opciones de tipo de bordes}
    grfFlags: UINT           {opciones de tipo de bordes}
): BOOL;                   {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja una línea o un rectángulo usando el efecto tridimensional de borde especificado.

**Parámetros**

*hdc*: Especifica un manejador de un contexto de dispositivo sobre el cual se dibuja el borde.

*qrc*: Un puntero a un registro *TRect* que contiene las coordenadas del rectángulo, en unidades lógicas, que definen el borde.

*edge*: Una combinación de opciones que indican el tipo de borde a dibujar. A este parámetro se le tiene que asignar una combinación de valores, uno tomado de la tabla de opciones para el borde interior (Tabla 3-8) y otro de la tabla de opciones para el borde exterior (Tabla 3-9). También puede utilizarse un único valor de la tabla de opciones de combinación de bordes (Tabla 3-10) en lugar de los valores combinados.

*grfFlags*: Una combinación de opciones que indican el tipo de borde a dibujar. A este parámetro se le puede asignar una combinación de opciones de la Tabla 3-11.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información adicional sobre un error se debe llamar a la función *GetLastError*.

Véase además

*MoveToEx, LineDDA, LineTo, Rectangle*

Ejemplo

#### Listado 3-10: Dibujando bordes tridimensionales

```

type
  TFlagsArray = array[0..18] of UINT;  // array de opciones de tipos de bordes

const
  {Inicializar el array de opciones de bordes}
  BorderFlags: TFlagsArray = (BF_ADJUST, BF_BOTTOM, BF_BOTTOMLEFT,
                              BF_BOTTOMRIGHT, BF_DIAGONAL,
                              BF_DIAGONAL_ENDBOTTOMLEFT,
                              BF_DIAGONAL_ENDBOTTOMRIGHT,
                              BF_DIAGONAL_ENDTOPLEFT, BF_DIAGONAL_ENDTOPRIGHT,
                              BF_FLAT, BF_LEFT, BF_MIDDLE, BF_MONO, BF_RECT,
                              BF_RIGHT, BF_SOFT, BF_TOP, BF_TOPLEFT,
                              BF_TOPRIGHT);

procedure TForm1.Button1Click(Sender: TObject);
var
  TheRect: TRect;           // define el rectángulo del borde
  Edge, Border: UINT;       // almacena los valores de las opciones del borde
  iCount: Integer;         // contador general para bucle
begin
  {Definir el rectángulo para el borde}
  TheRect := Rect(21, 200, 216, 300);

  {Borrar el último borde dibujado}
  Canvas.Brush.Color := clBtnFace;
  Canvas.FillRect(TheRect);

  {Definir el tipo de borde}
  case RadioGroup_Additional.ItemIndex of
    0: Edge := EDGE_BUMP;    // Combinación BDR_RAISEDOUTER and BDR_SUNKENINNER
    1: Edge := EDGE_ETCHED; // Combinación BDR_SUNKENOUTER and BDR_RAISEDINNER
    2: Edge := EDGE_RAISED;  // Combinación BDR_RAISEDOUTER and BDR_RAISEDINNER
    3: Edge := EDGE_SUNKEN;  // Combinación BDR_SUNKENOUTER and BDR_SUNKENINNER
  end;

  {Inicializar las opciones de bordes}
  Border := 0;

  {Determinar la opción del tipo de borde seleccionado}
  for iCount := 0 to 18 do
    if CheckListBox2.Checked[iCount] then Border:=Border or BorderFlags[iCount];

  {Dibujar el borde}
  DrawEdge(Canvas.Handle, TheRect, Edge, Border);
end;

```

Figur 3-14: Un rectángulo grabado

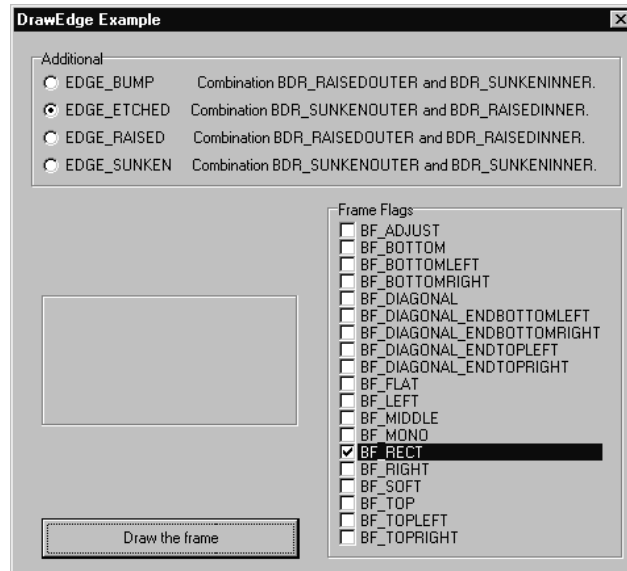


Tabla 3-8: Valores de opciones de bordes internos para el parámetro Flags de DrawEdge

Valor	Descripción
BDR_RAISEDINNER	Indica un borde interior realzado.
BDR_SUNKENINNER	Indica un borde interior rehundido.

Tabla 3-9: Valores de opciones de bordes externos para el parámetro Flags de DrawEdge

Valor	Descripción
BDR_RAISEDOUTER	Indica un borde exterior realzado.
BDR_SUNKENOUTER	Indica un borde exterior rehundido.

Tabla 3-10: Valores de opciones de combinación para el parámetro Flags de DrawEdge

Valor	Descripción
EDGE_BUMP	Combinación de BDR_RAISEDOUTER y BDR_SUNKENINNER.
EDGE_ETCHED	Combinación de BDR_SUNKENOUTER y BDR_RAISEDINNER.
EDGE_RAISED	Combinación de BDR_RAISEDOUTER y BDR_RAISEDINNER.
EDGE_SUNKEN	Combinación de BDR_SUNKENOUTER y BDR_SUNKENINNER.

Tabla 3-11: Valores del parámetro grfFlags de DrawEdge

Valor	Descripción
BF_ADJUST	Las coordenadas del rectángulo son decrementadas para tener en cuenta el ancho de las líneas del borde.

BF_BOTTOM	Dibuja el borde inferior del rectángulo.
BF_BOTTOMLEFT	Dibuja los bordes izquierdo e inferior del rectángulo.
BF_BOTTOMRIGHT	Dibuja los bordes derecho e inferior del rectángulo.
BF_DIAGONAL	Dibuja un borde diagonal.
BF_DIAGONAL_ENDBOTTOMLEFT	Dibuja un borde diagonal comenzando en la esquina superior derecha y terminando en la inferior izquierda.
BF_DIAGONAL_ENDBOTTOMRIGHT	Dibuja un borde diagonal comenzando en la esquina superior izquierda y terminando en la inferior derecha.
BF_DIAGONAL_ENDTOPLEFT	Dibuja un borde diagonal comenzando en la esquina inferior derecha y terminando en la superior izquierda.
BF_DIAGONAL_ENDTOPRIGHT	Dibuja un borde diagonal comenzando en la esquina inferior izquierda y terminando en la superior derecha.
BF_FLAT	Dibuja un borde plano.
BF_LEFT	Dibuja el borde izquierdo del rectángulo.
BF_MIDDLE	Rellena el interior del rectángulo.
BF_MONO	Dibuja un borde unidimensional.
BF_RECT	Dibuja un borde alrededor de todo el rectángulo.
BF_RIGHT	Dibuja el borde derecho del rectángulo.
BF_SOFT	Dibuja el borde con un estilo suave.
BF_TOP	Dibuja el borde superior del rectángulo.
BF_TOPLEFT	Dibuja los bordes superior e izquierdo del rectángulo.
BF_TOPRIGHT	Dibuja los bordes superior y derecho del rectángulo.

---

**DrawFocusRect****Windows.Pas****Sintaxis**

```

DrawFocusRect(
    HDC: HDC;           {contexto de dispositivo}
    const lprc: TRect   {coordenadas del rectángulo}
); BOOL;               {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja un rectángulo en un estilo que indica que tiene el foco. El rectángulo es dibujado utilizando una operación booleana **xor**. Debido a ésto, si la

función es llamada por segunda vez con las mismas coordenadas, se borrará el rectángulo.

#### Parámetros

*hDC*: Manejador de contexto de dispositivo sobre el cual se dibujará el rectángulo.

*lprc*: Especifica las coordenadas del rectángulo que define los bordes del rectángulo a dibujar.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información adicional sobre un error se debe llamar a la función *GetLastError*.

#### Véase además

*DrawCaption*, *DrawEdge*, *DrawFrameControl*, *FrameRect*, *Rectangle*, *RoundRect*

#### Ejemplo

##### Listado 3-II: Dibujando un rectángulo que tiene el foco

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRect: TRect; // las coordenadas del rectángulo enfocado
begin
  {Definir el rectángulo}
  MyRect := Rect(14, 10, 151, 90);

  {Dibujar el rectángulo enfocado}
  if not DrawFocusRect(Canvas.Handle, MyRect) then
    ShowMessage('DrawFocusRect not working');
end;
```

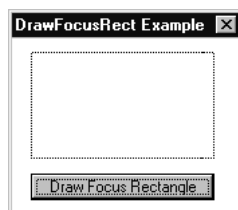


Figura 3-15:  
El rectángulo  
que tiene el  
foco

#### DrawFrameControl

#### Windows.Pas

#### Sintaxis

DrawFrameControl(	
DC: HDC;	{manejador de contexto de dispositivo}
<b>const</b> Rect: TRect;	{coordenadas del rectángulo}
uType: UINT;	{opciones de tipo de control}
uState: UINT	{opciones de estado de control}

); BOOL; {devuelve TRUE o FALSE}

#### Descripción

Esta función dibuja varios botones definidos por el sistema, con el estilo y estado especificados.

#### Parámetros

*DC*: Manejador del contexto del dispositivo sobre el que se dibujará el control.

*Rect*: Especifica las coordenadas del rectángulo que definen el tamaño del control.

*uType*: Una combinación de opciones que indican el tipo de control que será dibujado. A este parámetro se le puede asignar un valor de la Tabla 3-12.

*uState*: Una combinación de opciones que indican el estado del control que será dibujado. A este parámetro se le puede asignar un valor de las Tablas 3-13 a 3-16 dependiendo del valor del parámetro *uType*. Se cuenta con una tabla separada para cada posible valor del parámetro *uType*. También se puede tomar un valor apropiado para *uType* y combinarlo con uno o más valores de la tabla de opciones de estado generales (Tabla 3-17).

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información adicional sobre un error se debe llamar a la función *GetLastError*.

#### Véase además

*DrawCaption, DrawEdge, DrawFocusRect, DrawState*

#### Ejemplo

##### Listado 3-12: Dibujando varios controles de marco.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    TheRect: TRect;      // el rectángulo límite para la imagen de control
    TheType: UINT;       // almacena el tipo de control
    TheState: UINT;      // almacena el estado del control
begin
    {Inicializar las opciones de tipo y de estado}
    TheType := 0;
    TheState := 0;

    {Definir el rectángulo límite}
    TheRect := Rect(10, 10, 50, 50);

    {Elegir el tipo de control}
    case RadioGroup_ButtonType.ItemIndex of
        0:
            begin
                {Indicar que estamos dibujando un botón}
                TheType := DFC_BUTTON;
```

```

{Elegir el estado del control}
case RadioGroup1.ItemIndex of
  0: TheState := DFCS_BUTTON3STATE;
  1: TheState := DFCS_BUTTONCHECK;
  2: TheState := DFCS_BUTTONPUSH;
  3: TheState := DFCS_BUTTONRADIO;
  4: TheState := DFCS_BUTTONRADIOIMAGE;
  5: TheState := DFCS_BUTTONRADIOMASK;
end;
end;
1:
begin
  {Indicar que estamos dibujando un botón de barra de título}
  TheType := DFC_CAPTION;

  {Elegir el estado del control}
  case RadioGroup2.ItemIndex of
    0: TheState := DFCS_CAPTIONCLOSE;
    1: TheState := DFCS_CAPTIONHELP;
    2: TheState := DFCS_CAPTIONMAX;
    3: TheState := DFCS_CAPTIONMIN;
    4: TheState := DFCS_CAPTIONRESTORE;
  end;
end;
2:
begin
  {Indicar que estamos dibujando un mapa de bits de un elemento de menú}
  TheType := DFC_MENU;

  {Elegir el estado del control}
  case RadioGroup3.ItemIndex of
    0: TheState := DFCS_MENUARROW;
    1: TheState := DFCS_MENUBULLET;
    2: TheState := DFCS_MENUCHECK;
  end;
end;
3:
begin
  {Indicar que estamos dibujando un botón de la barra de desplazamiento}
  TheType := DFC_SCROLL;

  {Elegir el estado del control}
  case RadioGroup4.ItemIndex of
    0: TheState := DFCS_SCROLLCOMBOBOX;
    1: TheState := DFCS_SCROLLDOWN;
    2: TheState := DFCS_SCROLLLEFT;
    3: TheState := DFCS_SCROLLRIGHT;
    4: TheState := DFCS_SCROLLSIZEGRIP;
    5: TheState := DFCS_SCROLLUP;
  end;
end;
end;

```

```

{Identificar el estado del botón}
case RadioGroup5.ItemIndex of
  0: TheState := TheState or DFCS_CHECKED;
  1: TheState := TheState or DFCS_FLAT;
  2: TheState := TheState or DFCS_INACTIVE;
  3: TheState := TheState or DFCS_MONO;
  4: TheState := TheState or DFCS_PUSHED;
end;

{Borrar la imagen anterior}
Canvas.Brush.Color := clBtnFace;
Canvas.FillRect(TheRect);

{Dibujar el control del marco}
DrawFrameControl(Canvas.Handle, TheRect, TheType, TheState);
end;

```

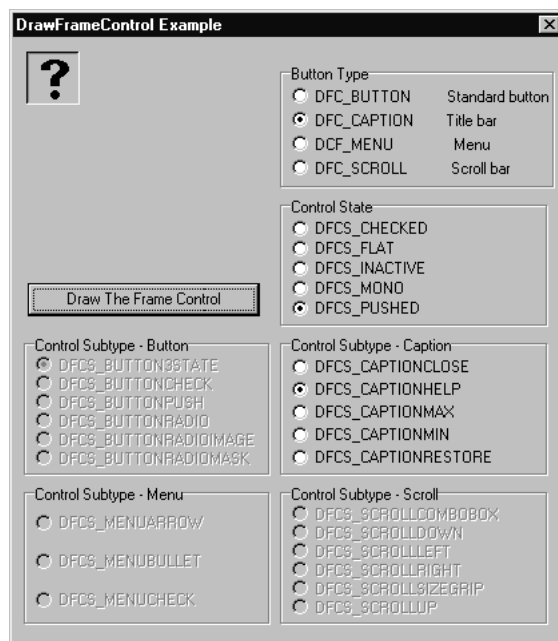


Figura 3-16:  
El programa  
de ejemplo  
“Frame  
Control”

Tabla 3-12: Valores del parámetro uType de DrawFrameControl

Valor	Descripción
DFC_BUTTON	Dibuja un botón estándar.
DFC_CAPTION	Dibuja un botón de barra de título.
DCF_MENU	Dibuja imágenes utilizadas en menús.
DFC_SCROLL	Dibuja un botón de barra de desplazamiento.

Tabla 3-13: Valores del parámetro uState de DrawFrameControl (para DFC\_BUTTON)

Valor	Descripción
DFCS_BUTTON3STATE	Dibuja un botón de tres estados.
DFCS_BUTTONCHECK	Dibuja una casilla de verificación.
DFCS_BUTTONPUSH	Dibuja un botón de pulsación corriente.
DFCS_BUTTONRADIO	Dibuja un botón de radio.
DFCS_BUTTONRADIOIMAGE	Dibuja la máscara <b>xor</b> del botón de radio.
DFCS_BUTTONRADIOMASK	Dibuja la máscara <b>and</b> del botón de radio.

Tabla 3-14: Valores del parámetro uState de DrawFrameControl (para DFC\_CAPTION)

Valor	Descripción
DFCS_CAPTIONCLOSE	Dibuja un botón de cerrar.
DFCS_CAPTIONHELP	Dibuja un botón de ayuda.
DFCS_CAPTIONMAX	Dibuja un botón de maximizar.
DFCS_CAPTIONMIN	Dibuja un botón de minimizar.
DFCS_CAPTIONRESTORE	Dibuja un botón de restaurar.

Tabla 3-15: Valores del parámetro uState de DrawFrameControl (para DFC\_MENU)

Valor	Descripción
DFCS_MENUARROW	Dibuja una flecha de submenú.
DFCS_MENUBULLET	Dibuja una viñeta.
DFCS_MENUCHECK	Dibuja un marca de verificación.

Tabla 3-16: Valores del parámetro uState de DrawFrameControl (para DFC\_SCROLL)

Valor	Descripción
DFCS_SCROLLCOMBOBOX	Dibuja un botón de menú desplegable de una caja de selección múltiple.
DFCS_SCROLLDOWN	Dibuja un botón hacia abajo de una barra de desplazamiento.
DFCS_SCROLLLEFT	Dibuja un botón hacia la izquierda de una barra de desplazamiento.
DFCS_SCROLLRIGHT	Dibuja un botón hacia la derecha de una barra de desplazamiento.
DFCS_SCROLLSIZEGRIP	Dibuja una doble flecha de redimensionamiento.
DFCS_SCROLLUP	Dibuja un botón hacia arriba de una barra de desplazamiento.

Tabla 3-17: Opciones generales del parámetro `uState` de `DrawFrameControl`

Valor	Descripción
<code>DFCS_ADJUSTRECT</code>	El rectángulo especificado es reducido para excluir el borde alrededor del control.
<code>DFCS_CHECKED</code>	Indica que el botón está pulsado o marcado.
<code>DFCS_FLAT</code>	Dibuja el botón con un borde plano.
<code>DFCS_INACTIVE</code>	Dibuja el botón como inactivo.
<code>DFCS_MONO</code>	Dibuja el botón con un borde monocromo.
<code>DFCS_PUSHED</code>	Indica que el botón está pulsado.

## DrawState      Windows.Pas

### Sintaxis

```

DrawState(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: HBRUSH;              {manejador de brocha}
    p3: TFNDrawStateProc;    {dirección de una función de respuesta (opcional)}
    p4: LPARAM;              {manejador de mapa de bits, icono o puntero a cadena}
    p5: WPARAM;              {longitud de la cadena}
    p6: Integer;              {coordenada horizontal de la ubicación de la imagen}
    p7: Integer;              {coordenada vertical de la ubicación de la imagen}
    p8: Integer;              {ancho de la imagen}
    p9: Integer;              {altura de la imagen}
    p10: UINT                 {tipo de imagen y opciones de estado}
): BOOL;                     {devuelve TRUE o FALSE}

```

### Descripción

Esta función muestra un icono, un mapa de bits o una cadena de texto, aplicando un efecto visual para indicar su estado. Se pueden aplicar varios efectos de estado utilizando las opciones válidas para el parámetro `p10`, o se puede llamar a una función de respuesta definida por la aplicación para dibujar efectos de estado complejos definidos por la aplicación.

### Parámetros

`DC`: Manejador del contexto de dispositivo sobre el cual la imagen es dibujada.

`p2`: Especifica un manejador de brocha. Esta brocha será utilizada si el parámetro `p10` contiene la opción `DSS_MONO`. Si el parámetro `p10` no contiene esa opción, este parámetro es ignorado.

`p3`: Especifica un puntero a una función de respuesta definida por la aplicación. Esta función es llamada para dibujar la imagen en un estado específico cuando el parámetro

*p10* contiene la opción *DST\_COMPLEX*. Si el parámetro *p10* no contiene esa opción, este parámetro es ignorado.

*p4*: Si el parámetro *p10* contiene la opción *DST\_BITMAP*, este parámetro contiene el manejador del mapa de bits que será dibujado. Si el parámetro *p10* contiene la opción *DST\_ICON*, este parámetro contendrá el manejador del icono que será dibujado. Si el parámetro *p10* contiene la opción *DST\_PREFIXTEXT* o la opción *DST\_TEXT*, este parámetro contendrá un puntero a la cadena que será dibujada. En caso contrario, a este parámetro puede asignársele un valor definido por la aplicación.

*p5*: Contiene la longitud de la cadena que será dibujada si el parámetro *p10* contiene la opción *DST\_PREFIXTEXT* o la opción *DST\_TEXT*. En este caso, a este parámetro se le puede asignar cero si la cadena termina en nulo. En caso contrario, a este parámetro se le puede asignar un valor definido por la aplicación.

*p6*: Especifica la coordenada horizontal en la cual la imagen será dibujada.

*p7*: Especifica la coordenada vertical en la cual la imagen será dibujada.

*p8*: Especifica el ancho de la imagen en unidades del dispositivo. Si el parámetro *p10* contiene la opción *DST\_COMPLEX*, este parámetro es obligatorio. En caso contrario, se le puede asignar cero, forzando al sistema a calcular el ancho de la imagen.

*p9*: Especifica la altura de la imagen, en unidades del dispositivo. Si el parámetro *p10* contiene la opción *DST\_COMPLEX*, este parámetro será obligatorio. En caso contrario, se le puede asignar cero, forzando al sistema a calcular la altura de la imagen.

*p10*: Una combinación de opciones que indican el tipo y estado de la imagen. A este parámetro se le asigna una combinación que toma una opción de la Tabla 3-18 y otra de la Tabla 3-19.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Sintaxis de la función de respuesta

```
DrawStateProc(
    hDC: HDC;           {manejador de contexto de dispositivo}
    lData: LPARAM;       {datos definidos por la aplicación}
    wParam: WPARAM;     {datos definidos por la aplicación}
    cx: Integer;         {ancho de la imagen}
    cy: Integer;         {altura de la imagen}
    ): BOOL;            {devuelve TRUE o FALSE}
```

#### Descripción

Esta función de respuesta es utilizada cuando el parámetro *p10* contiene la opción *DST\_COMPLEX*. Su propósito es proporcionar una imagen compleja de la manera deseada. Esta función de respuesta puede ejecutar cualquier acción deseada.

**Parámetros**

*hDC*: Manejador del contexto del dispositivo sobre el cual será dibujada la imagen.

*lData*: Proporciona datos específicos de la aplicación, que han sido pasados a la función *DrawState* en el parámetro *p4*.

*wData*: Proporciona datos específicos de la aplicación, que han sido pasados a la función *DrawState* en el parámetro *p5*.

*cx*: Especifica el ancho de la imagen, en unidades de dispositivo, que han sido pasados a la función *DrawState* en el parámetro *p8*.

*cy*: Especifica la altura de la imagen, en unidades de dispositivo, que han sido pasados a la función *DrawState* en el parámetro *p9*.

**Valor que devuelve**

La función de respuesta debe devolver TRUE para indicar que la función tuvo éxito o FALSE para indicar lo contrario.

**Véase además**

*DrawFocusRect*, *DrawText*, *TextOut*, *SetTextColor*

**Ejemplo****Listado 3-13: Dibujando imágenes en estado deshabilitado**

```
procedure TForm1.FormPaint(Sender: TObject);
var
  Text: PChar;          // almacena una cadena de texto
begin
  {Inicializar la cadena de texto}
  Text := 'A DISABLED ICON';

  {Dibujar el texto en la pantalla en estado deshabilitado}
  DrawState(Canvas.Handle, 0, nil, Integer(Text), 0, 20, 20, 0, 0,
    DST_TEXT or DSS_DISABLED);

  {Dibujar el icono de la aplicación en estado deshabilitado}
  DrawState(Canvas.Handle, 0, nil, Application.Icon.Handle, 0, 50, 50, 0, 0,
    DST_ICON or DSS_DISABLED);
end;
```



Figura 3-17:  
Las imágenes  
deshabilitadas

Tabla 3-18: Valores de tipo de imagen para el parámetro `pl0` de `DrawState`

Valor	Descripción
<code>DST_BITMAP</code>	Indica una imagen de mapa de bits. La palabra menos significativa del parámetro <code>p4</code> contiene el manejador del mapa de bits.
<code>DST_COMPLEX</code>	Indica una imagen compleja definida por la aplicación. La función de respuesta, identificada por el parámetro <code>p3</code> , es llamada para producir la imagen.
<code>DST_ICON</code>	Indica una imagen de un icono. La palabra menos significativa del parámetro <code>p4</code> contiene el manejador del icono.
<code>DST_PREFIXTEXT</code>	Indica que la imagen es texto y que puede contener un símbolo acelerador. Cualquier carácter ampersand (&) es convertido en un guión bajo en el siguiente carácter. El parámetro <code>p4</code> contiene un puntero a la cadena y el parámetro <code>p5</code> contiene la longitud de la cadena.
<code>DST_TEXT</code>	Indica que la imagen es texto. El parámetro <code>p4</code> contiene un puntero a la cadena y el parámetro <code>p5</code> la longitud de la cadena.

Tabla 3-19: Valores del estado de la imagen para el parámetro `pl0` de `DrawState`

Valor	Descripción
<code>DSS_NORMAL</code>	Dibuja la imagen en su forma original.
<code>DSS_UNION</code>	Dibuja la imagen en forma difuminada.
<code>DSS_DISABLED</code>	Dibuja la imagen en forma de relieve.
<code>DSS_MONO</code>	Dibuja la imagen utilizando la brocha especificada en el parámetro <code>p2</code> .

**Ellipse****Windows.Pas****Sintaxis**

```

Ellipse(
    DC: HDC;           {manejador del contexto de dispositivo}
    X1: Integer;        {coordenada horizontal de la esquina superior izquierda}
    Y1: Integer;        {coordenada vertical de la esquina superior izquierda}
    X2: Integer;        {coordenada horizontal de la esquina inferior derecha}
    Y2: Integer;        {coordenada vertical de la esquina inferior derecha}
); BOOL;               {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja una elipse dentro del rectángulo límite definido por los parámetros `X1`, `Y1`, `X2` e `Y2`. El centro del rectángulo límite define el centro de la elipse. La elipse

es rellenada con la brocha actual y dibujada con la pluma actual. La posición actual no es utilizada ni actualizada por esta función.

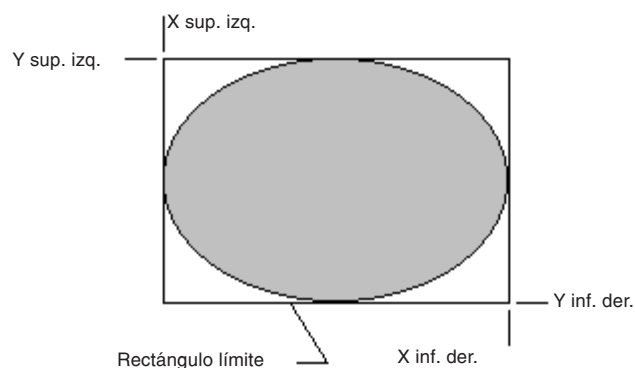


Figura 3-18:  
Coordenadas  
de la elipse

#### Parámetros

*DC*: Manejador del contexto de dispositivo sobre el cual será dibujada la elipse.

*X1*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo límite que define la forma de la elipse. Bajo Windows 95/98, la suma de los parámetros *X1* y *X2* tiene que ser menor que 32.767.

*Y1*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo límite que define la forma de la elipse. Bajo Windows 95/98, la suma de los parámetros *Y1* e *Y2* tiene que ser menor que 32.767.

*X2*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo límite que define la forma de la elipse.

*Y2*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo límite que define la forma de la elipse.

#### Valor del retorno

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*Arc*, *Chord*, *CreateEllipticRgn*, *CreateEllipticRgnIndirect*

#### Ejemplo

##### Listado 3-14: Dibujando elipses

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    {Asignar a la brocha del área de dibujo un color aleatorio}
    Canvas.Brush.Color := $01000000 or Random(10);
```

```

{Dibujar una elipse aleatoria}
Ellipse(Canvas.Handle, Random(ClientWidth), Random(ClientHeight),
        Random(ClientWidth), Random(ClientHeight))
end;

```

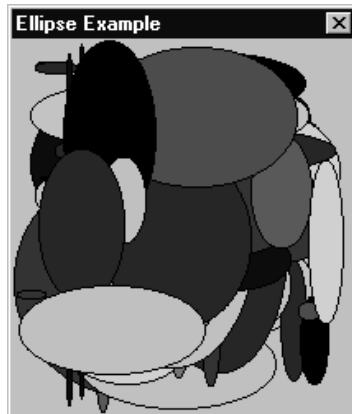


Figura 3-19:  
Elipses  
aleatorias

## EndPoint

## Windows.Pas

### Sintaxis

```

EndPoint(
    hWnd: HWND;                {manejador de ventana}
    const lpPaint: TPaintStruct {puntero a un registro TPaintStruct}
): BOOL;                      {esta función siempre devuelve TRUE}

```

### Descripción

Esta función es usada conjuntamente con la función *BeginPaint* para marcar el fin de la operación de pintado de la ventana especificada. Cualquier cursor de edición oculto por la función *BeginPaint* será restaurado.

### Parámetros

*hWnd*: Especifica el manejador de la ventana a redibujar.

*lpPaint*: Especifica un puntero a un registro *TPaintStruct* que contiene información sobre el redibujado. El registro *TPaintStruct* se define de la siguiente forma:

```

TPaintStruct = packed record
    hdc: HDC;                {manejador de contexto de dispositivo}
    fErase: BOOL;            {opción de borrado del fondo}
    rcPaint: TRect;          {coordenadas del rectángulo a
                             redibujar}
    fRestore: BOOL;          {reservado}
    fIncUpdate: BOOL;        {reservado}

```

```
    rgbReserved: array[0..31] of Byte;    {reservado}
end;
```

Consulte la función *BeginPaint* para ver una descripción de esta estructura de datos.

#### Valor que devuelve

Esta función siempre devuelve TRUE.

#### Véase además

*BeginPaint*

#### Ejemplo

Vea el Listado 3-29 bajo *InvalidateRect* y el Listado 3-30 bajo *InvalidateRgn*.

### **EnumObjects**      **Windows.Pas**

#### Sintaxis

```
EnumObjects(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: Integer;             {opción de tipo de objeto}
    p3: TFNGObjEnumProc;    {función de respuesta definida por la aplicación}
    p4: LPARAM              {datos definidos por la aplicación}
): Integer;                {devuelve un código de éxito}
```

#### Descripción

Esta función enumera todas la plumas o brochas accesibles en el contexto de dispositivo especificado. La información relativa a cada brocha o pluma es pasada a la función de respuesta definida por la aplicación a la que apunta el parámetro *p3*. Este proceso continúa hasta que todos los objetos hayan sido enumerados o la función de respuesta devuelva cero.

#### Parámetros

*DC*: Manejador de contexto de dispositivo que contiene los objetos que serán enumerados.

*p2*: Opción que indica el tipo de objetos que se desea enumerar. Si al parámetro se le asigna *OBJ\_BRUSH*, todas las brochas serán enumeradas. Si se le asigna *OBJ\_PEN*, serán enumeradas todas las plumas.

*p3*: Puntero a la función de respuesta definida por la aplicación.

*p4*: Especifica un valor de 32 bits definido por la aplicación que es pasado a la función de respuesta.

**Valor que devuelve**

Esta función suministra el último valor devuelto por la función de respuesta. Si hay demasiados objetos a enumerar la función devuelve -1. Esta función no indica errores en caso de fallo.

**Sintaxis de la función de respuesta**

```
EnumObjectsProc(
    lpLogObject: Pointer;           {puntero a registro de datos del objeto}
    lpData: LPARAM                 {datos definidos por la aplicación}
): Integer;                       {devuelve cero o uno}
```

**Descripción**

Esta función es llamada una vez para cada objeto del tipo especificado asociado al contexto de dispositivo. En ella se puede ejecutar cualquier acción deseada.

**Parámetros**

*lpLogObject*: Un puntero a un registro *TLogPen* si el parámetro *p2* de la función contiene la opción *OBJ\_PEN*, o un puntero a un registro *TLogBrush* si el parámetro *p2* contiene la opción *OBJ\_BRUSH*. Consulte la función *CreatePenIndirect* para ver una descripción del registro *TLogPen*, y la función *CreateBrushIndirect* para ver una descripción del parámetro *TLogBrush*.

*lpData*: Especifica un valor de 32 bits definido por la aplicación pasado a través del parámetro *p4* de la función *EnumObjects*. Este valor está destinado a uso específico de la aplicación.

**Valor que devuelve**

Esta función debe devolver uno para continuar la enumeración o cero para finalizarla.

**Véase además**

*GetObject*, *GetObjectType*

**Ejemplo****Listado 3-15: Enumerando todas las plumas en un contexto de dispositivo**

```
{El prototipo de la función de respuesta}
function EnumObjProc(ObjType: PLogPen; lData: lParam): Integer; stdcall;
var
    Form1: TForm1;

implementation

{$R *.DFM}

function EnumObjProc(ObjType: PLogPen; lData: lParam): Integer;
var
    LocalObjType: TLogPen;    // almacena información de la pluma lógica
```

```

    PenDescription: String; // almacena una descripción de una pluma
begin
    {Obtener la información de la pluma}
    LocalObjType := ObjType^;

    {Determinar el tipo de pluma}
    case LocalObjType.lpnStyle of
        PS_SOLID: PenDescription := 'PS_SOLID';
        PS_DASH: PenDescription := 'PS_DASH';
        PS_DOT: PenDescription := 'PS_DOT';
        PS_DASHDOT: PenDescription := 'PS_DASHDOT';
        PS_DASHDOTDOT: PenDescription := 'PS_DASHDOTDOT';
        PS_NULL: PenDescription := 'PS_NULL';
        PS_INSIDEFRAME: PenDescription := 'PS_INSIDEFRAME';
    end;

    {Determinar el color de la pluma}
    case LocalObjType.lpnColor of
        clBlack: PenDescription := PenDescription + ' Color: clBlack';
        clMaroon: PenDescription := PenDescription + ' Color: clMaroon';
        clGreen: PenDescription := PenDescription + ' Color: clGreen';
        clOlive: PenDescription := PenDescription + ' Color: clOlive';
        clNavy: PenDescription := PenDescription + ' Color: clNavy';
        clPurple: PenDescription := PenDescription + ' Color: clPurple';
        clTeal: PenDescription := PenDescription + ' Color: clTeal';
        clGray: PenDescription := PenDescription + ' Color: clGray';
        clSilver: PenDescription := PenDescription + ' Color: clSilver';
        clRed: PenDescription := PenDescription + ' Color: clRed';
        clLime: PenDescription := PenDescription + ' Color: clLime';
        clYellow: PenDescription := PenDescription + ' Color: clYellow';
        clBlue: PenDescription := PenDescription + ' Color: clBlue';
        clFuchsia: PenDescription := PenDescription + ' Color: clFuchsia';
        clAqua: PenDescription := PenDescription + ' Color: clAqua';
        clWhite: PenDescription := PenDescription + ' Color: clWhite';
    end;

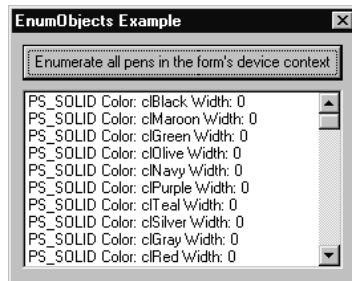
    {Indicar el ancho de la pluma}
    PenDescription := PenDescription + ' Width: ' + IntToStr(LocalObjType.lpnWidth.X);

    {Añadir la descripción al cuadro de lista}
    Form1.ListBox.Items.Add(PenDescription);
    {Indicar que la enumeración debe continuar}
    Result := 1;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Enumerar todas las plumas en el contexto de dispositivo del formulario}
    EnumObjects(Canvas.Handle, OBJ_PEN, @EnumObjProc, 0);
end;

```

Figura 3-20:  
Listando todas  
las plumas en  
el contexto de  
dispositivo



### ExtCreatePen

### Windows.Pas

#### Sintaxis

```
ExtCreatePen(
    PenStyle: DWORD;           {tipo, estilo, terminación y opción de unión de la pluma}
    Width: DWORD;              {ancho de la pluma}
    const Brush: TLogBrush;    {puntero a registro TLogBrush}
    StyleCount: DWORD;         {cantidad de entradas en el array de entradas a la medida}
    Style: Pointer              {puntero a array de longitudes de guiones y espacios}
): HPEN;                      {devuelve un manejador de pluma}
```

#### Descripción

Esta función crea una nueva pluma geométrica o cosmética con los atributos especificados. Las plumas geométricas pueden ser de cualquier ancho y tienen los mismos atributos que las brochas. Las plumas cosméticas tienen que ser siempre de un píxel de ancho, pero se dibujan más rápidamente que las geométricas. Adicionalmente, bajo Windows NT, esta función permite crear una pluma con un patrón definido por el usuario. Cuando la aplicación deje de necesitar la pluma, ésta deberá ser eliminada llamando a la función *DeleteObject*.

#### Parámetros

**PenStyle:** Una combinación de opciones que definen el tipo, estilo, terminación y unión de la pluma. Este parámetro puede contener una combinación de valores, donde un valor es tomado de la Tabla 3-20 y otro de la Tabla 3-21. Si este parámetro contiene la opción de estilo *PS\_GEOMETRIC*, puede contener una combinación adicional, con un valor tomado de la Tabla 3-22 y otro de la Tabla 3-23. Nótese que bajo Windows 95/98, los estilos del extremo y unión son soportados sólo para plumas geométricas cuando éstas son utilizadas para dibujar una ruta.

**Width:** Especifica el ancho de la pluma, en unidades lógicas. Si el parámetro *PenStyle* contiene la opción *PS\_COSMETIC*, a este parámetro se le debe asignar uno (1).

**Brush:** Un puntero al registro *TLogBrush* que define los atributos adicionales de la pluma. Si el parámetro *PenStyle* contiene la opción *PS\_COSMETIC*, el campo *lbColor* de este registro especifica el color de la pluma y al campo *lbStyle* hay que asignarle

*BS\_SOLID*. Si el parámetro *PenStyle* contiene la opción *PS\_GEOMETRIC*, todos los campos de este registro son usados para especificar los atributos de la pluma. El registro *TLogBrush* se define de la siguiente forma:

```
TLogBrush = packed record
    lbStyle: UINT;           {opción de estilo de la brocha}
    lbColor: COLORREF;      {un especificador de color}
    lbHatch: Longint;       {opción de estilo del sombreado}
end;
```

Observe que si el campo *lbHatch* apunta a un mapa de bits, este mapa de bits no puede haber sido creado con la función *CreateDIBSection*. Consulte la función *CreateBrushIndirect* para ver una descripción de esta estructura de datos.

*StyleCount*: Especifica la cantidad de entradas en el *array* de estilos de pluma definidos por el usuario al que apunta el parámetro *Style*. Si el parámetro *PenStyle* no contiene la opción *PS\_USERSTYLE*, este parámetro es ignorado.

*Style*: Un puntero a un *array* de valores que definen el patrón de los guiones y espacios para un estilo de pluma definido por el usuario. La primera entrada en el *array* especifica la longitud del primer guión, en unidades lógicas. La segunda entrada especifica la longitud del primer espacio, en unidades lógicas. Esto continúa hasta que el patrón está completamente definido. El patrón se repetirá tantas veces como sea necesario cuando se dibuje una línea utilizando la pluma. Si el parámetro *PenStyle* no contiene la opción *PS\_USERSTYLE*, este parámetro es ignorado.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la nueva pluma; en caso contrario, devuelve cero.

#### Véase además

*CreateBrushIndirect*, *CreatePen*, *CreatePenIndirect*, *DeleteObject*, *GetObject*, *SelectObject*, *SetMiterLimit*

#### Ejemplo

##### Listado 3-16: Dibujando rutas con plumas geométricas

```
procedure TForm1.Button1Click(Sender: TObject);
var
    NewPen, OldPen: HPen; // las plumas vieja y nueva
    FormDC: HDC;         // manejador del contexto de dispositivo del formulario
    BrushInfo: TLogBrush; // el registro lógico de la brocha
    MiterLimit: Single;   // el límite del inglete
begin
    {Obtener el contexto de dispositivo del formulario}
    FormDC := GetDC(Form1.Handle);

    {Definir la brocha}
    with BrushInfo do begin
```

```

    lbStyle := BS_SOLID;
    lbColor := clBlue;
    lbHatch := 0;
end;

{Crear pluma geométrica con terminación cuadrada y uniones en forma de inglete,
de 20 unidades de ancho}
NewPen := ExtCreatePen(PS_GEOMETRIC or PS_ENDCAP_SQUARE or PS_JOIN_MITER, 20,
    BrushInfo, 0, nil);

{Seleccionar la pluma dentro del contexto de dispositivo del formulario}
OldPen := SelectObject(FormDC, NewPen);

{Comenzar definición de la ruta}
BeginPath(FormDC);

{Definir una ruta triangular cerrada}
MoveToEx(FormDC, ClientWidth div 2, 20, nil);
LineTo(FormDC, ClientWidth - 20, 90);
LineTo(FormDC, 20, 90);
CloseFigure(FormDC);

{Terminar definición de la ruta}
EndPath(FormDC);

{Asegurar que el límite del inglete es de 2 unidades}
GetMiterLimit(FormDC, MiterLimit);
if MiterLimit > 2 then
    SetMiterLimit(FormDC, 2, nil);

{Dibujar la ruta con la pluma geométrica}
StrokePath(FormDC);

{Eliminar la pluma y el contexto de dispositivo}
SelectObject(FormDC, OldPen);
ReleaseDC(Form1.Handle, FormDC);
DeleteObject(NewPen);
end;

```

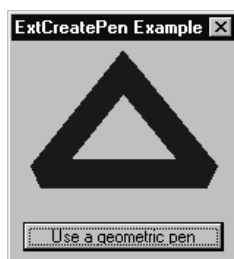


Figura 3-21:  
La pluma  
geométrica en  
acción

Tabla 3-20: Valores del tipo de pluma en el parámetro **PenStyle** de **ExtCreatePen**

Valor	Descripción
PS_GEOMETRIC	Indica una pluma geométrica.
PS_COSMETIC	Indica una pluma cosmética.

Tabla 3-21: Valores de estilo de pluma en el parámetro **PenStyle** de **ExtCreatePen**

Valor	Descripción
PS_ALTERNATE	Sólo Windows NT: Asigna todos los demás píxeles cuando se dibuja una línea (sólo para plumas cosméticas).
PS_SOLID	Crea una pluma sólida.
PS_DASH	Crea una pluma de guiones. Sólo Windows 95/98: Este estilo no está soportado para plumas geométricas.
PS_DOT	Crea una pluma de puntos. Sólo Windows 95/98: Este estilo no está soportado para plumas geométricas.
PS_DASHDOT	Crea una pluma que alterna guiones y puntos. Sólo Windows 95/98: Este estilo no está soportado para plumas geométricas.
PS_DASHDOTDOT	Crea una pluma que alterna un guión y dos puntos. Sólo Windows 95/98: Este estilo no está soportado para plumas geométricas.
PS_NULL	Crea una pluma invisible.
PS_USERSTYLE	Sólo Windows NT: Crea una pluma de estilo definido por el usuario. El parámetro <b>Style</b> apunta a un array de valores <b>DWORD</b> que especifica los guiones y espacios de la pluma.
PS_INSIDEFRAME	Crea una pluma sólida. Cuando esta pluma es usada con una función que especifica un rectángulo límite, las dimensiones de la figura son reducidas de manera que toda la figura, cuando sea dibujada, quepa dentro del rectángulo límite (sólo para plumas geométricas).

Tabla 3-22: Estilos de punto final en el parámetro **PenStyle** de **ExtCreatePen** (sólo para plumas geométricas)

Valor	Descripción
PS_ENDCAP_ROUND	Los extremos de las líneas son redondeados.
PS_ENDCAP_SQUARE	Los extremos de las líneas son cuadrados.
PS_ENDCAP_FLAT	Los extremos de las líneas son planos.

Tabla 3-23: Estilos de unión de líneas en el parámetro **PenStyle** de **ExtCreatePen** (sólo para plumas geométricas)

Valor	Descripción
PS_JOIN_BEVEL	Las uniones de línea son biseladas.

Valor	Descripción
PS_JOIN_MITER	Las uniones de línea tienen estilo de inglete cuando están dentro de los límites actuales asignados por la función SetMiterLimit. Si exceden esos límites, la unión es biselada.
PS_JOIN_ROUND	Las uniones de líneas son redondeadas.

**ExtFloodFill****Windows.Pas****Sintaxis**

```
ExtFloodFill(
    DC: HDC;                {manejador del contexto de dispositivo}
    X: Integer;              {coordenada horizontal del origen del relleno}
    Y: Integer;              {coordenada vertical del origen del relleno}
    Color: COLORREF;         {color de relleno}
    FillType: UINT           {opción de tipo de relleno}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función rellena un área del contexto de dispositivo especificado con la brocha actual.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el cual el relleno se dibujará.

*X*: Especifica la coordenada horizontal, en unidades lógicas, del origen del relleno.

*Y*: Especifica la coordenada vertical, en unidades lógicas, del origen del relleno.

*Color*: Un especificador de color que indica el color que delimita el borde del área que será rellena. El significado de este parámetro depende del valor del parámetro *FillType*.

*FillType*: Una opción que indica el tipo de relleno a ejecutar. A este parámetro se le puede asignar un valor de la Tabla 3-24.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*FillPath*, *FillRect*, *FillRgn*, *GetDeviceCaps*

*Ejemplo***Listado 3-17: Rellenando un área**

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Asignar el color de la brocha usada para el flujo de relleno}
    Canvas.Brush.Color := clLime;

    {Rellenar el cuadrado rojo con el nuevo color de la brocha}
    ExtFloodFill(Canvas.Handle, 20, 20, clRed, FLOODFILLSURFACE);
end;

```

**Tabla 3-24: Valores del parámetro FillType de ExtFloodFill**

<b>Valor</b>	<b>Descripción</b>
FLOODFILLBORDER	Indica que el área que será rellenada está rodeada por píxeles del color especificado en el parámetro Color. La función rellena píxeles en todas direcciones a partir del origen con el color de la brocha <u>hasta que</u> el color especificado por el parámetro Color es encontrado.
FLOODFILLSURFACE	Indica que el área que será rellenada está definida por un color sólido. La función rellena píxeles en todas direcciones a partir del origen con el color de la brocha <u>mientras</u> el color especificado por el parámetro Color es encontrado.

**FillPath****Windows.Pas***Sintaxis*

```

FillPath(
    DC: HDC                {manejador de contexto de dispositivo}
); BOOL;                  {devuelve TRUE o FALSE}

```

*Descripción*

Esta función cierra cualquier ruta abierta en el contexto de dispositivo especificado y rellena su interior con la brocha actual. La ruta es rellenada de acuerdo al modo de relleno de polígono actual. Observe que cuando esta función retorne, la ruta habrá sido descartada del contexto de dispositivo.

*Parámetros*

*DC*: Manejador del contexto de dispositivo que contiene la ruta válida que será rellenada.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*BeginPath*, *ExtFloodFill*, *FillRgn*, *SetPolyFillMode*, *StrokeAndFillPath*, *StrokePath*

**Ejemplo****Listado 3-18: Rellenando el interior de una ruta**

```

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Iniciar definición de ruta}
    BeginPath(Canvas.Handle);

    {Dibujar texto en la ruta, indicando que la ruta consiste en el texto interior}
    SetBkMode(Canvas.Handle, TRANSPARENT);
    Canvas.TextOut(10, 10, 'DELPHI ROCKS!');

    {Terminar definición de ruta}
    EndPath(Canvas.Handle);

    {Inicializar la brocha del área de dibujo}
    Canvas.Brush.Color := clBlue;
    Canvas.Brush.Style := bsDiagCross;

    {Rellenar la ruta con la brocha actual}
    FillPath(Canvas.Handle);
end;

```



Figura 3-22:  
La ruta  
rellenada

**FillRect****Windows.Pas****Sintaxis**

```

FillRect(
    hDC: HDC;                {manejador de contexto de dispositivo}
    const lprc: TRect;        {coordenadas del rectángulo}
    hbr: HBRUSH               {manejador de la brocha}
); Integer;                  {devuelve cero o uno}

```

**Descripción**

Esta función rellena el área del rectángulo especificado en el contexto de dispositivo, utilizando la brocha indicada. Los bordes superior e izquierdo del rectángulo son incluidos en el relleno, pero los bordes inferior y derecho son excluidos.

**Parámetros**

*hDC*: Manejador del contexto de dispositivo sobre el cual se dibujará el rectángulo relleno.

*lprc*: Un puntero al registro *TRect* que define las coordenadas del rectángulo, en unidades lógicas, del área que será rellena.

*hbr*: Especifica el manejador de la brocha que será usada para relleno el rectángulo. Opcionalmente, puede utilizarse un color del sistema para relleno el rectángulo, asignándole a este parámetro un valor de la Tabla 3-25. Observe que cuando se utilice un color del sistema, deberá añadirse uno (1) al valor (por ejemplo, *COLOR\_ACTIVEBORDER + 1*).

**Valor que devuelve**

Si la función tiene éxito, devuelve uno; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateHatchBrush*, *CreatePatternBrush*, *CreateSolidBrush*, *GetStockObject*, *ExtFloodFill*, *FrameRect*, *FillPath*, *FillRgn*

**Ejemplo**

Vea el Listado 3-5 bajo *CreatePatternBrush*.

**Tabla 3-25: Valores de color para el parámetro *hbr* de *FillRect***

Valor	Descripción
<i>COLOR_3DDKSHADOW</i>	El color de sombra oscura para elementos de visualización tridimensionales.
<i>COLOR_3DLIGHT</i>	El color de borde iluminado para elementos de visualización tridimensionales.
<i>COLOR_ACTIVEBORDER</i>	El color del borde de las ventanas activas.
<i>COLOR_ACTIVECAPTION</i>	El color de las barras de título de la ventana activa.
<i>COLOR_APPWORKSPACE</i>	El color de fondo usado en aplicaciones que siguen la Interfaz de Documentos Múltiples (MDI).
<i>COLOR_BACKGROUND</i>	El color del escritorio.
<i>COLOR_BTNFACE</i>	El color de la superficie de los botones.
<i>COLOR_BTNHIGHLIGHT</i>	El color de un botón resaltado.
<i>COLOR_BTNSHADOW</i>	El color del borde oscuro de los botones.

Valor	Descripción
COLOR_BTNTEXT	El color del texto de los botones.
COLOR_CAPTIONTEXT	El color del texto usado en las barras de título, cajas de redimensionamiento y controles de flechas de las barras de desplazamiento.
COLOR_GRAYTEXT	El color de un texto deshabilitado. Este valor será cero si el manipulador del dispositivo de visualización no soporta gris sólido.
COLOR_HIGHLIGHT	El color usado para los elementos seleccionados en un control.
COLOR_HIGHLIGHTTEXT	El color usado para el texto de los elementos seleccionados en un control.
COLOR_INACTIVEBORDER	El color del borde de las ventanas inactivas.
COLOR_INACTIVECAPTION	El color de la barra de título de las ventanas inactivas.
COLOR_INACTIVECAPTIONTEXT	El color del texto de una barra de título inactiva.
COLOR_INFOBK	El color del fondo para las indicaciones.
COLOR_INFOTEXT	El color del texto para las indicaciones.
COLOR_MENU	El color del fondo de los menús.
COLOR_MENUTEXT	El color del texto de los menús.
COLOR_SCROLLBAR	El color del área “gris” de las barras de desplazamiento.
COLOR_WINDOW	El color de fondo de las ventanas.
COLOR_WINDOWFRAME	El color del marco de las ventanas.
COLOR_WINDOWTEXT	El color del texto usado en las ventanas.

**FillRgn****Windows.Pas****Sintaxis**

```

FillRgn(
    DC: HDC;           {manejador de contexto de dispositivo}
    p2: HRGN;          {manejador de la región}
    p3: HBRUSH         {manejador de brocha}
); BOOL;              {devuelve TRUE o FALSE}

```

**Descripción**

Esta función rellena la región especificada con la brocha identificada por el parámetro *p3*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el cual se dibujará la región rellena.

*p2*: Especifica un manejador de la región que será rellenada.

*p3*: Especifica un manejador de la brocha que se utilizará para rellenar la región.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*CreateBrushIndirect*, *CreateHatchBrush*, *CreatePatternBrush*, *CreateSolidBrush*, *FrameRgn*, *FillPath*, *FillRect*, *PaintRgn*

#### Ejemplo

Vea el listado 3-3 bajo *CreateBrushIndirect*.

### FrameRect Windows.Pas

#### Sintaxis

```
FrameRect(
    hDC: HDC;                {manejador de contexto de dispositivo}
    const lprc: TRect;        {coordenadas del rectángulo}
    hbr: HBRUSH               {manejador de brocha}
): Integer;                 {devuelve cero o uno}
```

#### Descripción

Esta función dibuja un borde alrededor del rectángulo indicado en el contexto de dispositivo especificado, utilizando la brocha identificada mediante el parámetro *hbr*. Este borde siempre tiene el ancho de una unidad lógica.

#### Parámetros

*hDC*: Manejador del contexto de dispositivo sobre el cual el marco rectangular será dibujado.

*lprc*: Puntero al registro *TRect* que contiene las coordenadas del rectángulo que define el recuadro.

*hbr*: El manejador de la brocha utilizada para dibujar el recuadro rectangular.

#### Valor que devuelve

Si la función tiene éxito, devuelve uno; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateHatchBrush*, *CreatePatternBrush*, *CreateSolidBrush*, *FillRect*, *FrameRgn*, *GetStockObject*, *Rectangle*

*Ejemplo***Listado 3-19: Dibujando un marco rectangular**

```

procedure TForm1.Button1Click(Sender: TObject);
var
    TheRect: TRect;    // las coordenadas del rectángulo
begin
    {Definir el rectángulo}
    TheRect := Rect(10, 10, 110, 110);

    {Inicializar la brocha}
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsCross;

    {Enmarcar el rectángulo}
    FrameRect(Canvas.Handle, TheRect, Canvas.Brush.Handle);
end;

```

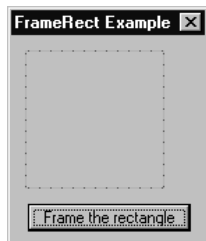


Figura 3-23:  
El rectángulo  
enmarcado

**FrameRgn****Windows.Pas***Sintaxis*

FrameRgn(	
DC: HDC;	{manejador de contexto de dispositivo}
p2: HRGN;	{manejador de región}
p3: HBRUSH;	{manejador de brocha}
p4: Integer;	{ancho de brochazo vertical}
p5: Integer	{altura del brochazo horizontal}
); BOOL;	{devuelve TRUE o FALSE}

*Descripción*

Esta función dibuja el perímetro de la región especificada, utilizando la brocha identificada por el parámetro *p3*.

*Parámetros*

*DC*: Manejador para un contexto de dispositivo sobre el cual la región enmarcada es dibujada.

*p2*: Manejador de la región cuyo perímetro está siendo dibujado.

*p3*: Manejador de la brocha a utilizar para dibujar el marco.

*p4*: Especifica el ancho vertical de la brocha cuando se dibuja el marco, en unidades lógicas.

*p5*: Especifica la altura horizontal de la brocha cuando se dibuja el marco, en unidades lógicas.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, FrameRect, FillRgn, PaintRgn*

#### Ejemplo

##### Listado 3-20: Enmarcando una región

```
procedure TForm1.Button1Click(Sender: TObject);
var
    RegionHandle: HRGN;           // manejador de la región
    PointsArray: array[0..5] of TPoint; // puntos que definen la región
begin
    {Definir y crear la región poligonal}
    PointsArray[0].x := 50;   PointsArray[0].y := 50;
    PointsArray[1].x := 100;  PointsArray[1].y := 50;
    PointsArray[2].x := 125;  PointsArray[2].y := 75;
    PointsArray[3].x := 100;  PointsArray[3].y := 100;
    PointsArray[4].x := 50;   PointsArray[4].y := 100;
    PointsArray[5].x := 25;   PointsArray[5].y := 75;
    {Crear la región}
    RegionHandle := CreatePolygonRgn(PointsArray, 6, ALTERNATE);

    {Enmarcar la región en negro}
    Canvas.Brush.Color := clBlack;
    FrameRgn(Canvas.Handle, RegionHandle, Canvas.Brush.Handle, 2, 2);
end;
```

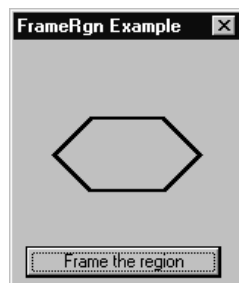


Figura 3-24:  
La región  
dentro del  
marco

**GetBkColor****Windows.Pas****Sintaxis**

```
GetBkColor(
    hDC: HDC                {manejador de contexto de dispositivo}
): COLORREF;               {devuelve el color de fondo}
```

**Descripción**

Esta función recupera el color de fondo del contexto de dispositivo especificado.

**Parámetros**

*hDC*: Manejador del contexto de dispositivo cuyo color de fondo será recuperado.

**Valor que devuelve**

Si la función tiene éxito, devuelve un especificador de color que describe el color de fondo; en caso contrario, devuelve *CLR\_INVALID*.

**Véase además**

*GetBkMode*, *SetBkColor*

**Ejemplo****Listado 3-21: Dibujando texto con y sin color de fondo**

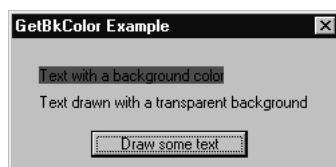
```
procedure TForm1.Button1Click(Sender: TObject);
begin
    {Si el color de fondo no es rojo, hacerlo rojo}
    if GetBkColor(Canvas.Handle) <> clRed then
        SetBkColor(Canvas.Handle, clRed);

    {Mostrar algún texto; el color de fondo será usado}
    Canvas.TextOut(20, 20, 'Text with a background color');

    {Si el modo del fondo no es transparente, hacerlo transparente}
    if GetBkMode(Canvas.Handle) <> TRANSPARENT then
        SetBkMode(Canvas.Handle, TRANSPARENT);

    {Dibujar algún texto; el color de fondo no será usado}
    Canvas.TextOut(20, 40, 'Text drawn with a transparent background');
end;
```

Figura 3-25:  
Texto con y sin  
color de fondo



**GetBkMode****Windows.Pas****Sintaxis**

```
GetBkMode(
    hDC: HDC;                {manejador de contexto de dispositivo}
): Integer;                 {devuelve el modo actual del fondo}
```

**Descripción**

Esta función recupera el modo de mezcla actual del fondo para el contexto de dispositivo especificado.

**Parámetros**

*hDC*: Manejador del contexto de dispositivo cuyo modo de mezcla actual del fondo será recuperado.

**Valor que devuelve**

Si la función tiene éxito, devuelve una opción que indica el modo de mezcla actual del fondo del contexto de dispositivo especificado. Esta opción puede ser *OPAQUE* o *TRANSPARENT*. Consulte la función *SetBkMode* para ver una descripción de estas opciones. Si la función falla, devuelve cero.

**Véase además**

*GetBkColor*, *SetBkMode*

**Ejemplo**

Vea el Listado 3-21 bajo *GetBkColor*.

**GetBoundsRect****Windows.Pas****Sintaxis**

```
GetBoundsRect(
    DC: HDC;                {manejador de contexto de dispositivo}
    var p2: TRect;          {puntero a un registro TRect}
    p3: UINT                {opciones de operación}
): UINT;                   {devuelve el estado del rectángulo límite acumulativo}
```

**Descripción**

Esta función recupera el rectángulo límite actual para el contexto de dispositivo especificado. Windows mantiene un rectángulo límite acumulativo para cada contexto de dispositivo, el cual identifica las dimensiones de la salida producida por las funciones de dibujo. Cuando una función de dibujo sobrepasa estas fronteras, el rectángulo límite es extendido. De esta manera, el rectángulo límite es el rectángulo

más pequeño que puede ser trazado alrededor del área afectada por las operaciones de dibujo efectuadas sobre el contexto de dispositivo.

#### Parámetros

*DC*: Manejador del contexto de dispositivo cuyo rectángulo límite acumulativo será recuperado.

*p2*: Puntero a un registro *TRect* que recibe las coordenadas del rectángulo límite del contexto de dispositivo.

*p3*: Opción que indica si el rectángulo límite será borrado. Si a este parámetro se le asigna cero, el rectángulo límite no será modificado. Si se le asigna *DCB\_RESET*, el rectángulo límite es borrado.

#### Valor que devuelve

Esta función devuelve un código que indica el estado del rectángulo límite; será una combinación de uno o más valores de la Tabla 3-26. En caso de fallo, la función devuelve una condición de error.

#### Véase además

*GetUpdateRect*, *SetBoundsRect*

#### Ejemplo

##### Listado 3-22: Asignando y recuperando el rectángulo límite del contexto de dispositivo

```
procedure TForm1.Button1Click(Sender: TObject);
var
    TheRect: TRect;           // recibe el rectángulo límite
    FormDC: HDC;              // manejador del contexto de dispositivo del formulario
    BoundRectState: UINT;     // almacena el estado del rectángulo límite
begin
    {Obtener el contexto de dispositivo del formulario}
    FormDC := GetDC(Form1.Handle);

    {Inicializar y asignar el rectángulo límite}
    TheRect := Rect(10, 10, 110, 110);
    SetBoundsRect(FormDC, @TheRect, DCB_ENABLE);

    {Recuperar el rectángulo límite}
    BoundRectState := GetBoundsRect(FormDC, TheRect, 0);

    {Liberar el contexto de dispositivo}
    ReleaseDC(Form1.Handle, FormDC);

    {Mostrar las coordenadas del rectángulo límite}
    with TheRect do
    begin
        Label1.Caption := 'Top: ' + IntToStr(Top) + ' Left: ' + IntToStr(Left) +
            ' Bottom: ' + IntToStr(Bottom) + ' Right: ' + IntToStr(Right);
    end;
```

```

{Mostrar el estado del rectángulo límite}
case BoundRectState of
  DCB_DISABLE: Label2.Caption := 'State: DCB_DISABLE';
  DCB_ENABLE:  Label2.Caption := 'State: DCB_ENABLE';
  DCB_RESET:   Label2.Caption := 'State: DCB_RESET';
  DCB_SET:     Label2.Caption := 'State: DCB_SET';
end;
end;

```

Figura 3-26:  
El rectángulo  
límite actual

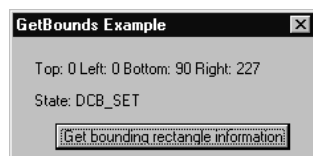


Tabla 3-26: Valores que devuelve GetBoundsRect

Valor	Descripción
0	Indica que ha ocurrido un error.
DCB_DISABLE	La acumulación de límite está desactivada.
DCB_ENABLE	La acumulación de límite está activada.
DCB_RESET	El rectángulo límite está vacío.
DCB_SET	El rectángulo límite no está vacío.

## GetBrushOrgEx

## Windows.Pas

### Sintaxis

```

GetBrushOrgEx(
  DC: HDC;                {manejador de contexto de dispositivo}
  var p2: TPoint           {puntero a un registro TPoint}
): BOOL;                  {devuelve TRUE o FALSE}

```

### Descripción

Esta función recupera el origen de la brocha para el contexto de dispositivo especificado. El origen de la brocha es relativo a la trama o al mapa de bits que define el patrón de la brocha. El origen por defecto de la brocha está situado en el punto (0,0). El patrón de una brocha no puede ser mayor de ocho píxeles cuadrados. Por eso, las coordenadas del origen pueden variar entre 0 y 7, vertical y horizontalmente. Cuando el origen es movido, el patrón de la brocha es desplazado en la cantidad especificada. Si una aplicación está utilizando un patrón de brocha para dibujar el fondo de ventanas hijas y madres, puede ser necesario mover el origen de la brocha para alinear los patrones. Observe que bajo Windows NT, el sistema mantiene automáticamente el origen de la brocha para que los patrones estén alineados.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo origen de brocha se desea recuperar.

*P2*: Puntero a registro *TPoint* que recibirá las coordenadas del origen de la brocha, en unidades del dispositivo.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateBrushIndirect*, *CreateHatchBrush*, *CreatePatternBrush*, *FillRect*, *FillRgn*, *SelectObject*, *SetBrushOrgEx*

**Ejemplo**

Vea el Listado 3-42 bajo *Rectangle*.

**GetCurrentObject****Windows.Pas****Sintaxis**

```
GetCurrentObject(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: UINT                 {opción de tipo de objeto}
): HGDIOBJ;                {devuelve un manejador de objeto del GDI}
```

**Descripción**

Esta función devuelve un manejador del objeto del tipo especificado que está actualmente seleccionado en el contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Manejador para el contexto de dispositivo cuyo objeto actualmente seleccionado se desea recuperar.

*p2*: Opción que especifica qué tipo de objeto se desea recuperar. A este parámetro se le puede asignar un valor de la Tabla 3-27.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del objeto del tipo especificado actualmente seleccionado. Si falla, devuelve cero.

**Véase además**

*DeleteObject*, *GetObject*, *GetObjectType*, *SelectObject*

**Ejemplo**

Vea el Listado 3-24 bajo *GetObject*.

**Tabla 3-27: Valores del parámetro p2 de *GetCurrentObject***

Valor	Descripción
OBJ_PEN	Recupera un manejador de la pluma actualmente seleccionada.
OBJ_BRUSH	Recupera un manejador de la brocha actualmente seleccionada.
OBJ_PAL	Recupera un manejador de la paleta actualmente seleccionada.
OBJ_FONT	Recupera un manejador de la fuente actualmente seleccionada.
OBJ_BITMAP	Recupera un manejador del mapa de bits actualmente seleccionado si el parámetro DC identifica un contexto de dispositivo.

## ***GetCurrentPositionEx*      *Windows.Pas***

**Sintaxis**

```
GetCurrentPositionEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    Point: PPoint            {puntero a un registro TPoint}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recupera las coordenadas de la posición actual en unidades lógicas.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuya posición actual es recuperada.

*Point*: Puntero a un registro *TPoint* que recibe las coordenadas de la posición actual, en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*LineTo*, *MoveToEx*, *PolyBezierTo*, *PolylineTo*

**Ejemplo**

**Listado 3-23: Mostrando la posición actual**

```
procedure TForm1.Button1Click(Sender: TObject);
var
    CurPosPt: TPoint;        // almacena la posición actual
begin
```

```

{Seleccionar el modo de fondo a transparente}
SetBkMode(Canvas.Handle, TRANSPARENT);

{Mostrar el primer punto}
MoveToEx(Canvas.Handle, 60, 20, nil);
GetCurrentPositionEx(Canvas.Handle, @CurPosPt);
TextOut(Canvas.Handle, CurPosPt.x - 55, CurPosPt.y,
        PChar('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)),
        Length('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)));

{Mostrar el segundo punto}
LineTo(Canvas.Handle, 160, 20);
GetCurrentPositionEx(Canvas.Handle, @CurPosPt);
TextOut(Canvas.Handle, CurPosPt.x + 2, CurPosPt.y,
        PChar('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)),
        Length('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)));

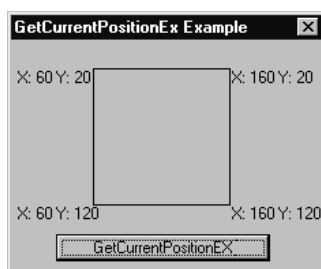
{Mostrar el tercer punto}
LineTo(Canvas.Handle, 160, 120);
GetCurrentPositionEx(Canvas.Handle, @CurPosPt);
TextOut(Canvas.Handle, CurPosPt.x + 2, CurPosPt.y,
        PChar('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)),
        Length('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)));

{Mostrar el cuarto punto}
LineTo(Canvas.Handle, 60, 120);
GetCurrentPositionEx(Canvas.Handle, @CurPosPt);
TextOut(Canvas.Handle, CurPosPt.x - 55, CurPosPt.y,
        PChar('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)),
        Length('X: ' + IntToStr(CurPosPt.X) + ' Y: ' + IntToStr(CurPosPt.Y)));

{Cerrar la figura}
LineTo(Canvas.Handle, 60, 20);
end;

```

Figura 3-27:  
Obteniendo la  
posición  
actual



## GetMiterLimit Windows.Pas

### Sintaxis

```

GetMiterLimit(
    DC: HDC;                {manejador de contexto de dispositivo}

```

```

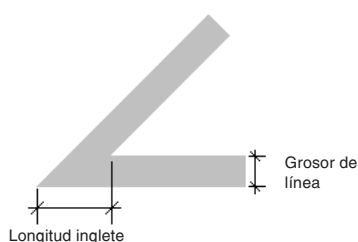
var Limit: Single    {puntero a la variable que recibe el límite del inglete}
): BOOL;             {devuelve TRUE o FALSE}

```

### Descripción

Esta función recupera el límite del inglete para el contexto de dispositivo especificado. El límite del inglete es usado para líneas geométricas que tienen uniones de inglete, y es la relación entre el largo del inglete y el ancho de la línea. La longitud del inglete es la distancia desde la intersección de la pared interna con la pared externa.

Figura 3-28:  
Dimensiones  
del límite del  
inglete



### Parámetros

*DC*: Un manejador para el contexto de dispositivo desde el cual el límite del inglete será recuperado.

*Limit*: Un puntero a una variable de tipo *Single* que recibe el límite del inglete del contexto de dispositivo.

### Valor que devuelve

Si la función tiene éxito, devuelve *TRUE*; en caso contrario, devuelve *FALSE*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*ExtCreatePen*, *SetMiterLimit*

### Ejemplo

Vea el Listado 3-16 bajo *ExtCreatePen*.

## GetObject Windows.Pas

### Sintaxis

```

GetObject(
  p1: HGDIOBJ;    {manejador de objeto gráfico}
  p2: Integer;     {tamaño del buffer al que apunta el parámetro p3}
  p3: Pointer      {puntero al buffer que recibe información sobre el objeto}
): Integer;        {devuelve el número de bytes escritos al buffer}

```

### Descripción

Esta función recupera información sobre el objeto gráfico identificado en el parámetro *p1*. Dependiendo del tipo de objeto, el parámetro *p3* debe apuntar a un *buffer* que recibe un registro de tipo *TBitmap*, *TDIBSection*, *TExtLogPen*, *TLogBrush*, *TLogFont* o *TLogPen* que contiene información sobre el objeto especificado. Nota: Si el parámetro *p1* contiene un manejador de un mapa de bits creado con una función distinta que *CreateDIBSection*, el registro de datos devuelto en el *buffer* contiene sólo el ancho, la altura y el formato de colores del mapa de bits.

### Parámetros

*p1*: Especifica un manejador del objeto gráfico cuya información se desea recuperar. Puede ser un manejador de un mapa de bits, una sección DIB, una brocha, una fuente, una pluma, o una paleta.

*p2*: Especifica el tamaño del *buffer* al que apunta el parámetro *p3*.

*p3*: Un puntero a un *buffer* que recibirá un registro de datos que contiene información acerca del objeto gráfico especificado. El tipo de registro recibido depende del tipo del objeto especificado en el parámetro *p1*. Si a este parámetro se le asigna **nil**, la función devuelve el tamaño requerido por el *buffer* para almacenar la información recuperada. Si el parámetro *p1* contiene un manejador de una paleta, el *buffer* al que apunta este parámetro recibirá un valor de 16 bits que indica el número de entradas en la paleta. Si *p1* contiene un manejador de un mapa de bits, una pluma, una brocha o una fuente, el *buffer* al que apunta este parámetro recibirá un registro de datos *TBitmap*, *TLogPen*, *TLogBrush*, o *TLogFont*, respectivamente. Consulte las funciones *CreateBitmapIndirect*, *CreatePenIndirect*, *CreateBrushIndirect*, o *CreateFontIndirect* para ver las descripciones de estos registros. Si el parámetro *p1* contiene un manejador de un mapa de bits devuelto por la función *CreateDIBSection*, el *buffer* al que apunta este parámetro recibirá un registro de tipo *TDIBSection*. Si el parámetro *p1* contiene un manejador de una pluma devuelto por la función *ExtCreatePen*, el *buffer* al que apunta este parámetro recibirá un registro de tipo *TExtLogPen*.

El registro *TDIBSection* se define de la siguiente forma:

*TDIBSection* = **packed record**

<i>dsBm</i> : <i>TBitmap</i> ;	{registro <i>TBitmap</i> }
<i>dsBmih</i> : <i>TBitmapInfoHeader</i> ;	{registro <i>TBitmapInfoHeader</i> }
<i>dsBitfields</i> : <b>array</b> [0..2] <b>of</b> <i>DWORD</i> ;	{máscaras de colores}
<i>dshSection</i> : <i>THandle</i> ;	{manejador de objeto de mapeado de fichero}
<i>dsOffset</i> : <i>DWORD</i> ;	{desplazamiento de valores de bits}

**end;**

*dsBm*: Especifica un registro *TBitmap* que contiene información acerca del tipo de mapa de bits, sus dimensiones y un puntero a los bits que lo componen. Consulte *CreateBitmapIndirect* para ver una descripción de este registro.

*dsBmih*: Especifica un registro *TBitmapInfoHeader* que contiene información acerca del formato de colores del mapa de bits. Consulte la función *CreateDIBSection* para ver una descripción de este registro.

*dsBitFields*: Un *array* que contiene las máscaras de los tres colores, si el mapa de bits tiene una profundidad de color mayor que ocho bits por píxel.

*dshSection*: Especifica un manejador del objeto de mapeado de fichero pasado a *CreateDIBSection* cuando el mapa de bits fue creado. Si el mapa de bits no fue creado a partir de un objeto de mapeado de fichero, este campo contendrá cero.

*dsOffset*: Especifica el desplazamiento dentro del objeto de mapeado de fichero para el inicio de los bits del mapa de bits. Si no se utilizó un objeto de mapeado para crear el mapa de bits este campo contendrá cero.

El registro *TextLogPen* se define de la siguiente forma:

**TextLogPen = packed record**

<i>elpPenStyle</i> : DWORD;	{opción de tipo, estilo, terminación y unión}
<i>elpWidth</i> : DWORD;	{ancho de la pluma}
<i>elpBrushStyle</i> : UINT;	{estilo de la brocha}
<i>elpColor</i> : COLORREF;	{color de la pluma}
<i>elpHatch</i> : Longint;	{estilo de la trama}
<i>elpNumEntries</i> : DWORD;	{número de entradas en el <i>array</i> }
<i>elpStyleEntry</i> : <b>array</b> [0..0] <b>of</b> DWORD;	{estilo definido por el usuario}

**end;**

*elpPenStyle*: Una combinación de opciones que definen el tipo, estilo, terminación y uniones de la pluma. Consulte la función *ExtCreatePen* para ver una lista de las opciones disponibles.

*elpWidth*: Ancho de la pluma en unidades lógicas. Si el parámetro *elpPenStyle* contiene la opción *PS\_COSMETIC*, a este parámetro se le debe asignar 1.

*elpBrushStyle*: Una opción que indica el estilo de la brocha de la pluma. Consulte la función *CreateBrushIndirect* para ver una lista de las opciones disponibles.

*elpColor*: Especifica el color de la pluma.

*elpHatch*: Especifica el patrón de trama de la pluma. Consulte la función *CreateHatchBrush* para ver una lista de las opciones disponibles.

*elpNumEntries*: Especifica la cantidad de entradas en el *array* de estilo de la pluma definido por el usuario a la que apunta el campo *elpStyleEntry*. Si *elpPenStyle* no contiene la opción *PS\_USERSTYLE*, este campo es ignorado.

*elpStyleEntry*: Puntero al *array* de valores DWORD que define el patrón de guiones y espacios para un estilo de pluma definido por el usuario. La primera entrada en el *array* especifica la longitud del primer guión, en unidades lógicas; la segunda entrada especifica la longitud del primer espacio, en unidades lógicas; y así sucesivamente, hasta que la línea esté definida. Este patrón se repetirá tantas veces como sea necesario cuando se dibuje una línea con la pluma. Si *elpPenStyle* no contiene la opción *PS\_USERSTYLE*, este campo es ignorado.

**Valor que devuelve**

Si la función tiene éxito, devuelve el número de bytes escritos en el *buffer* al que apunta el parámetro *p3*; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateBitmapIndirect, CreateBrushIndirect, CreateDIBSection, CreateFontIndirect, CreatePenIndirect, ExtCreatePen, GetBitmapBits, GetDIBits, GetCurrentObject, GetObjectType, GetPaletteEntries, GetRegionData, GetStockObject*

**Ejemplo****Listado 3-24: Recuperando información acerca de un objeto**

```
function GetStyle(Style: Integer): string;
begin
    {Mostrar el estilo de la brocha}
    case Style of
        BS_DIBPATTERN:    Result := 'BS_DIBPATTERN';
        BS_DIBPATTERN8X8: Result := 'BS_DIBPATTERN8X8';
        BS_DIBPATTERNPT:  Result := 'BS_DIBPATTERNPT';
        BS_HATCHED:       Result := 'BS_HATCHED';
        BS_HOLLOW:         Result := 'BS_HOLLOW';
        BS_PATTERN:        Result := 'BS_PATTERN';
        BS_PATTERN8X8:     Result := 'BS_PATTERN8X8';
        BS_SOLID:          Result := 'BS_SOLID';
    end;
end;

function GetHatch(Hatch: Integer): string;
begin
    {Mostrar el estilo de trama}
    case Hatch of
        HS_BDIAGONAL:    Result := 'HS_BDIAGONAL';
        HS_CROSS:         Result := 'HS_CROSS';
        HS_DIAGCROSS:     Result := 'HS_DIAGCROSS';
        HS_FDIAGONAL:     Result := 'HS_FDIAGONAL';
        HS_HORIZONTAL:    Result := 'HS_HORIZONTAL';
        HS_VERTICAL:      Result := 'HS_VERTICAL';
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    hObject: HGDIOBJ;    // manejador de la brocha
    LogBrush: TLogBrush; // almacena información de la brocha
    FormDC: HDC;         // manejador del contexto de dispositivo del formulario
begin
    {Recuperar el contexto de dispositivo del formulario}
    FormDC := GetDC(Form1.Handle);

    {Inicializar la brocha del formulario y recuperar su manejador}
```

```

Canvas.Brush.Color := clRed;
Canvas.Brush.Style := bsDiagCross;
hObject := GetCurrentObject(Canvas.Handle, OBJ_BRUSH);

{Recuperar información acerca del objeto}
GetObject(hObject, SizeOf(TLogBrush), @LogBrush);

{Indicar el tipo de objeto recuperado}
case GetObjectType(hObject) of
  OBJ_BITMAP:      Edit4.Text := 'Bitmap';
  OBJ_BRUSH:       Edit4.Text := 'Brush';
  OBJ_FONT:        Edit4.Text := 'Font';
  OBJ_PAL:         Edit4.Text := 'Palette';
  OBJ_PEN:         Edit4.Text := 'Pen';
  OBJ_EXTPEN:      Edit4.Text := 'Extended Pen';
  OBJ_REGION:      Edit4.Text := 'Region';
  OBJ_DC:          Edit4.Text := 'Device Context';
  OBJ_MEMDC:       Edit4.Text := 'Memory Device Context';
  OBJ_METAFILE:    Edit4.Text := 'Metafile';
  OBJ_METADC:      Edit4.Text := 'Metafile Device Context';
  OBJ_ENHMETAFILE: Edit4.Text := 'Enhanced Metafile';
  OBJ_ENHMETADC:   Edit4.Text := 'Enhanced Metafile Device Context';
end;

{Mostrar la información del objeto}
with LogBrush do begin
  Edit1.Text := GetStyle(lbStyle);
  Edit2.Text := IntToHex(lbColor, 8);
  Edit3.Text := GetHatch(lbHatch);
end;

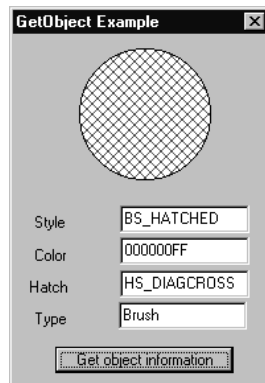
{Seleccionar la brocha dentro del contexto de dispositivo del formulario}
SelectObject(FormDC, hObject);

{Dibujar una elipse con la brocha}
Ellipse(FormDC, 50, 10, 150, 110);

{Borrar el contexto de dispositivo}
ReleaseDC(Form1.Handle, FormDC);
end;

```

Figura 3-29:  
La información  
del objeto



**GetObjectType****Windows.Pas****Sintaxis**

```
GetObjectType(
    h: HGDIOBJ           {manejador del objeto gráfico}
): DWORD;               {devuelve un indicador del tipo de objeto}
```

**Descripción**

Esta función devuelve un indicador que muestra el tipo de objeto al que hace referencia el parámetro *h*.

**Parámetros**

*h*: Manejador de un objeto gráfico cuyo tipo será recuperado.

**Valor que devuelve**

Si la función tiene éxito, devuelve una opción que indica el tipo del objeto, que puede ser un valor de la Tabla 3-28. Si falla, devuelve cero.

**Véase además**

*DeleteObject*, *GetCurrentObject*, *GetObject*, *GetStockObject*, *SelectObject*

**Ejemplo**

Vea el Listado 3-24 bajo *GetObject*.

**Tabla 3-28: Valores que devuelve GetObjectType**

Valor	Descripción
OBJ_BITMAP	Mapa de bits.
OBJ_BRUSH	Brocha.
OBJ_FONT	Fuente.
OBJ_PAL	Paleta.
OBJ_PEN	Pluma.
OBJ_EXTPEN	Pluma extendida.
OBJ_REGION	Región.
OBJ_DC	Contexto de dispositivo.
OBJ_MEMDC	Contexto de dispositivo en memoria.
OBJ_METAFILE	Metafichero.
OBJ_METADC	Contexto de dispositivo de metafichero.
OBJ_ENHMETAFILE	Metafichero mejorado.
OBJ_ENHMETADC	Contexto de dispositivo de metafichero mejorado.

**GetPixel****Windows.Pas****Sintaxis**

```
GetPixel(
    DC: HDC;           {manejador de contexto de dispositivo}
    X: Integer;         {coordenada horizontal del píxel}
    Y: Integer         {coordenada vertical del píxel}
): COLORREF;          {devuelve un especificador de color}
```

**Descripción**

Esta función recupera el color del píxel situado en las coordenadas especificadas en el contexto de dispositivo indicado. La coordenada tiene que estar dentro de las fronteras de la región de recorte actual.

**Parámetros**

*DC*: Manejador del contexto de dispositivo al que pertenece el píxel cuyo color se desea recuperar.

*X*: La coordenada horizontal del píxel dentro del contexto de dispositivo, en unidades lógicas.

*Y*: La coordenada vertical del píxel dentro del contexto de dispositivo, en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve el especificador de color del píxel en las coordenadas indicadas. Si falla, devuelve *CLR\_INVALID*.

**Véase además**

*SetPixel*, *SetPixelV*

**Ejemplo**

Vea el Listado 3-44 bajo *SetPixel*.

**GetPolyFillMode****Windows.Pas****Sintaxis**

```
GetPolyFillMode(
    DC: HDC           {manejador de contexto de dispositivo}
): Integer;          {devuelve el modo de relleno de polígonos}
```

**Descripción**

Esta función recupera el modo actual del relleno de polígonos, para el contexto de dispositivo especificado.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo modo actual de relleno de polígonos se desea recuperar.

**Valor que devuelve**

Si la función tiene éxito, devuelve un indicador que muestra el modo actual de relleno de polígonos del contexto de dispositivo, que puede ser un valor de la Tabla 3-29. Si falla, devuelve cero. Consulte la función *SetPolyFillMode* para ver una descripción de estas opciones.

**Véase además**

*FillPath, Polygon, PolyPolygon, SetPolyFillMode*

**Ejemplo****Listado 3-25: Seleccionando y recuperando el modo de relleno del polígono**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    PointsArray: Array[0..10] of TPoint; // almacena la definición del polígono
    FillMode: Integer; // almacena el modo de relleno
begin
    {Definir el polígono}
    PointsArray[0].X := 145; PointsArray[0].Y := 220;
    PointsArray[1].X := 145; PointsArray[1].Y := 20;
    PointsArray[2].X := 310; PointsArray[2].Y := 20;
    PointsArray[3].X := 310; PointsArray[3].Y := 135;
    PointsArray[4].X := 105; PointsArray[4].Y := 135;
    PointsArray[5].X := 105; PointsArray[5].Y := 105;
    PointsArray[6].X := 280; PointsArray[6].Y := 105;
    PointsArray[7].X := 280; PointsArray[7].Y := 50;
    PointsArray[8].X := 175; PointsArray[8].Y := 50;
    PointsArray[9].X := 175; PointsArray[9].Y := 220;

    {Asignar al modo de relleno de polígonos el valor seleccionado}
    if RadioGroup1.ItemIndex = 0 then
        SetPolyFillMode(Canvas.Handle, ALTERNATE)
    else
        SetPolyFillMode(Canvas.Handle, WINDING);

    {Mostrar el modo de relleno de polígonos del contexto de dispositivo}
    FillMode := GetPolyFillMode(Canvas.Handle);
    if FillMode = ALTERNATE then
        Caption := 'GetPolyFillMode Example - Alternate'
    else
        Caption := 'GetPolyFillMode Example - Winding';

    {Asignar rojo a la brocha y dibujar un polígono relleno}
    Canvas.Brush.Color := clRed;
    Polygon(Canvas.Handle, PointsArray, 10);
end;

```

Figura 3-30:  
Un modo  
específico de  
relleno de  
polígono

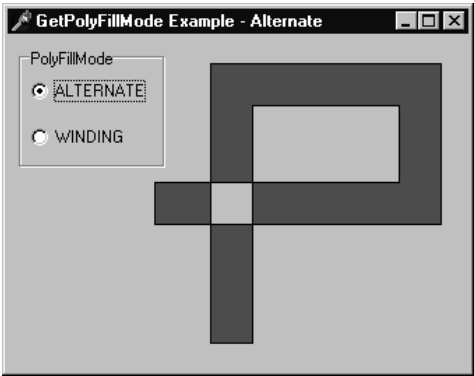


Tabla 3-29: Valores que devuelve GetPolyFillMode

Valor	Descripción
ALTERNATE	Rellena el polígono usando el método Alternate.
WINDING	Rellena el polígono usando el método Winding.

**GetROP2**                      **Windows.Pas**

*Sintaxis*

```
GetROP2(  
    DC: HDC                      {manejador de contexto de dispositivo}  
); Integer;                      {devuelve el modo de mezcla del primer plano}
```

*Descripción*

Esta función recupera el modo de mezcla del primer plano para el contexto de dispositivo especificado. El modo de mezcla del primer plano determina cómo el color de la pluma utilizada en las operaciones de dibujo se combina con el color anterior de los píxeles en el contexto de dispositivo especificado.

*Parámetros*

*DC*: Manejador del contexto de dispositivo cuyo modo de mezcla del primer plano se desea recuperar.

*Valor que devuelve*

Si la función tiene éxito, devuelve un indicador que muestra el modo de mezcla del primer plano del contexto de dispositivo, que puede ser una opción de la Tabla 3-30. Si falla, devuelve cero.

*Véase además*

*LineTo, PolyBezier, Polyline, Rectangle, SetROP2*

*Ejemplo***Listado 3-26: Usando el modo de mezcla para dibujar un rectángulo arrastrable**

```

var
  Form1: TForm1;
  RectDragging: Boolean;    // indica si una operación de arrastre ha comenzado
  OldRect: TRect;          // coordenadas anteriores del rectángulo

implementation

{$R *.DFM}

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {Indicar que una operación de arrastre ha comenzado}
  RectDragging := TRUE;
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  {Si estamos arrastrando un rectángulo...}
  if RectDragging then begin
    {Inicializar la pluma del área de dibujo}
    Canvas.Pen.Width := 5;

    {Si el modo de mezcla del primer plano no es R2_NOT, asignarlo}
    if GetROP2(Canvas.Handle) <> R2_NOT then SetROP2(Canvas.Handle, R2_NOT);

    {Seleccionar la brocha para que borre solo las líneas mostradas}
    Canvas.Brush.Style := bsClear;

    {Dibujar un rectángulo sobre el anterior para borrarlo}
    Canvas.Rectangle(OldRect.Left, OldRect.Top, OldRect.Right, OldRect.Bottom);

    {Dibujar un rectángulo en la nueva posición}
    Canvas.Rectangle(X-20, Y-20, X+20, Y+20);

    {Almacenar las coordenadas del rectángulo actuales para la próxima vez}
    OldRect := Rect(X-20, Y-20, X+20, Y+20);
  end;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  RectDragging := FALSE;  {El arrastre ha finalizado}
end;

```

Figura 3-31:  
Dibujando el  
rectángulo  
arrastrable



Tabla 3-30: Valores que devuelve GetROP2

Valor	Descripción
R2_BLACK	El píxel de destino siempre será negro.
R2_COPYPEN	Al píxel de destino se le asigna el color de la pluma.
R2_MASKNOTPEN	El píxel de destino es una combinación de los colores comunes a la pantalla y al inverso de la pluma.
R2_MASKPEN	El píxel de destino es una combinación de los colores comunes a la pantalla y a la pluma.
R2_MASKPENNOT	El píxel de destino es una combinación de los colores comunes a la pluma y al inverso de la pantalla.
R2_MERGEOTPEN	El píxel de destino es una combinación de la pantalla y el inverso de la pluma.
R2_MERGEPEN	El píxel de destino es una combinación de la pluma y la pantalla
R2_MERGEPENNOT	El píxel de destino es una combinación de la pluma y el inverso de la pantalla.
R2_NOP	El píxel de destino no será modificado.
R2_NOT	El píxel de destino es el inverso de la pantalla.
R2_NOTCOPYPEN	El píxel de destino es el inverso de la pluma.
R2_NOTMASKPEN	El píxel de destino es el inverso de la opción R2_MASKPEN.
R2_NOTMERGEPEN	El píxel de destino es el inverso de la opción R2_MERGEPEN.
R2_NOTXORPEN	El píxel de destino es el inverso de la opción R2_XORPEN.
R2_WHITE	El píxel de destino siempre será blanco.
R2_XORPEN	El píxel de destino es una combinación de los colores en la pluma o en la pantalla, pero no ambos.

### GetStockObject

### Windows.Pas

#### Sintaxis

```
GetStockObject(
    Index: Integer
): HGDIOBJ;
```

{opción del tipo de objeto del almacén del sistema}  
{devuelve un manejador del objeto gráfico}

**Descripción**

Esta función recupera un manejador de una pluma, brocha, fuente, o paleta predefinida. NO es necesario eliminar estos objetos cuando la aplicación deja de necesitarlos.



**Nota:** Utilice las brochas de almacén *DKGRAY\_BRUSH*, *GRAY\_BRUSH* y *LTGRAY\_BRUSH* sólo en ventanas con los estilos de clase *CS\_HREDRAW* y *CS\_VREDRAW*; en caso contrario, podrían ocurrir desajustes de los patrones de brochas, si la ventana es movida o redimensionada. El origen de las brochas del almacén no puede ser modificado.

**Parámetros**

*Index*: Opción que indica el tipo de objeto del almacén que se desea recuperar. A este parámetro se le puede asignar un valor de la Tabla 3-31.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del objeto gráfico predefinido; en caso contrario, devuelve cero.

**Véase además**

*DeleteObject*, *GetObject*, *GetObjectType*, *SelectObject*

**Ejemplo****Listado 3-27: Usando un objeto del almacén**

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ARegion: HRGN;    // almacena una región
begin
  {Crear una región que será rellenada}
  ARegion := CreateRectRgn(20, 20, 190, 110);

  {Rellenar la región con una brocha del almacén}
  FillRgn(Canvas.Handle, ARegion, GetStockObject(BLACK_BRUSH));
end;
```

**Tabla 3-31: Valores del campo Index de GetStockObject**

Valor	Descripción
BLACK_BRUSH	Recupera un manejador de una brocha negra.
DKGRAY_BRUSH	Recupera un manejador de una brocha gris oscuro.
GRAY_BRUSH	Recupera un manejador de una brocha gris.
HOLLOW_BRUSH	Recupera un manejador de una brocha “hueca”.
LTGRAY_BRUSH	Recupera un manejador de una brocha gris claro.

Valor	Descripción
WHITE_BRUSH	Recupera un manejador de una brocha blanca.
BLACK_PEN	Recupera un manejador de una pluma negra.
NULL_PEN	Recupera un manejador de una pluma nula.
WHITE_PEN	Recupera un manejador de una pluma blanca.
ANSI_FIXED_FONT	Recupera un manejador de una fuente del sistema de tamaño fijo ( <i>fixed-pitch</i> ).
ANSI_VAR_FONT	Recupera un manejador de una fuente del sistema proporcional ( <i>variable-pitch</i> ).
DEVICE_DEFAULT_FONT	Sólo Windows NT: Recupera un manejador de una fuente dependiente del dispositivo.
DEFAULT_GUI_FONT	Sólo Windows 95/98: Recupera un manejador de la fuente por defecto utilizada en los objetos de interfaz de usuario.
OEM_FIXED_FONT	Recupera un manejador de una fuente <i>fixed-pitch</i> del fabricante.
SYSTEM_FONT	Recupera un manejador de la fuente del sistema.
SYSTEM_FIXED_FONT	Recupera un manejador de la fuente de sistema <i>fixed-pitch</i> utilizada en versiones de Windows anteriores a 3.0.
DEFAULT_PALETTE	Recupera un manejador de la paleta por defecto que contiene los colores estáticos en la paleta del sistema.

### GetUpdateRect Windows.Pas

#### Sintaxis

```
GetUpdateRect(
    hWnd: HWND;           {manejador de ventana}
    var lpRect: TRect;     {puntero a registro TRect}
    bErase: BOOL           {opción de borrado del fondo}
): BOOL;                 {devuelve TRUE o FALSE}
```

#### Descripción

Esta función recupera las coordenadas del rectángulo más pequeño que puede ser dibujado alrededor de la región no válida (la región de actualización) de la ventana especificada. El rectángulo estará expresado en coordenadas de área cliente, a menos que la ventana haya sido creada con el estilo de clase *CS\_OWNDC* y su modo de mapeado sea distinto de *MM\_TEXT*. En este caso, el rectángulo estará expresado en coordenadas lógicas. Observe que esta función tiene que ser usada antes de que la función *BeginPaint* sea llamada, ya que *BeginPaint* valida la región a actualizar, lo que provocaría que esta función devolviese un rectángulo vacío.

**Parámetros**

*hWnd*: Manejador de la ventana cuyo rectángulo límite de la región de actualización será recuperado.

*lpRect*: Puntero a un registro *TRect* que recibirá las coordenadas de la región de actualización.

*bErase*: Una opción que indica si el fondo de la región no válida debe ser borrado. Si a este parámetro se le asigna *TRUE* y la región no está vacía, se enviará un mensaje *WM\_ERASEBKGD* a la ventana especificada.

**Valor que devuelve**

Si la función tiene éxito, devuelve *TRUE*; en caso contrario, devuelve *FALSE*.

**Véase además**

*BeginPaint*, *GetUpdateRgn*, *InvalidateRect*

**Ejemplo**

Vea el Listado 3-29 bajo *InvalidateRect*.

**GetUpdateRgn Windows.Pas****Sintaxis**

```
GetUpdateRgn(
    hWnd: HWND;           {manejador de ventana}
    hRgn: HRGN;           {manejador de región}
    bErase: BOOL           {opción de borrado del fondo}
): Integer;              {devuelve el tipo de región no válida}
```

**Descripción**

Esta función recupera el manejador de la región no válida de la ventana especificada. La región es relativa al área cliente de la ventana. Observe que esta función tiene que ser usada antes de que la función *BeginPaint* sea llamada, ya que *BeginPaint* valida la región a actualizar, lo que provocaría que esta función devolviera una región vacía.

**Parámetros**

*hWnd*: Manejador de la ventana cuya región a actualizar será recuperada.

*hRgn*: Un manejador de la región existente. Este manejador será inicializado para que apunte a la región no válida cuando la función retorne.

*bErase*: Una opción que indica si el fondo de la región no válida debe ser borrado y las áreas no cliente de las ventanas hijas redibujadas. Si a este parámetro se le asigna *TRUE* y la región no está vacía, se enviará un mensaje *WM\_ERASEBKGD* a la ventana especificada y las áreas no cliente de las ventanas hijas serán redibujadas.

**Valor que devuelve**

Esta función devuelve un resultado que indica el tipo de región recuperada o una condición de error y puede tomar un valor de la Tabla 3-32.

**Véase además**

*GetUpdateRect, InvalidateRgn*

**Ejemplo**

Vea el Listado 3-30 bajo *InvalidateRgn*.

**Tabla 3-32: Valores que devuelve GetUpdateRgn**

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región con más de un rectángulo.
ERROR	Indica que ha ocurrido un error.

**GrayString****Windows.Pas****Sintaxis**

```

GrayString(
    hDC: HDC;                {manejador de contexto de dispositivo}
    hBrush: HBRUSH;          {manejador de brocha}
    lpOutputFunc: TFNGrayStringProc; {puntero a función de respuesta}
    lpData: LPARAM;          {puntero a cadena}
    nCount: Integer;         {longitud de la cadena}
    X: Integer;              {coordenada de salida horizontal}
    Y: Integer;              {coordenada de salida vertical}
    nWidth: Integer;         {ancho del mapa de bits fuera de pantalla}
    nHeight: Integer;        {altura del mapa de bits fuera de pantalla}
): BOOL;                   {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja texto en la localización especificada del contexto de dispositivo. El texto es dibujado creando un mapa de bits fuera de pantalla, dibujando en el mapa de bits, modificando su color con la brocha especificada o la brocha por defecto y finalmente, copiando el mapa de bits en el área de dibujo especificada en las coordenadas indicadas. Para dibujar el texto se utiliza la fuente actualmente seleccionada en el contexto de dispositivo. Si al parámetro *lpOutputFunc* se le asigna **nil**, el parámetro *lpData* tiene que contener un puntero a una cadena y la función *TextOut* es utilizada para dibujar la cadena sobre el mapa de bits fuera de pantalla. En caso contrario, el parámetro *lpData* puede apuntar a cualquier tipo de datos definido

por el usuario, por ejemplo un mapa de bits, y la función de respuesta a la que apunta el parámetro *lpOutputFunc* tiene que dibujar los datos en el mapa de bits fuera de pantalla.

#### Parámetros:

*hDC*: Manejador del contexto de dispositivo sobre el cual la cadena es dibujada.

*hBrush*: Manejador de la brocha a utilizar para modificar el color del texto. Si a este parámetro se le asigna cero, esta función utilizará la brocha por defecto para dibujar el texto.

*lpOutputFunc*: Puntero a una función de respuesta que manejará la salida de texto. Si a este parámetro se le asigna **nil**, la función utilizará *TextOut* para dibujar el texto en el mapa de bits fuera de pantalla.

*lpData*: Si el parámetro *lpOutputFunc* contiene **nil**, *lpData* debe contener un puntero a la cadena que será dibujada. Esta debe ser una cadena terminada en nulo si al parámetro *nCount* se le asigna cero. En caso contrario, este parámetro contiene un puntero a datos que serán pasados a la función de respuesta.

*nCount*: Especifica la longitud, en caracteres, de la cadena a la que apunta el parámetro *lpData*. Si a este parámetro se le asigna cero, la función calculará la longitud de la cadena, que deberá terminar en carácter nulo. Si a este parámetro se le asigna -1 y la función de respuesta a la que apunta el parámetro *lpOutputFunc* devuelve FALSE, la cadena será mostrada en su forma original.

*X*: Especifica la coordenada horizontal de la posición en la cual se mostrará la cadena, en unidades del dispositivo.

*Y*: Especifica la coordenada vertical de la posición en la cual se mostrará la cadena, en unidades del dispositivo.

*nWidth*: Especifica el ancho, en unidades de dispositivo, del mapa de bits fuera de pantalla en el cual se dibujará la cadena. Si a este parámetro se le asigna cero, la función calculará el ancho del mapa de bits fuera de pantalla si el parámetro *lpData* contiene un puntero a una cadena.

*nHeight*: Especifica la altura, en unidades de dispositivo, del mapa de bits fuera de pantalla en el cual se dibujará la cadena. Si a este parámetro se le asigna cero, la función calculará la altura del mapa de bits fuera de pantalla si el parámetro *lpData* contiene un puntero a una cadena.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Sintaxis de la función de respuesta

```
GrayStringOutputProc(
    hdc: HDC;                {manejador del contexto de dispositivo
                             del mapa de bits fuera de pantalla}
```

<code>lpData: LPARAM;</code>	<code>{puntero a los datos que serán dibujados}</code>
<code>cchData: Integer</code>	<code>{longitud de la cadena}</code>
<code>): BOOL;</code>	<code>{devuelve TRUE o FALSE}</code>

**Descripción**

Esta función de respuesta es utilizada para dibujar la cadena especificada o los datos definidos por el usuario, de la manera específica definida por la aplicación. El parámetro *hDC* especifica un manejador de un contexto de dispositivo que representa al mapa de bits fuera de pantalla creado por la función *GrayString*. La función de respuesta tiene que dibujar los datos de la manera deseada en este contexto de dispositivo, los cuales serán copiados, a su vez, en el contexto de dispositivo especificado por el parámetro *hDC* de la función *GrayString* cuando la función de respuesta retorne. Esta función de respuesta puede ejecutar cualquier acción deseada.

**Parámetros**

*hdc*: Manejador del contexto de dispositivo del mapa de bits fuera de pantalla sobre el cual serán dibujados los datos o la cadena. El contexto de dispositivo tendrá el mismo ancho y altura que se especifica en los parámetros *nWidth* y *nHeight* de la función *GrayString*.

*lpData*: Un puntero a los datos que serán dibujados; corresponde al parámetro *lpData* de la función *GrayString*.

*cchData*: Especifica la longitud en caracteres de la cadena; corresponde al parámetro *nCount* de la función *GrayString*.

**Valor que devuelve**

La función de repuesta debe devolver TRUE para indicar que ha tenido éxito; en caso contrario, devuelve FALSE.

**Véase además**

*DrawText*, *GetSysColor*, *SetTextColor*, *TabbedTextOut*, *TextOut*

**Ejemplo****Listado 3-28: Dibujando texto con GrayString**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    Str: PChar;           // apunta a la cadena que será dibujada
begin
    {Inicializar el puntero a la cadena}
    Str := 'Delphi Rocks!';

    {Inicializar la brocha usada para dibujar la cadena}
    Canvas.Brush.Color := clRed;

    {Dibujar la cadena}
    GrayString(Canvas.Handle, Canvas.Brush.Handle, nil, LPARAM(Str), Length(Str),

```

```

10, 10, 0, 0);
end;

```

Figura 3-32:  
Uso de  
GrayString



## **InvalidateRect**      **Windows.Pas**

### **Sintaxis**

```

InvalidateRect(
  hWnd: HWND;           {manejador de ventana}
  lpRect: PRect;        {puntero a las coordenadas del rectángulo}
  bErase: BOOL          {opción de borrado del fondo}
): BOOL;                {devuelve TRUE o FALSE}

```

### **Descripción**

Esta función añade el rectángulo especificado a la región no válida de la ventana indicada, haciendo que ésta reciba un mensaje *WM\_PAINT*.

### **Parámetros**

*hWnd*: Manejador de la ventana que contiene la región no válida a la que se añadirá el rectángulo especificado. Si a este parámetro se le asigna cero, todas las ventanas son invalidadas y recibirán los mensajes *WM\_ERASEBKGD* y *WM\_NCPAINT* antes de que la función retorne.

*lpRect*: Puntero al registro *TRect* que contiene las coordenadas del rectángulo que será añadido a la región no válida. Si a este parámetro se le asigna **nil**, el área cliente completa será añadida a la región no válida.

*bErase*: Una opción que indica si el fondo en la región no válida debe ser borrado. Si a este parámetro se le asigna TRUE y la región no está vacía, el fondo completo de la región no válida será borrado cuando la función *BeginPaint* sea llamada.

### **Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### **Véase además**

*BeginPaint*, *GetUpdateRect*, *InvalidateRgn*

### **Ejemplo**

#### **Listado 3-29: Dibujando solo el rectángulo no válido del área de dibujo**

```

procedure TForm1.Button2Click(Sender: TObject);

```

```

var
  InvalidRectangle: TRect; //rectángulo a invalidar
begin
  {Definir el rectángulo}
  InvalidRectangle := Rect(10, 10, 110, 110);

  {Borrar solo el área rectangular}
  InvalidateRect(Form1.Handle, @InvalidRectangle, TRUE);
end;

procedure TForm1.WMPaint(var Msg: TWMPaint);
var
  InvalidRect: TRect;          // almacena el área rectangular no válida
  PaintStruct: TPaintStruct;   // almacena información de redibujado
begin
  {Recuperar el rectángulo no válido}
  GetUpdateRect(Handle, InvalidRect, TRUE);

  {Comenzar el proceso de repintado; esto valida la región no válida}
  BeginPaint(Handle, PaintStruct);

  {Si el área cliente completa es no válida...}
  if EqualRect(InvalidRect, ClientRect) then
    {...redibujar el mapa de bits que está en Image1}
    Canvas.Draw(0, 0, Image1.Picture.Bitmap)
  else
    begin
      {...en caso contrario, dibujar un rectángulo rojo en el área del rectángulo
        no válido y lo etiqueta como un área previamente inválida}
      Canvas.Brush.Color := clRed;
      Canvas.Rectangle(InvalidRect.Left, InvalidRect.Top, InvalidRect.Right,
        InvalidRect.Bottom);
      Canvas.TextOut(InvalidRect.Left+10, InvalidRect.Top + 10, 'Invalid Rect');
    end;

  {Finalizar la operación de repintado}
  EndPaint(Handle, PaintStruct);
end;

```



Figura 3-33:  
El rectángulo  
no válido

**InvalidateRgn**      **Windows.Pas****Sintaxis**

```

InvalidateRgn(
  hWnd: HWND;           {manejador de ventana}
  hRgn: HRGN;           {manejador de región}
  bErase: BOOL          {opción de borrado del fondo}
): BOOL;               {siempre devuelve TRUE}

```

**Descripción**

Esta función añade la región dada a la región no válida de la ventana especificada, haciendo que ésta reciba un mensaje *WM\_PAINT*.

**Parámetros**

*hWnd*: Manejador de la ventana que contiene la región no válida a la cual se añadirá la región especificada.

*hRgn*: Manejador de la región que será añadida a la región no válida de la ventana. Se asume que la región está expresada en coordenadas cliente. Si a este parámetro se le asigna cero, el área cliente completa es añadida a la región no válida.

*bErase*: Opción que indica si el fondo en la región no válida debe ser borrado. Si a este parámetro se le asigna TRUE y la región no está vacía, el fondo de la región no válida completa será borrado cuando la función *BeginPaint* sea llamada.

**Valor que devuelve**

Esta función siempre devuelve TRUE.

**Véase además**

*BeginPaint*, *GetUpdateRgn*, *InvalidateRect*

**Ejemplo****Listado 3-30: Dibujando solo la región no válida del área de dibujo**

```

procedure TForm1.Button2Click(Sender: TObject);
var
  PointsArray: array[0..2] of TPoint; // array de puntos que define la región
  RegionHandle: HRGN;                // un manejador de la región
begin
  {Definir la región}
  PointsArray[0].x := 20;  PointsArray[0].y := 20;
  PointsArray[1].x := 100; PointsArray[1].y := 65;
  PointsArray[2].x := 20;  PointsArray[2].y := 120;

  {Crear la región}
  RegionHandle := CreatePolygonRgn(PointsArray, 3, ALTERNATE);

```

```

    {Invalidar la región}
    InvalidateRgn(Form1.Handle, RegionHandle, TRUE);

    {La región ya no es necesaria; la eliminamos}
    DeleteObject(RegionHandle);
end;

procedure TForm1.WMPaint(var Msg: TWMPaint);
var
    InvalidRgn: HRGN;           // un manejador de la región no válida
    PaintStruct: TPaintStruct;  // almacena información sobre el redibujado
begin
    {GetUpdateRgn requiere un manejador de una región preexistente, por lo que se
    crea una}
    InvalidRgn := CreateRectRgn(0, 0, 1, 1);

    {Recuperar un manejador de la región a actualizar}
    GetUpdateRgn(Handle, InvalidRgn, FALSE);

    {Comienza la operación de redibujado}
    BeginPaint(Handle, PaintStruct);

    {Si la región es igual al área cliente completa...}
    if EqualRgn(InvalidRgn, CreateRectRgnIndirect(ClientRect)) then
        {...dibujar el mapa de bits Image1 en el área de dibujo del formulario}
        Canvas.Draw(0, 0, Image1.Picture.Bitmap)
    else
        begin
            {...en caso contrario, rellenar la región no válida en rojo}
            Canvas.Brush.Color := clRed;
            FillRgn(Canvas.Handle, InvalidRgn, Canvas.Brush.Handle);
        end;

    {Finalizar la operación de repintado}
    EndPaint(Handle, PaintStruct);

    {Borrar el objeto región, ya que no es necesario}
    DeleteObject(InvalidRgn);
end;

```



Figura 3-34:  
La región no  
válida

**LineDDA****Windows.Pas****Sintaxis**

```

LineDDA(
  p1: Integer;           {coordenada horizontal de punto de inicio}
  p2: Integer;           {coordenada vertical del punto de inicio}
  p3: Integer;           {coordenada horizontal del punto de terminación}
  p4: Integer;           {coordenada vertical del punto de terminación}
  p5: TFNLineDDAProc;    {un puntero a la función de respuesta}
  p6: LPARAM             {datos definidos por la aplicación}
): BOOL;                 {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja una línea pasando las coordenadas de cada punto que pertenece a la línea, exceptuando el punto de terminación, a una función de respuesta definida por la aplicación. La función de respuesta determina como la línea será dibujada. Si los modos de mapeado y de transformaciones por defecto están activos, las coordenadas pasadas a la función de respuesta se corresponden con píxeles en la pantalla.

**Parámetros**

*p1*: Especifica la coordenada horizontal del punto de inicio de la línea.  
*p2*: Especifica la coordenada vertical del punto de inicio de la línea.  
*p3*: Especifica la coordenada horizontal del punto de terminación de la línea.  
*p4*: Especifica la coordenada vertical del punto de terminación de la línea.  
*p5*: Puntero a la función de respuesta definida por la aplicación.  
*p6*: Especifica un valor definido por la aplicación.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Sintaxis de la función de respuesta**

```

LineDDAProc(
  X: Integer;           {coordenada horizontal de la línea}
  Y: Integer;           {coordenada vertical de la línea}
  lpData: LPARAM        {datos definidos por la aplicación}
);                       {este procedimiento no devuelve ningún valor}

```

**Descripción**

Este procedimiento es llamado para cada píxel de la línea definida por la función *LineDDA*. La función de respuesta puede ejecutar cualquier acción de dibujo basada en estas coordenadas, tal como dibujar un píxel, copiar un mapa de bits, etc.

**Parámetros**

*X*: La coordenada horizontal actual a lo largo de la línea.

*Y*: La coordenada vertical actual a lo largo de la línea.

*lpData*: Especifica un valor de 32 bits definido por la aplicación, correspondiente al valor pasado a la función *LineDDA* en el parámetro *p6*. Este valor se utilizará para propósitos específicos de la aplicación.

**Valor que devuelve**

Este procedimiento no devuelve valor alguno.

**Véase además**

*ExtCreatePen*, *LineTo*

**Ejemplo****Listado 3-31: Dibujando un rectángulo de selección animado**

```
{El prototipo de la función de respuesta}
procedure AnimLines(X, Y: Integer; lpData: lParam); stdcall;

var
  Form1: TForm1;
  Offset: Integer;

const
  AL_HORIZONTAL = 1;    // indica si la línea a dibujar es
  AL_VERTICAL   = 2;    // horizontal o vertical

implementation

{$R *.DFM}

procedure AnimLines(X, Y: Integer; lpData: lParam);
var
  Coord: Integer;      // almacena la coordenada utilizada en el cálculo
begin
  {Si la línea es horizontal, usar la coordenada X; en caso contrario, usar Y}
  if lpData = AL_HORIZONTAL then
    Coord := X
  else
    Coord := Y;

  {Determinar si el píxel de este punto debe ser negro o blanco}
  if (Coord mod 5 = Offset) then
    SetPixelV(Form1.Canvas.Handle, X, Y, clBlack)
  else
    SetPixelV(Form1.Canvas.Handle, X, Y, clWhite);
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
```

```

{Incrementar el desplazamiento}
Inc(Offset);

{Si el desplazamiento ha ido demasiado lejos, reinicializarlo}
if Offset>4 then Offset := 0;

{Dibujar un rectángulo con líneas animadas}
LineDDA(20, 20, 120, 20, @AnimLines, AL_HORIZONTAL);
LineDDA(120, 20, 120, 120, @AnimLines, AL_VERTICAL);
LineDDA(20, 20, 20, 120, @AnimLines, AL_VERTICAL);
LineDDA(20, 120, 120, 120, @AnimLines, AL_HORIZONTAL);
end;

```

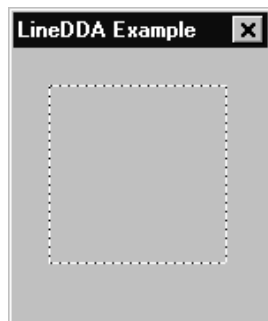


Figura 3-35:  
El rectángulo  
animado

## LineTo Windows.Pas

### Sintaxis

```

LineTo(
  DC: HDC;           {manejador de contexto de dispositivo}
  X: Integer;         {coordenada horizontal del destino de la línea}
  Y: Integer;         {coordenada vertical del destino de la línea}
): BOOL;             {devuelve TRUE o FALSE}

```

### Descripción

Esta función dibuja una línea utilizando la pluma seleccionada en el contexto de dispositivo especificado. La línea es dibujada desde la *posición actual* hasta la posición que indican las coordenadas *X* e *Y*. El punto con las coordenadas especificadas es excluido de los píxeles dibujados en la línea. La posición actual será actualizada a las coordenadas especificadas en *X* e *Y* cuando la función retorne.

### Parámetros

*DC*: Manejador del contexto de dispositivo sobre el cual la línea será dibujada.  
*X*: Especifica la coordenada horizontal del punto de terminación de la línea, en unidades lógicas.

Y: Especifica la coordenada vertical del punto de terminación de la línea, en unidades lógicas.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*LineDDA, MoveToEx, PolyBezier, PolyBezierTo, Polyline, PolylineTo, PolyPolyline*

#### Ejemplo

Vea el Listado 3-23 bajo *GetCurrentPositionEx*.

### LockWindowUpdate      Windows.Pas

#### Sintaxis

```
LockWindowUpdate(
  hWndLock: HWND           {manejador de ventana}
): BOOL;                   {devuelve TRUE o FALSE}
```

#### Descripción

Esta función deshabilita o habilita todas las operaciones de dibujo y repintado en la ventana especificada. Una ventana *bloqueada* fijada no puede ser movida y sólo una ventana puede estar bloqueada en un momento dado. Windows registra las áreas de la ventana bloqueada en las que se producen operaciones de dibujo o pintura. Cuando la ventana es desbloqueada, el área afectada por esas operaciones es invalidada, haciendo que un mensaje *WM\_PAINT* sea enviado a la ventana. Si las funciones *GetDC* o *BeginPaint* son aplicadas a una ventana bloqueada, el contexto de dispositivo devuelto contendrá una región vacía visible.

#### Parámetros

*hWndLock*: Manejador de la ventana para la cual las operaciones de dibujo serán deshabilitadas. Si a este parámetro se le asigna cero, la ventana que esté actualmente bloqueada será desbloqueada.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*BeginPaint, GetDC*

#### Ejemplo

#### Listado 3-32: Habilitando y deshabilitando la actualización de una ventana

```
procedure TForm1.Button1Click(Sender: TObject);
```

```

begin
  {Deshabilitar actualización de la ventana}
  LockWindowUpdate(Form1.Handle);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  {Habilitar el repintado de la ventana}
  LockWindowUpdate(0);
end;

```

## MoveToEx Windows.Pas

### Sintaxis

```

MoveToEx(
  DC: HDC;           {manejador de contexto de dispositivo}
  p2: Integer;       {coordenada horizontal}
  p3: Integer;       {coordenada vertical}
  p4: PPoint         {puntero a un registro TPoint}
): BOOL;            {si tiene éxito, devuelve TRUE}

```

### Descripción

Esta función mueve la posición actual del contexto de dispositivo a las coordenadas especificadas, devolviendo la posición anterior. Esto afectará a todas las llamadas posteriores a funciones de dibujo que utilizan la posición actual como punto de inicio.

### Parámetros

*DC*: Manejador de un contexto de dispositivo cuya posición actual será asignada.

*p2*: Especifica la coordenada horizontal de la nueva posición actual, en unidades lógicas.

*p3*: Especifica la coordenada vertical de la nueva posición actual, en unidades lógicas.

*p4*: Puntero a un registro *TPoint* que recibirá las coordenadas de la vieja posición. A este parámetro puede asignársele **nil**, si las coordenadas de la antigua posición actual no son necesarias.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### Véase además

*LineTo*, *PolyBezierTo*, *PolylineTo*

### Ejemplo

Vea el Listado 3-23 bajo *GetCurrentPositionEx*.

**PaintDesktop**      **Windows.Pas***Sintaxis*

```

PaintDesktop(
    hdc: HDC           {manejador de contexto de dispositivo}
): BOOL;              {devuelve TRUE o FALSE}

```

*Descripción*

Esta función rellena la región de recorte del contexto de dispositivo especificado con el mapa de bits del papel tapiz o patrón del escritorio.

*Parámetros*

*hdc*: Manejador del contexto de dispositivo sobre el cual el papel tapiz o el patrón del escritorio será dibujado.

*Valor que devuelve*

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

*Véase además*

*BitBlt*, *GetDC*, *SystemParametersInfo*

*Ejemplo***Listado 3-33: Dibujando el escritorio sobre un formulario**

```

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Mostrar el papel del escritorio}
    PaintDesktop(Canvas.Handle);
end;

procedure TForm1.WMMoving(var Msg: TMessage);
begin
    {Mostrar el papel del escritorio cuando se mueve}
    PaintDesktop(Canvas.Handle);
end;

```

**PaintRgn**      **Windows.Pas***Sintaxis*

```

PaintRgn(
    DC: HDC;           {manejador de contexto de dispositivo}
    RGN: HRGN          {manejador de región}
): BOOL;              {devuelve TRUE o FALSE}

```

**Descripción**

Esta función rellena la región especificada del contexto de dispositivo, utilizando la brocha actualmente seleccionada.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuya región será rellena.

*RGN*: Manejador de la región que será rellena.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*ExtFloodFill*, *FillPath*, *FillRect*, *FillRgn*, *FrameRect*, *FrameRgn*

**Ejemplo****Listado 3-34: Rellenando una región con la brocha actual**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    PointsArray: array[0..5] of TPoint;    // los puntos que definen la región
    RegionHandle: HRgn;                     // el manejador de la región
begin
    {Definir la región}
    PointsArray[0].x := 50;  PointsArray[0].y := 50;
    PointsArray[1].x := 100; PointsArray[1].y := 50;
    PointsArray[2].x := 125; PointsArray[2].y := 75;
    PointsArray[3].x := 100; PointsArray[3].y := 100;
    PointsArray[4].x := 50;  PointsArray[4].y := 100;
    PointsArray[5].x := 25;  PointsArray[5].y := 75;
    {Crear la región}
    RegionHandle := CreatePolygonRgn(PointsArray, 6, ALTERNATE);
    {Pintar la región usando la brocha del área de dibujo}
    Canvas.Brush.Color := clGreen;
    Canvas.Brush.Style := bsBDiagonal;
    PaintRgn(Canvas.Handle, RegionHandle);
end;

```

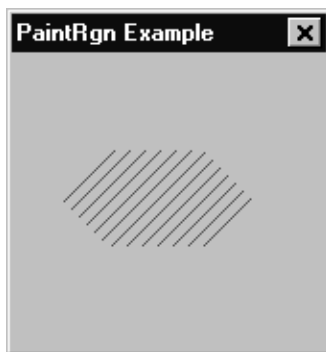


Figura 3-36:  
La región  
rellenada

**Pie**      **Windows.Pas****Sintaxis**

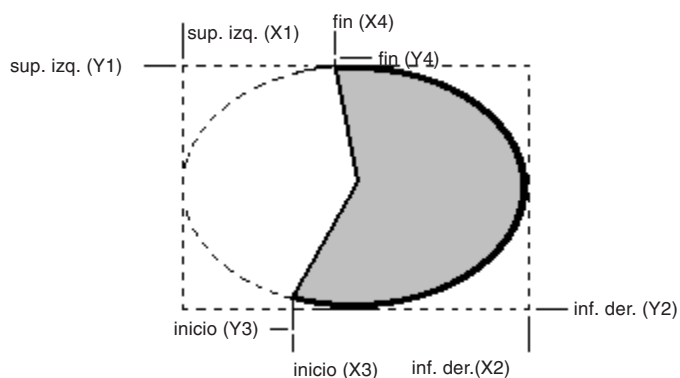
```

Pie(
  DC: HDC;                {manejador de contexto de dispositivo}
  X1: Integer;             {coordenada x de la esquina superior izquierda}
  Y1: Integer;             {coordenada y de la esquina superior izquierda}
  X2: Integer;             {coordenada x de la esquina inferior derecha}
  Y2: Integer;             {coordenada y de la esquina inferior derecha}
  X3: Integer;             {coordenada x del primer punto de terminación radial}
  Y3: Integer;             {coordenada y del primer punto de terminación radial}
  X4: Integer;             {coordenada x del segundo punto de terminación radial}
  Y4: Integer;             {coordenada y del segundo punto de terminación radial}
); BOOL;                  {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja una cuña de tarta utilizando la pluma seleccionada y rellena esa cuña utilizando la brocha seleccionada. Una *cuña de tarta* es una región delimitada por una elipse y dos segmentos de línea radiales. Las dimensiones de la cuña dependen del rectángulo límite que encierra la elipse. Los parámetros *X3* e *Y3* definen los puntos de terminación de la línea radial, comenzando en el centro del rectángulo límite, e identifican la ubicación del inicio del área de ésta. Los parámetros *X4* e *Y4* definen los puntos de terminación de la línea radial, comenzando desde el centro del rectángulo límite, e identifican la ubicación final del área de la misma. La cuña es dibujada en sentido contrario a las manecillas del reloj. Esta función no afecta la posición actual.



**Figura 3-37:**  
Coordenadas  
de la tarta

**Parámetros**

*DC*: Especifica el contexto de dispositivo en el que la cuña será dibujada.

*X1*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo límite, en unidades lógicas. Bajo Windows 95/98 la suma de los parámetros *X1* y *X2* tiene que ser menor que 32.767.

*Y1*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo límite, en unidades lógicas. Bajo Windows 95/98 la suma de los parámetros *Y1* e *Y2* tiene que ser menor que 32.767.

*X2*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo límite, en unidades lógicas.

*Y2*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo límite, en unidades lógicas.

*X3*: Especifica la coordenada horizontal, en unidades lógicas, del punto de terminación de la línea radial que define el punto de inicio de la cuña de tarta.

*Y3*: Especifica la coordenada vertical, en unidades lógicas, del punto de terminación de la línea radial que define el punto de inicio de la cuña de la tarta.

*X4*: Especifica la coordenada horizontal, en unidades lógicas, del punto de terminación de la línea radial que define el punto de terminación de la tarta.

*Y4*: Especifica la coordenada vertical, en unidades lógicas, del punto de terminación de la línea radial que define el punto de terminación de la tarta.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*Arc*, *Chord*, *Ellipse*

#### Ejemplo

##### Listado 3-35: Dibujando una cuña de tarta

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    {Dibujar una cuña de tarta}
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Pie(Canvas.Handle, 10, 10, 110, 110, 10, 60, 60, 10);
end;
```

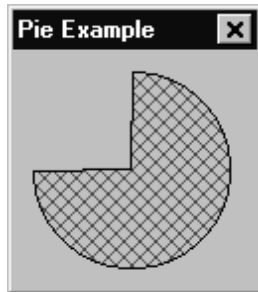


Figura 3-38:  
La cuña de la  
tarta

### PolyBezier

### Windows.Pas

#### Sintaxis

```
PolyBezier(
    DC: HDC;                {manejador de contexto de dispositivo}
    const Points;            {puntero a un array de coordenadas}
    Count: DWORD             {cantidad de entradas en el array}
): BOOL;                   {devuelve TRUE o FALSE}
```

#### Descripción

Esta función dibuja una o más curvas cúbicas de Bézier en el contexto de dispositivo especificado utilizando la pluma seleccionada. El parámetro *Points* apunta a un *array* de registros de tipo *TPoint* que contienen el punto de inicio, los puntos de control y el punto de terminación de las curvas de Bézier. El primer punto del *array* define el punto de inicio de la curva. Los dos próximos son usados como puntos de control, y el cuarto define el punto de terminación. Cada trío posterior define los dos puntos de control y el punto de terminación de otra curva de Bézier, utilizando el punto de terminación de la curva anterior como punto de inicio. Esta función no modifica la posición actual.

#### Parámetros

*DC*: Manejador del contexto de dispositivo sobre el cual la curva de Bézier es dibujada.

*Points*: Puntero a un *array* de registros *TPoint* que contiene los puntos de control y los puntos de terminación de las curvas de Bézier.

*Count*: Especifica la cantidad de entradas en el *array* al que apunta el parámetro *Points*.

#### Valores que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*MoveToEx*, *PolyBezierTo*

*Ejemplo***Listado 3-36: Dibujando una curva de Bézier**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    Points: array[0..6] of TPoint; // puntos que definen la curva de Bézier
begin
    {Definir la curva de Bézier}
    Points[0].X := 10;   Points[0].Y := 50;
    Points[1].X := 40;   Points[1].Y := 90;
    Points[2].X := 80;   Points[2].Y := 10;
    Points[3].X := 110;  Points[3].Y := 50;
    Points[4].X := 140;  Points[4].Y := 10;
    Points[5].X := 180;  Points[5].Y := 90;
    Points[6].X := 210;  Points[6].Y := 50;

    {Dibujar la curva de Bézier}
    PolyBezier(Canvas.Handle, Points, 7);
end;

```

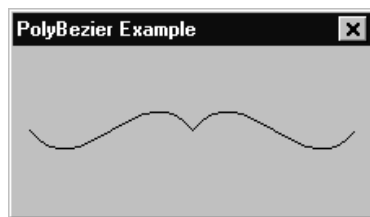


Figura 3-39:  
La curva de  
Bézier

**PolyBezierTo      Windows.Pas****Sintaxis**

```

PolyBezierTo(
    DC: HDC;                {manejador de contexto de dispositivo}
    const Points;            {puntero a un array de coordenadas}
    Count: DWORD             {número de entradas en el array}
): BOOL;                   {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja una o más curvas cúbicas de Bézier en el contexto de dispositivo especificado, utilizando la pluma seleccionada. El parámetro *Points* apunta a un *array* de registros de tipo *TPoint* que contienen el punto de inicio, los puntos de control y los puntos de terminación de las curvas de Bézier. El primer punto del *array* define el punto de inicio de la curva. Los dos próximos puntos son utilizados como puntos de control, y el cuarto define el punto de terminación. Cada trío de puntos siguiente determina los dos puntos de control y el punto de terminación de otra curva de Bézier, que utilizará el punto de terminación de la curva anterior como punto de inicio. La posición actual será actualizada con el último punto del *array Points*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el que la curva de Bézier será dibujada.

*Points*: Puntero a un *array* de registros *TPoint* que contiene los puntos de control y de terminación de las curvas de Bézier.

*Count*: Especifica la cantidad de entradas en el *array* al que apunta el parámetro *Points*.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*MoveToEx*, *PolyBezier*

**Ejemplo****Listado 3-37: Dibujando una curva de Bézier y actualizando la posición actual**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    Points: array[0..2] of TPoint; // puntos que definen la curva de Bézier
begin
    {Definir la curva de Bézier}
    Points[0].X := 40;  Points[0].Y := 110;
    Points[1].X := 80;  Points[1].Y := 30;
    Points[2].X := 110; Points[2].Y := 70;

    {Mover la posición actual al punto de inicio correcto}
    MoveToEx(Canvas.Handle, 10, 70, nil);

    {Dibuja la curva de Bézier}
    PolyBezierTo(Canvas.Handle, Points, 3);

    {La posición fue actualizada, podemos usarla para continuar dibujando la imagen}
    LineTo(Canvas.Handle, 110, 10);
    LineTo(Canvas.Handle, 10, 10);
    LineTo(Canvas.Handle, 10, 70);
end;

```

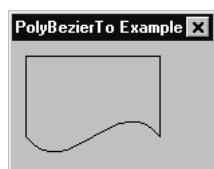


Figura 3-40:  
La curva de  
Bézier

**Polygon Windows.Pas****Sintaxis**

```
Polygon(
    DC: HDC;                {manejador de contexto de dispositivo}
    const Points;            {puntero a un array de coordenadas}
    Count: DWORD             {cantidad de entradas en el array}
); BOOL;                    {devuelve TRUE o FALSE}
```

**Descripción**

Esta función dibuja un polígono en el contexto de dispositivo especificado utilizando la pluma seleccionada, y rellena el polígono utilizando la brocha seleccionada y el modo de relleno de polígono actual del contexto de dispositivo. El parámetro *Points* apunta a un *array* de registros de tipo *TPoint* que definen los vértices del polígono. El polígono será cerrado automáticamente con una línea desde el último al primer vértice en el *array*. Esta función no afecta a la posición actual.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el que el polígono será dibujado.

*Points*: Puntero a un *array* de registros de tipo *TPoint* que contienen los vértices del polígono. Este *array* tiene que contener al menos dos vértices, o la función fallará.

*Count*: Especifica la cantidad de entradas en el *array* al que apunta el parámetro *Points*.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetPolyFillMode*, *Polyline*, *PolylineTo*, *PolyPolygon*, *PolyPolyline*, *SetPolyFillMode*

**Ejemplo**

Vea el Listado 3-25 bajo *GetPolyFillMode*.

**Polyline Windows.Pas****Sintaxis**

```
Polyline(
    DC: HDC;                {manejador de contexto de dispositivo}
    const Points;            {puntero a un array de coordenadas}
    Count: DWORD             {cantidad de entradas en el array}
); BOOL;                    {devuelve TRUE o FALSE}
```

**Descripción**

Esta función dibuja un polígono en el contexto de dispositivo especificado utilizando la pluma actual. El parámetro *Points* apunta a un *array* de registros de tipo *TPoint* que definen los vértices del polígono. Este es dibujado uniendo los puntos del *array* mediante segmentos de línea. Esta función no afecta la posición actual.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el que el polígono será dibujado.

*Points*: Puntero a un *array* de registros *TPoint* que contiene los vértices del polígono.

*Count*: Especifica la cantidad de entradas en el *array* al que apunta el parámetro *Points*.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*LineTo*, *MoveToEx*, *PolylineTo*, *PolyPolyline*

**Ejemplo****Listado 3-38: Dibujando un polígono delineado**

```
procedure TForm1.FormPaint(Sender: TObject);
var
  PointsArray: array[0..6] of TPoint;      // puntos que definen el polígono
begin
  {Definir los vértices del polígono}
  PointsArray[0].X := 50;  PointsArray[0].Y := 50;
  PointsArray[1].X := 100; PointsArray[1].Y := 50;
  PointsArray[2].X := 125; PointsArray[2].Y := 75;
  PointsArray[3].X := 100; PointsArray[3].Y := 100;
  PointsArray[4].X := 50;  PointsArray[4].Y := 100;
  PointsArray[5].X := 25;  PointsArray[5].Y := 75;
  PointsArray[6].X := 50;  PointsArray[6].Y := 50;

  {Dibujar el polígono}
  Polyline(Canvas.Handle, PointsArray, 7);
end;
```

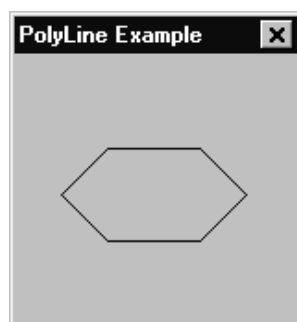


Figura 3-41:  
El polígono  
vacío

**PolylineTo****Windows.Pas****Sintaxis**

```

PolylineTo(
    DC: HDC;                {manejador de contexto de dispositivo}
    const Points;            {puntero a un array de coordenadas}
    Count: DWORD             {cantidad de entradas en el array}
); BOOL;                    {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja un polígono en el contexto de dispositivo especificado, utilizando la pluma seleccionada. El parámetro *Points* apunta a un *array* de registros de tipo *TPoint* que definen los vértices del polígono. Este es dibujado uniendo los puntos del *array* mediante segmentos de línea, comenzando por la posición actual. Cuando la función retorna, la posición actual es actualizada con las coordenadas del último punto del *array*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el cual el polígono será dibujado.

*Points*: Puntero a un *array* de registros *TPoint* que contiene los vértices del polígono.

*Count*: Especifica la cantidad de entradas en el *array* al que apunta el parámetro *Points*.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*LineTo*, *MoveToEx*, *Polyline*, *PolyPolyline*

**Ejemplo****Listado 3-39: Dibujando un polígono no relleno comenzando desde la posición actual**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    PointsArray: array[0..5] of TPoint; // los puntos que definen el polígono
begin
    {Mover la posición actual al punto donde comienza el polígono}
    MoveToEx(Canvas.Handle, 50, 50, nil);

    {Definir el polígono}
    PointsArray[0].x := 100; PointsArray[0].y := 50;
    PointsArray[1].x := 125; PointsArray[1].y := 75;
    PointsArray[2].x := 100; PointsArray[2].y := 100;
    PointsArray[3].x := 50;  PointsArray[3].y := 100;
    PointsArray[4].x := 25;  PointsArray[4].y := 75;
    PointsArray[5].x := 50;  PointsArray[5].y := 50;

```

```

    {Dibujar el polígono, comenzando en la posición actual}
    PolylineTo(Canvas.Handle, PointsArray, 6);
end;

```

## **PolyPolygon**      **Windows.Pas**

### **Sintaxis**

```

PolyPolygon(
  DC: HDC;                            {manejador de contexto de dispositivo}
  var Points;                        {puntero a un array de coordenadas}
  var nPoints;                       {puntero a un array de cantidades de vértices}
  p4: Integer                        {cantidad de polígonos}
): BOOL;                            {devuelve TRUE o FALSE}

```

### **Descripción**

Esta función dibuja una serie de polígonos cerrados en el contexto de dispositivo especificado utilizando la pluma seleccionada, y rellena los polígonos usando la brocha seleccionada y el modo de relleno de polígono actual del contexto de dispositivo. El parámetro *Points* apunta a un *array* de registros *TPoint* que definen los vértices de cada polígono. El parámetro *nPoints* apunta a un *array* de enteros, en el que cada entero especifica la cantidad de elementos en el *array Points* que definen un polígono. Cada uno será automáticamente cerrado con una línea que va desde el último vértice hasta el primero. Esta función no modifica la posición actual.

### **Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el que los polígonos serán dibujados.

*Points*: Puntero a un *array* de registros de tipo *TPoint* que contiene los vértices de los polígonos. Todos los vértices se utilizan en orden consecutivo y deben ser especificados una vez. Los polígonos definidos por este *array* pueden solaparse.

*nPoints*: Puntero a un *array* de enteros, en el que cada entero especifica el número de elementos del *array Points* que define cada polígono individual.

*p4*: Indica la cantidad total de polígonos que serán dibujados.

### **Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### **Véase además**

*GetPolyFillMode*, *Polygon*, *Polyline*, *PolylineTo*, *PolyPolyline*, *SetPolyFillMode*

## Ejemplo

## Listado 3-40: Dibujando múltiples polígonos

```

procedure TForm1.FormPaint(Sender: TObject);
var
    PointsArray: array[0..9] of TPoint; // almacena los vértices de los polígonos
    NPoints: array[0..1] of Integer;    // cantidad de vértices en cada polígono
begin
    {Definir los polígonos}
    {primer polígono}
    PointsArray[0].X := 50;  PointsArray[0].Y := 50;
    PointsArray[1].X := 100; PointsArray[1].Y := 50;
    PointsArray[2].X := 125; PointsArray[2].Y := 75;
    PointsArray[3].X := 100; PointsArray[3].Y := 100;
    PointsArray[4].X := 50;  PointsArray[4].Y := 100;
    PointsArray[5].X := 25;  PointsArray[5].Y := 75;
    {segundo polígono}
    PointsArray[6].X := 200; PointsArray[6].Y := 25;
    PointsArray[7].X := 300; PointsArray[7].Y := 25;
    PointsArray[8].X := 300; PointsArray[8].Y := 125;
    PointsArray[9].X := 200; PointsArray[9].Y := 125;

    {Indicar cuantos vértices hay en cada polígono}
    NPoints[0] := 6;
    NPoints[1] := 4;

    {Dibujar los polígonos}
    PolyPolygon(Canvas.Handle, PointsArray, NPoints, 2);
end;

```

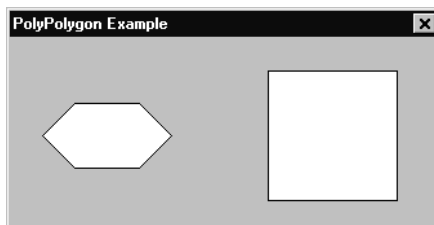


Figura 3-42:  
Múltiples  
polígonos

## PolyPolyline      Windows.Pas

### Sintaxis

PolyPolyline(	
DC: HDC;	{manejador de contexto de dispositivo}
<b>const</b> PointStructs;	{puntero a un <i>array</i> de coordenadas}
<b>const</b> Points;	{puntero a un <i>array</i> de cantidades de vértices}
p4: DWORD	{cantidad de polígonos}
); BOOL;	{devuelve TRUE o FALSE}

**Descripción**

Esta función dibuja una serie de polígonos en el contexto de dispositivo especificado, utilizando la pluma seleccionada. El parámetro *PointStructs* apunta a un *array* de registros de tipo *TPoint* que contiene los vértices de cada polígono. El parámetro *Points* apunta a un *array* de enteros, donde cada entero especifica la cantidad de elementos del *array PointStructs* que definen un polígono. Cada polígono es dibujado uniendo los puntos del *array* mediante segmentos de línea. Esta función no afecta a la posición actual.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el que los polígonos serán dibujados.

*PointStructs*: Puntero a un *array* de registros *TPoint* que contiene los vértices de cada polígono. Los vértices son utilizados en orden consecutivo y deben ser especificados una sola vez.

*Points*: Puntero a un *array* de enteros, donde cada entero especifica la cantidad de elementos en el *array PointStructs* que definen cada polígono individual.

*p4*: Indica la cantidad total de polígonos que serán dibujados.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*Polygon*, *Polyline*, *PolylineTo*, *PolyPolygon*

**Ejemplo****Listado 3-41: Dibujando múltiples polígonos sin rellenar**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    PointsArray: array[0..11] of TPoint; // los vértices que define el polígono
    NPoints: array[0..1] of Integer;    // cantidad de vértices en cada polígono
begin
    {Definir los polígonos}
    {primer polígono}
    PointsArray[0].x := 50;  PointsArray[0].y := 50;
    PointsArray[1].x := 100; PointsArray[1].y := 50;
    PointsArray[2].x := 125; PointsArray[2].y := 75;
    PointsArray[3].x := 100; PointsArray[3].y := 100;
    PointsArray[4].x := 50;  PointsArray[4].y := 100;
    PointsArray[5].x := 25;  PointsArray[5].y := 75;
    PointsArray[6].x := 50;  PointsArray[6].y := 50;
    {segundo polígono}
    PointsArray[7].x := 200; PointsArray[7].y := 25;
    PointsArray[8].x := 300; PointsArray[8].y := 25;
    PointsArray[9].x := 300; PointsArray[9].y := 125;
    PointsArray[10].x := 200; PointsArray[10].y := 125;

```

```

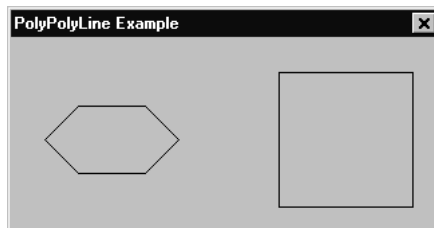
PointsArray[11].X := 200; PointsArray[11].Y := 25;

{Indicar cuantos vértices tiene cada polígono}
NPoints[0] := 7;
NPoints[1] := 5;

{Dibujar el polígono sin relleno}
PolyPolyline(Canvas.Handle, PointsArray, NPoints, 2);
end;

```

Figura 3-43:  
Múltiples  
polígonos no  
rellenados



## Rectangle

## Windows.Pas

### Sintaxis

Rectangle(	
DC: HDC;	{manejador de contexto de dispositivo}
X1: Integer;	{coordenada horizontal de la esquina superior izquierda}
Y1: Integer;	{coordenada vertical de la esquina superior izquierda}
X2: Integer;	{coordenada horizontal de la esquina inferior derecha}
Y2: Integer	{coordenada vertical de la esquina inferior derecha}
): BOOL;	{devuelve TRUE o FALSE}

### Descripción

Esta función dibuja un rectángulo en las coordenadas indicadas del contexto de dispositivo especificado utilizando la pluma seleccionada y rellenándolo con la brocha seleccionada. Esta función no modifica la posición actual.

### Parámetros

**DC:** Manejador del contexto de dispositivo sobre el cual el rectángulo es dibujado.

**X1:** Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo, en unidades lógicas.

**Y1:** Especifica la coordenada vertical de la esquina superior izquierda del rectángulo, en unidades lógicas.

**X2:** Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo, en unidades lógicas.

**Y2:** Especifica la coordenada vertical de la esquina inferior derecha del rectángulo, en unidades lógicas.

*Valor que devuelve*

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

*Véase además*

*CreateRectRgn, CreateRectRgnIndirect, FillRect, FrameRect, Polygon, Polyline, RoundRect*

*Ejemplo***Listado 3-42: Dibujando un rectángulo con un relleno animado**

```

var
    Form1: TForm1;
    BrushOffset: Integer;    // almacena el desplazamiento de la brocha actual

implementation

{$R *.DFM}

procedure TForm1.Timer1Timer(Sender: TObject);
var
    BrushPt: TPoint;          // almacena el origen de la brocha actual
    BrushHndl, OldBrush: HBRUSH; // manejadores para las brochas
    FormDC: HDC;              // el contexto de dispositivo del formulario
begin
    {Recuperar el contexto de dispositivo del formulario}
    FormDC := GetDC(Form1.Handle);

    {Incrementar el desplazamiento de la brocha}
    Inc(BrushOffset);

    {Crear una brocha con trama}
    BrushHndl := CreateHatchBrush(HS_DIAGCROSS, clRed);

    {Asignar el origen de la brocha}
    SetBrushOrgEx(FormDC, BrushOffset, BrushOffset, nil);

    {Seleccionar la brocha dentro del contexto de dispositivo}
    OldBrush := SelectObject(FormDC, BrushHndl);

    {Recuperar el origen de la brocha actual}
    GetBrushOrgEx(FormDC, BrushPt);

    {Si el origen de la brocha está más allá del límite, inicializarlo}
    if BrushPt.X > 7 then begin
        BrushOffset := 0;
        SetBrushOrgEx(FormDC, BrushOffset, BrushOffset, nil);
    end;

    {Dibujar el rectángulo}

```

```

Rectangle(FormDC, 10, 10, 110, 110);

{Borrar la brocha nueva}
SelectObject(FormDC, OldBrush);
DeleteObject(BrushHnd1);

{Liberar el contexto de dispositivo del formulario}
ReleaseDC(Form1.Handle, FormDC);
end;

```

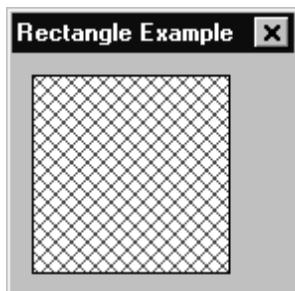


Figura 3-44:  
El rectángulo  
animado

### RoundRect

### Windows.Pas

#### Sintaxis

RoundRect(	
DC: HDC;	{manejador de contexto de dispositivo}
X1: Integer;	{coordenada horizontal de la esquina superior izquierda}
Y1: Integer;	{coordenada vertical de la esquina superior izquierda}
X2: Integer;	{coordenada horizontal de la esquina inferior derecha}
Y2: Integer;	{coordenada vertical de la esquina inferior derecha}
X3: Integer;	{ancho de elipse de la esquina}
Y3: Integer	{altura de elipse de la esquina}
); BOOL;	{devuelve TRUE o FALSE}

#### Descripción

Esta función dibuja un rectángulo en las coordenadas especificadas del contexto de dispositivo indicado, utilizando la pluma seleccionada y rellenándolo con la brocha seleccionada. Las esquinas del rectángulo serán redondeadas de acuerdo a los valores de la elipse formada por los parámetros *X3* e *Y3*. Esta función no modifica la posición actual.

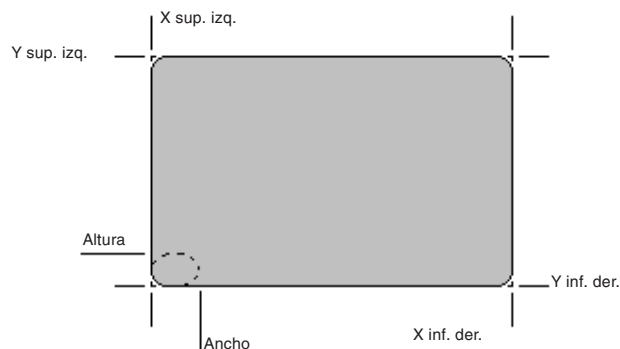


Figura 3-45:  
Coordenadas  
de RoundRect

#### Parámetros

*DC*: Manejador del contexto de dispositivo sobre el que se dibujará el rectángulo redondeado.

*X1*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo redondeado, en unidades lógicas.

*Y1*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo redondeado, en unidades lógicas.

*X2*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo redondeado, en unidades lógicas.

*Y2*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo redondeado, en unidades lógicas.

*X3*: Indica el ancho de la elipse usada para dibujar las esquinas.

*Y3*: Indica la altura de la elipse usada para dibujar las esquinas.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

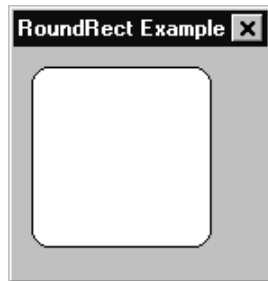
*CreateRoundRectRgn*, *FillRect*, *FrameRect*, *Polygon*, *Polyline*, *Rectangle*

#### Ejemplo

##### Listado 3-43: Dibujando un rectángulo redondeado

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    {Crear un rectángulo redondeado}
    RoundRect(Canvas.Handle, 10, 10, 110, 110, 20, 20);
end;
```

Figura 3-46:  
El rectángulo  
redondeado



### SelectObject Windows.Pas

#### Sintaxis

```
SelectObject(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: HGDI OBJ            {manejador de objeto gráfico}
): HGDI OBJ;               {devuelve un manejador del anterior objeto seleccionado}
```

#### Descripción

Esta función selecciona el objeto gráfico especificado dentro del contexto de dispositivo. La mayoría de los objetos gráficos tienen que ser seleccionados dentro de un contexto de dispositivo antes de que puedan ser usados en funciones de dibujo. El último objeto seleccionado reemplaza al objeto del mismo tipo seleccionado previamente. La aplicación deberá volver a seleccionar el objeto anterior en el contexto de dispositivo cuando el nuevo objeto deje de ser necesario. Un objeto gráfico no puede ser destruido mientras está seleccionado dentro de un contexto de dispositivo.

#### Parámetros

*DC*: Manejador del contexto de dispositivo en el que el objeto es seleccionado.

*p2*: Especifica un manejador del objeto gráfico que será seleccionado dentro del contexto de dispositivo, y puede ser una brocha, una pluma, un mapa de bits, una región, o una fuente. Tenga en cuenta que los mapas de bits sólo pueden estar seleccionados dentro de un contexto de dispositivo a la vez.

#### Valor que devuelve

Si el objeto seleccionado no es una región y la función tiene éxito, ésta devuelve un manejador del objeto previamente seleccionado del mismo tipo. Si la función tiene éxito y el objeto seleccionado es una región, devuelve un valor de la Tabla 3-33. Si la función falla y el objeto seleccionado no es una región, devuelve cero; en caso contrario, devuelve *GDI\_ERROR*.

#### Véase además

*CombineRgn*, *CreateBitmap*, *CreateBitmapIndirect*, *CreateBrushIndirect*, *CreateCompatibleBitmap*, *CreateDIBitmap*, *CreateEllipticRgn*,

*CreateEllipticRgnIndirect, CreateFont, CreateFontIndirect, CreateHatchBrush, CreatePatternBrush, CreatePen, CreatePenIndirect, CreatePolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateSolidBrush, DeleteObject, SelectClipRgn, SelectPalette*

### Ejemplo

Vea el Listado 3-8 bajo *CreateSolidBrush* y otros ejemplos a lo largo de este libro.

**Tabla 3-33: Valores que devuelve SelectObject**

Valor	Descripción
SIMPLEREGION	La región es un rectángulo simple.
COMPLEXREGION	La región consta de múltiples rectángulos.
NULLREGION	La región está vacía.

## SetBkColor

## Windows.Pas

### Sintaxis

```
SetBkColor(
    DC: HDC;           {manejador de contexto de dispositivo}
    Color: COLORREF    {el nuevo color de fondo}
): COLORREF;          {devuelve el color de fondo anterior}
```

### Descripción

Esta función selecciona el color de fondo para el contexto de dispositivo especificado. Si el dispositivo no puede representar el color especificado, utilizará el color disponible más cercano.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuyo color de fondo va a ser asignado.

*Color*: Un especificador de color que identifica el nuevo color de fondo.

### Valor que devuelve

Si la función tiene éxito, devuelve el color de fondo anterior; en caso contrario, devuelve *CLR\_INVALID*.

### Véase además

*CreatePen, ExtCreatePen, GetBkColor, GetBkMode, SetBkMode*

### Ejemplo

Vea el Listado 3-21 bajo *GetBkColor*.

**SetBkMode**      **Windows.Pas****Sintaxis**

```
SetBkMode(
    DC: HDC;                {manejador de contexto de dispositivo}
    BkMode: Integer          {opción de modo de fondo}
): Integer;                {devuelve el modo de fondo previo}
```

**Descripción**

Esta función selecciona el modo de mezcla del fondo del contexto de dispositivo dado.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo modo de mezcla del fondo será modificado.

*BkMode*: Una opción que indica el nuevo modo de mezcla del fondo. Este parámetro puede tomar un valor de la Tabla 3-34.

**Valor que devuelve**

Si la función tiene éxito, devuelve el modo de mezcla previo del fondo; en caso contrario, devuelve cero.

**Véase además**

*CreatePen*, *ExtCreatePen*, *GetBkColor*, *GetBkMode*, *SetBkColor*

**Ejemplo**

Vea el Listado 3-21 bajo *GetBkColor*.

**Tabla 3-34: Valores del parámetro BkMode de SetBkMode**

Valor	Descripción
OPAQUE	El color de fondo es usado para rellenar vacíos en el texto, las brochas con tramas y los patrones de plumas.
TRANSPARENT	El color del contexto de dispositivo se dejará ver a través de los vacíos en el texto, las brochas con tramas y los patrones de plumas.

**SetBoundsRect****Windows.Pas****Sintaxis**

```
SetBoundsRect(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: TRect;              {puntero a un registro TRect}
    p3: UINT                {opciones de operaciones}
```

); UINT; {devuelve el estado anterior del rectángulo límite}

### Descripción

Esta función modifica el comportamiento de acumulación del rectángulo límite del contexto de dispositivo dado. Para cada contexto de dispositivo, Windows mantiene un rectángulo límite acumulado que indica las dimensiones máximas de la salida producida por las funciones de dibujo. Cuando una función de dibujo sobrepasa esas fronteras, el rectángulo límite es extendido. De esta manera, el rectángulo límite es el rectángulo más pequeño que puede ser dibujado alrededor del área afectada por todas las operaciones de dibujo en el contexto de dispositivo.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuyo comportamiento de acumulación del rectángulo límite será modificado.

*p2*: Puntero a un registro *TRect* que contiene las coordenadas de rectángulo, en unidades lógicas, del nuevo rectángulo límite. A este parámetro se le puede asignar **nil** si no se necesita establecer el rectángulo límite.

*p3*: Una combinación de opciones que indican cómo el rectángulo especificado será combinado con el rectángulo límite actual y si la acumulación de rectángulo límite está habilitada. A este parámetro se le puede asignar una combinación de valores de la Tabla 3-35.

### Valor que devuelve

Esta función devuelve un código que indica el estado del rectángulo límite o una condición de error, y será uno o más valores de la Tabla 3-36.

### Véase además

*GetBoundsRect*, *GetUpdateRect*

### Ejemplo

Vea el Listado 3-22 bajo *GetBoundsRect*.

**Tabla 3-35: Valores del parámetro *p3* de *SetBoundsRect***

Valor	Descripción
DCB_ACCUMULATE	Añade el rectángulo especificado por el parámetro <i>p2</i> al rectángulo límite actual mediante una unión. Si las opciones DCB_RESET y DCB_ACCUMULATE son ambas especificadas, el rectángulo límite se iguala exactamente al rectángulo especificado por el parámetro <i>p2</i> .
DCB_DISABLE	Deshabilita la acumulación del rectángulo límite. Este es el estado por defecto.
DCB_ENABLE	Habilita la acumulación del rectángulo límite.
DCB_RESET	Borra el rectángulo límite.

Tabla 3-36: Valores que devuelve SetBoundsRect

Valor	Descripción
0	Indica que ha ocurrido un error.
DCB_DISABLE	La acumulación del rectángulo límite está deshabilitada.
DCB_ENABLE	La acumulación del rectángulo límite está habilitada.
DCB_RESET	El rectángulo límite está vacío.
DCB_SET	El rectángulo límite no está vacío.

**SetBrushOrgEx****Windows.Pas****Sintaxis**

```
SetBrushOrgEx(
    DC: HDC;           {manejador de contexto de dispositivo}
    X: Integer;         {coordenada horizontal del origen}
    Y: Integer;         {coordenada vertical del origen}
    PrevPt: PPoint     {puntero a registro TPoint}
): BOOL;              {devuelve TRUE o FALSE}
```

**Descripción**

Esta función asigna el origen de la próxima brocha que sea seleccionada en el contexto de dispositivo especificado. El origen de la brocha es relativo a la trama o al mapa de bits que definen el patrón de la brocha. El origen por defecto de la brocha es (0,0). Un patrón de brocha no puede tener más de ocho píxeles cuadrados. De esta manera, el origen puede estar entre 0 y 7, tanto vertical como horizontalmente. Cuando el origen es movido, el patrón de la brocha es desplazado en la cantidad indicada. Si una aplicación está usando un mismo patrón de brocha para dibujar el fondo de ventanas madres e hijas, puede ser necesario mover el origen de la brocha para alinear los patrones. Tenga en cuenta que, bajo Windows NT, el sistema ajusta automáticamente el origen de las brochas para que los patrones estén alineados.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo origen de brocha será asignado.

*X*: Especifica la coordenada horizontal del origen de la brocha, en unidades del dispositivo.

*Y*: Especifica la coordenada vertical del origen de la brocha, en unidades del dispositivo.

*PrevPt*: Puntero a un registro *TPoint* que recibirá las coordenadas del origen anterior de la brocha. A este parámetro se le puede asignar **nil**, si las coordenadas previas no son necesarias.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateBrushIndirect, CreateHatchBrush, CreatePatternBrush, FillRect, FillRgn, GetBrushOrgEx, SelectObject*

**Ejemplo**

Vea el Listado 3-42 bajo *Rectangle*.

**SetMiterLimit****Windows.Pas****Sintaxis**

```
SetMiterLimit(
    DC: HDC;                {manejador de contexto de dispositivo}
    NewLimit: Single;        {el nuevo límite del inglete}
    OldLimit: PSingle        {recibe el límite anterior del inglete}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función asigna el límite del inglete para el contexto de dispositivo especificado. El límite del inglete es utilizado al trazar líneas geométricas que tienen uniones a inglete, y es la relación máxima entre el largo del inglete y el ancho de la línea. La longitud del inglete es la distancia desde la intersección de la pared interna hasta la intersección de la pared externa. El límite del inglete por defecto es 10,0.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo límite del inglete será asignado.

*NewLimit*: Especifica el nuevo límite del inglete para el contexto de dispositivo.

*OldLimit*: Puntero a una variable que recibirá el límite anterior del inglete. A este parámetro se le puede asignar **nil** si el límite anterior no se necesita.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*ExtCreatePen, GetMiterLimit*

**Ejemplo**

Vea el Listado 3-16 bajo *ExtCreatePen*.

**SetPixel Windows.Pas****Sintaxis**

```
SetPixel(
  DC: HDC;           {manejador de contexto de dispositivo}
  X: Integer;         {coordenada horizontal del píxel}
  Y: Integer;         {coordenada vertical del píxel}
  Color: COLORREF     {el nuevo color del píxel}
): COLORREF;         {devuelve un especificador de color}
```

**Descripción**

Esta función asigna el color del píxel situado en las coordenadas especificadas dentro del contexto de dispositivo indicado. Las coordenadas tienen que estar dentro de las fronteras de la región de recorte actual.

**Parámetros**

*DC*: Manejador del contexto de dispositivo en el que se modifica el color de un píxel.

*X*: La coordenada horizontal del píxel dentro del contexto de dispositivo, en unidades lógicas.

*Y*: La coordenada vertical del píxel dentro del contexto de dispositivo, en unidades lógicas.

*Color*: Especifica el nuevo color del píxel.

**Valor que devuelve**

Si la función tiene éxito, devuelve el color que se le asignó al píxel. Este puede ser diferente del especificado si no se puede hallar un color correspondiente exacto. Si la función falla, devuelve *CLR\_INVALID*.

**Véase además**

*GetPixel*, *SetPixelV*

**Ejemplo****Listado 3-44: Implementando un efecto sencillo de degradado de mapa de bits**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {Borrar la imagen actual}
  Canvas.Brush.Color := Color;
  Canvas.FillRect(ClientRect);
```

```

    {Comenzar el efecto}
    Timer1.Enabled := TRUE;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var
    X, Y: Integer;      // coordenadas del píxel
    iCount: Integer;    // contador de bucle
begin
    {Comenzar el efecto de degradado sencillo}
    for iCount := 0 to 20000 do
    begin
        {Recuperar una coordenada aleatoria}
        X := Random(Image1.Width - 1);
        Y := Random(Image1.Height - 1);

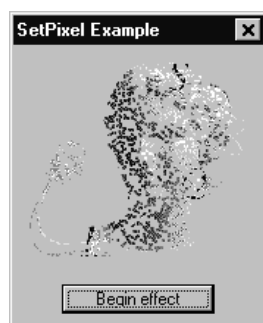
        {En un cuadrado de 4x4 píxeles a partir de este punto, recuperar los píxeles
        de la imagen original, y asignarlos en el área de dibujo del formulario}
        SetPixel(Canvas.Handle, X+Image1.Left, Y+Image1.Top,
            GetPixel(Image1.Picture.Bitmap.Canvas.Handle, X, Y));
        SetPixel(Canvas.Handle, X+1+Image1.Left, Y+Image1.Top,
            GetPixel(Image1.Picture.Bitmap.Canvas.Handle, X+1, Y));
        SetPixel(Canvas.Handle, X+Image1.Left, Y+1+Image1.Top,
            GetPixel(Image1.Picture.Bitmap.Canvas.Handle, X, Y+1));
        SetPixel(Canvas.Handle, X+1+Image1.Left, Y+1+Image1.Top,
            GetPixel(Image1.Picture.Bitmap.Canvas.Handle, X+1, Y+1));
    end;

    {Dibujar la imagen terminada de modo que no haya huecos a la izquierda}
    Canvas.Draw(Image1.Left, Image1.Top, Image1.Picture.Bitmap);

    {Deshabilitar temporizador}
    Timer1.Enabled := FALSE;
end;

```

Figura 3-47:  
Degradado  
del mapa de  
bits  
ejecutándose



### SetPixelV

### Windows.Pas

#### Sintaxis

```

SetPixelV(
    DC: HDC;                      {manejador de contexto de dispositivo}

```

```

X: Integer;           {coordenada horizontal del píxel}
Y: Integer;           {coordenada vertical del píxel}
Color: COLORREF       {el nuevo color del píxel}
): BOOL;              {devuelve TRUE o FALSE}

```

**Descripción**

Esta función asigna el color del píxel en las coordenadas especificadas del contexto de dispositivo indicado. Es generalmente más rápida que *SetPixel* porque no tiene que devolver un color. Las coordenadas tienen que estar dentro de las fronteras de la región actual de recorte.

**Parámetros**

*DC*: Manejador del contexto de dispositivo en el cual se asigna el color del nuevo píxel.

*X*: La coordenada horizontal del píxel dentro del contexto de dispositivo, en unidades lógicas.

*Y*: La coordenada vertical del píxel dentro del contexto de dispositivo, en unidades lógicas.

*Color*: Especifica el color del píxel.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetPixel*, *SetPixel*

**Ejemplo**

Vea el Listado 3-31 bajo *LineDDA*.

**SetPolyFillMode****Windows.Pas****Sintaxis**

```

SetPolyFillMode(
    DC: HDC;           {manejador de contexto de dispositivo}
    PolyFillMode: Integer {modo de relleno de polígonos}
): Integer;           {devuelve el modo anterior de relleno de polígonos}

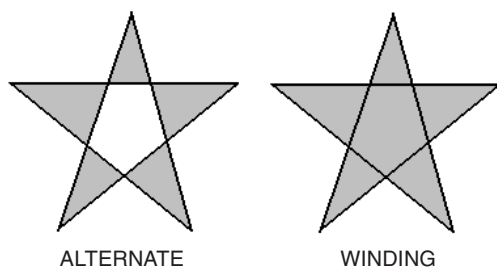
```

**Descripción**

Esta función asigna el modo de relleno de polígonos del contexto de dispositivo especificado. El modo de relleno de polígonos determina cómo serán rellenados los

polígonos y las regiones. Para determinar qué píxeles serán rellenados cuando se utilice el modo *ALTERNATE*, seleccione cualquier píxel del interior del polígono y dibuje una línea imaginaria en la dirección positiva de las X, hasta el infinito. Para cada línea del polígono que intersecte la línea imaginaria, incremente un contador en uno. El píxel será rellenado si este valor es un número impar. Para determinar qué píxeles serán rellenados cuando se utilice el modo *WINDING*, seleccione cualquier píxel en el interior del polígono y dibuje una línea imaginaria en la dirección positiva de las X, hasta el infinito. Para cada línea del polígono que intersecte la línea imaginaria, si la línea del polígono fue dibujada en la dirección positiva de las Y, se incrementa un contador. Si la línea del polígono fue dibujada en la dirección negativa de las Y, se decrementa el contador. El píxel será rellenado si el valor del contador es distinto de cero.

Figura 3-48:  
Resultados de  
modos de  
relleno de  
polígonos



#### Parámetros

*DC*: Manejador del contexto de dispositivo cuyo modo actual de relleno de polígonos será seleccionado.

*PolyFillMode*: Un indicador que especifica el nuevo modo de relleno de polígonos. A este parámetro se le puede asignar un valor de la Tabla 3-37.

#### Valor que devuelve

Si la función tiene éxito, devuelve un valor de la Tabla 3-37 que indica el modo de relleno de polígonos anterior. Si la función falla, devuelve cero.

#### Véase además

*FillPath*, *GetPolyFillMode*, *Polygon*, *PolyPolygon*

#### Ejemplo

Vea el Listado 3-25 bajo *GetPolyFillMode*.

**Tabla 3-37: Valores del parámetro *PolyFillMode* y valores que devuelve *SetPolyFillMode***

Valor	Descripción
ALTERNATE	Rellena el polígono utilizando el método Alternate.
WINDING	Rellena el polígono utilizando el método Winding.

**SetROP2****Windows.Pas****Sintaxis**

```
SetROP2(
    DC: HDC;           {manejador de contexto de dispositivo}
    p2: Integer         {indicador del modo de mezcla del primer plano}
); Integer;           {devuelve el modo anterior de mezcla del primer plano}
```

**Descripción**

Esta función selecciona el modo de mezcla del primer plano para el contexto de dispositivo especificado. El modo de mezcla del primer plano determina cómo se combinará el color de la pluma utilizada en las operaciones de dibujo con el color de los píxeles en el contexto de dispositivo especificado.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo modo de mezcla del primer plano será seleccionado.

*p2*: Indicador que especifica el nuevo modo de mezcla del primer plano. A este parámetro se le puede asignar un valor de la Tabla 3-38.

**Valor que devuelve**

Si la función tiene éxito, devuelve un valor de la Tabla 3-38, que indica el modo de mezcla del primer plano anterior. Si la función falla, devuelve cero.

**Véase además**

*GetROP2*, *LineTo*, *PolyBezier*, *Polyline*, *Rectangle*

**Ejemplo**

Vea el Listado 3-26 bajo *GetROP2*.

**Tabla 3-38: Valores del parámetro p2 y valores que devuelve SetROP2**

Valor	Descripción
R2_BLACK	El píxel de destino siempre será negro.
R2_COPYPEN	Al píxel de destino se le asigna el color de la pluma.
R2_MASKNOTPEN	El píxel de destino es una combinación de los colores comunes a la pantalla y al inverso de la pluma.
R2_MASKPEN	El píxel de destino es una combinación de los colores comunes a la pantalla y a la pluma.
R2_MASKPENNNOT	El píxel de destino es una combinación de los colores comunes a la pluma y al inverso de la pantalla.

Valor	Descripción
R2_MERGENOTPEN	El píxel de destino es una combinación de la pantalla y del inverso de la pluma.
R2_MERGEPEPEN	El píxel de destino es una combinación de la pluma y la pantalla.
R2_MERGEPEPNOT	El píxel de destino es una combinación de la pluma y del inverso de la pantalla.
R2_NOP	El píxel de destino no será modificado.
R2_NOT	El píxel de destino es el inverso de la pantalla.
R2_NOTCOPYPEN	El píxel de destino es el inverso de la pluma.
R2_NOTMASKPEN	El píxel de destino es el inverso de la opción R2_MASKPEN.
R2_NOTMERGEPEPEN	El píxel de destino es el inverso de la opción R2_MERGEPEPEN.
R2_NOTXORPEN	El píxel de destino es el inverso de la opción R2_XORPEN.
R2_WHITE	El píxel de destino siempre será blanco.
R2_XORPEN	El píxel de destino es una combinación de los colores en la pluma o en la pantalla, pero no ambos.

**StrokeAndFillPath****Windows.Pas****Sintaxis**

```
StrokeAndFillPath(
    DC: HDC                                {manejador de contexto de dispositivo}
); BOOL;                                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función cierra las figuras abiertas en la ruta del contexto de dispositivo especificado y traza y rellena la ruta utilizando la pluma y la brocha actualmente seleccionadas. La ruta es rellenada según el modo actual de relleno de polígonos del contexto de dispositivo. Tenga en cuenta que cuando esta función retorne, la ruta será descartada del contexto de dispositivo.

**Parámetros**

*DC*: Manejador del contexto de dispositivo que contiene la ruta a trazar y rellenar.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*BeginPath*, *FillPath*, *SetPolyFillMode*, *StrokePath*

*Ejemplo***Listado 3-45: Trazando y rellenando una ruta simultáneamente**

```

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Comenzar una definición de ruta}
    BeginPath(Canvas.Handle);

    {Dibujar algún texto agradable}
    SetBkMode(Canvas.Handle, TRANSPARENT);
    Canvas.TextOut(10, 10, 'DELPHI ROCKS!');

    {Cerrar la ruta}
    EndPath(Canvas.Handle);

    {Inicializar la pluma y la brocha a utilizar}
    Canvas.Pen.Color := clRed;
    Canvas.Pen.Style := psSolid;
    Canvas.Brush.Color := clBlue;
    Canvas.Brush.Style := bsDiagCross;

    {Trazar y rellenar la ruta}
    StrokeAndFillPath(Canvas.Handle);
end;

```

Figura 3-49:  
La ruta  
trazada y  
rellenada

**StrokePath****Windows.Pas***Sintaxis*

```

StrokePath(
    DC: HDC           {manejador de contexto de dispositivo}
): BOOL;             {devuelve TRUE o FALSE}

```

*Descripción*

Esta función traza la ruta contenida en el contexto de dispositivo especificado con la pluma actualmente seleccionada. Observe que cuando esta función retorne la ruta será desechada del contexto de dispositivo.

*Parámetros*

*DC*: Manejador del contexto de dispositivo que contiene la ruta a trazar.

*Valor que devuelve*

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

*Véase además*

*BeginPath*, *EndPath*, *FillPath*, *ExtCreatePen*, *StrokeAndFillPath*

*Ejemplo***Listado 3-46: Trazando una ruta**

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    {Comenzar definición de ruta}
    BeginPath(Canvas.Handle);

    {Dibujar algún texto agradable}
    SetBkMode(Canvas.Handle, TRANSPARENT);
    Canvas.TextOut(10, 10, 'DELPHI ROCKS!');

    {Cerrar la ruta}
    EndPath(Canvas.Handle);

    {Inicializar la pluma a utilizar para dibujar el borde de la ruta}
    Canvas.Pen.Color := clRed;
    Canvas.Pen.Style := psSolid;

    {Trazar la ruta}
    StrokePath(Canvas.Handle);
end;
```

*Figura 3-50:*  
La ruta  
delineada



**Capítulo 4**

# Funciones de regiones y rutas

Cuando se producen salidas gráficas sobre un contexto de dispositivo, a menudo es necesario confinar la salida a un área más pequeña que el área cliente, o a un área no rectangular. Las máscaras monocromáticas, los *buffers* fuera de pantalla y las operaciones de barrido pueden combinarse para producir los efectos deseados, resultando en un complicado método de salidas gráficas. Alternativamente, el desarrollador puede aprovechar unas herramientas especiales de Windows conocidas como regiones y rutas. Las funciones de regiones y rutas no están encapsuladas en la VCL y casi no están documentadas en la literatura sobre Delphi. Estas funciones pueden ser utilizadas para crear efectos sorprendentes y pueden ofrecer soluciones elegantes que en caso contrario, requerirían la complicada serie de pasos que se sugieren en el capítulo anterior. Este capítulo discute las funciones de rutas y regiones que ofrece el API Win32.

## Regiones y rutas

Inicialmente, una ruta y una región pueden parecer muy similares. Ambas definen una figura. Pueden ser trazadas y rellenadas utilizando las plumas y brochas que el programador determine. Sin embargo, con una inspección más detallada, las diferencias entre rutas y regiones se hacen tangibles.

### Regiones

Una región es una figura poligonal cerrada. No es una figura en el sentido visual; se comporta como una definición de figura, que puede ser creada mediante diversas técnicas. Generalmente, una región se construye mediante funciones específicas que crean una definición de figura en forma de primitiva gráfica, como puede ser un rectángulo o una elipse. Como tal, las regiones tienden a ser más simples en cuanto a forma que las rutas. Sin embargo, las regiones pueden combinarse entre sí de varias maneras para producir figuras más complejas. La función *CombineRgn* ejecuta este servicio usando varias opciones, que representan operaciones booleanas para combinar regiones de diferentes maneras, como se ilustra a continuación.

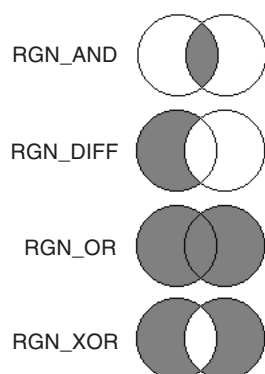


Figura 4-1:  
Métodos de  
combinación  
de regiones de  
CombineRgn

A diferencia de una ruta, una región puede usarse para comprobar una pulsación del ratón (*hit testing*). Dicha comprobación es el acto de determinar dónde está situado el cursor del ratón con respecto a un área determinada, normalmente en respuesta a un clic. Empleando las funciones *PtInRect* o *PtInRegion* combinadas con rectángulos o regiones, el desarrollador puede crear zonas calientes (*hot spots*) de formas muy complejas. Por ejemplo, una aplicación puede definir cincuenta regiones diferentes con la forma de los cincuenta estados de los Estados Unidos. Utilizando esas regiones como zonas calientes en un mapa de los Estados Unidos, un estado en particular puede resaltarse cuando el cursor del ratón entre en su perímetro, o se puede mostrar determinada información cuando se haga clic sobre él. Esto puede llevarse a cabo fácilmente en el evento *OnMouseDown* de un componente *TImage* o del propio formulario, utilizando la función *PtInRegion* para comparar las coordenadas del clic con cada región hasta que se encuentre la región correcta. También, a diferencia de las rutas, una región puede desplazarse a posiciones diferentes de sus coordenadas originales y comparada con otras regiones.

Si se necesita obtener información más detallada sobre una región, una aplicación puede usar la función *GetRegionData* para recuperar los atributos de una región. En particular, una región se define internamente como una serie de rectángulos, ordenados de arriba a abajo y de izquierda a derecha. La función *GetRegionData* puede utilizarse para recuperar el rectángulo individual que define una región, como se ilustra en el siguiente ejemplo.

#### Listado 4-1: Recuperando información acerca de una región

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TheRegion: HRGN;           // almacena la región
  RegionDataSize: DWORD;     // almacena el tamaño de la información de la región
  RegionData: Pointer;       // puntero a la información de la región
  iCount: Integer;           // variable de control del bucle general
  RectPointer: ^TRect;       // puntero utilizado para extraer las coordenadas
                              // del rectángulo
begin
```

```

{Crear una región rectangular redondeada}
TheRegion := CreateRoundRectRgn(10, 10, 110, 110, 30, 30);

{Inicializar la brocha del área de dibujo y dibujar la región}
Canvas.Brush.Color := clRed;
FillRgn(Canvas.Handle, TheRegion, Canvas.Brush.Handle);

{Recuperar el tamaño del buffer requerido para almacenar los datos de la región,
 y reservar la memoria especificada}
RegionDataSize := GetRegionData(TheRegion, 0, nil);
GetMem(RegionData, RegionDataSize);

{Recuperar la información acerca de la región rectangular redondeada}
GetRegionData(TheRegion, RegionDataSize, RegionData);

{Mostrar la información}
with ListBox1.Items do
begin
    {Mostrar la cantidad de rectángulos y el tamaño del rectángulo límite}
    Add('Number of rectangles: ' + IntToStr(TRgnData(RegionData^).rdh.nCount));
    Add('Región bounding rectangle -');
    Add('Left: ' + IntToStr(TRgnData(RegionData^).rdh.rcBound.Left) +
        ' Top: ' + IntToStr(TRgnData(RegionData^).rdh.rcBound.Top) +
        ' Right: ' + IntToStr(TRgnData(RegionData^).rdh.rcBound.Right) +
        ' Bottom: ' + IntToStr(TRgnData(RegionData^).rdh.rcBound.Bottom));
    Add('');

    {Inicializar un puntero a la dirección del buffer que contiene las coordenadas
     del rectángulo que define la región}
    RectPointer := @TRgnData(RegionData^).Buffer;

    {Asignar a la pluma activa un color diferente para que se vean los rectángulos}
    Canvas.Pen.Color := clBlack;

    {Recorrer los rectángulos}
    for iCount := 0 to TRgnData(RegionData^).rdh.nCount-1 do
    begin
        {El puntero RectPointer por definición convertirá los tipos de los valores
         en el buffer al tipo TRect, lo que permitirá a la aplicación extraer los
         campos necesarios}
        Add('Rect: ' + IntToStr(iCount) +
            ' - L: ' + IntToStr(RectPointer^.Left) +
            ', T: ' + IntToStr(RectPointer^.Top) +
            ', R: ' + IntToStr(RectPointer^.Right) +
            ', B: ' + IntToStr(RectPointer^.Bottom));

        {Dibujar este rectángulo específico sobre la región}
        Canvas.Rectangle(RectPointer^.Left, RectPointer^.Top, RectPointer^.Right,
            RectPointer^.Bottom);

        {Como RectPointer es un puntero a TRect, cuando se incremente su valor
         pasará a apuntar al próximo rectángulo de la serie}
        Inc(RectPointer);
    end;
end;

```

```

    {Eliminar la región y liberar la memoria}
    FreeMem(RegionData);
    DeleteObject(TheRegion);
end;

```

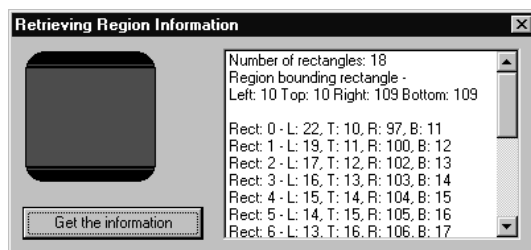


Figura 4-2: La información sobre la región

## Rutas

Al igual que una región, una ruta es una definición de figura. Sin embargo, una ruta no necesita formar una figura poligonal cerrada. Una ruta puede ser cualquier cosa, desde un rectángulo hasta una serie compleja de líneas y curvas de Bézier. Una ruta se crea encerrando una serie de llamadas a funciones de dibujo del GDI en lo que se conoce como una *definición de ruta* (*path bracket*). Una definición de ruta es una sección de código que comienza con una llamada a la función *BeginPath* y termina con una llamada a la función *EndPath*. Las funciones de dibujo específicas que sean llamadas entre estas dos funciones no producirán salida gráfica en la pantalla; en lugar de ello, definirán la forma de la ruta. Generalmente las rutas son mucho más complejas que las regiones en cuanto a forma. Consulte la función *BeginPath* para ver una lista de las funciones de dibujo que pueden utilizarse en una definición de ruta.

A diferencia de las regiones, de las cuales pueden crearse múltiples instancias, una ruta se asocia con el contexto de dispositivo en el cual ha sido definida. En un contexto de dispositivo cualquiera puede existir solamente una ruta a la vez; cuando se inicia la definición de otra ruta o se destruye el contexto de dispositivo, cualquier ruta existente es destruida. Sin embargo, una ruta puede convertirse en una región usando la función *PathToRegion*. Esto permite al desarrollador crear regiones de forma extremadamente compleja. Los puntos que definen la ruta pueden ser recuperados llamando a la función *GetPath*. Esta función devuelve un *array* de registros *TPoint* que contienen las coordenadas de los puntos que definen la región, en unidades lógicas. Un uso común de esta función puede verse en algoritmos que ajustan un texto a lo largo de una ruta o una figura, como por ejemplo una curva.

## Efectos especiales

Quizás el uso más común de una región o una ruta sea definir una *región de recorte*. Cuando una región de recorte es definida y seleccionada dentro de un contexto de dispositivo, cualquier salida gráfica al contexto de dispositivo se confina dentro de los límites de la región. Cualquier salida que quede fuera de la región será descartada o

“recortada”. Combinando la funcionalidad de las regiones con las rutas y usando el resultado como una región de recorte se pueden obtener efectos especiales asombrosos. Por ejemplo, puede crearse una ruta utilizando la función *TextOut* para definir una palabra o una frase. Esta ruta puede luego convertirse en una región y utilizarse en combinación con funciones de manejo de mapas de bits y algunas técnicas de animación para producir una pantalla de presentación sensacional. El siguiente ejemplo demuestra esta técnica. Observe que el mapa de bits utilizado dentro del texto se mueve de derecha a izquierda.

#### Listado 4-2: Atractivos efectos especiales producidos con regiones y rutas

```

var
    Form1: TForm1;
    Offset: Integer;           // desplazamiento del mapa de bits
    Buffer, TileBitmap: TBitmap; // mapas de bits fuera de pantalla y de textura

implementation

{$R *.DFM}

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Dibuja un marco del efecto}
    DrawEffect;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Inicializar el desplazamiento}
    Offset := 0;

    {Crear un buffer fuera de pantalla del tamaño del área cliente del formulario}
    Buffer := TBitmap.Create;
    Buffer.Width := ClientWidth;
    Buffer.Height := ClientHeight;

    {Crear y cargar el mapa de bits de la textura utilizada en las letras}
    TileBitmap := TBitmap.Create;
    TileBitmap.LoadFromFile(ExtractFilePath(ParamStr(0)) + 'Tile.bmp');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Liberar los mapas de bits de la textura y el buffer fuera de pantalla}
    Buffer.Free;
    TileBitmap.Free;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    {Incrementar el desplazamiento}
    Inc(Offset);

```

```

    {Si el desplazamiento supera el ancho del mapa de bits de la textura
    (64 píxeles), reinicializarlo}
    if Offset > 63 then
        Offset := 0;

    {Dibujar marco con el efecto}
    DrawEffect;
end;

procedure TForm1.DrawEffect;
var
    iCount: Integer;          // contador de bucle
    ClipRgn: HRGN;            // almacena la región
begin
    {Comenzar definición de ruta}
    BeginPath(Canvas.Handle);

    {Mostrar algún texto, definiendo la ruta como el interior del texto}
    SetBkMode(Canvas.Handle, TRANSPARENT);
    TextOut(Canvas.Handle, 10, 60, 'DELPHI', 6);

    {Terminar definición de ruta}
    EndPath(Canvas.Handle);

    {Convertir la ruta en región y seleccionar esta región como el área de recorte
    del buffer fuera de pantalla}
    ClipRgn := PathToRegion(Canvas.Handle);
    SelectClipRgn(Buffer.Canvas.Handle, ClipRgn);

    {Dibujar el mapa de bits de la textura dentro del área definida por la región, la
    imagen quedará recortada al interior de las letras}
    for iCount := 0 to 4 do
        Buffer.Canvas.Draw(iCount*64-Offset, 60, TileBitmap);

    {Eliminar la región de recorte del buffer fuera de pantalla}
    SelectClipRgn(Buffer.Canvas.Handle, 0);

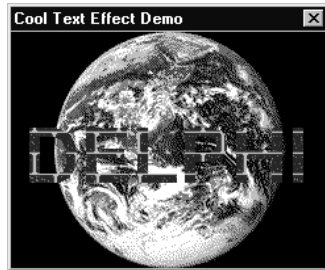
    {Reinicializar la región de recorte del buffer fuera de pantalla, ahora
    definiendo la región de recorte como el área exterior de las letras}
    ExtSelectClipRgn(Buffer.Canvas.Handle, ClipRgn, RGN_DIFF);

    {Dibujar la imagen de la Tierra en el buffer fuera de pantalla. Las letras
    previamente dibujadas no serán "tapadas" por el mapa de bits, debido a que
    están protegidas por la región de recorte actual}
    Buffer.Canvas.Draw(0, 0, Image1.Picture.Bitmap);

    {Dibujar el buffer fuera de pantalla sobre el formulario. Esto elimina el
    parpadeo; es una técnica de animación conocida como doble bufferización}
    Canvas.Draw(0, 0, Buffer);
end;

```

Figura 4-3:  
Una agradable  
pantalla de  
presentación



## Funciones de regiones y rutas

Este capítulo describe las siguientes funciones de regiones y rutas:

**Tabla 4-1: Funciones de regiones y rutas**

Función	Descripción
AbortPath	Descarta una ruta y cierra una definición de ruta abierta.
BeginPath	Comienza una definición de ruta.
CloseFigure	Cierra una figura abierta en una definición de ruta.
CombineRgn	Combina dos regiones usando una operación booleana.
CopyRect	Copia las coordenadas de un rectángulo en otro.
CreateEllipticRgn	Crea una región elíptica.
CreateEllipticRgnIndirect	Crea una región elíptica basada en propiedades definidas en un registro de datos.
CreatePolygonRgn	Crea una región poligonal.
CreatePolyPolygonRgn	Crea una región consistente en múltiples polígonos.
CreateRectRgn	Crea una región rectangular.
CreateRectRgnIndirect	Crea una región rectangular basada en propiedades definidas en un registro de datos.
CreateRoundRectRgn	Crea una región rectangular redondeada.
EndPath	Termina una definición de ruta.
EqualRect	Determina si las coordenadas de dos rectángulos son iguales.
EqualRgn	Determina si el tamaño y la forma de dos regiones son iguales.
ExcludeClipRect	Crea una nueva región de recorte, excluyendo la región especificada.
ExtCreateRegion	Transforma una región existente.
ExtSelectClipRgn	Selecciona una región de recorte, combinándola con la región de recorte existente mediante operaciones booleanas.
FlattenPath	Convierte las curvas en una ruta en segmentos de líneas.
GetClipBox	Recupera el rectángulo límite de la región de recorte.
GetClipRgn	Recupera un manejador de la región de recorte actual.

Función	Descripción
GetPath	Recupera los puntos que definen una ruta.
GetRegionData	Recupera información sobre una región.
GetRgnBox	Recupera el rectángulo límite de una región.
InflateRect	Modifica el tamaño de un rectángulo.
IntersectRect	Crea un nuevo rectángulo a partir de la intersección de dos rectángulos.
InvertRect	Invierte los colores de los píxeles dentro del área definida por un rectángulo.
InvertRgn	Invierte los colores de los píxeles dentro del área definida por una región.
IsRectEmpty	Determina si un rectángulo está vacío.
OffsetClipRgn	Mueve un área de recorte.
OffsetRect	Mueve un rectángulo.
OffsetRgn	Mueve una región.
PathToRegion	Convierte una ruta en una región.
PtInRect	Determina si una coordenada específica está situada dentro de un rectángulo.
PtInRegion	Determina si una coordenada específica está situada dentro de una región.
PtVisible	Determina si una coordenada específica está situada dentro de la región de recorte.
RectInRegion	Determina si un rectángulo está situado dentro de una región.
RectVisible	Determina si un rectángulo está situado dentro de la región de recorte.
SelectClipPath	Selecciona la ruta actual como la región de recorte.
SelectClipRgn	Selecciona una región como la región de recorte.
SetRect	Inicializa un rectángulo.
SetRectEmpty	Vacía un rectángulo.
SetRectRgn	Convierte una región en una región rectangular.
SetWindowRgn	Asigna la región de una ventana.
SubtractRect	Calcula la diferencia entre dos rectángulos.
UnionRect	Crea un rectángulo a partir de la suma de dos rectángulos.
WidenPath	Redefine la forma de una ruta con respecto a la pluma actual.

**AbortPath****Windows.Pas****Sintaxis**

```
AbortPath(
    DC: HDC                {manejador de contexto de dispositivo})
```

); BOOL; {devuelve TRUE o FALSE}

### Descripción

Esta función descarta cualquier ruta definida en el contexto de dispositivo identificado por el parámetro *DC*. Si la función es llamada dentro de una definición de ruta abierta, ésta es cerrada y la ruta es descartada.

### Parámetros

*DC*: Manejador del contexto de dispositivo que contiene la ruta que será eliminada.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*BeginPath*, *CloseFigure*, *EndPath*

### Ejemplo

Vea el Listado 4-3 bajo *CloseFigure*.

## **BeginPath**      **Windows.Pas**

### Sintaxis

```
BeginPath(
    DC: HDC           {manejador de contexto de dispositivo}
); BOOL;             {devuelve TRUE o FALSE}
```

### Descripción

Esta función abre una definición de ruta en el contexto de dispositivo especificado. Cualquier ruta que exista previamente en el contexto de dispositivo especificado es descartada. Utilice la función *EndPath* para cerrar una definición de ruta abierta. Una vez que una definición ha comenzado, ciertas funciones de dibujo aplicadas en el contexto de dispositivo especificado serán convertidas a información de la ruta y no mostrarán ninguna salida visible. Una vez que una definición de ruta es cerrada, ésta es asociada con el contexto de dispositivo especificado. La ruta puede ser convertida en una región mediante una llamada a la función *PathToRegion*.

Bajo Windows NT, las siguientes funciones pueden ser usadas dentro de una definición de ruta: *AngleArc*, *Arc*, *ArcTo*, *Chord*, *CloseFigure*, *Ellipse*, *ExtTextOut*, *LineTo*, *MoveToEx*, *Pie*, *PolyBezier*, *PolyBezierTo*, *PolyDraw*, *Polygon*, *Polyline*, *PolylineTo*, *PolyPolygon*, *PolyPolyline*, *Rectangle*, *RoundRect* y *TextOut*.

Bajo Windows 95/98, las siguientes funciones pueden ser usadas dentro de una definición de ruta: *CloseFigure*, *ExtTextOut*, *LineTo*, *MoveToEx*, *PolyBezier*, *PolyBezierTo*, *Polygon*, *Polyline*, *PolylineTo*, *PolyPolygon*, *PolyPolyline* y *TextOut*.

#### Parámetros

*DC*: Manejador del contexto de dispositivo en el que ciertas funciones de dibujo serán convertidas en información de la ruta.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CloseFigure*, *EndPath*, *ExtTextOut*, *FillPath*, *LineTo*, *MoveToEx*, *PathToRegion*, *PolyBezier*, *PolyBezierTo*, *Polygon*, *Polyline*, *PolylineTo*, *PolyPolygon*, *PolyPolyline*, *SelectClipPath*, *StrokeAndFillPath*, *StrokePath*, *TextOut*, *WidenPath*

#### Ejemplo

Vea el Listado 4-19 bajo *SelectClipPath*, y otros ejemplos a lo largo de este capítulo.

### **CloseFigure**      **Windows.Pas**

#### Sintaxis

```
CloseFigure(
    DC: HDC                      {manejador de contexto de dispositivo}
): BOOL;                        {devuelve TRUE o FALSE}
```

#### Descripción

Esta función cierra la figura creada en una definición de ruta en el contexto de dispositivo especificado. La figura es cerrada ejecutando la operación *LineTo* desde el punto actual hasta el punto especificado en la llamada más reciente a la función *MoveToEx*. Las líneas serán conectadas usando el estilo de unión de líneas identificado por la pluma geométrica que se encuentre seleccionada. Si la función *LineTo* es llamada explícitamente para cerrar la figura antes de llamar a la función *CloseFigure*, el estilo de punto final de la pluma geométrica que se encuentre seleccionada es usado para dibujar los finales de las líneas. Esta función es útil sólo cuando es llamada dentro de una definición de ruta abierta. Una figura en una definición de ruta está abierta a menos que sea explícitamente cerrada mediante una llamada a esta función. Después de que esta función sea llamada, cualquier otra función de dibujo que sea usada en la ruta comenzará una nueva figura.

**Parámetros**

*DC*: Especifica el contexto de dispositivo que contiene la ruta cuya figura actual será cerrada.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*BeginPath*, *EndPath*, *ExtCreatePen*, *LineTo*, *MoveToEx*

**Ejemplo****Listado 4-3: Cerrando una figura abierta en una definición de ruta**

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Comenzar una definición de ruta. Todas las subsiguientes llamadas a funciones
      de dibujo definirán una ruta y no producirán salida visible}
    BeginPath(Canvas.Handle);

    {Comenzar a dibujar la ruta}
    Canvas.MoveTo(65, 15);
    Canvas.LineTo(25, 234);
    Canvas.MoveTo(78, 111);
    Canvas.LineTo(98, 79);

    {Por cualquier razón la ruta puede ser abandonada...}
    AbortPath(Canvas.Handle);

    {La ruta anterior fue cerrada y abandonada; comenzar una nueva definición}
    BeginPath(Canvas.Handle);

    {Dibujar tres líneas dentro de la ruta}
    Canvas.MoveTo(25, 10);
    Canvas.LineTo(125, 10);
    Canvas.LineTo(125, 110);
    Canvas.LineTo(25, 110);

    {Cerrar la figura actual. Esto crea una ruta cuadrada}
    CloseFigure(Canvas.Handle);

    {Terminar definición de ruta, que será asociada con el contexto de dispositivo}
    EndPath(Canvas.Handle);

    {Inicializar la pluma y la brocha del contexto de dispositivo}
    Canvas.Pen.Width := 3;
    Canvas.Pen.Color := clRed;
    Canvas.Brush.Color := clLime;

    {Trazar la ruta sobre el contexto de dispositivo}

```

```
StrokeAndFillPath(Canvas.Handle);
end;
```

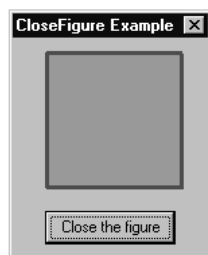


Figura 4-4: La figura cerrada

## CombineRgn Windows.Pas

### Sintaxis

```
CombineRgn(
  p1: HRGN;           {manejador de la región combinada}
  p2: HRGN;           {manejador de la primera región}
  p3: HRGN;           {manejador de la segunda región}
  p4: Integer         {opción de combinación de región}
): Integer;           {opción de tipo de la región combinada}
```

### Descripción

Esta función combina las regiones identificadas por los parámetros *p2* y *p3* de acuerdo a la operación indicada en el parámetro *p4*. La región identificada por el parámetro *p1* es inicializada para apuntar a la región resultante. Las regiones identificadas por los parámetros *p1*, *p2* y *p3* no tienen que ser únicas (por ejemplo, la región resultado identificada por el parámetro *p1* puede coincidir con la misma región identificada por alguno de los parámetros *p2* ó *p3*). Cuando la región combinada deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

### Parámetros

- p1*: Manejador de la región que recibirá la región combinada. Este parámetro tiene que especificar un manejador de una región existente.
- p2*: Manejador de la primera región que será combinada.
- p3*: Manejador de la segunda región que será combinada.
- p4*: Opción que indica el modo de combinación de las regiones identificadas por los parámetros *p2* y *p3*. Este parámetro puede contener un valor de la Tabla 4-2.

### Valor que devuelve

Esta función devuelve un resultado que indica el tipo de región creada o una condición de error y puede ser un valor de la Tabla 4-3.

Véase además

*CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, DeleteObject*

Ejemplo

#### Listado 4-4: Combinando dos regiones para crear un efecto especial

```

var
    Form1: TForm1;
    BinocularRgn: HRGN;    // manejador de la región combinada

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
    Circle1, Circle2: HRGN; // almacena dos regiones circulares
begin
    {El manejador de la región combinada tiene que identificar una región ya
    existente, por eso creamos una región "fantasma"}
    BinocularRgn := CreateEllipticRgnIndirect(BoundsRect);

    {Crear dos regiones circulares. La primera ocupa 3/4 del lado izquierdo del área
    cubierta por Image1, y la segunda ocupa 3/4 del lado derecho de ese área}
    Circle1 := CreateEllipticRgn(Image1.Left, Image1.Top,
                                Image1.Left + MulDiv(Image1.Width,3,4),
                                Image1.Top + Image1.Height);
    Circle2 := CreateEllipticRgn(Image1.Left + (Image1.Width div 4),
                                Image1.Top, Image1.Left + Image1.Width,
                                Image1.Top + Image1.Height);

    {Combinar las dos regiones, creando una región que recuerda a unos binoculares}
    CombineRgn(BinocularRgn, Circle1, Circle2, RGN_OR);

    {Eliminar las dos regiones circulares porque ya no son necesarias}
    DeleteObject(Circle1);
    DeleteObject(Circle2);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    ClipRect: TRect; // almacena las coordenadas de la región de recorte actual
begin
    {Seleccionar la región combinada como región de recorte}
    SelectClipRgn(Canvas.Handle, BinocularRgn);

    {Dibujar el contenido de la imagen (que es invisible) sobre la superficie del
    formulario. Será recortado según la región de recorte actual, resultando en lo
    que parece la imagen de un barco vista a través de unos binoculares}
    Canvas.Draw(Image1.Left, Image1.Top, Image1.Picture.Bitmap);
  
```

```

{Dibujar el perímetro de la región en rojo para que destaque}
Canvas.Brush.Color := clRed;
FrameRgn(Canvas.Handle, BinocularRgn, Canvas.Brush.Handle, 2, 2);

{Recuperar el rectángulo más pequeño que podrá circunscribir la zona actualmente
visible del contexto de dispositivo}
GetClipBox(Canvas.Handle, ClipRect);

{Eliminar la región de recorte, de manera que el dibujo pueda ser ejecutado
sobre la superficie del contexto de dispositivo}
SelectClipRgn(Canvas.Handle, 0);

{Dibujar el borde de la región de recorte seleccionada anteriormente}
Canvas.Brush.Style := bsClear;
Canvas.Pen.Color := clBlack;
Rectangle(Canvas.Handle, ClipRect.Left, ClipRect.Top, ClipRect.Right,
ClipRect.Bottom);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
{Eliminar la región combinada}
DeleteObject(BinocularRgn);
end;

```

Figura 4-5: La  
región  
combinada  
usada como  
región de  
recorte

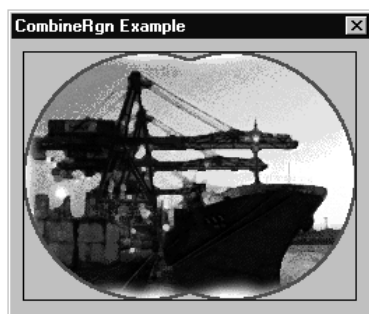


Tabla 4-2: Valores del parámetro p4 de CombineRgn

Valor	Descripción
RGN_AND	La región resultante es la intersección de las dos regiones especificadas.
RGN_COPY	La región resultante es una copia de la región identificada por el parámetro p2.
RGN_DIFF	La región resultante es el área de la región identificada por el parámetro p2 que no forma parte de la región identificada por el parámetro p3.
RGN_OR	La región resultante es la unión de las dos regiones especificadas.
RGN_XOR	La región resultante es la unión de las dos áreas especificadas, excluyendo las áreas comunes.

Tabla 4-3: Valores que devuelve CombineRgn

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ha ocurrido un error y que no ha sido creada ninguna región.

**CopyRect**      **Windows.Pas***Sintaxis*

```
CopyRect(
  var lprcDst: TRect;      {puntero al rectángulo de destino}
  const lprcSrc: TRect    {puntero al rectángulo fuente}
): BOOL;                  {devuelve TRUE o FALSE}
```

*Descripción*

Esta función copia las coordenadas del rectángulo al que apunta el parámetro *lprcSrc* en las coordenadas del rectángulo al que apunta el parámetro *lprcDst*.

*Parámetros*

*lprcDst*: Puntero a un registro *TRect* que recibirá las coordenadas del rectángulo al que apunta el parámetro *lprcSrc*.

*lprcSrc*: Puntero a un registro *TRect* que contiene las coordenadas que serán copiadas al rectángulo al que apunta el parámetro *lprcDst*.

*Valor que devuelve*

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

*Véase además*

*IsRectEmpty*, *SetRect*, *SetRectEmpty*, *SetRectRgn*

*Ejemplo*

Vea el Listado 4-16 bajo *OffsetRect*.

**CreateEllipticRgn****Windows.Pas****Sintaxis**

```

CreateEllipticRgn(
  p1: Integer;      {la coordenada horizontal de la esquina superior izquierda
                    del rectángulo límite}
  p2: Integer;      {la coordenada vertical de la esquina superior izquierda
                    del rectángulo límite}
  p3: Integer;      {la coordenada horizontal de la esquina inferior derecha
                    del rectángulo límite}
  p4: Integer;      {la coordenada vertical de la esquina inferior derecha
                    del rectángulo límite}
): HRGN;           {devuelve un manejador de la región}

```

**Descripción**

Esta función crea una región elíptica. Las coordenadas especificadas representan el rectángulo más pequeño que puede ser trazado alrededor de la elipse resultante. Cuando la región deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

**Parámetros**

*p1*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo límite de la elipse, en unidades lógicas.

*p2*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo límite de la elipse, en unidades lógicas.

*p3*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo límite de la elipse, en unidades lógicas.

*p4*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo límite de la elipse, en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de la región elíptica; en caso contrario, devuelve cero.

**Véase además**

*CreateEllipticRgnIndirect*, *DeleteObject*

**Ejemplo**

Vea el Listado 4-4 bajo *CombineRgn*.

**CreateEllipticRgnIndirect****Windows.Pas****Sintaxis**

```
CreateEllipticRgnIndirect(
  const p1: TRect      {puntero a las coordenadas del rectángulo}
): HRGN;              {devuelve un manejador de la región}
```

**Descripción**

Esta función crea una región elíptica basada en las coordenadas del rectángulo al que apunta el parámetro *p1*. Las coordenadas especificadas representan el rectángulo más pequeño que puede ser dibujado alrededor de la elipse resultante. Cuando la región deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

**Parámetros**

*p1*: Puntero a un registro *TRect* que contiene las coordenadas del rectángulo más pequeño que puede ser dibujado alrededor de la elipse resultante, en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de una región elíptica; en caso contrario, devuelve cero.

**Véase además**

*CreateEllipticRgn*, *DeleteObject*

**Ejemplo****Listado 4-5: Creando dinámicamente una región elíptica basada en el tamaño del formulario**

```
var
  Form1: TForm1;
  TheRegion: HRGN;      // almacena la región elíptica

implementation

{$R *.DFM}

procedure TForm1.FormPaint(Sender: TObject);
begin
  {Borrar la imagen actual en el formulario}
  Canvas.Brush.Color := clBtnFace;
  Canvas.FillRect(BoundsRect);

  {Trazar la región elíptica en color rojo}
  Canvas.Brush.Color := clRed;
  FrameRgn(Canvas.Handle, TheRegion, Canvas.Brush.Handle, 2, 2);
end;
```

```

procedure TForm1.FormResize(Sender: TObject);
begin
    {Eliminar la región actual, si existe}
    if TheRegion <> 0 then
        DeleteObject(TheRegion);

    {Crear una nueva región elíptica basada en las fronteras del área cliente}
    TheRegion := CreateEllipticRgnIndirect(ClientRect);

    {Redibujar el formulario}
    Repaint;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Eliminar la región elíptica}
    DeleteObject(TheRegion);
end;

```

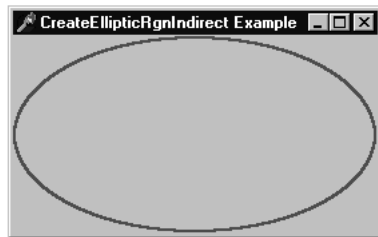


Figura 4-6: La región elíptica creada

## CreatePolygonRgn

## Windows.Pas

### Sintaxis

```

CreatePolygonRgn(
    const Points;           {array de puntos}
    Count: Integer;         {cantidad de puntos en el array}
    FillMode: Integer;      {opción de modo de relleno}
): HRGN;                  {devuelve un manejador de región}

```

### Descripción

Esta función crea una región poligonal con vértices en las coordenadas almacenadas en el *array* de vértices al que apunta el parámetro *Points*. Cuando la región deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

### Parámetros

*Points*: Puntero a un *array* de registros de tipo *TPoint* que contiene las coordenadas de los vértices del polígono, en unidades del dispositivo. Se asume que el polígono es cerrado y cada vértice puede ser especificado una sola vez.

*Count*: Especifica la cantidad de elementos de tipo *TPoint* que contiene el *array* al que apunta el parámetro *Points*.

*FillMode*: Opción que indica el modo de relleno a utilizar para determinar qué píxeles son incluidos en la región. Si este parámetro toma el valor *ALTERNATE*, la región se rellena entre los lados impares y pares del polígono especificado. Si el valor es *WINDING*, se rellenará cualquier parte de la región con un valor *WINDING* distinto de cero. Consulte la función *SetPolyFillMode* para más información sobre el significado de estas opciones.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la región poligonal; en caso contrario, devuelve cero.

#### Véase además

*CreatePolyPolygonRgn*, *DeleteObject*, *SetPolyFillMode*

#### Ejemplo

##### Listado 4-6: Creando una región con forma de estrella

```
var
  Form1: TForm1;
  PolygonRgn, ScaledRgn: HRGN; // almacena la región original y la escalada

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  Vertices: array[0..9] of TPoint; // vértices de la región poligonal
  RegionData: Pointer;             // puntero a datos de la región
  RgnDataSize: DWORD;              // tamaño de los datos de la región
  Transform: TXForm;                // matriz de transformación de escala
begin
  {Especificar un polígono en forma de estrella}
  Vertices[0] := Point(120, 5);
  Vertices[1] := Point(140, 70);
  Vertices[2] := Point(210, 70);
  Vertices[3] := Point(150, 100);
  Vertices[4] := Point(180, 175);
  Vertices[5] := Point(120, 120);
  Vertices[6] := Point(60, 175);
  Vertices[7] := Point(90, 100);
  Vertices[8] := Point(30, 70);
  Vertices[9] := Point(100, 70);

  {Crear una región poligonal en forma de estrella}
  PolygonRgn := CreatePolyPolygonRgn(Vertices, 10, WINDING);

  {Recuperar el tamaño de los datos de la región}
```

```

RgnDataSize := GetRegionData(PolygonRgn, 0, nil);

{Reservar memoria suficiente para almacenar los datos de la región}
GetMem(RegionData, RgnDataSize);

{Recuperar los datos de la región}
GetRegionData(PolygonRgn, RgnDataSize, RegionData);

{Inicializar una matriz de transformación para indicar un ligero aumento de
 tamaño y una traslación}
with Transform do
begin
    eM11 := 1.35;
    eM12 := 0;
    eM21 := 0;
    eM22 := 1.35;
    eDx  := -42;
    eDy  := -35;
end;

{Crear una nueva región escalada a partir de la original}
ScaledRgn := ExtCreateRegion(@Transform, RgnDataSize, TRgnData(RegionData^));

{Liberar los datos de la región porque ya no son necesarios}
FreeMem(RegionData, RgnDataSize);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    TempRgn: HRGN;    // manejador de región
begin
    {Seleccionar la región en forma de estrella escalada como la región de recorte}
    SelectClipRgn(Canvas.Handle, ScaledRgn);

    {Dibujar la imagen de la ciudad en el formulario. Será recortada por las
     fronteras de la región en forma de estrella}
    Canvas.Draw(0, 0, Image1.Picture.Bitmap);

    {Aún si sabemos explícitamente cuál es la región de recorte, la podemos recuperar
     del contexto de dispositivo utilizando la función GetClipRgn. Esta función
     requiere que se le pase un manejador de región existente, por lo que usaremos
     la región en forma de estrella original. Con esta llamada se obtendrá la región
     actual de recorte, que es la región escalada}
    TempRgn := PolygonRgn;
    GetClipRgn(Canvas.Handle, TempRgn);

    {Dibujar las esquinas de la región para destacarla}
    Canvas.Brush.Color := clRed;
    FrameRgn(Canvas.Handle, TempRgn, Canvas.Brush.Handle, 2, 2);
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin

```

```

{Seleccionar la región en forma de estrella escalada como la región de recorte}
SelectClipRgn(Canvas.Handle, ScaledRgn);

{Indicar si el área donde se ha hecho clic es visible dentro de la región de
recorte (la región en forma de estrella escalada)}
if PtVisible(Canvas.Handle, X, Y) then
  Caption := 'CreatePolygonRgn Example - Visible'
else
  Caption := 'CreatePolygonRgn Example - Invisible';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {Liberar todos los recursos asociados con las dos regiones}
  DeleteObject(PolygonRgn);
  DeleteObject(ScaledRgn);
end;

```

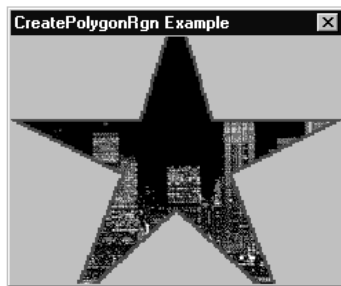


Figura 4-7: La región en forma de estrella

## CreatePolyPolygonRgn Windows.Pas

### Sintaxis

```

CreatePolyPolygonRgn(
  const pPtStructs;           {array de puntos}
  const pIntArray;           {array de cantidades de vértices}
  p3: Integer;                {cantidad de entradas en el array de cantidad de vértices}
  p4: Integer                 {opción de modo de relleno}
): HRGN;                     {devuelve un manejador de región}

```

### Descripción

Esta función crea una región definida por múltiples polígonos. Los vértices de cada polígono son especificados consecutivamente en el *array* de registros *TPoint* al que apunta el parámetro *pPtStructs*. Cada entrada en el *array* al que apunta el parámetro *pIntArray* indica la cantidad de puntos en el *array* de registros *TPoint* que definen los vértices de cada polígono. Los polígonos definidos por este *array* pueden solaparse.

Cuando la región deje de ser necesaria, debe ser eliminada llamando a la función *DeleteObject*.

#### Parámetros

*pPtStructs*: Puntero a un *array* de registros *TPoint* que describen los vértices de cada polígono, en unidades del dispositivo. Cada polígono es descrito consecutivamente y se asume que es cerrado. Cada vértice puede ser especificado una sola vez.

*pIntArray*: Puntero a un *array* de enteros. Cada entero especifica la cantidad de puntos del *array* al que apunta el parámetro *pPtStructs* que definen cada polígono.

*p3*: Especifica la cantidad de entradas en el *array* al que apunta el parámetro *pIntArray*.

*p4*: Opción que indica el modo de relleno a utilizar para determinar qué píxeles son incluidos en la región. Si este parámetro toma el valor *ALTERNATE*, la región se rellena entre los lados impares y pares del polígono especificado. Si el valor es *WINDING*, se rellena cualquier parte de la región con un valor *WINDING* distinto de cero. Consulte la función *SetPolyFillMode* para más información sobre el significado de estas opciones.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la región poligonal; en caso contrario, devuelve cero.

#### Véase además

*CreatePolygonRgn*, *DeleteObject*, *SetPolyFillMode*

#### Ejemplo

##### Listado 4-7: Creando una región de polígonos múltiples

```
var
  Form1: TForm1;
  HotSpotRgn: HRGN;    // almacena la región de polígonos múltiples

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  PolyPoints: array[0..11] of TPoint;    // almacena los puntos de los polígonos
  VertexCounts: array[0..1] of Integer;  // almacena la cantidad de vértices
begin
  {Definir un polígono en la región}
  PolyPoints[0] := Point(68, 80);   PolyPoints[1] := Point(76, 72);
  PolyPoints[2] := Point(87, 80);   PolyPoints[3] := Point(86, 96);
  PolyPoints[4] := Point(100, 96);  PolyPoints[5] := Point(100, 160);
  PolyPoints[6] := Point(68, 160);

  {Definir otro polígono en la región}
```

```

PolyPoints[7] := Point(173, 53); PolyPoints[8] := Point(184, 66);
PolyPoints[9] := Point(184, 146); PolyPoints[10] := Point(160, 146);
PolyPoints[11] := Point(160, 66);

{Indicar que el primer polígono consta de 7 puntos y el segundo de 5}
VertexCounts[0] := 7;
VertexCounts[1] := 5;
{Crear la región de polígonos múltiples}
HotSpotRgn := CreatePolyPolygonRgn(PolyPoints, VertexCounts, 2, WINDING);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Invertir el área definida por la región de polígonos múltiples}
    InvertRgn(Canvas.Handle, HotSpotRgn);
end;

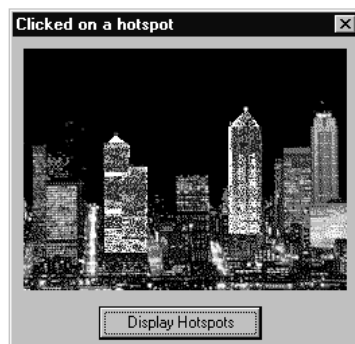
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
var
    TranslatedPt: TPoint;    // almacena una coordenada específica del formulario
begin
    {Como la región está definida en coordenadas lógicas relativas al formulario,
    Las coordenadas del ratón deben ser traducidas adecuadamente}
    TranslatedPt := Image1.ClientToScreen(Point(X,Y));
    TranslatedPt := Form1.ScreenToClient(TranslatedPt);

    {Indicar si el punto está dentro de la zona caliente definida por la región de
    polígonos múltiples}
    if PtInRegion(HotSpotRgn, TranslatedPt.X, TranslatedPt.Y) then
        Caption := 'Clicked on a hotspot'
    else
        Caption := 'No hot spot clicked';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Eliminar la región}
    DeleteObject(HotSpotRgn);
end;

```

Figura 4-8:  
Usando una  
región de  
polígonos  
múltiples  
como “zona  
caliente”



**CreateRectRgn    Windows.Pas****Sintaxis**

```
CreateRectRgn(
  p1: Integer;           {la coordenada horizontal superior izquierda}
  p2: Integer;           {la coordenada vertical superior izquierda}
  p3: Integer;           {la coordenada horizontal inferior derecha}
  p4: Integer;           {la coordenada vertical inferior derecha}
): HRGN;                {devuelve el manejador de la región}
```

**Descripción**

Esta función crea una región rectangular basada en las coordenadas especificadas. Cuando la región deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

**Parámetros**

*p1*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo, en unidades del dispositivo.

*p2*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo, en unidades del dispositivo.

*p3*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo, en unidades del dispositivo.

*p4*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo, en unidades del dispositivo.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de la región; en caso contrario, devuelve cero.

**Véase además**

*CreateRectRgnIndirect*, *CreateRoundRectRgn*, *DeleteObject*

**Ejemplo****Listado 4-8: Creando una región rectangular**

```
procedure TForm1.Button1Click(Sender: TObject);
var
  RegionHandle: HRGN;    // almacena la región rectangular
begin
  {Inicializar la brocha del área de dibujo}
  Canvas.Brush.Style := bsCross;
  Canvas.Brush.Color := clRed;

  {Crear la región rectangular}
  RegionHandle := CreateRectRgn(10, 40, 175, 175);
```

```

{Dibujar la región}
FillRgn(Canvas.Handle, RegionHandle, Canvas.Brush.Handle);

{Eliminar la región}
DeleteObject(RegionHandle);
end;

```

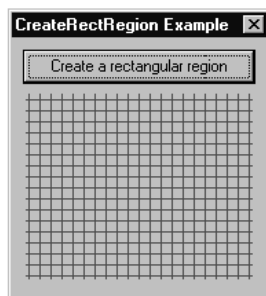


Figura 4-9: La región rectangular

## CreateRectRgnIndirect Windows.Pas

### Sintaxis

```

CreateRectRgnIndirect(
    const p1: TRect          {coordenadas de la región rectangular}
): HRGN;                   {devuelve un manejador de región}

```

### Descripción

Esta función crea una región rectangular basada en las coordenadas del rectángulo identificado por el parámetro *p1*. Cuando la región deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

### Parámetros

*p1*: Un registro *TRect* que contiene las coordenadas del rectángulo que define la región, en unidades del dispositivo.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la región; en caso contrario, devuelve cero.

### Véase además

*CreateRectRgn*, *CreateRoundRectRgn*, *DeleteObject*

### Ejemplo

#### Listado 4-9: Creando indirectamente una región rectangular

```

procedure TForm1.Button1Click(Sender: TObject);
var

```

```

RegionHandle: HRGN;    // manejador de región
begin
  {Crear una región rectangular del tamaño del área cliente del formulario}
  RegionHandle := CreateRectRgnIndirect(Form1.ClientRect);

  {Inicializar la brocha}
  Canvas.Brush.Style := bsDiagCross;
  Canvas.Brush.Color := clRed;

  {Rellenar la región rectangular}
  FillRgn(Canvas.Handle, RegionHandle, Canvas.Brush.Handle);

  {La región ya no es necesaria y es eliminada}
  DeleteObject(RegionHandle);
end;

```

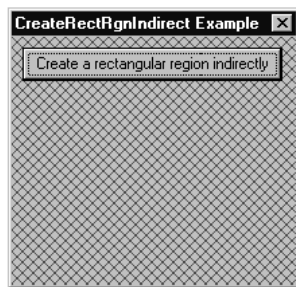


Figura 4-10:  
La región  
rectangular

### CreateRoundRectRgn Windows.Pas

#### Sintaxis

```

CreateRoundRectRgn(
  p1: Integer;           {la coordenada horizontal superior izquierda}
  p2: Integer;           {la coordenada vertical superior izquierda}
  p3: Integer;           {la coordenada horizontal inferior derecha}
  p4: Integer;           {la coordenada vertical inferior derecha}
  p5: Integer;           {el ancho de la elipse de la esquina redondeada}
  p6: Integer;           {la altura de la elipse de la esquina redondeada}
): HRGN;                {devuelve el manejador de la región}

```

#### Descripción

Esta función crea una región rectangular con esquinas redondeadas, basada en las coordenadas especificadas. Cuando la región deje de ser necesaria, debe ser eliminada llamando a la función *DeleteObject*.

**Parámetros**

*p1*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo, en unidades del dispositivo.

*p2*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo, en unidades del dispositivo.

*p3*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo, en unidades del dispositivo.

*p4*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo, en unidades del dispositivo.

*p5*: Especifica el ancho de la elipse utilizada para definir las esquinas redondeadas del rectángulo, en unidades del dispositivo.

*p6*: Especifica la altura de la elipse utilizada para definir las esquinas redondeadas del rectángulo, en unidades del dispositivo.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de la región; en caso contrario, devuelve cero.

**Véase además**

*CreateRectRgn*, *CreateRectRgnIndirect*, *CreateRoundRectRgn*, *DeleteObject*

**Ejemplo****Listado 4-10: Creando una región rectangular redondeada**

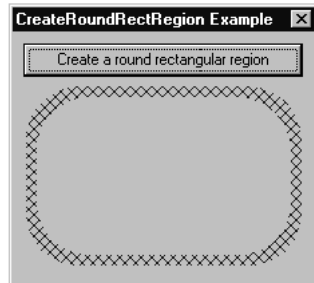
```
procedure TForm1.Button1Click(Sender: TObject);
var
    RegionHandle: HRGN; // almacena la región
begin
    {Crear una región rectangular redondeada}
    RegionHandle := CreateRoundRectRgn(10, 40, 217, 175, 80, 80);

    {Inicializar la brocha}
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Brush.Color := clBlue;

    {Dibujar el perímetro de la región}
    FrameRgn(Canvas.Handle, RegionHandle, Canvas.Brush.Handle, 8, 8);

    {Eliminar la región}
    DeleteObject(RegionHandle);
end;
```

Figura 4-11:  
La región  
rectangular  
redondeada



### **EndPath    Windows.Pas**

#### **Sintaxis**

```
EndPath(
    DC: HDC           {manejador de contexto de dispositivo}
): BOOL;             {devuelve TRUE o FALSE}
```

#### **Descripción**

Esta función cierra una definición de ruta abierta. La ruta resultante es asociada con el contexto de dispositivo identificado por el parámetro *DC*.

#### **Parámetros**

*DC*: Especifica el contexto de dispositivo que contendrá la ruta resultante.

#### **Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### **Véase además**

*BeginPath*

#### **Ejemplo**

Vea el Listado 4-19 bajo *SelectClipPath* y otros ejemplos a lo largo de este capítulo.

### **EqualRect    Windows.Pas**

#### **Sintaxis**

```
EqualRect(
    const lprc1: TRect;   {el primer rectángulo a comparar}
    const lprc2: TRect;   {el segundo rectángulo a comparar}
): BOOL;                 {devuelve TRUE o FALSE}
```

**Descripción**

Esta función determina si las coordenadas de dos rectángulos son idénticas.

**Parámetros**

*lprc1*: Puntero al primer registro *TRect* que contiene las coordenadas a comparar.

*lprc2*: Puntero al segundo registro *TRect* que contiene las coordenadas a comparar.

**Valor que devuelve**

Si la función tiene éxito, y las coordenadas del rectángulo identificado por el parámetro *lprc1* son idénticas a las coordenadas del rectángulo identificado por el parámetro *lprc2*, devuelve TRUE. Si la función falla o las coordenadas no son idénticas, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*EqualRgn*, *IsRectEmpty*, *PtInRect*

**Ejemplo**

Vea el Listado 4-16 bajo *OffsetRect*.

**EqualRgn****Windows.Pas****Sintaxis**

```
EqualRgn(
    p1: HRGN;           {manejador de la primera región a comparar}
    p2: HRGN           {manejador de la segunda región a comparar}
): BOOL;               {devuelve TRUE o FALSE}
```

**Descripción**

Esta función determina si dos regiones son idénticas en tamaño y forma, y residen en las mismas coordenadas.

**Parámetros**

*p1*: Manejador de la primera región a comparar.

*p2*: Manejador de la segunda región a comparar.

**Valor que devuelve**

Si las dos regiones son idénticas en tamaño y forma, y residen en las mismas coordenadas, la función devuelve TRUE; de lo contrario, devuelve FALSE. El valor *ERROR* indica que al menos uno de los manejadores de región especificados no es válido.

Véase además

*CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn*

Ejemplo

#### Listado 4-II: Comparando dos regiones

```

procedure TForm1.FormPaint(Sender: TObject);
var
    Region1, Region2: HRGN; // almacena las regiones a comparar
begin
    {Crear una región elíptica}
    Region1 := CreateEllipticRgn(50, 50, 150, 150);

    {Transformar la región en una región rectangular. Esta función puede ser
    ejecutada sobre cualquier región preexistente}
    SetRectRgn(Region1, 50, 50, 150, 150);

    {Crear una región rectangular idéntica a Region1}
    Region2 := CreateRectRgn(50, 50, 150, 150);

    {Rellenar ambas regiones de rojo}
    Canvas.Brush.Color := clRed;
    PaintRgn(Canvas.Handle, Region1);
    PaintRgn(Canvas.Handle, Region2);

    {Indicar que las regiones son idénticas}
    if EqualRgn(Region1, Region2) then
        Label1.Caption := 'Regions Equal'
    else
        Label1.Caption := 'Regions Not Equal';

    {Eliminar ambas regiones, pues no son necesarias}
    DeleteObject(Region1);
    DeleteObject(Region2);
end;

```

#### **ExcludeClipRect** Windows.Pas

Sintaxis

```

ExcludeClipRect(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: Integer;             {la coordenada horizontal superior izquierda}
    p3: Integer;             {la coordenada vertical superior izquierda}
    p4: Integer;             {la coordenada horizontal inferior derecha}
    p5: Integer;             {la coordenada vertical inferior derecha}
): Integer;                 {devuelve el tipo de región de recorte}

```

**Descripción**

Esta función excluye el rectángulo definido por las coordenadas dadas de la región de recorte del contexto de dispositivo especificado. Los bordes superior e izquierdo del rectángulo especificados son excluidos de la región de recorte, pero no los bordes inferior y derecho.

**Parámetros**

*DC*: Manejador del contexto de dispositivo que contiene la región de recorte que será modificada.

*p2*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo, en unidades lógicas.

*p3*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo, en unidades lógicas.

*p4*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo, en unidades lógicas.

*p5*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo, en unidades lógicas.

**Valor que devuelve**

Esta función devuelve un resultado que indica el tipo de región creada o una condición de error y puede ser un valor de la Tabla 4-4.

**Véase además**

*OffsetClipRgn*, *SetRect*, *SetRectRgn*

**Ejemplo****Listado 4-12: Dibujando una imagen de primer plano sólo una vez**

```
{Estructura del registro que define un punto que se mueve}
TDot = record
  Pos: TPoint;
  Vel: TPoint;
end;

var
  Form1: TForm1;
  Dots: array[0..9] of TDot; // array de puntos que se mueven
  Offscreen: TBitmap;       // el doble buffer fuera de pantalla
```

**implementation**

```
{ $R *.DFM }
```

```
procedure TForm1.FormPaint(Sender: TObject);
begin
```

```

        {Dibujar la imagen de primer plano. Esta se dibujará sólo una vez}
        Canvas.Draw(Image2.Left, Image2.Top, Image2.Picture.Bitmap);
    end;

    procedure TForm1.FormCreate(Sender: TObject);
    var
        iCount: Integer;    // variable de control de bucle
    begin
        {Crear e inicializar el mapa de bits fuera de pantalla}
        OffScreen := TBitmap.Create;
        OffScreen.Width := Form1.ClientWidth;
        OffScreen.Height := Form1.ClientHeight;

        {Crear e inicializar el array de puntos móviles}
        for iCount := 0 to 9 do
            begin
                Dots[iCount].Pos.X := Random(ClientWidth);
                Dots[iCount].Pos.Y := Random(ClientHeight);
                if Random(2)=0 then Dots[iCount].Vel.X := -1 else Dots[iCount].Vel.X := 1;
                if Random(2)=0 then Dots[iCount].Vel.Y := -1 else Dots[iCount].Vel.Y := 1;
            end;
        end;

    procedure TForm1.FormDestroy(Sender: TObject);
    begin
        {El mapa de bits fuera de pantalla ya no es necesario, se libera}
        Offscreen.Free;
    end;

    procedure TForm1.Timer1Timer(Sender: TObject);
    var
        iCount: Integer;    // contador de bucle
    begin
        {Borrar el último marco de animación en el mapa de bits fuera de pantalla}
        Offscreen.Canvas.Brush.Color := clBlack;
        Offscreen.Canvas.FillRect(Offscreen.Canvas.ClipRect);

        {Recorrer los 10 puntos móviles}
        for iCount := 0 to 9 do
            begin
                {Cambiar la posición del punto de acuerdo con la velocidad}
                Dots[iCount].Pos.X := Dots[iCount].Pos.X + Dots[iCount].Vel.X;
                Dots[iCount].Pos.Y := Dots[iCount].Pos.Y + Dots[iCount].Vel.Y;

                {Invertir la velocidad del punto si ha llegado al borde de la pantalla}
                if (Dots[iCount].Pos.X < 0) or (Dots[iCount].Pos.X > ClientWidth) then
                    Dots[iCount].Vel.X := 0-Dots[iCount].Vel.X;
                if (Dots[iCount].Pos.Y < 0) or (Dots[iCount].Pos.Y > ClientHeight) then
                    Dots[iCount].Vel.Y := 0-Dots[iCount].Vel.Y;

                {Dibujar un punto rojo en el mapa de bits fuera de pantalla (2X2 píxeles)}
                Offscreen.Canvas.Pixels[Dots[iCount].Pos.X,Dots[iCount].Pos.Y] := clRed;
                Offscreen.Canvas.Pixels[Dots[iCount].Pos.X+1,Dots[iCount].Pos.Y] := clRed;
                Offscreen.Canvas.Pixels[Dots[iCount].Pos.X,Dots[iCount].Pos.Y+1] := clRed;
            end;
        end;
    end;

```

```

    Offscreen.Canvas.Pixels[Dots[iCount].Pos.X+1,Dots[iCount].Pos.Y+1] := clRed;
end;

{El mapa de bits almacenado en Image1 ya ha sido dibujado sobre el formulario.
 Esto se produce sólo una vez, cuando el evento Paint se dispara, lo que
 ocurre cuando el formulario es mostrado la primera vez o después que sea
 descubierto por una ventana de nivel superior. Debido a que no queremos destruir
 esta imagen de primer plano, excluimos su área rectangular de la región de
 recorte. Esto abrirá un "hueco" en la región de recorte, y cualquier intento de
 dibujar en esta área será denegado}
ExcludeClipRect(Canvas.Handle, Image2.Left, Image2.Top,
                Image2.Left + Image2.Width, Image2.Top + Image2.Height);

{Dibuja el mapa de bits fuera de pantalla en la pantalla. El "hueco" en la región
 de recorte evita que el mapa de bits sea dibujado sobre el mapa de bits de
 primer plano}
Canvas.Draw(0, 0, Offscreen);
end;

```

Figura 4-12:  
La imagen de  
primer plano  
no es afectada  
durante la  
animación  
continua

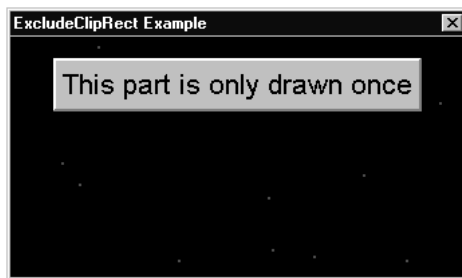


Tabla 4-4: Valores que devuelve ExcludeClipRect

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ocurrió un error.

### ExtCreateRegion Windows.Pas

#### Sintaxis

```

ExtCreateRegion(
  p1: PXForm;           {puntero a un registro TXForm}
  p2: DWORD;            {tamaño del registro de datos de la región}
  const p3: TRgnData    {puntero a un registro TRgnData}
): HRGN;               {devuelve un manejador de región}

```

### Descripción

Esta función crea una nueva región aplicando la matriz de transformación especificada en el parámetro *p1* a los datos de región especificados en el parámetro *p3*. Tenga en cuenta que bajo Windows 95/98, las transformaciones de corte (*shearing*) y rotación no están soportadas y la función fallará si el registro indicado en el parámetro *p1* contiene otra cosa que no sean valores de escalamiento y traslación.

### Parámetros

*p1*: Puntero al registro *TXForm* que contiene una matriz de transformación que es aplicada a la región identificada por el parámetro *p3*. Si este parámetro es **nil**, la región no es transformada en modo alguno. Consulte la Tabla 4-5, que describe cómo los campos de este registro son usados en caso de diferentes transformaciones. El registro *TXForm* se define de la siguiente forma:

TXForm = **packed record**

eM11: Single;	{valor de rotación, escalamiento o reflexión}
eM12: Single;	{valor de rotación o corte}
eM21: Single;	{valor de rotación o corte}
eM22: Single;	{valor de rotación, escalamiento o reflexión}
eDx: Single;	{traslación horizontal}
eDy: Single;	{traslación vertical}

**end;**

*eM11*: Especifica el valor del escalamiento horizontal, el coseno del ángulo de rotación o el valor de la reflexión horizontal.

*eM12*: Especifica la constante de proporcionalidad horizontal para el corte o el seno del ángulo de rotación.

*eM21*: Especifica la constante de proporcionalidad vertical para el corte o el seno negativo del ángulo de rotación.

*eM22*: Especifica el valor del escalamiento vertical, el coseno del ángulo de rotación o el valor de la reflexión vertical.

*eDx*: Especifica el valor de la traslación horizontal.

*eDy*: Especifica el valor de la traslación vertical.

*p2*: Especifica el tamaño de los datos de la región a los que apunta el parámetro *p3*, en bytes.

*p3*: Puntero al registro *TRgnData* que contiene información sobre la región que será transformada. Es una estructura de longitud variable que tiene que ser inicializada mediante una llamada previa a la función *GetRegionData*. El registro *TRgnData* se define como:

TRgnData = **record**

rdh: TRgnDataHeader;	{información sobre los datos de la región}
Buffer: <b>array</b> [0..0] <b>of</b> CHAR;	{un array de rectángulos}

**end;**

*rdh*: Especifica un registro *TRgnDataHeader* que contiene información sobre la definición de la región. El registro *TRgnDataHeader* se define como:

*TRgnDataHeader* = **packed record**

<i>dwSize</i> : DWORD;	{tamaño del registro}
<i>iType</i> : DWORD;	{opción de tipo de región}
<i>nCount</i> : DWORD;	{cantidad de rectángulos}
<i>nRgnSize</i> : DWORD;	{tamaño del <i>buffer</i> de las coordenadas rectangulares}
<i>rcBound</i> : TRect;	{coordenadas del rectángulo límite}

**end;**

*dwSize*: Especifica el tamaño del registro *TRgnDataHeader*, en bytes. A este campo debe asignársele el valor *SizeOf(TRgnDataHeader)*.

*iType*: Especifica una opción que indica el tipo de región. Actualmente este campo sólo puede contener el valor *RDH\_RECTANGLES*.

*nCount*: Especifica la cantidad de rectángulos que definen la región.

*nRgnSize*: Especifica el tamaño del *buffer* requerido para recibir las coordenadas de los rectángulos que definen la región. Este es el tamaño del *buffer* identificado por el campo *Buffer* del registro *TRgnData*.

*rcBound*: Especifica un registro *TRect* que contiene la coordenadas del rectángulo límite para la región, en unidades lógicas.

*Buffer*: Especifica un *buffer* de longitud variable que contiene las coordenadas de los rectángulos que definen la región.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la nueva región transformada; en caso contrario, devuelve cero.

#### Véase además

*CreateEllipticRgn*, *CreateEllipticRgnIndirect*, *CreatePolygonRgn*, *CreatePolyPolygonRgn*, *CreateRectRgn*, *CreateRectRgnIndirect*, *CreateRoundRectRgn*, *GetRegionData*

#### Ejemplo

Vea el Listado 4-6 bajo *CreatePolygonRgn*.

Tabla 4-5: Valores de transformación del parámetro *pl* de `ExtCreateRegion`

Transformación	Valor eM11	Valor eM12	Valor eM21	Valor eM22
Rotación	Coseno del ángulo de rotación	Seno del ángulo de rotación	Seno negativo del ángulo de rotación	Coseno del ángulo de rotación
Escalamiento	Valor de escalamiento horizontal	Cero	Cero	Valor de escalamiento vertical
Corte	Cero	Valor de proporcionalidad horizontal	Valor de proporcionalidad vertical	Cero
Reflexión	Valor de reflexión horizontal	Cero	Cero	Valor de reflexión vertical

***ExtSelectClipRgn Windows.Pas******Sintaxis***

```
ExtSelectClipRgn(
    DC: HDC;           {manejador de contexto de dispositivo}
    p2: HRGN;          {manejador de región}
    p3: Integer         {opción de combinación de región}
): Integer;           {devuelve el tipo de la región combinada}
```

***Descripción***

Esta función combina la región de recorte del contexto de dispositivo identificado por el parámetro *DC*, con la región identificada por el parámetro *p2*, de acuerdo a la opción especificada en el parámetro *p3*. Se asume que las coordenadas de la región identificada por el parámetro *p2* están dadas en unidades del dispositivo. Esta función crea una copia de la región identificada por el parámetro *p2*; la región original no es afectada y puede ser utilizada en otras llamadas a funciones posteriores.

***Parámetros***

*DC*: Manejador del contexto de dispositivo que contiene la región de recorte que será combinada con la región especificada.

*p2*: Manejador de la región que será combinada con la región de recorte del contexto de dispositivo especificado.

*p3*: Opción que indica cómo la región de recorte del contexto de dispositivo y la región especificada serán combinadas. Este parámetro puede contener un valor de la Tabla 4-6.

**Valor que devuelve**

Esta función devuelve un resultado que indica el tipo de región creada o una condición de error y puede ser un valor de la Tabla 4-7. Si ocurre un error, la región de recorte del contexto de dispositivo especificado no es afectada.

**Véase además**

*GetClipBox, GetClipRgn, OffsetClipRgn, SelectClipPath, SelectClipRgn*

**Ejemplo**

Vea el Listado 4-15 bajo *OffsetClipRgn*.

**Tabla 4-6: Valores del parámetro p3 de ExtSelectClipRgn**

Valor	Descripción
RGN_AND	La región resultante es la intersección de las dos regiones especificadas.
RGN_COPY	La región resultante es una copia de la región identificada por el parámetro p2. Si se especifica esta opción y el parámetro p2 es igual a cero, la región de recorte actual es reiniciada a la región de recorte por defecto para el contexto de dispositivo especificado.
RGN_DIFF	La región resultante es el área de la región identificada por el parámetro p2 que no forma parte de la región actualmente recortada.
RGN_OR	La región resultante es la unión de las dos regiones especificadas.
RGN_XOR	La región resultante es la unión de las dos áreas especificadas, excluyendo las áreas comunes a ambas.

**Tabla 4-7: Valores que devuelve ExtSelectClipRgn**

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ha ocurrido un error.

**FlattenPath****Windows.Pas****Sintaxis**

```
FlattenPath(
    DC: HDC           {manejador de contexto de dispositivo}
); BOOL;             {devuelve TRUE o FALSE}
```

**Descripción**

Esta función convierte cualquier curva ubicada en la ruta seleccionada en el contexto de dispositivo especificado en una serie de segmentos de línea recta.

**Parámetros**

*DC*: Manejador del contexto de dispositivo que contiene la ruta que será convertida en segmentos de líneas.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetPath*, *PathToRegion*, *WidenPath*

**Ejemplo**

Vea el Listado 4-13 bajo *GetPath*.

**GetClipBox****Windows.Pas****Sintaxis**

```
GetClipBox(
    DC: HDC;                {manejador de contexto de dispositivo}
    var Rect: TRect          {puntero a registro TRect}
): Integer;                {devuelve el tipo de la región de recorte}
```

**Descripción**

Esta función recupera las coordenadas del rectángulo más pequeño que puede ser dibujado alrededor del área actualmente visible en el contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo rectángulo límite del área visible será recuperado.

*Rect*: Puntero a un registro *TRect* que recibe las coordenadas del rectángulo más pequeño que engloba el área visible del contexto de dispositivo especificado, en unidades lógicas.

**Valor que devuelve**

Esta función devuelve un resultado que indica el tipo de región creada o una condición de error y puede ser un valor de la Tabla 4-8.

**Véase además**

*ExtSelectClipRgn*, *GetClipRgn*, *GetRgnBox*, *OffsetClipRgn*, *SelectClipPath*, *SelectClipRgn*

**Ejemplo**

Vea el Listado 4-4 bajo *CombineRgn*.

**Tabla 4-8: Valores que devuelve GetClipBox**

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ha ocurrido un error.

**GetClipRgn**      **Windows.Pas****Sintaxis**

```
GetClipRgn(
    DC: HDC;           {manejador de contexto de dispositivo}
    rgn: HRGN          {manejador de una región preexistente}
); Integer;           {devuelve un código de error}
```

**Descripción**

Esta función recupera un manejador de la región de recorte definida por la aplicación asignada por la última llamada a la función *SelectClipRgn*. La región identificada por el parámetro *rgn* tiene que ser una región preexistente.

**Parámetros**

*DC*: Especifica un manejador del contexto de dispositivo que contiene la región de recorte definida por la aplicación que será recuperada.

*rgn*: Especifica un manejador de una región preexistente. Este manejador identificará una copia de la región de recorte definida por la aplicación cuando la función retorne. Cualquier cambio en esta región copiada no afectará a la región de recorte actual.

**Valor que devuelve**

Si la función tiene éxito, y el contexto de dispositivo no contiene una región de recorte, devuelve cero. Si la función tiene éxito, y el contexto de dispositivo contiene una región de recorte, devuelve 1. Si la función falla, devuelve -1.

**Véase además**

*GetClipBox*, *GetRgnBox*, *SelectClipRgn*

**Ejemplo**

Vea el Listado 4-6 bajo *CreatePolygonRgn*.

**GetPath****Windows.Pas****Sintaxis**

```

GetPath(
    DC: HDC;                {manejador de un contexto de dispositivo}
    var Points;              {puntero a array de registros TPoint}
    var Types;              {puntero a array de bytes}
    nSize: Integer          {cantidad de elementos en los arrays}
): Integer;                {devuelve la cantidad de puntos recuperados}

```

**Descripción**

Esta función recupera las coordenadas y tipos de vértices de los puntos de terminación de los segmentos de línea y los puntos de control de las curvas de Bézier, que definen la ruta en el contexto de dispositivo especificado. Los puntos de terminación y puntos de control de la ruta son almacenados en el *array* de registros *TPoint* al que apunta el parámetro *Points*, y los tipos de vértices son almacenados en el *array* de bytes al que apunta el parámetro *Types*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo que contiene la ruta cuyos puntos y tipos de vértices serán recuperados.

*Points*: Puntero a un *array* de registros *TPoint* reservado por la aplicación, que recibirá los puntos de terminación de las líneas y los puntos de control de las curvas de la ruta. Estas coordenadas estarán especificadas en unidades lógicas.

*Types*: Puntero a un *array* de bytes reservado por la aplicación, donde cada entrada recibirá un valor que indica el tipo de vértice correspondiente. Deberá existir un elemento en este *array* por cada elemento del *array* al que apunta el parámetro *Points*. Los posibles valores que tomará este *array* se muestran en la Tabla 4-9.

*nSize*: Especifica el número total de elementos de los *arrays* a los que apuntan los parámetros *Points* y *Types*. Si a estos parámetros se les asigna cero, la función devolverá la cantidad total de entradas que se requieren para almacenar todos los puntos que definen la ruta.

**Valor que devuelve**

Si la función tiene éxito, devuelve la cantidad total de puntos recuperados de la ruta. Si la función falla, si el parámetro *nSize* especifica una cantidad menor que el número de puntos en la ruta o no hay suficiente espacio en los *arrays* a los que apuntan los parámetros *Points* y *Types*, la función devuelve -1. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*BeginPath*, *EndPath*, *FlattenPath*, *PathToRegion*, *WidenPath*

## Ejemplo

**Listado 4-13: Recuperando los puntos que definen una curva aplanada**

```

procedure TForm1.Button1Click(Sender: TObject);
type
  TPointsArray = array[0..0] of TPoint;    // array para los vértices
  TTypesArray = array[0..0] of Byte;      // array para los tipos de vértices
var
  CurvePts: array[0..3] of TPoint;        // array de puntos que definen la curva
  Points: ^TPointsArray;                  // puntero a un array de puntos
  Types: ^TTypesArray;                    // puntero a un array de bytes
  PtCount: Integer;                      // el número de puntos en la ruta
  iCount: Integer;                      // variable de control de bucle general
  FormDC: HDC;                          // un manejador del DC del formulario
  ThePen, OldPen: HPEN;                  // manejadores de plumas
  InfoString: String;                  // una cadena que describe un punto
begin
  {Define puntos utilizados para dibujar una curva de Bézier}
  CurvePts[0] := Point(30, 80);
  CurvePts[1] := Point(55, 30);
  CurvePts[2] := Point(105, 30);
  CurvePts[3] := Point(130, 80);

  {Recupera un manejador del contexto de dispositivo del formulario}
  FormDC := GetDC(Form1.Handle);

  {Comienza una definición de ruta}
  BeginPath(FormDC);

  {Dibuja una curva de Bézier}
  PolyBezier(FormDC, CurvePts, 4);

  {Termina la definición de ruta}
  EndPath(FormDC);

  {Convierte la ruta en una serie de segmentos de línea}
  FlattenPath(FormDC);

  {Recupera el número de puntos que definen la ruta}
  PtCount := GetPath(FormDC, Points^, Types^, 0);

  {Reserva suficiente memoria para almacenar los puntos y sus tipos}
  GetMem(Points, SizeOf(TPoint) * PtCount);
  GetMem(Types, PtCount);

  {Recupera los puntos y tipos de vértices de la ruta}
  GetPath(FormDC, Points^, Types^, PtCount);

  {Para cada punto en la ruta...}
  for iCount := 0 to PtCount-1 do
  begin
    {Registra las coordenadas de los puntos}
    InfoString := 'X: ' + IntToStr(Points[iCount].X) +

```

```

        ' Y: ' + IntToStr(Points[iCount].Y);

{Registra el tipo de punto}
case (Types[iCount] and not PT_CLOSEFIGURE) of
    PT_MOVETO:   InfoString := InfoString+' Type: MoveTo';
    PT_LINETO:   InfoString := InfoString+' Type: LineTo';
    PT_BEZIERTO: InfoString := InfoString+' Type: BezierTo';
end;

{Debido a que la opción PT_CLOSEFIGURE puede ser combinada con otras, se
verifica separadamente y se registra si la figura en la ruta está cerrada}
if (Types[iCount] and PT_CLOSEFIGURE) =PT_CLOSEFIGURE then
    InfoString := InfoString+', Close Figure';

{Muestra la información sobre este punto de la ruta}
ListBox1.Items.Add(InfoString);
end;

{Crea y selecciona una pluma en el contexto de dispositivo}
ThePen := CreatePen(PS_SOLID, 1, clBlack);
OldPen := SelectObject(FormDC, ThePen);

{Dibuja la ruta}
StrokePath(FormDC);

{La pluma ya no es necesaria y es eliminada}
SelectObject(FormDC, OldPen);
DeleteObject(ThePen);

{Libera la memoria usada para almacenar los puntos y los tipos de vértices}
FreeMem(Points);
FreeMem(Types);
end;

```

Figura 4-13:  
Los puntos  
finales del  
segmento de  
línea de una  
curva plana

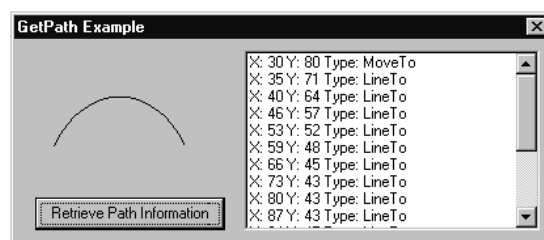


Tabla 4-9: Valores del campo Types de GetPath

Valor	Descripción
PT_MOVETO	En el punto asociado comienza una nueva figura.
PT_LINETO	El punto asociado y el punto previo forman un segmento de línea.
PT_BEZIERTO	El punto asociado es un punto de control o de terminación para una curva de Bézier. El punto que precede al primer punto PT_BEZIERTO es el punto de inicio para la curva de Bézier. Los siguientes dos puntos PT_BEZIERTO son los puntos de control para la curva. Estos serán seguidos por otro punto PT_BEZIERTO que identifica el punto de terminación de la curva de Bézier, si una fue especificada.
PT_CLOSEFIGURE	Este valor puede ser combinado con las opciones PT_LINETO o PT_BEZIERTO mediante el operador booleano <b>or</b> , e indica el último punto en una figura cerrada.

**GetRegionData**    *Windows.Pas***Sintaxis**

```

GetRegionData(
    RGN: HRGN;           {manejador de la región}
    p2: DWORD;           {tamaño del buffer de datos de la región}
    p3: PRgnData         {puntero a un registro TRgnData}
): DWORD;               {si tiene éxito, devuelve 1}

```

**Descripción**

Esta función recupera información sobre la región identificada por el parámetro *RGN*; esencialmente, se trata de información concerniente al rectángulo que define la región. Esta información es almacenada en el registro de longitud variable al que apunta el parámetro *p3*.

**Parámetros**

*RGN*: Un manejador de la región cuya información se desea recuperar.

*p2*: Especifica el tamaño del registro de datos al que apunta el parámetro *p3*, en bytes. Si este valor no es lo suficientemente grande para almacenar los datos de la región, la función devuelve el tamaño requerido del *buffer*, en bytes.

*p3*: Puntero al registro *TRgnData* que recibirá la información sobre la región especificada. El registro *TRgnData* es un registro de longitud variable para el cual la aplicación debe reservar memoria. Si a este parámetro se le asigna **nil**, la función devuelve el tamaño de *buffer* necesario para almacenar los datos de la región (en bytes). El registro *TRgnData* se define como:

```

TRgnData = record
    rdh: TRgnDataHeader;           {información de la región}
    Buffer: array[0..0] of CHAR;   {array de rectángulos}
end;

```

Consulte la función *ExtCreateRegion* para ver una descripción de este registro.

#### Valor que devuelve

Si la función tiene éxito, devuelve uno; en caso contrario, devuelve cero.

#### Véase además

*ExtCreateRegion*, *GetClipRgn*

#### Ejemplo

Vea el Listado 4-6 bajo *CreatePolygonRgn*.

## GetRgnBox Windows.Pas

#### Sintaxis

```

GetRgnBox(
    RGN: HRGN;           {manejador de una región}
    var p2: TRect         {puntero a un registro TRect}
): Integer;             {devuelve el tipo de región}

```

#### Descripción

Esta función recupera las coordenadas del rectángulo más pequeño que puede ser dibujado alrededor de la región especificada.

#### Parámetros

*RGN*: Un manejador de la región cuyo rectángulo límite se desea recuperar.

*p2*: Puntero a un registro *TRect* que recibirá las coordenadas del rectángulo más pequeño que circunda la región especificada, en unidades lógicas.

#### Valor que devuelve

El resultado indica el tipo de región cuyo rectángulo límite fue recuperado o una condición de error, y puede ser uno de los valores de la Tabla 4-10.

#### Véase además

*GetClipBox*, *GetClipRgn*, *GetRegionData*

#### Ejemplo

Vea el Listado 4-17 bajo *OffsetRgn* y el Listado 4-18 bajo *PathToRegion*.

Tabla 4-10: Valores que devuelve GetRgnBox

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ocurrió un error.

**InflateRect**      **Windows.Pas****Sintaxis**

```
InflateRect(
    var lprc: TRect;           {puntero a un registro TRect}
    dx: Integer;               {valor de incremento o decremento horizontal}
    dy: Integer;               {valor de incremento o decremento vertical}
); BOOL;                       {devuelve TRUE o FALSE}
```

**Descripción**

Esta función modifica el tamaño del rectángulo identificado por el parámetro *lprc*, añadiendo el valor del parámetro *dx* a los lados izquierdo y derecho, y el valor del parámetro *dy* a los lados superior e inferior.

**Parámetros**

*lprc*: Puntero a un registro *TRect* que contiene el rectángulo que será incrementado o decrementado en tamaño.

*dx*: Especifica la cantidad en la cual incrementar o decrementar el ancho del rectángulo. Un valor positivo incrementa el ancho; un valor negativo lo decremента.

*dy*: Especifica la cantidad en la cual incrementar o decrementar la altura del rectángulo. Un valor positivo incrementa la altura; un valor negativo la decremента.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CopyRect*, *IntersectRect*, *OffsetRect*, *PtInRect*, *SetRect*, *UnionRect*

**Ejemplo**

Vea el Listado 4-16 bajo *OffsetRect*.

**IntersectRect**      **Windows.Pas***Sintaxis*

```
IntersectRect(
  var lprcDst: TRect;           {rectángulo que recibe las coordenadas de la
                                intersección}
  const lprcSrc1: TRect;        {primer rectángulo}
  const lprcSrc2: TRect;        {segundo rectángulo}
): BOOL;                       {devuelve TRUE o FALSE}
```

*Descripción*

Esta función determina la intersección entre los rectángulos identificados por los parámetros *lprcSrc1* y *lprcSrc2*. Las coordenadas del rectángulo intersección son almacenadas en el registro *TRect* al que apunta el parámetro *lprcDst*. Si los rectángulos no se intersectan, las coordenadas del parámetro *lprcDst* serán puestas a cero.

*Parámetros*

*lprcDst*: Puntero a un registro *TRect* que recibe las coordenadas de la intersección entre dos rectángulos identificados por los parámetros *lprcSrc1* y *lprcSrc2*.

*lprcSrc1*: Puntero al registro *TRect* que contiene las coordenadas del primer rectángulo.

*lprcSrc2*: Puntero al registro *TRect* que contiene las coordenadas del segundo rectángulo.

*Valor que devuelve*

Si la función tiene éxito, y los rectángulos se intersectan, devuelve TRUE. Si la función falla o los rectángulos no se intersectan, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

*Véase además*

*InflateRect*, *OffsetRect*, *PtInRect*, *SetRect*, *UnionRect*

*Ejemplo*

Vea el Listado 4-16 bajo *OffsetRect*.

**InvertRect**      **Windows.Pas***Sintaxis*

```
InvertRect(
  hDC: HDC;                     {manejador de un contexto de dispositivo}
  const lprc: TRect;            {puntero a un registro TRect}
): BOOL;                       {devuelve TRUE o FALSE}
```

**Descripción**

Esta función ejecuta una operación booleana **not** sobre el valor del color de cada píxel del contexto de dispositivo especificado que cae dentro del área rectangular definida por el rectángulo al que apunta el parámetro *lprc*.

**Parámetros**

*hDC*: Un manejador del contexto de dispositivo que contiene los píxeles cuyos colores serán invertidos.

*lprc*: Puntero a un registro *TRect* que contiene las coordenadas del área rectangular que será invertida, en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a *GetLastError*.

**Véase además**

*FillRect*, *InvertRgn*, *SetRect*

**Ejemplo****Listado 4-14: Invirtiendo una porción rectangular de una imagen**

```

procedure TForm1.Button1Click(Sender: TObject);
var
    TheRect: TRect;    // almacena las coordenadas del rectángulo
begin
    {Crea un rectángulo}
    SetRect(TheRect, 46, 40, 106, 100);

    {Invierte los píxeles dentro del rectángulo}
    InvertRect(Image1.Canvas.Handle, TheRect);

    {Repinta la nueva imagen}
    Image1.Refresh;
end;

```



Figura 4-14:  
El área  
rectangular  
invertida

**InvertRgn**      **Windows.Pas****Sintaxis**

```

InvertRgn(
    DC: HDC;           {manejador de un contexto de dispositivo}
    p2: HRGN           {manejador de una región}
): BOOL;              {devuelve TRUE o FALSE}

```

**Descripción**

Esta función ejecuta una operación booleana **not** sobre el valor del color de cada píxel del contexto de dispositivo *DC* que cae dentro de la región identificada por el parámetro *p2*.

**Parámetros**

*DC*: Un manejador del contexto de dispositivo que contiene los colores de los píxeles que serán invertidos.

*p2*: Un manejador de la región que define el área que será invertida. Se asume que las coordenadas de esta región están en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*FillRgn, InvertRect, PaintRgn*

**Ejemplo**

Vea el Listado 4-7 bajo *CreatePolyPolygonRgn*.

**IsRectEmpty**      **Windows.Pas****Sintaxis**

```

IsRectEmpty(
    const lprc: TRect   {puntero a un registro TRect}
): BOOL;               {devuelve TRUE o FALSE}

```

**Descripción**

Esta función determina si el rectángulo especificado está vacío. Un rectángulo se considera vacío si su lado inferior es menor o igual que su lado superior, o si su lado derecho es menor o igual que su lado izquierdo.

**Parámetros**

*lprc*: Un puntero al registro *TRect* que será comprobado. Las coordenadas de este rectángulo están en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito y el rectángulo está vacío, devuelve TRUE. Si la función falla o el rectángulo no está vacío, la función devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*EqualRect*, *PtInRect*, *SetRect*, *SetRectEmpty*, *SetRectRgn*

**Ejemplo**

Vea el Listado 4-21 bajo *SetRectEmpty*.

**OffsetClipRgn Windows.Pas****Sintaxis**

```
OffsetClipRgn(
    DC: HDC;                {manejador de un contexto de dispositivo}
    p2: Integer;             {desplazamiento horizontal}
    p3: Integer;             {desplazamiento vertical}
): Integer;                 {devuelve el tipo de región}
```

**Descripción**

Esta función mueve la región de recorte del contexto de dispositivo especificado en las cantidades horizontal y vertical dadas en los parámetros *p2* y *p3*.

**Parámetros**

*DC*: Un manejador del contexto de dispositivo que contiene la región de recorte que será movida.

*p2*: Especifica el desplazamiento horizontal en que será movida la región de recorte, en unidades lógicas.

*p3*: Especifica el desplazamiento vertical en que será movida la región de recorte, en unidades lógicas.

**Valor que devuelve**

Esta función devuelve un resultado que indica el tipo de región de recorte resultante del movimiento o una condición de error, y puede ser un valor de la Tabla 4-11.

**Véase además**

*OffsetRgn*, *SelectClipRgn*

*Ejemplo***Listado 4-15: Ejecutando efectos especiales de animación al mover la región de recorte**

```

var
  Form1: TForm1;
  MovingRgn: HRGN;           // almacena una región
  XPos, YPos, XVel, YVel: Integer; // almacena la velocidad y posición de la
                                // región

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {Crea una pequeña región circular que será usada como región de recorte}
  MovingRgn := CreateEllipticRgn(0, 0, 75, 75);

  {Inicializa la posición y velocidad de la región}
  XPos := 1;
  YPos := 1;
  XVel := 1;
  YVel := 1;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var
  TempBitmap: TBitmap;       // almacena un mapa de bits fuera de la pantalla
begin
  {Crea el mapa de bits fuera de la pantalla e inicializa su tamaño con el del
  TImage invisible. Este mapa de bits fuera de la pantalla es usado para eliminar
  el parpadeo}
  TempBitmap := TBitmap.Create;
  TempBitmap.Width := Image1.Width;
  TempBitmap.Height := Image1.Height;

  {Incrementa la posición de la región según la velocidad}
  Inc(XPos, XVel);
  Inc(YPos, YVel);

  {Si la región ha llegado al borde de la pantalla, invertir la velocidad}
  if (Xpos < 0) or (Xpos > ClientRect.Right-75) then
    XVel := 0-XVel;
  if (Ypos < 0) or (Ypos > ClientRect.Bottom-75) then
    YVel := 0-YVel;

  {Selecciona una región circular en el contexto de dispositivo del mapa de bits
  fuera de pantalla que indica que debe aplicársele un AND con la región actual
  de recorte del mapa de bits}
  ExtSelectClipRgn(TempBitmap.Canvas.Handle, MovingRgn, RGN_AND);

  {Mueve la región de recorte a la posición que está siendo seguida}
  OffsetClipRgn(TempBitmap.Canvas.Handle, XPos, YPos);

```

```

{Dibuja la imagen almacenada en Image1 en el mapa de bits. La región de recorte
sólo permitirá dibujar la pequeña área circular de la región en el mapa de bits}
TempBitmap.Canvas.Draw(0, 0, Image1.Picture.Bitmap);

{Dibuja el mapa de bits fuera de la pantalla en el formulario. Esto resultará en
una animación de un pequeño círculo que rebota}
Canvas.Draw(Image1.Left, Image1.Top, TempBitmap);

{Libera el mapa de bits fuera de pantalla}
TempBitmap.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {La región no es ya necesaria y se elimina}
    DeleteObject(MovingRgn);
end;

```



Figura 4-15:  
La región de  
recorte  
desplazada

Tabla 4-II: Valores que devuelve OffsetClipRgn

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ha ocurrido un error.

## OffsetRect

## Windows.Pas

### Sintaxis

```

OffsetRect(
    var lprc: TRect;           {puntero a un registro TRect}
    dx: Integer;               {desplazamiento horizontal}
    dy: Integer;               {desplazamiento vertical}
); BOOL;                      {devuelve TRUE o FALSE}

```

**Descripción**

Esta función mueve el rectángulo especificado horizontal y verticalmente en las cantidades indicadas por los parámetros *dx* y *dy*.

**Parámetros**

*lprc*: Puntero a un registro *TRect* que contiene las coordenadas del rectángulo que será desplazado, en unidades lógicas.

*dx*: Especifica el desplazamiento horizontal en el que será movido el rectángulo, en unidades lógicas.

*dy*: Especifica el desplazamiento vertical en el que será movido el rectángulo, en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CopyRect*, *InflateRect*, *IntersectRect*, *OffsetRgn*, *SubtractRect*, *UnionRect*

**Ejemplo****Listado 4-16: Una demostración de varias funciones de manipulación de rectángulos**

```
var
  Form1: TForm1;
  Rect1, Rect2: TRect;           // los dos rectángulos que serán
                                  // comprobados
  DragRect: PRect;              // apunta al rectángulo arrastrado
  DraggingRect: Boolean;         // indica si el arrastre está ocurriendo
  MouseOffsetX, MouseOffsetY: Integer; // usado para desplazar el rectángulo
                                  // arrastrado

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {Inicializa los dos rectángulos que serán comprobados}
  SetRect(Rect1, 10, 30, 110, 130);
  SetRect(Rect2, 60, 80, 160, 180);

  {Inicializa el indicador de que el arrastre está ocurriendo}
  DraggingRect := FALSE;
end;

procedure TForm1.FormPaint(Sender: TObject);
var
```

```

Intersection, Union: TRect; // muestra la unión y la intersección
begin
  {Recupera la unión de los dos rectángulos que serán comprobados}
  UnionRect(Union, Rect1, Rect2);

  {Dibuja esta unión de los rectángulos en verde}
  Form1.Canvas.Brush.Color := clGreen;
  Form1.Canvas.FillRect(Union);

  {Dibuja los dos rectángulos que serán comprobados en rojo}
  Form1.Canvas.Brush.Color := clRed;
  Form1.Canvas.FillRect(Rect1);
  Form1.Canvas.FillRect(Rect2);

  {Recupera la intersección de los dos rectángulos a comprobar}
  IntersectRect(Intersection, Rect1, Rect2);

  {Dibuja esta intersección en azul}
  Form1.Canvas.Brush.Color := clBlue;
  Form1.Canvas.FillRect(Intersection);

  {Indica si los dos rectángulos tienen exactamente las mismas coordenadas}
  if EqualRect(Rect1, Rect2) then
    Form1.Caption := 'OffsetRectExample - Rectangles are equal'
  else
    Form1.Caption := 'OffsetRectExample - Rectangles are not equal';
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {Si se hizo clic dentro del primer rectángulo...}
  if PtInRect(Rect1, Point(X, Y)) then
    begin
      {Indica que el arrastre ha comenzado}
      DraggingRect := TRUE;

      {Indica que estamos arrastrando el rectángulo 1}
      DragRect := @Rect1;
    end;

  {Si se hizo click dentro del segundo rectángulo...}
  if PtInRect(Rect2, Point(X, Y)) then
    begin
      {Indica que el arrastre ha comenzado}
      DraggingRect := TRUE;

      {Indica que estamos arrastrando el rectángulo 2}
      DragRect := @Rect2;
    end;

  {Si el arrastre ha comenzado...}
  if DraggingRect then
    begin

```

```

        {Recupera el desplazamiento de las coordenadas actuales del ratón dentro del
        rectángulo arrastrado. Esto es usado para que cuando se mueve el rectángulo
        se mantenga el lugar donde se hizo clic dentro del rectángulo. En caso
        contrario, cuando el rectángulo es movido la esquina superior izquierda de
        éste será ubicada en la posición del cursor del ratón.}
        MouseOffsetX := X - DragRect^.Left;
        MouseOffsetY := Y - DragRect^.Top;
    end;
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
Y: Integer);
begin
    {Si una operación de arrastre está ocurriendo...}
    if DraggingRect then
        begin
            {...borra el formulario}
            Form1.Canvas.Brush.Color := clBtnFace;
            Form1.Canvas.FillRect(DragRect^);

            {Mueve el rectángulo arrastrado, desplazándolo de la posición actual del ratón
            de manera que la ubicación del clic original dentro del rectángulo se
            mantenga}
            OffsetRect(DragRect^, X - DragRect^.Left - MouseOffsetX,
                Y - DragRect^.Top - MouseOffsetY);

            {Repinta el formulario}
            Form1.Repaint;
        end;
    end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
    {Indica que el arrastre ha finalizado}
    DraggingRect := FALSE;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    {Incrementa o decrementa el tamaño del último rectángulo arrastrado. La cantidad
    en la cual se incrementa o decrementa el tamaño es almacenada en la propiedad
    Tag de los botones de la barra de herramientas}
    if DragRect<>NIL then
        InflateRect(DragRect^, TSpeedButton(Sender).Tag, TSpeedButton(Sender).Tag);

    {Redibuja el formulario para mostrar los resultados}
    Form1.Repaint;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Fuerza al rectángulo 2 a convertirse en un duplicado exacto del rectángulo 1}
    CopyRect(Rect2, Rect1);

```

```
{Redibuja el formulario para mostrar los resultados}
Form1.Repaint;
end;
```

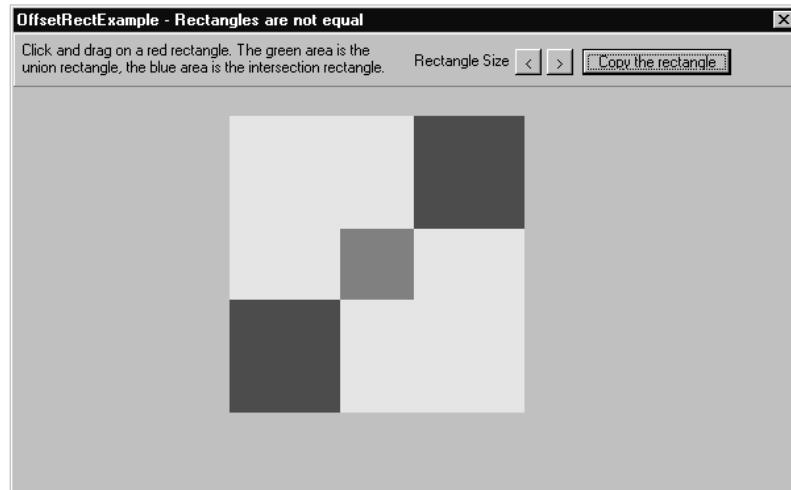


Figura 4-16:  
La prueba de  
la función  
Rectangle

## OffsetRgn

## Windows.Pas

### Sintaxis

```
OffsetRgn(
  RGN: HRGN;           {manejador de la región}
  p2: Integer;          {desplazamiento horizontal}
  p3: Integer           {desplazamiento vertical}
): Integer;             {devuelve el tipo de región}
```

### Descripción

Esta función mueve la región especificada en las cantidades indicadas en los parámetros *p2* y *p3*.

### Parámetros

*RGN*: Un manejador de la región que será movida.

*p2*: Especifica el desplazamiento horizontal en que será movida la región, en unidades lógicas.

*p3*: Especifica el desplazamiento vertical en que será movida la región, en unidades lógicas.

### Valor que devuelve

Esta función devuelve un resultado que indica el tipo de región resultante del movimiento o una condición de error, y puede ser un valor de la Tabla 4-12.

Véase además

*EqualRgn, OffsetClipRgn, OffsetRect*

Ejemplo

#### Listado 4-17: Moviendo una región para producir efectos especiales de animación

```

var
    Form1: TForm1;
    MovingRgn: HRGN;    // un manejador de la región en movimiento
    XVel, YVel: Integer; // la velocidad de la región

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Crea una región elíptica}
    MovingRgn := CreateEllipticRgn(0, 0, 75, 75);

    {Inicializa la velocidad de movimiento}
    XVel := 1;
    YVel := 1;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Selecciona la región circular como la región de recorte para el formulario}
    SelectClipRgn(Canvas.Handle, MovingRgn);

    {Dibuja la imagen en el TImage oculto sobre el formulario. La región de
    recorte circular evita cualquier dibujo fuera de la región circular}
    Canvas.Draw(Image1.Left, Image1.Top, Image1.Picture.Bitmap);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var
    RegionBounds: TRect; // almacena el rectángulo límite de la región
begin
    {Recupera el rectángulo más pequeño que puede ser dibujado alrededor de la región
    circular}
    GetRgnBox(MovingRgn, RegionBounds);

    {El rectángulo límite es usado para determinar si la región circular ha
    alcanzado los bordes de la pantalla. En ese caso, se invierte la velocidad}
    if (RegionBounds.Left < 0) or (RegionBounds.Left > ClientRect.Right-75) then
        XVel := 0-XVel;
    if (RegionBounds.Top < 0) or (RegionBounds.Top > ClientRect.Bottom-75) then
        YVel := 0-YVel;

    {Mueve la región a su velocidad actual}
    OffsetRgn(MovingRgn, XVel, YVel);

```

```

    {Redibuja el formulario para mostrar los resultados}
    Repaint;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {La región ya no es necesaria y se destruye}
    DeleteObject(MovingRgn);
end;

```



Figura 4-17:  
La región en  
movimiento

Tabla 4-12: Valores que devuelve OffsetRgn

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ha ocurrido un error.

## **PathToRegion**      **Windows.Pas**

### **Sintaxis**

```

PathToRegion(
    DC: HDC                {manejador de un contexto de dispositivo}
): HRGN;                  {devuelve un manejador de una región}

```

### **Descripción**

Esta función convierte la ruta en el contexto de dispositivo especificado en una región. La ruta tiene que haber sido cerrada, y es descartada del contexto de dispositivo cuando la función retorna. Cuando la región ya no es necesaria, debe ser eliminada llamando a la función *DeleteObject*.

### **Parámetros**

*DC*: Un manejador del contexto de dispositivo que contiene la ruta cerrada que será convertida en una región.

*Valor que devuelve*

Si la función tiene éxito, devuelve un manejador de una nueva región; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

*Véase además*

*BeginPath, EndPath, GetPath*

*Ejemplo***Listado 4-18: Convirtiendo una ruta en una región****var**

```
Form1: TForm1;
TextRgn: HRGN;           // almacena la región de texto
YVel: Integer;           // la velocidad vertical de la región
TempBitmap: TBitmap;     // un mapa de bits fuera de la pantalla para eliminar el
                          // parpadeo
```

**implementation**

```
{ $R *.DFM }
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

**begin**

```
{Comienza una definición de ruta para el contexto de dispositivo del formulario}
BeginPath(Canvas.Handle);
```

```
{Asigna al fondo un modo transparente. Esto es necesario porque la ruta
será el área interior del texto. Sin esto, la ruta queda definida como el
área exterior del texto.}
SetBkMode(Canvas.Handle, TRANSPARENT);
```

```
{Salida de una palabra al formulario. Esto es capturado como parte de la ruta.
Observe que a la fuente del formulario se le ha asignado un tamaño 48 Arial}
TextOut(Canvas.Handle, 1, 1, 'DELPHI', Length('DELPHI'));
```

```
{Termina la definición de ruta}
EndPath(Canvas.Handle);
```

```
{Convierte la ruta en una región. Observe que esto descarta la ruta en el
contexto de dispositivo}
TextRgn := PathToRegion(Canvas.Handle);
```

```
{Crea el mapa de bits fuera de pantalla y lo inicializa al tamaño del TImage
invisible}
TempBitmap := TBitmap.Create;
TempBitmap.Width := Image1.Width;
TempBitmap.Height := Image1.Height;
```

```
{Inicializa la velocidad vertical}
YVel := 1;
```

```

end;

procedure TForm1.Timer1Timer(Sender: TObject);
var
  RegionBounds: TRect; // almacena el rectángulo límite de la región
begin
  {Recupera el rectángulo límite de la región}
  GetRgnBox(TextRgn, RegionBounds);

  {Si la región está en el borde superior o inferior del formulario, se invierte su
  velocidad}
  if (RegionBounds.Top < 0) or (RegionBounds.Top > ClientRect.Bottom -
    (RegionBounds.Bottom - RegionBounds.Top)) then
    YVel := 0-YVel;

  {Desplaza la región verticalmente de acuerdo a su velocidad}
  OffsetRgn(TextRgn, 0, YVel);

  {Dibuja el gráfico en la imagen invisible al mapa de bits fuera de la pantalla}
  TempBitmap.Canvas.Draw(0, 0, Image1.Picture.Bitmap);

  {Invierte el área dentro de la región de texto}
  InvertRgn(TempBitmap.Canvas.Handle, TextRgn);

  {Copia el mapa de bits fuera de pantalla en el formulario, sin parpadeo}
  Canvas.Draw(0, 0, TempBitmap);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {La región ya no es necesaria y se elimina}
  DeleteObject(TextRgn);
end;

```

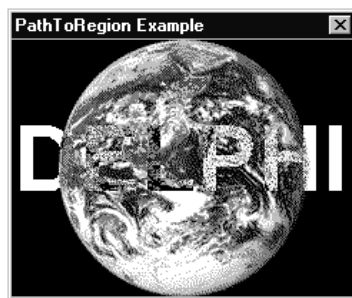


Figura 4-18:  
La ruta  
convertida

## **PtInRect Windows.Pas**

### **Sintaxis**

```

PtInRect(
  const lprc: TRect;           {puntero a un registro TRect}

```

```

    pt: TPoint                {puntero a un registro TPoint}
); BOOL;                    {devuelve TRUE o FALSE}

```

**Descripción**

Esta función determina si el punto identificado por el parámetro *pt* se encuentra dentro del rectángulo al que apunta el parámetro *lprc*. El punto es considerado *externo* al rectángulo si se encuentra exactamente sobre el borde inferior o sobre el derecho del rectángulo.

**Parámetros**

*lprc*: Puntero a un registro *TRect* que contiene las coordenadas del rectángulo.

*pt*: Puntero a un registro *TPoint* que contiene el punto que será comprobado.

**Valor que devuelve**

Si la función tiene éxito y el punto se encuentra dentro del rectángulo, devuelve TRUE.

Si la función falla o el punto no se encuentra dentro del rectángulo, devuelve FALSE.

Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*EqualRect*, *IsRectEmpty*, *PtInRegion*, *PtVisible*, *RectInRegion*, *SetRect*

**Ejemplo**

Vea el Listado 4-16 bajo *OffsetRect*.

**PtInRegion****Windows.Pas****Sintaxis**

```

PtInRegion(
    RGN: HRGN;                {manejador de una región}
    p2: Integer;               {coordenada horizontal}
    p3: Integer;               {coordenada vertical}
); BOOL;                      {devuelve TRUE o FALSE}

```

**Descripción**

Esta función determina si el punto identificado por los parámetros *p2* y *p3* se encuentra dentro de la región especificada en el parámetro *RGN*.

**Parámetros**

*RGN*: Un manejador de la región en la que el punto será comprobado.

*p2*: Especifica la coordenada horizontal del punto que será comprobado.

*p3*: Especifica la coordenada vertical del punto que será comprobado.

**Valor que devuelve**

Si la función tiene éxito y el punto se encuentra dentro de la región, devuelve TRUE. Si la función falla o el punto no se encuentra dentro de la región, devuelve FALSE.

**Véase además**

*PtInRect, PtVisible, RectInRegion*

**Ejemplo**

Vea el Listado 4-7 bajo *CreatePolyPolygonRgn*.

**PtVisible Windows.Pas****Sintaxis**

```
PtVisible(
    DC: HDC;           {manejador de un contexto de dispositivo}
    p2: Integer;        {coordenada horizontal}
    p3: Integer;        {coordenada vertical}
); BOOL;              {devuelve TRUE o FALSE}
```

**Descripción**

Esta función determina si el punto identificado por los parámetros *p2* y *p3* se encuentra dentro de la región de recorte del contexto de dispositivo especificado.

**Parámetros**

*DC*: Un manejador del contexto de dispositivo que contiene la región de recorte en la que el punto será comprobado.

*p2*: Especifica la coordenada horizontal del punto que será comprobado.

*p3*: Especifica la coordenada vertical del punto que será comprobado.

**Valor que devuelve**

Si la función tiene éxito y el punto se encuentra dentro de la región de recorte, devuelve TRUE. Si la función falla o el punto no se encuentra dentro de la región de recorte, devuelve FALSE.

**Véase además**

*PtInRect, PtInRegion, RectInRegion, RectVisible*

**Ejemplo**

Vea el Listado 4-6 bajo *CreatePolygonRgn*.

**RectInRegion**      **Windows.Pas***Sintaxis*

```

RectInRegion(
    RGN: HRGN;           {manejador de una región}
    const p2: TRect       {puntero a un registro TRect}
): BOOL;                {devuelve TRUE o FALSE}

```

*Descripción*

Esta función determina si alguna porción del rectángulo al que apunta el parámetro *p2* se encuentra dentro de la región identificada por el parámetro *RGN*. Observe que los lados inferior y derecho del rectángulo son excluidos de la comparación.

*Parámetros*

*RGN*: Un manejador de la región en la que el rectángulo será comprobado.

*p2*: Puntero a un registro *TRect* que contiene las coordenadas del rectángulo a comprobar.

*Valor que devuelve*

Si la función tiene éxito y alguna porción del rectángulo se encuentra dentro de la región, devuelve TRUE. Si la función falla o ninguna parte del rectángulo se encuentra dentro de la región, devuelve FALSE.

*Véase además*

*PtInRect*, *PtInRegion*, *PtVisible*, *RectVisible*, *SelectClipRegion*

*Ejemplo*

Vea el Listado 4-20 bajo *SelectClipRgn*.

**RectVisible**      **Windows.Pas***Sintaxis*

```

RectVisible(
    DC: HDC;             {manejador de un contexto de dispositivo}
    const Rect: TRect     {puntero a un registro TRect}
): BOOL;                {devuelve TRUE o FALSE}

```

*Descripción*

Esta función determina si alguna porción del rectángulo al que apunta el parámetro *Rect* se encuentra dentro de la región de recorte del contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Un manejador del contexto de dispositivo que contiene la región de recorte en la que el rectángulo será comprobado.

*Rect*: Puntero a un registro *TRect* que contiene las coordenadas del rectángulo a comprobar.

**Valor que devuelve**

Si la función tiene éxito y una porción del rectángulo se encuentra dentro de la región de recorte, devuelve TRUE. Si la función falla o ninguna parte del rectángulo se encuentra dentro de la región de recorte, devuelve FALSE.

**Véase además**

*GetClipRgn*, *PtInRegion*, *PtVisible*, *RectInRegion*, *SelectClipRgn*

**Ejemplo**

Vea el Listado 4-20 bajo *SelectClipRgn*.

**SelectClipPath    Windows.Pas****Sintaxis**

```
SelectClipPath(
    DC: HDC;                {manejador de un contexto de dispositivo}
    Mode: Integer            {modo de combinación de región}
); BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función selecciona la ruta actual en el contexto de dispositivo especificado como la región de recorte del contexto de dispositivo, combinándola con la región de recorte actual de acuerdo a las opciones especificadas en el parámetro *Mode*.

**Parámetros**

*DC*: Un manejador del contexto de dispositivo que contiene la ruta que será usada como región de recorte. Tiene que ser una ruta cerrada.

*Mode*: Opción que indica cómo la región de recorte formada por la ruta será combinada con la región de recorte actual del contexto de dispositivo. Este parámetro puede contener un valor de la Tabla 4-13.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*BeginPath, EndPath, PathToRegion, SelectClipRgn*

Ejemplo

#### Listado 4-19: Creando efectos de texto especiales

```

var
    Form1: TForm1;
    ThePalette: HPalette; // manejador de la paleta definida por la aplicación

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
    NewPalette: PLogPalette; // puntero a información de la paleta lógica
    iCount: Integer;         // contador de bucle general
begin
    {Inicializa la fuente del formulario}
    Font.Name := 'Arial';
    Font.Size := 48;

    {Asigna memoria suficiente para crear una paleta de 75 entradas}
    GetMem(NewPalette, SizeOf(TLogPalette)+75*SizeOf(TPaletteEntry));

    {Inicializa información específica de la paleta}
    NewPalette^.palVersion := $300;
    NewPalette^.palNumEntries := 75;

    {Recupera las primeras diez entradas de la paleta del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 10, NewPalette^.palPalEntry);

    {Crea una paleta de gradiente para las entradas restantes}
    for iCount := 10 to 74 do
    begin
        NewPalette^.palPalEntry[iCount].peRed := 255;
        NewPalette^.palPalEntry[iCount].peGreen := ((256 div 64) * (iCount-10));
        NewPalette^.palPalEntry[iCount].peBlue := 0;
        NewPalette^.palPalEntry[iCount].peFlags := PC_NOCOLLAPSE;
    end;

    {Crea una nueva paleta}
    ThePalette := CreatePalette(NewPalette^);

    {Libera la memoria reservada para la información de la paleta lógica}
    FreeMem(NewPalette);
end;

    {Dibuja líneas radiales gradientes, cuyo origen es el centro del texto}
procedure TForm1.DrawRadial;
var

```

```

iCount: Integer;           // variable de contador general de bucle
RayOrigin: TPoint;         // origen de las líneas radiales
Radius: Integer;           // radio dentro del cual se dibujarán las líneas
NewPen, OldPen: HPen;      // almacena una nueva y vieja pluma
begin
  {Comienza una definición de ruta dentro del contexto de dispositivo del
  formulario}
  BeginPath(Canvas.Handle);

  {Asigna al fondo un modo transparente. Esto es necesario porque la ruta
  será el área interior del texto. Sin esto, la ruta queda definida como el
  área exterior del texto.}
  SetBkMode(Canvas.Handle, TRANSPARENT);

  {Salida de una palabra en el formulario. Esto es capturado como parte de la ruta}
  TextOut(Canvas.Handle, 50, 50, 'Delphi Rocks!', Length('Delphi Rocks!'));

  {Termina la definición de ruta}
  EndPath(Canvas.Handle);

  {Selecciona esta ruta como la región de recorte para el contexto de dispositivo
  del formulario}
  SelectClipPath(Canvas.Handle, RGN_COPY);

  {Las líneas radiales deben dibujarse desde el centro del texto}
  RayOrigin.X := (Canvas.TextWidth('Delphi Rocks!') div 2) + 50;
  RayOrigin.Y := (Canvas.TextHeight('Delphi Rocks!') div 2) + 50;

  {El radio del círculo dentro del cual las líneas serán dibujadas será igual a la
  longitud del texto}
  Radius := Canvas.TextWidth('Delphi Rocks!');

  {Dibuja líneas en un arco de 90 grados}
  for iCount := 0 to 89 do
  begin
    {Crea una nueva pluma y especifica un color de la nueva paleta}
    NewPen := CreatePen(PS_SOLID, 1, PaletteIndex(75 - Trunc(iCount*(64/90)+10)));

    {Selecciona esta pluma en el contexto de dispositivo}
    OldPen := SelectObject(Canvas.Handle, NewPen);

    {Dibuja una línea comenzando en el centro del texto. Estas líneas se
    dibujarán hacia fuera de un modo circular. El siguiente código dibuja una
    línea en el primer cuadrante de un área circular dentro del texto y refleja
    esta línea en los otros tres cuadrantes}
    MoveToEx(Canvas.Handle, RayOrigin.X, RayOrigin.Y, nil);
    LineTo(Canvas.Handle, RayOrigin.X + Trunc(Radius*cos(iCount/(180/PI))),
            RayOrigin.Y + Trunc(Radius*sin(iCount/(180/PI))));
    MoveToEx(Canvas.Handle, RayOrigin.X, RayOrigin.Y, nil);
    LineTo(Canvas.Handle, RayOrigin.X + Trunc(Radius*cos(iCount/(180/PI))),
            RayOrigin.Y - Trunc(Radius*sin(iCount/(180/PI))));
    MoveToEx(Canvas.Handle, RayOrigin.X, RayOrigin.Y, nil);
    LineTo(Canvas.Handle, RayOrigin.X - Trunc(Radius*cos(iCount/(180/PI))),
            RayOrigin.Y - Trunc(Radius*sin(iCount/(180/PI))));
  end
end

```

```

MoveToEx(Canvas.Handle, RayOrigin.X, RayOrigin.Y, nil);
LineTo(Canvas.Handle, RayOrigin.X - Trunc(Radius*cos(iCount/(180/PI))),
      RayOrigin.Y + Trunc(Radius*sin(iCount/(180/PI))));

{Elimina la nueva pluma}
SelectObject(Canvas.Handle, OldPen);
DeleteObject(NewPen);
end;
end;

{Esta función dibuja texto relleno con gradiente}
procedure TForm1.DrawGradient;
var
  iCount: Integer;           // contador de bucle general
  TempRect: TRect;          // almacena un rectángulo temporal
  NewBrush, OldBrush: HBrush; // almacena una brocha nueva y vieja
begin
  {Comienza una definición de ruta dentro del contexto de dispositivo del
  formulario}
  BeginPath(Canvas.Handle);

  {Asigna al fondo un modo transparente. Esto es necesario porque la ruta será el
  área interior del texto. Sin esto, la ruta queda definida como el área exterior
  al texto}
  SetBkMode(Canvas.Handle, TRANSPARENT);

  {Salida de una palabra en el formulario. Esto es capturado como parte de la ruta}
  TextOut(Canvas.Handle, 50, 150, 'Delphi Rocks!', Length('Delphi Rocks!'));

  {Termina la definición de ruta}
  EndPath(Canvas.Handle);

  {Selecciona esta ruta como región de recorte para el contexto del formulario}
  SelectClipPath(Canvas.Handle, RGN_COPY);

  {Dibuja una serie de rectángulos dentro del texto, resultando un relleno de
  gradiente}
  for iCount := 0 to 64 do
    begin
      {Crea una nueva brocha y especifica un color de la nueva paleta}
      NewBrush := CreateSolidBrush(PaletteIndex(iCount+10));

      {Selecciona la brocha en el contexto de dispositivo}
      OldBrush := SelectObject(Form1.Canvas.Handle, NewBrush);

      {Crea un rectángulo, incrementado desde el lado izquierdo del texto}
      TempRect := Rect(Trunc(50 + iCount*Canvas.TextWidth('Delphi Rocks!')/64), 150,
        Trunc(50 + (iCount*Canvas.TextWidth('Delphi Rocks!')/64) +
          (Canvas.TextWidth('Delphi Rocks!')/64)),
        150 + Canvas.TextHeight('Delphi Rocks!'));

      {Rellena el rectángulo con la brocha. El producto final será la ilusión del
      texto relleno con gradiente}
      FillRect(Canvas.Handle, TempRect, NewBrush);
    end;
  end;

```

```

        {Elimina la nueva brocha}
        SelectObject(Form1.Canvas.Handle, OldBrush);
        DeleteObject(NewBrush);
    end;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Selecciona y activa la nueva paleta en el contexto del formulario}
    SelectPalette(Form1.Canvas.Handle, ThePalette, FALSE);
    RealizePalette(Form1.Canvas.Handle);

    {Dibuja radialmente texto relleno}
    DrawRadial;

    {Dibuja texto relleno con gradiente}
    DrawGradient;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {La paleta ya no es necesaria y se elimina}
    DeleteObject(ThePalette);
end;

```

Figura 4-19:  
Usando rutas  
para crear  
efectos de  
texto  
especiales

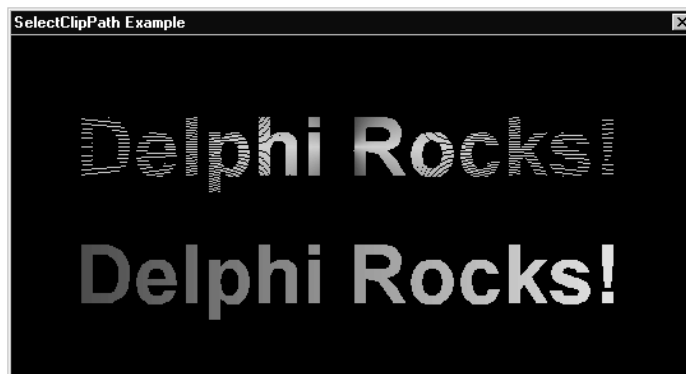


Tabla 4-13: Valores del parámetro Mode de SelectClipPath

Valor	Descripción
RGN_AND	La región resultante es la intersección de la región de recorte actual y la ruta.
RGN_COPY	La región resultante es la ruta.
RGN_DIFF	La región resultante es el área de la región de recorte actual que no está en el área de la ruta.
RGN_OR	La región resultante es la unión de la región de recorte actual y la ruta.

RGN\_XOR

La región resultante es la unión de la región de recorte actual y la ruta excluyendo cualquier zona solapada.

## SelectClipRgn Windows.Pas

### Sintaxis

```
SelectClipRgn(
    DC: HDC;           {manejador de un contexto de dispositivo}
    p2: HRGN           {manejador de una región}
): Integer;           {devuelve el tipo de región}
```

### Descripción

Esta función selecciona la región identificada por el parámetro *p2* como la región de recorte para el contexto de dispositivo especificado.

### Parámetros

*DC*: Un manejador del contexto de dispositivo cuya región de recorte será seleccionada.

*p2*: Un manejador de la región que será seleccionada como la región de recorte del contexto de dispositivo especificado. Esta función usa una copia de la región; la región original no se ve afectada y puede utilizarse en otras funciones. Se asume que las coordenadas están en unidades del dispositivo. Si a este parámetro se le asigna cero, la región de recorte actual del contexto de dispositivo es eliminada.

### Valor que devuelve

Esta función devuelve un resultado indicando el tipo de región de recorte seleccionada o una condición de error y puede ser un valor de la Tabla 4-14.

### Véase además

*ExtSelectClipRgn*, *GetClipRgn*, *OffsetClipRgn*, *SelectClipPath*

### Ejemplo

#### Listado 4-20: Dibujando sobre una región de recorte

```
var
    Form1: TForm1;
    ClippingRegion: HRGN;           // manejador de la región de recorte
    DraggingRect: Boolean;          // indica que una operación de arrastre está
                                    // ocurriendo
    TheRect: TRect;                 // rectángulo arrastrado
    MouseOffsetX,                   // usado para desplazar el rectángulo arrastrado
    MouseOffsetY: Integer;
```

#### implementation

```
{ $R *.DFM }
```

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Crea una región elíptica para ser usada en el recorte}
    ClippingRegion := CreateEllipticRgn(40, 40, ClientWidth-50, ClientHeight-50);

    {Crea un rectángulo}
    SetRect(TheRect, (ClientWidth div 2)-30, (ClientHeight div 2)-30,
        (ClientWidth div 2)+30, (ClientHeight div 2)+30);

    {Inicializa indicador de arrastre}
    DraggingRect := FALSE;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Selecciona la región elíptica como la región de recorte}
    SelectClipRgn(Canvas.Handle, ClippingRegion);

    {Indica si el rectángulo arrastrado es visible dentro de la región de recorte}
    if RectVisible(Canvas.Handle, TheRect) then
        Caption := Caption + 'Rect Visible'
    else
        Caption := Caption + 'Rect Not Visible';

    {Dibuja el perímetro de la región de recorte en rojo}
    Canvas.Brush.Color := clRed;
    FrameRgn(Canvas.Handle, ClippingRegion, Canvas.Brush.Handle, 4, 4);

    {Dibuja el rectángulo arrastrable en azul}
    Canvas.Brush.Color := clBlue;
    Canvas.FillRect(TheRect);
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    {Si se pulsó el ratón dentro del rectángulo arrastrable}
    if PtInRect(TheRect, Point(X, Y)) then
        begin
            {Indicar que una operación de arrastre ha comenzado}
            DraggingRect := TRUE;

            {Recupera el desplazamiento de las coordenadas actuales del ratón dentro del
            rectángulo arrastrado. Esto es usado cuando se mueve el rectángulo para que
            Se mantenga la posición original donde se hizo clic dentro del rectángulo.
            De lo contrario, cuando el rectángulo sea movido, la esquina superior
            izquierda del rectángulo sería ubicada en la posición del cursor del ratón.}
            MouseOffsetX := X - TheRect.Left;
            MouseOffsetY := Y - TheRect.Top;
        end;
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);

```

```

begin
    {Si una operación de arrastre está ocurriendo...}
    if DraggingRect then
    begin
        {...borra el área de dibujo del formulario}
        Form1.Canvas.Brush.Color := clBtnFace;
        Form1.Canvas.FillRect(TheRect);

        {Mueve el rectángulo arrastrado, desplazándolo a partir de la posición actual
        del ratón para que se mantenga el lugar original donde se hizo clic dentro
        del rectángulo}
        OffsetRect(TheRect, X - TheRect.Left - MouseOffsetX,
            Y - TheRect.Top - MouseOffsetY);

        {Inicializa la barra de título del formulario}
        Caption := 'SelectClipRgn Example - ';

        {Indica si el rectángulo está dentro la región elíptica}
        if RectInRegion(ClippingRegion, TheRect) then
            Caption := Caption+'Rect In Región - '
        else
            Caption := Caption+'Rect Not In Región - ';

        {Redibuja el formulario para mostrar los cambios}
        Form1.Repaint;
    end;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    {Indicar que la operación de arrastre se ha detenido}
    DraggingRect := FALSE;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {La región ya no es necesaria y se elimina}
    DeleteObject(ClippingRegion);
end;

```

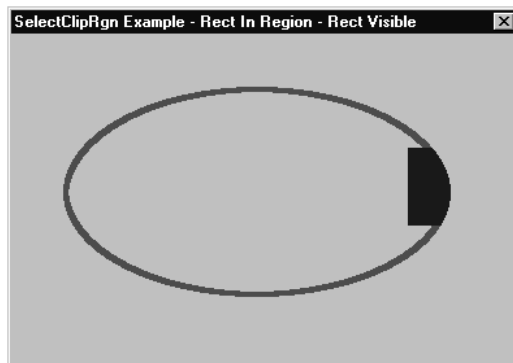


Figura 4-20:  
La región de  
recorte

Tabla 4-14: Valores que devuelve SelectClipRgn

Valor	Descripción
NULLREGION	Indica una región vacía.
SIMPLEREGION	Indica una región rectangular simple.
COMPLEXREGION	Indica una región que consta de más de un rectángulo.
ERROR	Indica que ha ocurrido un error.

**SetRect**      **Windows.Pas****Sintaxis**

```
SetRect(
    var lprc: TRect;           {puntero a un registro TRect}
    xLeft: Integer;            {coordenada horizontal superior izquierda}
    yTop: Integer;             {coordenada vertical superior izquierda}
    xRight: Integer;           {coordenada horizontal inferior derecha}
    yBottom: Integer;          {coordenada vertical inferior derecha}
); BOOL;                      {devuelve TRUE o FALSE}
```

**Descripción**

Esta función asigna a las coordenadas del rectángulo al que apunta el parámetro *lprc* las coordenadas especificadas.

**Parámetros**

*lprc*: Un puntero a un registro *TRect* cuyas coordenadas serán seleccionadas.

*xLeft*: Especifica la coordenada horizontal de la esquina superior izquierda del rectángulo.

*yTop*: Especifica la coordenada vertical de la esquina superior izquierda del rectángulo.

*xRight*: Especifica la coordenada horizontal de la esquina inferior derecha del rectángulo.

*yBottom*: Especifica la coordenada vertical de la esquina inferior derecha del rectángulo.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CopyRect*, *IntersectRect*, *SetRectEmpty*, *SetRectRgn*, *SubtractRect*, *UnionRect*

**Ejemplo**

Vea el Listado 4-20 bajo *SelectClipRgn*, y otros ejemplos a lo largo este capítulo.

**SetRectEmpty Windows.Pas****Sintaxis**

```
SetRectEmpty(
    var lprc: TRect          {puntero a un registro TRect}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función pone a cero todas las coordenadas del rectángulo especificado.

**Parámetros**

*lprc*: Puntero a un registro *TRect* cuyas coordenadas serán puestas a cero.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CopyRect*, *IntersectRect*, *SetRect*, *SetRectRgn*, *SubtractRect*, *UnionRect*

**Ejemplo****Listado 4-21: Vaciando un rectángulo**

```
var
    Form1: TForm1;
    TheRect: TRect;    // almacena un rectángulo

implementation

{$R *.DFM}

procedure TForm1.FormActivate(Sender: TObject);
begin
    {Crea un nuevo rectángulo}
    SetRect(TheRect, 8, 40, 169, 160);
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
    {Muestra el rectángulo}
    Form1.Canvas.Brush.Color := clRed;
    Form1.Canvas.FillRect(TheRect);
```

```

end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {Vacía el rectángulo}
  SetRectEmpty(TheRect);

  {Indica si el rectángulo está vacío}
  if IsRectEmpty(TheRect) then
    Button1.Caption := 'Rectangle is empty'
  else
    Button1.Caption := 'Rectangle is not empty';

  {Redibuja el formulario para mostrar los cambios}
  Form1.Repaint;
end;

```

## SetRectRgn Windows.Pas

### Sintaxis

```

SetRectRgn(
  Rgn: HRgn;           {manejador de una región preexistente}
  X1: Integer;         {coordenada horizontal superior izquierda}
  Y1: Integer;         {coordenada vertical superior izquierda}
  X2: Integer;         {coordenada horizontal inferior derecha}
  Y2: Integer;         {coordenada vertical inferior derecha}
); BOOL;               {devuelve TRUE o FALSE}

```

### Descripción

Esta función convierte la región identificada por el parámetro *Rgn* en una región rectangular situada en las coordenadas especificadas. Observe que los bordes inferior y derecho del rectángulo son excluidos de la región.

### Parámetros

*Rgn*: Especifica un manejador de la región que será convertida en una región rectangular.

*X1*: Especifica la coordenada horizontal superior izquierda de la región rectangular, en unidades lógicas.

*Y1*: Especifica la coordenada vertical superior izquierda de la región rectangular, en unidades lógicas.

*X2*: Especifica la coordenada horizontal inferior derecha de la región rectangular, en unidades lógicas.

*Y2*: Especifica la coordenada vertical inferior derecha de la región rectangular, en unidades lógicas.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*CreateRectRgn*, *CreateRectRgnIndirect*, *CreateRoundRectRgn*, *SetRect*

**Ejemplo**

Vea el Listado 4-11 bajo *EqualRgn*.

**SetWindowRgn****Windows.Pas****Sintaxis**

```
SetWindowRgn(
    hWnd: HWND;           {manejador de una ventana}
    hRgn: HRGN;           {manejador de una región}
    bRedraw: BOOL         {opción de redibujo de ventana}
): BOOL;                 {devuelve TRUE o FALSE}
```

**Descripción**

Esta función asigna la región de la ventana especificada a la región identificada por el parámetro *hRgn*. La región de la ventana determina el área dentro de la ventana donde está permitido dibujar y Windows no permitirá que se haga ningún dibujo fuera de la región de la ventana. Cuando esta función retorna, el sistema operativo es responsable de la región especificada y ésta no debe ser usada por funciones subsiguientes. Esta función es típicamente usada para crear ventanas con una forma no rectangular.

**Parámetros**

*hWnd*: Un manejador de la ventana cuya región será asignada.

*hRgn*: Un manejador de la región que será usada como la región de la ventana. Las coordenadas de esta región son relativas a la ventana, no al área cliente. Si a este parámetro se le asigna cero, la región de la ventana es reiniciada a la región por defecto.

*bRedraw*: Indica si la ventana debe ser redibujada cuando la región es asignada. Si a este valor se le asigna TRUE la ventana es redibujada y los mensajes *WM\_WINDOWPOSCHANGING* y *WM\_WINDOWPOSCHANGED* son enviados a la ventana. Si se le asigna FALSE, la ventana no es redibujada.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

Véase además

*CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, ExtCreateRegion, ExtSelectClipRgn, SelectClipRgn*

Ejemplo

#### Listado 4-22: Creando una ventana redonda para una aplicación de un reloj analógico

```

var
    Form1: TForm1;
    OriginX, OriginY: Integer;    // almacena las coordenadas centrales la ventana

implementation

uses Math;

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
    NewShape: HRGN;    // almacena la región
begin
    {Crea una región circular}
    NewShape := CreateEllipticRgn(GetSystemMetrics(SM_CXBORDER)+3,
                                   GetSystemMetrics(SM_CYCAPTION)+3,
                                   GetSystemMetrics(SM_CXBORDER)+103,
                                   GetSystemMetrics(SM_CYCAPTION)+103);

    {Determina el centro del círculo. Se utiliza al dibujar los números del reloj}
    OriginX := (GetSystemMetrics(SM_CXBORDER) + 90) div 2;
    OriginY := ((GetSystemMetrics(SM_CXBORDER) + 90) div 2) - 3;

    {Asigna la región de la ventana a una región circular. Esto creará una ventana
    redondeada}
    SetWindowRgn(Handle, NewShape, TRUE);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    iCount: Integer;           // variable de control de bucle general
    Hour, Minute, Second, MilSec: Word; // usado para decodificar el tiempo
begin
    {Asigna el modo de fondo a transparente para dibujar texto}
    SetBkMode(Canvas.Handle, TRANSPARENT);

    {Dibuja un bisel resaltado}
    Canvas.Pen.Color := clWhite;
    Canvas.Pen.Width := 2;
    Arc(Canvas.Handle, 1, 1, 98, 98, 98, 1, 1, 98);

    {Dibuja un bisel opaco}
    Canvas.Pen.Color := clBtnShadow;
  
```

```

Arc(Canvas.Handle, 1, 1, 98, 98, 1, 98, 98, 1);

{Para todas las horas del día...}
for iCount := 1 to 12 do
begin
    {...dibuja las horas de forma circular alrededor de la ventana}
    Canvas.TextOut(Trunc(Sin(((360/12)*iCount)*(PI/180))*40) + OriginX,
        Trunc(-Cos(-((360/12)*iCount)*(PI/180))*40) + OriginY,
        IntToStr(iCount));
end;

{Recupera la hora actual en un formato usable}
DecodeTime(Now, Hour, Minute, Second, MilSec);

{Convierte la hora militar en civil}
if Hour>12 then Hour := Hour-12;

{Dibuja el horario}
Canvas.Pen.Color := clBlack;
Canvas.MoveTo(50, 50);
Canvas.LineTo(Trunc(Sin(((360/12)*Hour)*(PI/180))*30)+50,
    Trunc(-Cos(-((360/12)*Hour)*(PI/180))*30)+50);

{Dibuja el minuterero}
Canvas.MoveTo(50, 50);
Canvas.LineTo(Trunc(Sin(((360/60)*Minute)*(PI/180))*40)+50,
    Trunc(-Cos(-((360/60)*Minute)*(PI/180))*40)+50);

{Dibuja el secundario}
Canvas.Pen.Color := clRed;
Canvas.MoveTo(50, 50);
Canvas.LineTo(Trunc(Sin(((360/60)*Second)*(PI/180))*40)+50,
    Trunc(-Cos(-((360/60)*Second)*(PI/180))*40)+50);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    {Redibuja el formulario una vez por segundo}
    Repaint;
end;

procedure TForm1.WMNCHitTest(var Msg: TWMNCHitTest);
begin
    {Esto permite al usuario arrastrar la ventana haciendo clic en cualquier lugar
    del formulario}
    inherited;
    Msg.Result := HTCAPTION;
end;

```

Figura 4-21:  
El reloj  
analógico



## SubtractRect Windows.Pas

### Sintaxis

```
SubtractRect(
  var lprcDst: TRect;      {puntero al registro TRect de destino}
  const lprcSrc1: TRect;   {puntero al primer rectángulo}
  const lprcSrc2: TRect    {puntero al segundo rectángulo}
): BOOL;                  {devuelve TRUE o FALSE}
```

### Descripción

Esta función sustrae las coordenadas del rectángulo al que apunta el parámetro *lprcSrc2*, de las coordenadas del rectángulo al que apunta el parámetro *lprcSrc1*. Observe que esta función tiene éxito sólo cuando los dos rectángulos se intersectan completamente en uno de los dos ejes, horizontal o vertical.

### Parámetros

*lprcDst*: Un puntero a un registro *TRect* que recibe las coordenadas resultantes de la sustracción del rectángulo al que apunta el parámetro *lprcSrc2*, del rectángulo al que apunta el parámetro *lprcSrc1*.

*lprcSrc1*: Un puntero a un registro *TRect* desde el cual, el rectángulo al que apunta el parámetro *lprcSrc2* es sustraído.

*lprcSrc2*: Un puntero a un registro *TRect* que contiene el rectángulo que será sustraído del rectángulo al que apunta el parámetro *lprcSrc1*.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### Véase además

*EqualRect*, *IntersectRect*, *SetRect*, *UnionRect*

### Ejemplo

#### Listado 4-23: Sustrayendo un rectángulo de otro

```
procedure TForm1.FormPaint(Sender: TObject);
var
  Rect1, Rect2, Subtract: TRect;    // almacena los rectángulos
begin
```

```

{Asigna las coordenadas de los dos rectángulos}
SetRect(Rect1, 10, 10, 110, 110);
SetRect(Rect2, 60, 10, 160, 160);

{Sustraer el rectángulo 2 del rectángulo 1}
SubtractRect(Subtract, Rect1, Rect2);

with Form1.Canvas do
begin
  {Inicializa objetos de áreas de dibujo para dibujar contornos}
  Brush.Style := bsClear;
  Pen.Style := psSolid;

  {Dibuja los contornos de los rectángulos 1 y 2}
  Rectangle(Rect1.Left, Rect1.Top, Rect1.Right, Rect1.Bottom);
  Rectangle(Rect2.Left, Rect2.Top, Rect2.Right, Rect2.Bottom);

  {Inicializa objetos de áreas de dibujo para dibujar el resultado}
  Brush.Style := bsSolid;
  Brush.Color := clRed;

  {Rellena el rectángulo resultante con rojo}
  FillRect(Subtract);
end;
end;

```

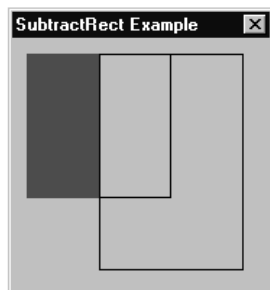


Figura 4-22:  
El rectángulo  
resultante

## UnionRect

## Windows.Pas

### Sintaxis

```

UnionRect(
  var lprcDst: TRect;           {puntero al registro TRect de destino}
  const lprcSrc1: TRect;        {puntero al primer rectángulo}
  const lprcSrc2: TRect;        {puntero al segundo rectángulo}
): BOOL;                       {devuelve TRUE o FALSE}

```

### Descripción

Esta función crea un rectángulo que es la unión de los rectángulos a los que apuntan los parámetros *lprcSrc1* y *lprcSrc2*.

**Parámetros**

*lprcDst*: Un puntero a un registro *TRect* que recibe las coordenadas resultantes de la unión de los rectángulos a los que apuntan los parámetros *lprcSrc1* y *lprcSrc2*.

*lprcSrc1*: Un puntero a un registro *TRect* que contiene uno de los rectángulos que será unido.

*lprcSrc2*: Un puntero a un registro *TRect* que contiene uno de los rectángulos que será unido.

**Valor que devuelve**

Si la función tiene éxito y el rectángulo al que apunta el parámetro *lprcDst* no está vacío, devuelve TRUE. Si la función falla o el rectángulo al que apunta el parámetro *lprcDst* está vacío, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*EqualRect*, *InflateRect*, *IntersectRect*, *IsRectEmpty*, *SetRect*, *SetRectEmpty*, *SubtractRect*

**Ejemplo**

Vea el Listado 4-16 bajo *OffsetRect*.

**WidenPath****Windows.Pas****Sintaxis**

```
WidenPath(
    DC: HDC                      {manejador de un contexto de dispositivo}
); BOOL;                        {devuelve TRUE o FALSE}
```

**Descripción**

Esta función ensancha la ruta contenida en el contexto de dispositivo especificado. La nueva ruta se define como el área que sería dibujada si la función *StrokePath* fuese llamada usando la pluma actualmente seleccionada. Cualquier curva de Bézier que defina una parte de la ruta es convertida a segmentos de líneas.

**Parámetros**

*DC*: Un manejador del contexto de dispositivo que contiene la ruta que será ensanchada.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*BeginPath, CreatePen, EndPath, ExtCreatePen, FlattenPath, GetPath, PathToRegion, SetMiterLimit*

Ejemplo

**Listado 4-24: Dibujando el contorno de un texto**

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    {Comienza una definición de ruta}
    BeginPath(Canvas.Handle);

    {Asigna el modo TRANSPARENT al fondo para que la ruta sea definida como el área
    dentro del texto}
    SetBkMode(Canvas.Handle, TRANSPARENT);

    {Dibuja algún texto. Observe que a la fuente del formulario se le asigna 48
    Arial, negrita}
    TextOut(Canvas.Handle, 20, 20, 'Delphi Rocks!', Length('Delphi Rocks!'));

    {Termina la definición de ruta}
    EndPath(Canvas.Handle);

    {Modifica la pluma para que sea de 4 píxeles de ancho}
    Canvas.Pen.Width := 4;

    {Ensancha la ruta definida por el texto. Debido al ancho de la pluma, el
    ancho del contorno de las letras de la ruta es de 4 píxeles}
    WidenPath(Canvas.Handle);

    {Reinicia el ancho de la pluma y el color de la brocha para dibujar la ruta}
    Canvas.Pen.Width := 1;
    Canvas.Brush.Color := clRed;

    {Asigna el modo de relleno para que la ruta sea rellenada correctamente}
    SetPolyFillMode(Canvas.Handle, WINDING);

    {Rellena la ruta con la brocha roja}
    FillPath(Canvas.Handle);
end;
```

Figura 4-23:  
El texto  
delineado



**Capítulo 5**

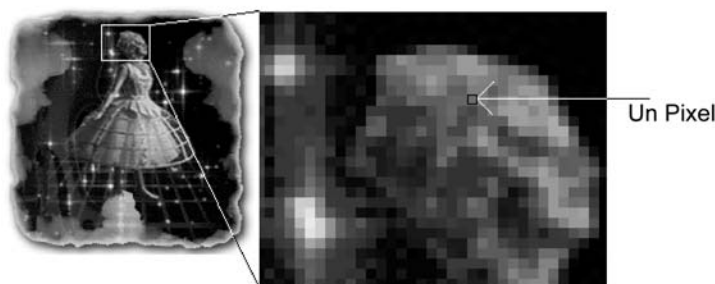
# Funciones de mapas de bits y metaфicheros

Es difícil que se escriba una aplicación para Windows que no ejecute alguna clase de manipulación de imágenes. Las imágenes gráficas pueden ser clasificadas en dos categorías: de mapas de bits y vectoriales. El API Win32 ofrece al desarrollador una amplia variedad de funciones para manipular estos dos tipos de imágenes. Windows originariamente soporta formatos de imágenes de mapas de bits y de vectores conocidos como metaфicheros. Este capítulo describe las funciones del API Win32 que pueden utilizarse para manipular ambos tipos de gráficos.

Observe que los programas de ejemplo de este capítulo asumen que el controlador de vídeo ha sido configurado para trabajar con 256 colores. Algunos ejemplos pueden no funcionar apropiadamente si la profundidad de colores es diferente.

## Mapas de bits

Figura 5-1:  
Un píxel es el  
elemento más  
pequeño de  
una imagen



Un *mapa de bits* es un *array* de bytes que almacena la información sobre el color de los elementos de la imagen conocidos como píxeles. Un píxel es un pequeño cuadrado de color que, cuando es visto en conjunto con los que lo rodean, conforma una imagen de mapa de bits, como se ilustra en la Figura 5-1.

La cantidad de bits que son utilizados para describir un píxel individual puede variar ampliamente, de acuerdo con la profundidad del color de la imagen. Los píxeles de una

imagen de 16 colores pueden ser descritos con 4 bits por píxel; de esta manera, un byte de memoria puede contener 2 píxeles de la imagen. Una imagen de 256 colores utiliza 1 byte para cada píxel y una imagen en color verdadero (16.7 millones de colores) usa 3 bytes para cada píxel. Vea la Tabla 5-6 bajo *CreateDIBSection* para ver una descripción completa de la profundidad de colores de los mapas de bits.

### **Mapas de bits dependientes del dispositivo**

Los mapas de bits dependientes del dispositivo son llamados de esa manera precisamente porque dependen fuertemente del dispositivo sobre el cual son mostrados. Estos mapas de bits sólo almacenan información sobre el ancho y la altura, el formato de colores y el *array* de píxeles que describe la imagen. No contienen ninguna información concerniente a la paleta de colores de la imagen o sobre su resolución original. Este tipo de mapa de bits era el único accesible para los programadores iniciales de Windows y sólo existe en la actualidad por problemas de compatibilidad. Los desarrolladores de Win32 deben usar mapas de bits independientes del dispositivo.

### **Mapas de bits independientes del dispositivo**

Los mapas de bits independientes del dispositivo contienen más información sobre su imagen que los mapas de bits dependientes del dispositivo. Por ejemplo, los mapas de bits independientes del dispositivo contienen la paleta de colores para la imagen, la resolución del dispositivo sobre el cual fue originalmente creado el mapa de bits y un indicador de compresión de datos. Quizás la ventaja más grande de los mapas de bits independientes del dispositivo es que el desarrollador tiene acceso directo a los bytes que conforman los píxeles del mapa de bits. Esto permite a un desarrollador modificar la imagen directamente, al contrario de los mapas de bits dependientes del dispositivo, que requieren que el desarrollador utilice las funciones del GDI para manipular la imagen del mapa de bits.

Por defecto, los mapas de bits independientes del dispositivo son orientados de un modo abajo-arriba, lo cual significa que el origen de los píxeles del mapa de bits comienza en la esquina inferior izquierda de la imagen. Sin embargo, un mapa de bits independiente del dispositivo puede ser orientado de un modo arriba-abajo como los mapas de bits dependientes del dispositivo, simplemente dándole un valor negativo a su altura.

### **Operaciones de mapa de bits**

Existen numerosas funciones para mostrar imágenes de mapas de bits en la pantalla. La acción de copiar los píxeles de un mapa de bits en la pantalla es conocida como *Blt* (pronunciada “blit”), que significa *Bit bLock Transfer* (transferencia de bloque de bits). Algunas funciones, tales como *BitBlt* y *StretchBlt*, están concebidas para ser usadas con mapas de bits dependientes del dispositivo y requieren contextos de dispositivos como origen y destino de la acción de transferencia de píxeles. Los mapas de bits

independientes del dispositivo utilizan las funciones *SetDIBitsToDevice* y *StretchDIBits* para copiar el *DIB* directamente al contexto de dispositivo.

Algunas funciones, tales como *StretchBlt* y *StretchDIBits*, permiten que el mapa de bits sea dibujado con un tamaño diferente del original. Windows le añadirá o quitará píxeles según determine el modo de compresión del contexto de dispositivo de destino. El modo de compresión del contexto de dispositivo de destino puede ser seleccionado llamando la función *SetStretchBltMode*.

**Escalado** Un mapa de bits puede ser escalado y mantener aún su relación de aspecto, si se encuentra el lado más pequeño del área rectangular que define el nuevo tamaño del mapa de bits y se determina la relación de esta nueva dimensión respecto al tamaño original del mismo lado del mapa de bits. Por ejemplo, si se amplía un mapa de bits de 5 x 10 píxeles para que se muestre sobre un área de 10 x 20 píxeles, el lado más pequeño de esta nueva área es 10 (la altura). La altura del mapa de bits original es 5, y  $10 \div 5$  es 2, para un incremento de tamaño de un 200 por ciento. Multiplicando todos los lados de las dimensiones originales del mapa de bits por 2 dará como resultado un nuevo tamaño del mapa de bits de 10 x 20 píxeles, conservando así su relación de aspecto original. El siguiente ejemplo muestra cómo usar esta fórmula para permitir al usuario escalar un mapa de bits a cualquier tamaño conservando siempre la relación de aspecto original del mapa de bits.

#### Listado 5-1: Escalando un mapa de bits y conservando su relación de aspecto original

```
var
  Form1: TForm1;
  ScaleRect: TRect;      // almacena las coordenadas del rectángulo a dibujar
  IsDragging: Boolean;   // indica si el usuario está dibujando un rectángulo
  ScaledImage: TBitmap;  // almacena la imagen que será escalada

implementation

{$R *.DFM}

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  {Indica que el usuario está arrastrando un rectángulo}
  IsDragging := TRUE;

  {Inicializa el rectángulo}
  ScaleRect := Rect(X, Y, X, Y);
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  {Si estamos arrastrando un rectángulo}
  if IsDragging then
  begin
    {Dibuja sobre el rectángulo actual para borrarlo}
```

```

Canvas.Pen.Style := psSolid;
Canvas.Pen.Color := clBlack;
Canvas.Pen.Mode := pmNot;
Canvas.Brush.Style := bsClear;
Canvas.Rectangle(ScaleRect.Left, ScaleRect.Top, ScaleRect.Right,
                  ScaleRect.Bottom);

{Convierte las coordenadas del rectángulo dibujado a las nuevas coordenadas}
ScaleRect := Rect(ScaleRect.Left, ScaleRect.Top, X, Y);

{Dibuja el nuevo rectángulo}
Canvas.Rectangle(ScaleRect.Left, ScaleRect.Top, ScaleRect.Right,
                  ScaleRect.Bottom);
end;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
var
    Ratio: Real;           // almacena la relación de compresión
begin
    {Indica que el usuario no está arrastrando ya un rectángulo}
    IsDragging := FALSE;

    {Borra la ventana completa}
    Canvas.Brush.Color := clBtnFace;
    Canvas.Brush.Style := bsSolid;
    Canvas.FillRect(Form1.ClientRect);

    {Dibuja un rectángulo vacío en las actuales coordenadas del rectángulo}
    Canvas.Brush.Style := bsClear;
    Canvas.Rectangle(ScaleRect.Left, ScaleRect.Top, ScaleRect.Right,
                     ScaleRect.Bottom);

    {Selecciona la paleta de la imagen en el área de dibujo del formulario
    y la activa}
    SelectPalette(Canvas.Handle, ScaledImage.Palette, FALSE);
    RealizePalette(Canvas.Handle);

    {Determina la relación de aspecto apropiada}
    if ScaleRect.Right - ScaleRect.Left < ScaleRect.Bottom - ScaleRect.Top then
        Ratio := (ScaleRect.Right - ScaleRect.Left) / ScaledImage.Width
    else
        Ratio := (ScaleRect.Bottom - ScaleRect.Top) / ScaledImage.Height;

    {Copia la imagen al área de dibujo, centrada en el rectángulo y escalada de
    manera que la relación de aspecto es conservada}
    StretchBlt(Canvas.Handle,
                ScaleRect.Left + (((ScaleRect.Right - ScaleRect.Left) div 2) -
                (Trunc(ScaledImage.Width * Ratio) div 2)),
                ScaleRect.Top + (((ScaleRect.Bottom - ScaleRect.Top) div 2) -
                (Trunc(ScaledImage.Height * Ratio) div 2)),
                Trunc(ScaledImage.Width * Ratio),
                Trunc(ScaledImage.Height * Ratio),

```

```

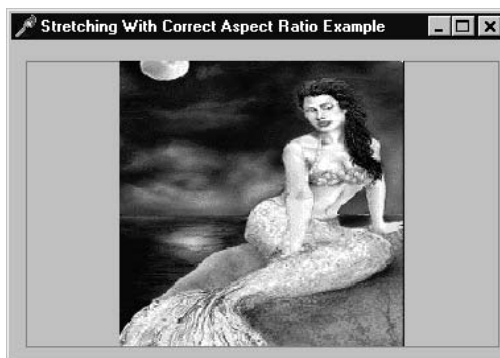
        ScaledImage.Canvas.Handle, 0, 0,
        ScaledImage.Width, ScaledImage.Height, SRCCOPY);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Crea y carga la imagen que será escalada}
    ScaledImage := TBitmap.Create;
    ScaledImage.LoadFromFile('Image9.bmp');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Libera la imagen del mapa de bits}
    ScaledImage.Free;
end;

```

Figura 5-2: El mapa de bits escalado, con la relación de aspecto corregida



**Operaciones de barrido** En adición a la simple copia de píxeles de un mapa de bits a la pantalla, ciertas funciones pueden ejecutar operaciones de barrido sobre los píxeles. Una operación de barrido determina cómo los píxeles del origen, el destino y la brocha seleccionada del contexto de dispositivo de destino son combinados. Las operaciones de barrido más comúnmente utilizadas se describen con las funciones correspondientes a lo largo de este capítulo. Si bien hay 256 operaciones de barrido, algunas no son aplicables a todas las funciones. En el Apéndice B se ofrece una descripción completa de todas las operaciones de barrido.

Ciertas operaciones de barrido pueden ser usadas para producir efectos especiales, tales como la ilusión de transparencia. Para copiar un mapa de bits a un destino usando operaciones de barrido, simulando el efecto de píxeles transparentes, la aplicación debe tener dos versiones del mapa de bits que será copiado, conocidas como las máscaras **and** y **or**. La imagen de la máscara **and** es una silueta monocromática del mapa de bits original. Los píxeles blancos indican dónde se mostrará el fondo (a través de los píxeles transparentes) y los píxeles negros indican dónde aparecerá la imagen actual del mapa de bits. La máscara **or** contiene la imagen real que será copiada y en ella los

píxeles negros de la imagen indican transparencia. Primero, la aplicación copia la máscara **and** al destino usando la operación de barrido *SRCAND*. Esto combina los píxeles del origen y el destino usando una operación booleana **and**. Los píxeles blancos de la máscara **and** preservarán los píxeles originales de la imagen de fondo, mientras que los píxeles negros de la máscara **and** convertirán los píxeles en la imagen de fondo en negros, resultando en un área labrada para la imagen final del mapa de bits. Una vez que este primer proceso ha sido ejecutado, la aplicación copia la máscara **or** en el destino usando la operación de barrido *SRCPAINT*. Esto combina los píxeles del origen y el destino usando la operación booleana **or**. Los píxeles negros de la máscara **or** preservarán los píxeles originales del mapa de bits, mientras que los píxeles originales de la imagen caerán sobre los píxeles negros del fondo producido en el primer paso. El resultado es la ilusión de una operación de copia transparente. El siguiente ejemplo demuestra la técnica de usar máscaras para producir transparencias con mapas de bits.

#### Listado 5-2: Mostrando un mapa de bits con píxeles transparentes

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {Copia la imagen de fondo al destino}
  Image3.Canvas.Draw(0, 0, Image1.Picture.Bitmap);

  {Combina la imagen de la máscara and con la imagen de fondo en el destino,
  usando una operación booleana and. Esto produce una primera imagen "labrada"}
  BitBlt(Image3.Canvas.Handle, (Image3.Width div 2) - (Image2.Width div 2),
    (Image3.Height div 2) - (Image2.Height div 2), Image2.Width,
    Image2.Height, Image2.Canvas.Handle, 0, 0, SRCAND);

  {Copia el resultado del paso 1 en la imagen de fondo usada en el paso 2}
  Image4.Canvas.Draw(0, 0, Image3.Picture.Bitmap);

  {Copia la imagen de fondo resultante del paso 1 en el destino}
  Image6.Canvas.Draw(0, 0, Image4.Picture.Bitmap);

  {Combina la imagen de la máscara or con el resultado del paso 1 en el destino
  usando una operación booleana or. Esto copia la imagen de primer plano en el
  área "labrada" del paso 1, al tiempo que mantiene los píxeles alrededor,
  creando así la ilusión de transparencia}
  BitBlt(Image6.Canvas.Handle, (Image6.Width div 2) - (Image5.Width div 2),
    (Image6.Height div 2) - (Image5.Height div 2), Image5.Width,
    Image5.Height, Image5.Canvas.Handle, 0, 0, SRCPAINT);
end;
```

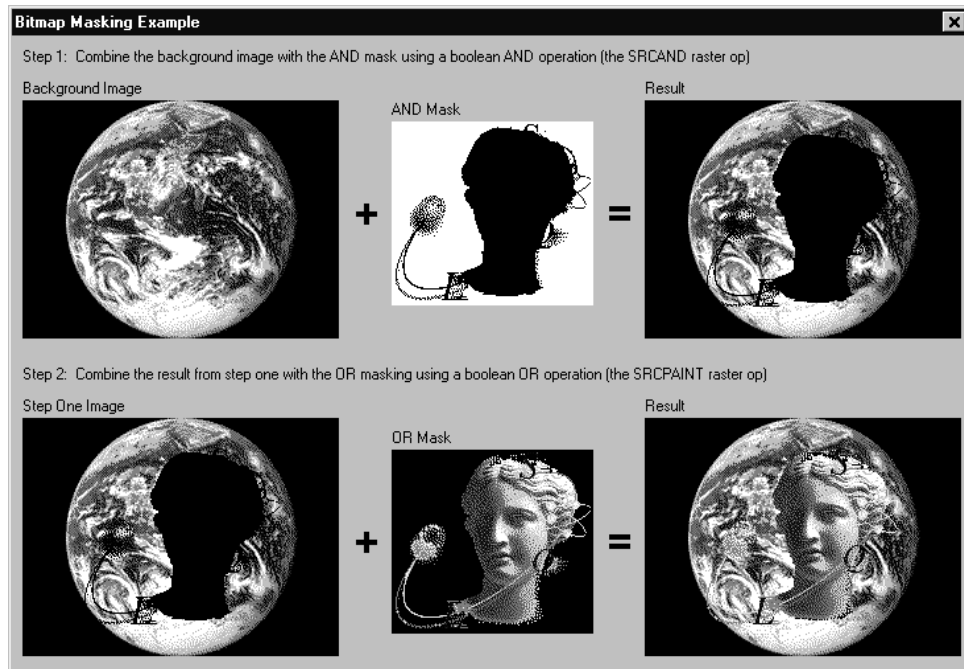


Figura 5-3: El mapa de bits copiado transparente-

**Los DIB y el GDI** Si bien un mapa de bits independiente de dispositivo difiere de muchas maneras de un mapa de bits dependiente de dispositivo, un DIB puede ser seleccionado dentro de un contexto de dispositivo y modificado usando funciones del GDI, como si fuera un mapa de bits dependiente de dispositivo corriente. Esto da al desarrollador una tremenda flexibilidad en el tratamiento de mapas de bits, dado que las funciones de dibujo a la medida pueden ser utilizadas junto con funciones corrientes de dibujo del GDI para manipular imágenes de mapas de bits. El siguiente ejemplo muestra cómo se selecciona un DIB en un contexto de dispositivo y se dibuja sobre el mapa de bits usando funciones de dibujo del GDI.

#### Listado 5-3: Manipulando un DIB usando funciones de dibujo del GDI

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Dib: HBITMAP;           // almacena un manejador del DIB
  DibInfo: PBitmapInfo;   // puntero a la estructura de información del mapa de
                          // bits
  BitsPtr: PByte;         // almacena un puntero a los bits del mapa de bits
  ReferenceDC: HDC;       // puntero al contexto de dispositivo de referencia
  iCount: Integer;       // contador de bucle general
  OldBitmap: HBITMAP;     // almacena un manejador del viejo mapa de bits del DC
  ScratchCanvas: TCanvas; // almacena un área de dibujo temporal para dibujar

  APolygon: array[0..2] of TPoint;           // almacena un polígono
```

```

SystemPalette: array[0..255] of TPaletteEntry; // necesario para convertir
                                                // la paleta del sistema en una
                                                // paleta compatible de DIB

begin
  {Reserva la memoria necesaria para el registro de información del mapa de bits}
  GetMem(DibInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));

  {Inicializa la información del mapa de bits}
  DibInfo^.bmiHeader.biWidth      := 64;    // crea un DIB de 64 X 64 píxeles
  DibInfo^.bmiHeader.biHeight     := -64;   // orientado de arriba abajo
  DibInfo^.bmiHeader.biPlanes     := 1;
  DibInfo^.bmiHeader.biBitCount   := 8;     // 256 colores
  DibInfo^.bmiHeader.biCompression := BI_RGB; // sin compresión
  DibInfo^.bmiHeader.biSizeImage  := 0;     // Windows determina el tamaño
  DibInfo^.bmiHeader.biXPelsPerMeter := 0;
  DibInfo^.bmiHeader.biYPelsPerMeter := 0;
  DibInfo^.bmiHeader.biClrUsed    := 0;
  DibInfo^.bmiHeader.biClrImportant := 0;
  DibInfo^.bmiHeader.biSize       := SizeOf(TBitmapInfoHeader);

  {Recupera la paleta actual del sistema}
  GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256, SystemPalette);

  {La paleta del sistema es devuelta como un array de registros TPaletteEntry,
   que almacena los colores de la paleta en la forma (Red, Green, Blue). Sin
   embargo, el campo bmiColors del registro TBitmapInfo del registro requiere un
   array de registros del tipo TRGBQuad, que almacena los colores de la paleta en
   la forma (Blue, Green, Red). Debido a ésto, debemos convertir los registros
   TPaletteEntry en registros TRGBQuad para obtener los colores correctos.}
  for iCount := 0 to 255 do
  begin
    DibInfo^.bmiColors[iCount].rgbBlue   := SystemPalette[iCount].peBlue;
    DibInfo^.bmiColors[iCount].rgbRed    := SystemPalette[iCount].peRed;
    DibInfo^.bmiColors[iCount].rgbGreen  := SystemPalette[iCount].peGreen;
    DibInfo^.bmiColors[iCount].rgbReserved := 0;
  end;

  {Crea un contexto de dispositivo basado en memoria}
  ReferenceDC := CreateCompatibleDC(0);

  {Crea un DIB sobre el contexto de dispositivo en memoria, a partir de la
   información inicializada del mapa de bits}
  Dib := CreateDIBSection(ReferenceDC, DibInfo^, DIB_RGB_COLORS,
    Pointer(BitsPtr), 0, 0);

  {Selecciona el DIB en el contexto de dispositivo}
  OldBitmap := SelectObject(ReferenceDC, Dib);

  {Crea un área de dibujo y asigna su manejador del contexto de dispositivo creado}
  ScratchCanvas := TCanvas.Create;
  ScratchCanvas.Handle := ReferenceDC;

  {Rellena el área de dibujo con rojo}
  ScratchCanvas.Brush.Color := clRed;
  ScratchCanvas.FillRect(ScratchCanvas.ClipRect);

```

```

{Dibuja un círculo verde}
ScratchCanvas.Brush.Color := clLime;
ScratchCanvas.Ellipse(0, 0, 32, 32);

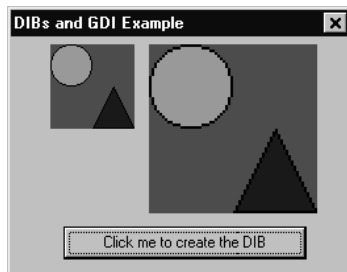
{Dibuja un triángulo}
ScratchCanvas.Brush.Color := clBlue;
APolygon[0] := Point(63, 63);
APolygon[1] := Point(32, 63);
APolygon[2] := Point(48, 32);
ScratchCanvas.Polygon(APolygon);

{Arriba se han utilizado funciones DIB}
SetDIBitsToDevice(Form1.Canvas.Handle, 30, 5, 64, 64, 0, 0, 0, 64, BitsPtr,
                  DibInfo^, DIB_RGB_COLORS);

{Dibuja el DIB de nuevo, pero esta vez lo agranda al doble de su tamaño}
StretchDIBits(Form1.Canvas.Handle, 105, 5, 128, 128, 0, 0, 64, 64, BitsPtr,
               DibInfo^, DIB_RGB_COLORS, SRCCOPY);

{DIB no es ya necesario y se elimina junto con el área de dibujo y la memoria
 reservada para el registro de los datos}
SelectObject(ReferenceDC, OldBitmap);
ScratchCanvas.Free;
DeleteObject(Dib);
DeleteDC(ReferenceDC);
FreeMem(DibInfo, SizeOf(TBitmapInfo)+256*SizeOf(TRGBQuad));
end;
```

Figura 5-4: El mapa de bits independiente del dispositivo modificado



## Metaфicheros

Un *metaфichero* es un formato gráfico basado en vectores para el almacenamiento de imágenes. La imagen es almacenada como una serie de instrucciones que describen cómo dibujarla, en lugar de un *array* de píxeles que explícitamente describen la imagen como un mapa de bits. Esto da a los metaфicheros una cierta independencia de los dispositivos ya que pueden ser mostrados en su tamaño y resolución original tanto en una impresora como en una pantalla.

Específicamente, un metaфichero es una colección de registros de metaфichero que se corresponden con llamadas a funciones del GDI para dibujar líneas y figuras, rellenar

regiones, etc. Al mostrarse un metaarchivo, se ejecutan en secuencia las funciones del GDI almacenadas en él sobre el contexto de dispositivo especificado, creando la imagen como si las funciones del GDI hubieran sido llamadas desde el programa.

Este método utilizado por los metaarchivos para almacenar sus imágenes permite escalarlas con poca o ninguna pérdida de resolución. Así, los gráficos de metaarchivos son comúnmente para “*clip arts*” o para almacenar dibujos técnicos, como diseños CAD o planos arquitectónicos. Además, debido a que los registros del metaarchivo sólo describen la imagen y no almacenan los píxeles individuales, los metaarchivos son normalmente mucho más pequeños que los mapas de bits para imágenes iguales. Sin embargo, como las funciones del GDI usadas para describir la imagen son ejecutadas cada vez que el metaarchivo es dibujado en pantalla, los metaarchivos se dibujan más lentamente que los mapas de bits.

### Metaarchivos mejorados

Las aplicaciones Win32 deben usar el formato de *metaarchivo mejorado* en lugar del formato Win16 de metaarchivo. El formato de metaarchivo mejorado contiene un encabezamiento que describe la resolución y tamaño originales para los cuales el metaarchivo fue creado. También almacena una paleta de colores para la imagen. El formato de metaarchivo Win16 no incluye nada de esto. Observe que el metaarchivo mejorado está sujeto a las limitaciones del GDI de Windows 95/98. Específicamente, el GDI de Windows 95/98 sólo soporta coordenadas con signo de 16 bits. Por esta razón, cualquier registro de metaarchivo mejorado con coordenadas más allá del rango -32,768 a 32,767 fallará al tratar de ser mostrado. La clase *TMetafile* de la VCL de Delphi encapsula ambos formatos, Win16 y metaarchivo mejorado.

## Funciones de mapas de bits y metaarchivos

Las siguientes funciones de mapas de bits y metaarchivos son tratadas en este capítulo:

**Tabla 5-1: Funciones de mapas de bits y metaarchivos**

Función	Descripción
BitBlt	Copia bits de un contexto de dispositivo a otro.
CloseEnhMetaFile	Cierra un contexto de dispositivo de metaarchivo mejorado y devuelve un manejador del metaarchivo.
CopyEnhMetaFile	Crea un duplicado del metaarchivo mejorado.
CopyImage	Crea un duplicado de un icono, un mapa de bits o un cursor.
CreateBitmap	Crea un mapa de bits dependiente del dispositivo.
CreateBitmapIndirect	Crea un mapa de bits dependiente del dispositivo a partir de la información en un registro.
CreateCompatibleBitmap	Crea un mapa de bits dependiente del dispositivo compatible con un contexto de dispositivo especificado.

Función	Descripción
CreateDIBitmap	Crea un mapa de bits dependiente del dispositivo a partir de un mapa de bits independiente del dispositivo.
CreateDIBSection	Crea un mapa de bits independiente del dispositivo.
CreateEnhMetaFile	Crea un metaarchivo mejorado.
DeleteEnhMetaFile	Elimina un metaarchivo mejorado.
EnumEnhMetaFile	Enumera los registros de metaarchivo en un metaarchivo mejorado.
GetBitmapBits	Recupera píxeles de un mapa de bits hacia un array.
GetBitmapDimensionEx	Recupera las dimensiones preferidas de un mapa de bits.
GetDIBits	Crea un mapa de bits independiente del dispositivo a partir de un mapa de bits dependiente del dispositivo.
GetEnhMetaFile	Abre un metaarchivo mejorado.
GetEnhMetaFileDescription	Recupera la cadena de descripción de un metaarchivo mejorado.
GetEnhMetaFileHeader	Recupera el encabezamiento de un metaarchivo mejorado.
GetStretchBltMode	Recupera el modo actual de compresión de mapas de bits.
LoadBitmap	Carga un recurso de mapa de bits.
LoadImage	Carga un icono, un cursor o un mapa de bits desde un recurso o un de mapa de bits.
PatBlt	Rellena el rectángulo especificado con la brocha del DC destino y puede ejecutar ciertas operaciones de barrido.
PlayEnhMetaFile	Dibuja un metaarchivo mejorado sobre un contexto de dispositivo.
PlayEnhMetaFileRecord	Dibuja un solo registro de un metaarchivo sobre un contexto de dispositivo.
SetBitmapBits	Asigna los píxeles de un mapa de bits dependiente del dispositivo.
SetBitmapDimensionEx	Asigna las dimensiones preferidas de un mapa de bits.
SetDIBits	Asigna los píxeles de un mapa de bits dependiente del dispositivo a los píxeles de un mapa de bits independiente del dispositivo.
SetDIBitsToDevice	Dibuja un mapa de bits independiente del dispositivo sobre un contexto de dispositivo.
SetStretchBltMode	Asigna el modo de compresión de mapas de bits.
StretchBlt	Dibuja y escala píxeles de un contexto de dispositivo sobre otro.
StretchDIBits	Dibuja y escala un mapa de bits independiente del dispositivo sobre un contexto de dispositivo.

**BitBlt**      **Windows.Pas****Sintaxis**

```

BitBlt(
    DestDC: HDC;           {manejador del contexto de dispositivo de destino}
    X: Integer;             {coordenada horizontal del rectángulo de destino}
    Y: Integer;             {coordenada vertical del rectángulo de destino}
    Width: Integer;         {ancho del rectángulo de origen y destino}
    Height: Integer;        {altura del rectángulo de origen y destino}
    SrcDC: HDC;             {manejador del contexto de dispositivo de origen}
    XSrc: Integer;          {coordenada horizontal del rectángulo de origen}
    YSrc: Integer;          {coordenada vertical del rectángulo de destino}
    ROP: DWORD              {código de la operación de barrido}
): BOOL;                   {devuelve TRUE o FALSE}

```

**Descripción**

Esta función copia un rectángulo de píxeles del mapa de bits del contexto de dispositivo especificado en el contexto de dispositivo de destino. El ancho y la altura del rectángulo de destino determinan el ancho y la altura del rectángulo de origen. Si el formato de colores de los contextos de dispositivo de origen y destino difieren, esta función convierte el formato de color del origen al formato de color del destino.

**Parámetros**

*DestDC*: Manejador del contexto de dispositivo al cual los píxeles son copiados.

*X*: La coordenada horizontal de la esquina superior izquierda del rectángulo destino en el contexto de dispositivo destino, en unidades lógicas.

*Y*: La coordenada vertical de la esquina superior izquierda del rectángulo destino en el contexto de dispositivo destino, en unidades lógicas.

*Width*: El ancho de los rectángulos de origen y destino, en unidades lógicas.

*Height*: La altura de los rectángulos de origen y destino, en unidades lógicas.

*SrcDC*: Manejador del contexto de dispositivo desde el cual los píxeles son copiados. Este no puede ser el manejador de un contexto de dispositivo de metaarchivo.

*XSrc*: La coordenada horizontal de la esquina superior izquierda del rectángulo de origen en el contexto de dispositivo de destino, en unidades lógicas.

*YSrc*: La coordenada vertical de la esquina superior izquierda del rectángulo de origen en el contexto de dispositivo de destino, en unidades lógicas.

*ROP*: Código de operación de barrido que determina cómo los colores de los píxeles en la imagen original son combinados con los colores de los píxeles en el dispositivo de destino. Este parámetro puede tomar un valor de la Tabla 5-2.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetDC*, *CreateCompatibleDC*, *CreateBitmap*, *LoadBitmap*, *StretchBlt*

#### Ejemplo

Vea el Listado 5-16 bajo *LoadImage* y otros ejemplos a lo largo de este capítulo.

**Tabla 5-2: Valores del parámetro ROP de BitBlt**

Valor	Descripción
BLACKNESS	Rellena los píxeles del rectángulo especificado en el destino con el color especificado en el índice 0 de la paleta física. Por defecto este color es el negro.
DSTINVERT	Invierte los colores de los píxeles del rectángulo especificado en el destino.
MERGECOPY	Combina los colores de los píxeles del rectángulo de origen con los colores de los píxeles del patrón de la brocha seleccionada en el contexto de dispositivo de destino, usando el operador booleano <b>and</b> .
MERGEPAINT	Invierte los colores de los píxeles del rectángulo de origen y los combina con los colores de los píxeles del rectángulo de destino, usando el operador booleano <b>or</b> .
NOTSRCCOPY	Invierte los colores de los píxeles del rectángulo de origen y los copia en el rectángulo de destino.
NOTSRCERASE	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>or</b> y luego invierte el color resultante.
PATCOPY	Copia el patrón de la brocha seleccionada en el contexto de dispositivo de destino directamente en el destino.
PATINVERT	Combina los colores de los píxeles del patrón de la brocha seleccionada en el contexto de dispositivo de destino, con los colores de los píxeles en el destino, usando el operador booleano <b>xor</b> .
PATPAINT	Combina los colores del patrón de la brocha seleccionada en el contexto de dispositivo de destino con los colores invertidos de los píxeles del rectángulo de origen, usando el operador booleano <b>or</b> , y entonces combina el resultado con los colores de los píxeles del rectángulo de destino, usando el operador booleano <b>or</b> .
SRCAND	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>and</b> .

Valor	Descripción
SRCCOPY	Copia los colores de los píxeles del rectángulo de origen directamente en el rectángulo destino.
SRCERASE	Combina los colores de los píxeles del rectángulo de origen con los colores invertidos del rectángulo destino, usando el operador booleano <b>and</b> .
SRCINVERT	Combina los colores de los píxeles de los rectángulos de origen y destino, usando el operador booleano <b>xor</b> .
SRCPAINT	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>or</b> .
WHITENESS	Rellena los píxeles del rectángulo especificado en el destino con el color especificado en el índice 255 de la paleta física. Por defecto este color es el blanco.

**CloseEnhMetaFile****Windows.Pas****Sintaxis**

```
CloseEnhMetaFile(
    DC: HDC                {manejador de contexto de dispositivo de metaarchivo}
): HENHMETAFILE;          {devuelve un manejador de metaarchivo mejorado}
```

**Descripción**

Esta función cierra el contexto de dispositivo de metaarchivo mejorado especificado y devuelve un manejador del metaarchivo mejorado. Este manejador puede ser utilizado en todas las funciones que requieren un manejador de metaarchivo mejorado. Cuando el metaarchivo deje de ser necesario deberá ser eliminado llamando a la función *DeleteEnhMetaFile*.

**Parámetros**

*DC*: Manejador de un contexto de dispositivo de metaarchivo mejorado.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de un metaarchivo mejorado; en caso contrario, devuelve cero.

**Véase además**

*CopyEnhMetaFile*, *CreateEnhMetaFile*, *DeleteEnhMetaFile*,  
*GetEnhMetaFileDescription*, *GetEnhMetaFileHeader*, *PlayEnhMetaFile*

**Ejemplo**

Vea el Listado 5-10 bajo *CreateEnhMetaFile*.

**CopyEnhMetaFile****Windows.Pas****Sintaxis**

```
CopyEnhMetaFile(
    p1: HENHMETAFILE;    {manejador de metaarchivo mejorado}
    p2: PChar             {cadena que especifica un nombre de fichero}
): HENHMETAFILE;        {devuelve un manejador de un metaarchivo mejorado}
```

**Descripción**

Esta función copia el metaarchivo mejorado especificado a un fichero o a memoria, devolviendo un manejador del metaarchivo mejorado copiado. Cuando el metaarchivo no sea necesario deberá ser eliminado llamando a la función *DeleteEnhMetaFile*.

**Parámetros**

*p1*: Manejador del metaarchivo mejorado que será copiado.

*p2*: Cadena terminada en nulo que especifica el nombre del fichero de destino. Si a este parámetro se le asigna **nil**, la función simplemente copia el metaarchivo mejorado en memoria.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del metaarchivo mejorado copiado; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateEnhMetaFile*, *DeleteEnhMetaFile*, *GetEnhMetaFileDescription*, *GetEnhMetaFileHeader*, *PlayEnhMetaFile*

**Ejemplo**

Vea el Listado 5-14 bajo *GetEnhMetaFile*.

**CopyImage****Windows.Pas****Sintaxis**

```
CopyImage(
    hImage: THandle;      {manejador de imagen}
    ImageType: UINT;      {indicador de tipo de imagen}
    X: Integer;           {ancho de la nueva imagen}
    Y: Integer;           {altura de la nueva imagen}
    Flags: UINT            {opciones de la operación de copia}
): THandle;              {devuelve un manejador de la imagen copiada}
```

**Descripción**

Esta función crea un duplicado de la imagen especificada (mapa de bits, icono o cursor). La nueva imagen puede ser ampliada o reducida como se desee y convertida a un formato monocromático o de color.

**Parámetros**

*hImage*: Manejador de la imagen que es copiada.

*ImageType*: Valor que indica el tipo de imagen que será copiada. Este parámetro puede ser un valor de la Tabla 5-3.

*X*: Indica el ancho deseado para la imagen copiada, en píxeles.

*Y*: Indica la altura deseada para la imagen copiada, en píxeles.

*Flags*: Un valor que indica cómo la imagen debe ser copiada. Este parámetro puede incluir una o más opciones de la Tabla 5-4.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de la imagen copiada; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*LoadBitmap*, *LoadCursor*, *LoadCursorFromFile*, *LoadIcon*, *LoadImage*

**Ejemplo****Listado 5-4: Creando una imagen monocromática para realizar una copia transparente**

```
var
  ForegroundImage: TBitmap;    // almacena la imagen de primer plano

implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  {Crea el mapa de bits del primer plano y lo carga}
  ForegroundImage := TBitmap.Create;
  ForegroundImage.LoadFromFile('Foreground.bmp');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {Libera el mapa de bits del primer plano}
  ForegroundImage.Free;
end;

procedure TForm1.FormPaint(Sender: TObject);
var
  TempBitmap: HBITMAP;        // un manejador de la imagen copiada
```

```

OldBitmap: HBITMAP;      // almacena el mapa de bits anterior del DC
OffscreenDC: HDC;        // manejador de contexto de dispositivo fuera de pantalla
begin
  {Crea una máscara monocromática de la imagen del primer plano}
  TempBitmap := CopyImage(ForegroundImage.Handle, IMAGE_BITMAP,
                          ForegroundImage.Width, ForegroundImage.Height,
                          LR_MONOCHROME);

  {Crea un contexto de dispositivo en memoria}
  OffscreenDC := CreateCompatibleDC(0);

  {Selecciona la imagen de la máscara monocromática en el contexto de dispositivo
  en memoria}
  OldBitmap := SelectObject(OffscreenDC, TempBitmap);

  {Efectúa un "blit" de la máscara monocromática sobre la imagen de fondo.
  $00220326 es una operación de barrido que invierte los píxeles del rectángulo
  de origen y combina esos píxeles con los del mapa de bits del destino usando el
  operador booleano and. Esto "labra" un área para el mapa de bits regular de
  primer plano}
  BitBlt(Image1.Picture.Bitmap.Canvas.Handle, 150, 50, 100, 100, OffscreenDC,
        0, 0, $00220326);

  {Efectúa un "blit" del mapa de bits del primer plano sobre el fondo, combinando
  los píxeles del primer plano y el fondo mediante el operador booleano or. El
  resultado es el globo terráqueo del primer plano copiado sobre el fondo,
  mientras los bordes de la imagen del globo terráqueo se ven transparente}
  BitBlt(Image1.Picture.Bitmap.Canvas.Handle, 150, 50, 100, 100,
        ForegroundImage.Canvas.Handle, 0, 0, SRCPAINT);

  {Selecciona el mapa de bits anterior en el contexto de dispositivo en memoria}
  SelectObject(OffscreenDC, OldBitmap);

  {Elimina el mapa de bits de máscara y el contexto de dispositivo en memoria}
  DeleteObject(TempBitmap);
  DeleteDC(OffscreenDC);
end;
```

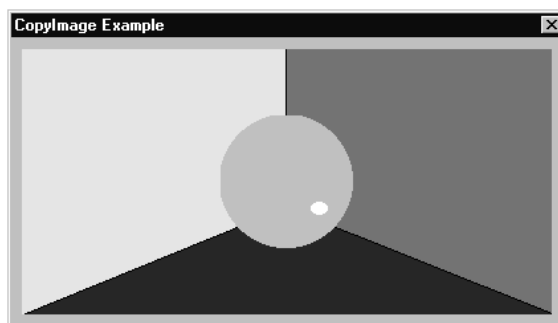


Figura 5-5: La copia de la imagen transparente

Tabla 5-3: Valores de tipo de imagen de CopyImage

Valor	Descripción
IMAGE_BITMAP	La imagen es un mapa de bits.
IMAGE_CURSOR	La imagen es un cursor.
IMAGE_ENHMETAFILE	La imagen es un metaarchivo mejorado.
IMAGE_ICON	La imagen es un icono.

Tabla 5-4: Valores del parámetro Flags de CopyImage

Valor	Descripción
LR_COPYDELETEORG	La imagen original es borrada después que se haga la copia.
LR_COPYFROMRESOURCE	La función trata de cargar un recurso de icono o cursor desde el fichero de recursos, en lugar de hacer una copia. La imagen recuperada desde el fichero de recursos es la imagen más cercana al tamaño deseado; no se escala la imagen al ancho y altura indicados. Si la imagen no fue cargada mediante las funciones <i>LoadIcon</i> o <i>LoadCursor</i> , o mediante la función <i>LoadImage</i> con la opción LR_SHARED seleccionada, la función falla.
LR_COPYRETURNORG	Crea un duplicado exacto de la imagen original. Los parámetros X e Y son ignorados.
LR_MONOCHROME	Crea una versión en blanco y negro de la imagen original.

**CreateBitmap****Windows.Pas****Sintaxis**

```

CreateBitmap(
    Width: Integer;           {ancho del mapa de bits en píxeles}
    Height: Integer;          {altura del mapa de bits en píxeles}
    Planes: Longint;          {cantidad de planos de colores}
    BitCount: Longint;        {bits necesarios para identificar un color}
    Bits: Pointer             {puntero a un array de datos de colores}
): HBITMAP;                  {devuelve un manejador de mapa de bits}

```

**Descripción**

Esta función crea un nuevo mapa de bits con el ancho, altura y profundidad de color especificados. Un *array* de información sobre los píxeles puede ser especificado para crear un mapa de bits con una imagen inicial. Si a los parámetros *Width* y *Height* se les asigna cero, esta función devuelve un manejador de un mapa de bits monocromático de 1 x 1 píxeles. Una vez que el mapa de bits es creado, puede ser seleccionado sobre un

contexto de dispositivo mediante la función *SelectObject*. Cuando el mapa de bits deje de ser necesario, deberá ser eliminado llamando a la función *DeleteObject*.

Aunque esta función puede ser usada para crear mapas de bits de colores, por razones de ejecución las aplicaciones deben usar las funciones *CreateBitmap* para crear mapas de bits monocromáticos y *CreateCompatibleBitmap* para crear mapas de bits en color. *CreateCompatibleBitmap* requiere un contexto de dispositivo y devuelve un mapa de bits que tiene el mismo formato de color que el dispositivo dado. Por esta razón las llamadas a *SelectObject* se ejecutan más rápidamente si el mapa de bits en color ha sido creado mediante *CreateCompatibleBitmap*.

### Parámetros

*Width*: El ancho del mapa de bits en píxeles.

*Height*: La altura del mapa de bits en píxeles.

*Planes*: El número de planos de colores usados por el dispositivo.

*BitCount*: La cantidad de bits requeridos para describir el color de un píxel (por ejemplo, 8 bits para imágenes de 256 colores, 24 bits para imágenes de 16,7 millones de colores, etc.).

*Bits*: Un puntero a un *array* de bytes que contiene los datos de los colores que describen la imagen del mapa de bits. Este *array* especifica el color de los píxeles en un área rectangular. A cada fila horizontal de píxeles en el rectángulo se le llama *línea de barrido*. Cada línea de barrido tiene que estar alineada a frontera de palabra, lo cual significa que su ancho tiene que ser un múltiplo de 2. Una línea de barrido puede ser rellena con ceros para facilitar la alineación. Si este parámetro es **nil**, el nuevo mapa de bits no está definido y no contiene ninguna imagen.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de un mapa de bits; en caso contrario, devuelve cero.

### Véase además

*CreateBitmapIndirect*, *CreateCompatibleBitmap*, *CreateDIBitmap*, *DeleteObject*, *GetBitmapBits*, *GetBitmapDimensionEx*, *SelectObject*, *SetBitmapBits*, *SetBitmapDimensionEx*

### Ejemplo

#### Listado 5-5: Creando un mapa de bits

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TheBitmap: HBitmap;           // manejador para un nuevo mapa de bits
  TheBits: array[0..4095] of Byte; // array de bits del mapa de bits original
  GotBits: array[0..4095] of Byte; // array para recuperar un mapa de bits
  LoopX,           // variables de contador de bucle general
```

```

    LoopY: Integer;
    OffScreen: HDC;           // contexto de dispositivo fuera de pantalla
    TheSize: TSize;           // almacena las dimensiones del mapa de bits
begin
    {Establece cada bit del nuevo mapa de bits con el color almacenado en la
    posición 3 de la paleta del sistema}
    FillMemory(@TheBits, 4096, 3);

    {Establece un cuadrado de 10 X 10 en el centro de la imagen con el color
    especificado en la posición 1 de la paleta del sistema}
    for LoopX:=27 to 37 do
    begin
        TheBits[LoopX*64+27] := 1;
        TheBits[LoopX*64+28] := 1;
        TheBits[LoopX*64+29] := 1;
        TheBits[LoopX*64+30] := 1;
        TheBits[LoopX*64+31] := 1;
        TheBits[LoopX*64+32] := 1;
        TheBits[LoopX*64+33] := 1;
        TheBits[LoopX*64+34] := 1;
        TheBits[LoopX*64+35] := 1;
        TheBits[LoopX*64+36] := 1;
    end;

    {Crea un mapa de bits de 64 X 64 píxeles, usando la información en el array}
    TheBitmap:=CreateBitmap(64, 64, 1, 8, @TheBits);

    {Establece las dimensiones preferidas del mapa de bits. Esto no es usado por
    Windows; simplemente se asigna información definida por el usuario}
    SetBitmapDimensionEx(TheBitmap, 100, 100, nil);

    {Crea un contexto de dispositivo fuera de pantalla compatible con ésta}
    OffScreen:=CreateCompatibleDC(0);

    {Selecciona el mapa de bits en el contexto de dispositivo fuera de pantalla}
    SelectObject(OffScreen, TheBitmap);

    {Copia el mapa de bits del contexto de dispositivo fuera de pantalla
    sobre el área de dibujo del formulario. Esto mostrará el mapa de bits}
    BitBlt(Form1.Canvas.Handle, 162, 16, 64, 64, OffScreen, 0, 0, SRCCOPY);

    {Recupera los bits que conforman la imagen}
    GetBitmapBits(TheBitmap, 4096, @GotBits);

    {Muestra los bits en la rejilla de cadenas}
    for LoopX:=0 to 63 do
    for LoopY:=0 to 63 do
        StringGrid1.Cells[LoopX,LoopY] := IntToStr(GotBits[LoopX*64+LoopY]);

    {Recupera dimensiones preferidas, definidas por el usuario, del mapa de bits}
    GetBitmapDimensionEx(TheBitmap, TheSize);

    {Muestra esas dimensiones}
    Label1.Caption:='Preferred bitmap dimensions - Width: ' + IntToStr(TheSize.CX)
        + ' Height: ' + IntToStr(TheSize.CY);

```

```
{Elimina el contexto de dispositivo fuera de pantalla}
DeleteDC(OffScreen);

{Elimina el nuevo mapa de bits}
DeleteObject(TheBitmap);
```

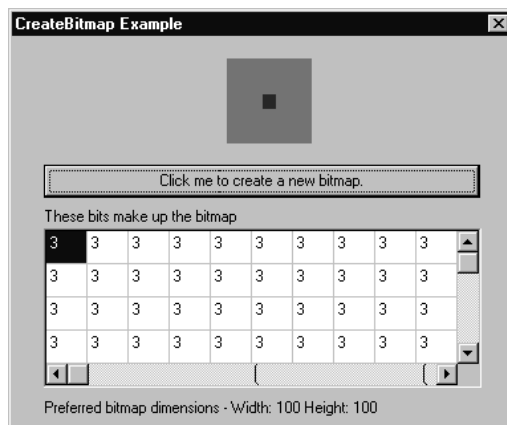


Figura 5-6: El nuevo mapa de bits

end;

### CreateBitmapIndirect

### Windows.Pas

#### Sintaxis

```
CreateBitmapIndirect(
    const p1: TBitmap      {puntero a estructura con información del mapa de bits}
): HBITMAP;               {devuelve un manejador de mapa de bits}
```

#### Descripción

Esta función crea un nuevo mapa de bits con el ancho, altura y profundidad de color especificados. Un *array* de información sobre píxeles puede ser especificado para crear un mapa de bits con una imagen inicial. Si a los parámetros *Width* y *Height* se les asigna cero, esta función devuelve un manejador de un mapa de bits monocromático de 1 x 1 píxeles. Una vez que el mapa de bits es creado, puede ser seleccionado sobre el contexto de dispositivo mediante la función *SelectObject*. Cuando el mapa de bits deje de ser necesario deberá ser eliminado llamando a la función *DeleteObject*.

Aunque esta función puede ser usada para crear mapas de bits a color, por razones de ejecución las aplicaciones deben usar *CreateBitmapIndirect* para crear mapas de bits monocromáticos y *CreateCompatibleBitmap* para crear mapas de bits a color. *CreateCompatibleBitmap* requiere un contexto de dispositivo y devuelve un mapa de bits que tiene el mismo formato de color que el dispositivo dado. Por esta razón, las

llamadas a *SelectObject* se ejecutan más rápidamente si el mapa de bits a color ha sido creado mediante *CreateCompatibleBitmap*.

#### Parámetros

*p1*: Identifica un registro *TBitmap* que contiene información sobre la imagen que se desea crear. El registro *TBitmap* se define como:

*TBitmap* = **packed record**

<i>bmType</i> : Longint;	{tipo de mapa de bits}
<i>bmWidth</i> : Longint;	{ancho del mapa de bits en píxeles}
<i>bmHeight</i> : Longint;	{altura del mapa de bits en píxeles}
<i>bmWidthBytes</i> : Longint;	{número de bytes en cada línea de barrido}
<i>bmPlanes</i> : Word;	{número de planos de colores}
<i>bmBitsPixel</i> : Word;	{número de bits que describen un píxel}
<i>bmBits</i> : Pointer;	{puntero a la imagen del mapa de bits}

**end;**

*bmType*: Indica el tipo de mapa de bits. En el momento en que se escribe este libro, a este campo hay que asignarle cero.

*bmWidth*: El ancho del mapa de bits en píxeles.

*bmHeight*: La altura del mapa de bits en píxeles.

*bmWidthBytes*: La cantidad de bytes en cada línea de barrido del *array* al que apunta el parámetro *bmBits*. Las líneas de barrido almacenadas en este *array* deberán estar alineadas a frontera de palabra, de manera que el valor de este campo tiene que ser un múltiplo de 2.

*bmPlanes*: La cantidad de planos de colores usados por el dispositivo.

*bmBitsPixel*: La cantidad de bits requeridos para describir el color de un píxel (por ejemplo, 8 bits para imágenes de 256 colores, 24 bits para imágenes de 16,7 millones de colores, etc.).

*bmBits*: Un puntero a un *array* de bytes que contiene los datos de los colores que componen la imagen del mapa de bits. Este *array* especifica el color de los píxeles en un área rectangular. A cada fila horizontal de píxeles en el rectángulo se le llama *línea de barrido*. Cada línea de barrido tiene que estar alineada a frontera de palabra, lo cual significa que su ancho tiene que ser un múltiplo de 2. Una línea de barrido puede ser rellenada con ceros para facilitar la alineación. Si este parámetro es **nil**, el nuevo mapa de bits no está definido y no contiene una imagen.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de un nuevo mapa de bits; en caso contrario, devuelve cero.

#### Véase además

*BitBlt*, *CreateBitmap*, *CreateCompatibleBitmap*, *CreateDIBitmap*, *DeleteObject*, *SelectObject*

*Ejemplo***Listado 5-6: Creando un mapa de bits indirectamente**

```

procedure TForm1.Button1Click(Sender: TObject);
var
    TheBitmap: HBITMAP;           // manejador del nuevo mapa de bits
    BitmapInfo: Windows.TBitmap;  // registro con información del mapa de bits
    OffscreenDC: HDC;             // manejador de contexto de dispositivo en
                                // memoria
    BitmapBits: array[0..4095] of Byte; // almacena la imagen del mapa de bits
begin
    {Inicializa los píxeles del mapa de bits con el color en la posición 5 de la
      paleta del sistema}
    FillMemory(@BitmapBits, 4096, 5);

    {Define el nuevo mapa de bits}
    BitmapInfo.bmType      := 0;
    BitmapInfo.bmWidth     := 64;
    BitmapInfo.bmHeight    := 64;
    BitmapInfo.bmWidthBytes := 64;
    BitmapInfo.bmPlanes    := 1;
    BitmapInfo.bmBitsPixel := 8; // 8 bits/píxel, un mapa de bits de 256 colores
    BitmapInfo.bmBits      := @BitmapBits;

    {Crea el mapa de bits a partir de la información del mapa de bits}
    TheBitmap := CreateBitmapIndirect(BitmapInfo);

    {Crea un contexto de dispositivo en memoria compatible con la pantalla}
    OffscreenDC := CreateCompatibleDC(0);

    {Selecciona el nuevo mapa de bits y una pluma de almacén sobre el contexto
      de dispositivo en memoria}
    SelectObject(OffscreenDC, TheBitmap);
    SelectObject(OffscreenDC, GetStockObject(WHITE_PEN));

    {Dibuja una línea sobre el mapa de bits}
    MoveToEx(OffscreenDC, 0, 0, nil);
    LineTo(OffscreenDC, 64, 64);

    {Muestra el mapa de bits}
    BitBlt(PaintBox1.Canvas.Handle, (PaintBox1.Width div 2) - 32,
           (PaintBox1.Height div 2) - 32, 64, 64, OffscreenDC, 0, 0, SRCCOPY);

    {Elimina el contexto de dispositivo en memoria y el mapa de bits}
    DeleteDC(OffscreenDC);
    DeleteObject(TheBitmap);
end;

```

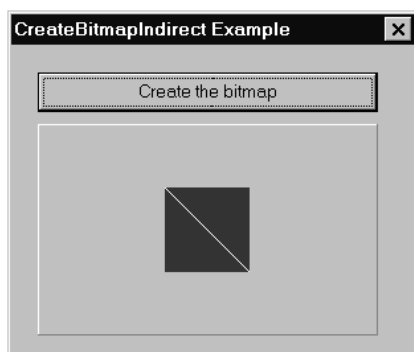


Figura 5-7: El nuevo mapa de bits creado de forma indirecta

### CreateCompatibleBitmap

### Windows.Pas

#### Sintaxis

CreateCompatibleBitmap(	
DC: HDC;	{manejador de contexto de dispositivo}
Width: Integer;	{ancho del mapa de bits en píxeles}
Height: Integer	{altura del mapa de bits en píxeles}
); HBitmap;	{devuelve un manejador del mapa de bits}

#### Descripción

Esta función crea un mapa de bits cuyo formato de color (por ejemplo, 8 bits por píxel, 24 bits por píxel, etc.) y su paleta se corresponden con el formato de color y paleta del dispositivo de visualización asociado con el contexto de dispositivo especificado. Si una sección DIB, creada con la función *CreateDIBSection*, es seleccionada en el contexto de dispositivo especificado, esta función crea un mapa de bits DIB. Utilice la función *DeleteObject* para eliminar el mapa de bits cuando deje de ser necesario. Si a los parámetros *Width* y *Height* se les asigna cero, esta función devuelve un manejador a un mapa de bits monocromático de 1 x 1 píxeles.

#### Parámetros

*DC*: Manejador del contexto de dispositivo desde el cual el mapa de bits recupera su formato y paleta de colores.

*Width*: El ancho del mapa de bits en píxeles.

*Height*: La altura del mapa de bits en píxeles.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del nuevo mapa de bits; en caso contrario, devuelve cero.

#### Véase además

*CreateBitmap*, *CreateBitmapIndirect*, *CreateDIBSection*, *DeleteObject*, *SelectObject*

*Ejemplo***Listado 5-7: Creando un mapa de bits compatible con el dispositivo de visualización actual**

```

procedure TForm1.Button1Click(Sender: TObject);
var
    TheBitmap: HBitmap;           // manejador para el nuevo mapa de bits
    TheBits: array[0..4095] of Byte; // array para el mapa de bits original
    LoopX: Integer;               // contador de bucle
    OffScreen: HDC;               // contexto de dispositivo fuera de pantalla
    ScreenDC: HDC;                // contexto de dispositivo temporal
begin
    {Inicializa los píxeles del mapa de bits con el color en la posición 3 de la
    paleta del sistema}
    FillMemory(@TheBits, 4095, 3);

    {Pone un cuadrado de 10 X 10 en el centro de la imagen al color especificado
    en la posición 1 de la paleta del sistema}
    for LoopX:=27 to 37 do
        begin
            TheBits[LoopX*64+27] := 1;
            TheBits[LoopX*64+28] := 1;
            TheBits[LoopX*64+29] := 1;
            TheBits[LoopX*64+30] := 1;
            TheBits[LoopX*64+31] := 1;
            TheBits[LoopX*64+32] := 1;
            TheBits[LoopX*64+33] := 1;
            TheBits[LoopX*64+34] := 1;
            TheBits[LoopX*64+35] := 1;
            TheBits[LoopX*64+36] := 1;
        end;

    {Recupera un contexto de dispositivo para el escritorio}
    ScreenDC := GetDC(0);

    {Crea un mapa de bits a colores de 64 X 64 píxeles compatible con el dispositivo
    de visualización}
    TheBitmap := CreateCompatibleBitmap(ScreenDC, 64, 64);

    {Libera el contexto de dispositivo del escritorio}
    ReleaseDC(0, ScreenDC);

    {Selecciona la imagen del mapa de bits}
    SetBitmapBits(TheBitmap, 64*64, @TheBits);

    {Crea un contexto de dispositivo fuera de pantalla compatible con la pantalla}
    OffScreen := CreateCompatibleDC(0);

    {Selecciona el nuevo mapa de bits en el contexto fuera de pantalla}
    SelectObject(OffScreen, TheBitmap);

    {Copia el mapa de bits del contexto de dispositivo fuera de pantalla sobre el
    área de dibujo del formulario. Esto mostrará el mapa de bits}
    BitBlt(Form1.Canvas.Handle, (Width div 2) - 32, 16, 64, 64,

```

```

        OffScreen, 0, 0, SRCCOPY);
    {Elimina el contexto de dispositivo fuera de pantalla}
    DeleteDC(OffScreen);

    {Elimina el nuevo mapa de bits}
    DeleteObject(TheBitmap);
end;

```

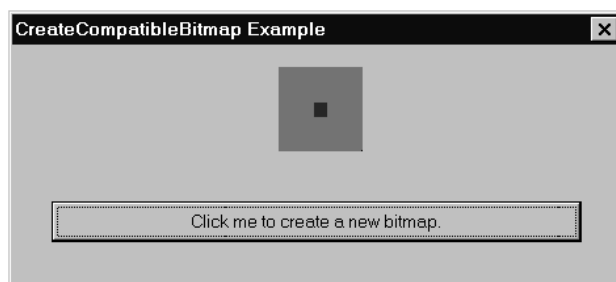


Figura 5-8: El mapa de bits compatible

## CreateDIBitmap

## Windows.Pas

### Sintaxis

CreateDIBitmap(	
DC: HDC;	{manejador de contexto de dispositivo de referencia}
<b>var</b> InfoHeader: TBitmapInfoHeader;	{puntero a registro TBitmapInfoHeader}
dwUsage: DWORD;	{opciones de inicialización del mapa de bits}
InitBits: PChar;	{puntero a los bits del mapa de bits DIB}
<b>var</b> InitInfo: TBitmapInfo;	{puntero a registro TBitmapInfo}
wUsage: UINT	{opciones de tipos de color}
): Hbitmap;	{devuelve un manejador de un mapa de bits dependiente del dispositivo}

### Descripción

Esta función crea un mapa de bits *dependiente* del dispositivo a partir de los atributos e imagen del mapa de bits independiente del dispositivo especificado. Cuando el nuevo mapa de bits deje de ser necesario, deberá ser eliminado mediante la función *DeleteObject*.

### Parámetros

**DC:** Manejador de un contexto de dispositivo. El formato del nuevo mapa de bits dependiente del dispositivo está basado en este contexto de dispositivo. Por eso no puede tratarse de un contexto de dispositivo en memoria. A este parámetro se le puede asignar un valor devuelto por las funciones *GetDC* o *CreateDC*.

*InfoHeader*: Puntero a un registro de tipo *TBitmapInfoHeader*. *CreateDIBitmap* utiliza la información de este registro para asignar los atributos del nuevo mapa de bits dependiente del dispositivo, tales como su ancho y altura. El registro *TBitmapInfoHeader* se define de la siguiente manera:

*TBitmapInfoHeader* = **packed record**

<i>biSize</i> : DWORD;	{tamaño del registro en bytes}
<i>biWidth</i> : Longint;	{ancho del mapa de bits en píxeles}
<i>biHeight</i> : Longint;	{altura del mapa de bits en píxeles}
<i>biPlanes</i> : Word;	{número de planos de colores}
<i>biBitCount</i> : Word;	{bits necesarios para describir un color}
<i>biCompression</i> : DWORD;	{opciones de compresión}
<i>biSizeImage</i> : DWORD;	{tamaño de la imagen en bytes}
<i>biXPelsPerMeter</i> : Longint;	{píxeles horizontales por unidad de medida del dispositivo de destino}
<i>biYPelsPerMeter</i> : Longint;	{píxeles verticales por unidad de medida del dispositivo de destino}
<i>biClrUsed</i> : DWORD;	{cantidad de índices de color usados}
<i>biClrImportant</i> : DWORD;	{cantidad de índices de color importantes}

**end;**

Consulte la función *CreateDIBSection* para ver una descripción de este registro.

*dwUsage*: Indicador que especifica cómo el nuevo mapa de bits dependiente del dispositivo será inicializado. Si a este parámetro se le asigna cero, los bits de la nueva imagen del mapa de bits no serán inicializados. Si a este parámetro se le asigna *CBM\_INIT*, Windows utiliza la información a la que apuntan los parámetros *InitBits* y *InitInfo* para inicializar los bits del nuevo mapa de bits dependiente del dispositivo de manera que se correspondan con los del mapa de bits independiente del dispositivo.

*InitBits*: Puntero a la imagen del mapa de bits DIB, en forma de un *array* de bytes. Si al parámetro *dwUsage* se le asigna cero, *InitBits* es ignorado.

*InitInfo*: Puntero a un registro de tipo *TBitmapInfo* que describe las dimensiones y formato de color del mapa de bits DIB al que apunta el parámetro *InitBits*. Si al parámetro *dwUsage* se le asigna cero, *InitInfo* es ignorado. El registro *TBitmapInfo* se define de la siguiente forma:

*TBitmapInfo* = **packed record**

<i>bmiHeader</i> : <i>TBitmapInfoHeader</i> ;	{información de encabezamiento}
<i>bmiColors</i> : <b>array</b> [0..0] <b>of</b> TRGBQuad;	{tabla de colores usada por el mapa de bits}

**end;**

Consulte la función *CreateDIBSection* para ver una descripción de este registro.

*wUsage*: Opción que indica el tipo de información sobre el color, almacenada en el campo *bmiColors* del registro *TBitmapInfo* al que apunta el parámetro *InitInfo*. Este parámetro puede tomar un valor de la Tabla 5-5.

*Valor que devuelve*

Si la función tiene éxito, devuelve un manejador del mapa de bits dependiente del dispositivo; en caso contrario, devuelve cero.

*Véase además*

*CreateBitmap, CreateBitmapIndirect, CreateDIBSection, DeleteObject*

*Ejemplo*

**Listado 5-8: Creando un mapa de bits dependiente del dispositivo desde uno independiente del dispositivo**

```

procedure TForm1.Button1Click(Sender: TObject);
var
    Dib: HBITMAP;           // manejador del mapa de bits independiente del
                           // dispositivo
    DDB: HBITMAP;           // manejador del mapa de bits dependiente del dispositivo
    DibInfo: PBitmapInfo;   // puntero a un registro TBitmapInfo
    BitsPtr: PByte;         // un puntero a los bits del mapa de bits DIB
    ReferenceDC: HDC;       // manejador de un contexto de dispositivo de referencia
    ScreenDC: HDC;         // manejador de un contexto fuera de pantalla
    iCount: Integer;       // contador de bucle

    SystemPalette: array[0..255] of TPaletteEntry;
                           // necesario para convertir la paleta del sistema
                           // en paletas compatibles DIB
begin
    {Reserva memoria para el DIB}
    GetMem(DibInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));

    {Inicializa la información del DIB}
    DibInfo^.bmiHeader.biWidth      := 64;    // crea un DIB de 64 X 64 píxeles,
    DibInfo^.bmiHeader.biHeight     := -64;   // orientado de arriba a abajo
    DibInfo^.bmiHeader.biPlanes     := 1;
    DibInfo^.bmiHeader.biBitCount   := 8;     // 256 colores
    DibInfo^.bmiHeader.biCompression := BI_RGB; // sin compresión
    DibInfo^.bmiHeader.biSizeImage   := 0;    // Windows determina el tamaño
    DibInfo^.bmiHeader.biXPelsPerMeter := 0;
    DibInfo^.bmiHeader.biYPelsPerMeter := 0;
    DibInfo^.bmiHeader.biClrUsed     := 0;
    DibInfo^.bmiHeader.biClrImportant := 0;
    DibInfo^.bmiHeader.biSize        := SizeOf(TBitmapInfoHeader);

    {Recupera la paleta actual del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256, SystemPalette);

    {La paleta del sistema es devuelta como un array de registros TPaletteEntry,
    que almacena los colores de la paleta en formato (Red, Green, Blue). Sin
    embargo, el campo bmiColors del registro TBitmapInfo necesita un array de
    registros TRGBQuad, que almacena los colores en formato (Blue, Green, Red).
    Por eso tenemos que convertir los registros TPaletteEntry en registros TRGBQuad}
    for iCount := 0 to 255 do
    begin

```

```

    DibInfo^.bmiColors[iCount].rgbBlue      := SystemPalette[iCount].peBlue;
    DibInfo^.bmiColors[iCount].rgbRed       := SystemPalette[iCount].peRed;
    DibInfo^.bmiColors[iCount].rgbGreen     := SystemPalette[iCount].peGreen;
    DibInfo^.bmiColors[iCount].rgbReserved := 0;
end;

{Crea un contexto de dispositivo basado en memoria}
ReferenceDC := CreateCompatibleDC(0);

{Crea el DIB sobre el contexto de dispositivo en memoria, a partir de la
información del mapa de bits inicializado}
Dib := CreateDIBSection(ReferenceDC, DibInfo^, DIB_RGB_COLORS,
    Pointer(BitsPtr), 0, 0);

{Dibuja bandas de colores en el DIB}
FillMemory(BitsPtr, 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+8*64), 8*64, $05);
FillMemory(Pointer(LongInt(BitsPtr)+2*(8*64)), 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+3*(8*64)), 8*64, $05);
FillMemory(Pointer(LongInt(BitsPtr)+4*(8*64)), 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+5*(8*64)), 8*64, $05);
FillMemory(Pointer(LongInt(BitsPtr)+6*(8*64)), 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+7*(8*64)), 8*64, $05);

{Obtiene un DC basado en la pantalla, que es utilizado como dispositivo de
referencia cuando se crea el mapa de bits dependiente del dispositivo}
ScreenDC := GetDC(0);

{Crea un mapa de bits dependiente del dispositivo desde el DIB}
DDB := CreateDIBitmap(ScreenDC, DibInfo^.bmiHeader, CBM_INIT, PChar(BitsPtr),
    DibInfo^, DIB_RGB_COLORS);

{Elimina el contexto de dispositivo basado en la pantalla}
ReleaseDC(0, ScreenDC);

{Selecciona el mapa de bits dependiente del dispositivo sobre el contexto
fuera de pantalla}
SelectObject(ReferenceDC, DDB);

{Copia el mapa de bits independiente del dispositivo al formulario}
SetDIBitsToDevice(Form1.Canvas.Handle, 50, 5, 64, 64, 0, 0, 0, 64, BitsPtr,
    DibInfo^, DIB_RGB_COLORS);

{Copia el mapa de bits dependiente del dispositivo al formulario}
BitBlt(Form1.Canvas.Handle, 166, 5, 64, 64, ReferenceDC, 0, 0, SRCCOPY);

{No necesitamos más ni el mapa de bits ni el contexto de dispositivo}
DeleteDC(ReferenceDC);
DeleteObject(Dib);
DeleteObject(DDB);
FreeMem(DibInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));
end;

```

Figura 5-9: El nuevo mapa de bits dependiente del dispositivo

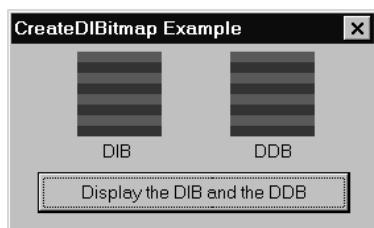


Tabla 5-5: Valores del parámetro wUsage de CreateDIBitmap

Valor	Descripción
DIB_PAL_COLORS	El campo bmiColors del registro TBitmapInfo es un array de índices de 16 bits en la paleta lógica actualmente activa del contexto de dispositivo especificado. Este valor no debe ser usado si el mapa de bits va a ser guardado en disco.
DIB_RGB_COLORS	El campo bmiColors del registro TBitmapInfo es un array de valores de colores RGB literales.

## CreateDIBSection

## Windows.Pas

### Sintaxis

```
CreateDIBSection(
    DC: HDC;                {manejador de contexto de dispositivo}
    const p2: TBitmapInfo;   {puntero a registro TBitmapInfo}
    p3: UINT;                {opciones de tipo de color}
    var p4: Pointer;         {variable que toma un puntero a los bits del mapa de bits}
    p5: THandle;             {manejador de un objeto de mapeado de fichero}
    p6: DWORD                {desplazamiento de los valores de bits del mapa de bits}
): HBITMAP;                {devuelve un manejador de DIB}
```

### Descripción

Esta función crea un mapa de bits independiente del dispositivo a partir de los atributos especificados. Devuelve un manejador del nuevo mapa de bits y un puntero a los valores de bits que conforman la imagen de éste. El desarrollador puede especificar un objeto de mapeado de fichero para almacenar los bits de la imagen o dejar que Windows automáticamente reserve la memoria. Cuando el mapa de bits deje de ser necesario, deberá ser eliminado llamando a la función *DeleteObject*.

### Parámetros

*DC*: Manejador de un contexto de dispositivo. Si el parámetro *p3* contiene la opción *DIB\_PAL\_COLORS*, la nueva paleta de colores del DIB se corresponderá con la paleta lógica del contexto de dispositivo identificado por este parámetro.

*p2*: Puntero a un registro *TBitmapInfo*. Este registro contiene información que describe el tipo de mapa de bits DIB que será creado, tal como su tamaño, formato de color y compresión. El registro *TBitmapInfo* se define como:

```
TBitmapInfo = packed record
    bmiHeader: TBitmapInfoHeader;           {información de encabezamiento}
    bmiColors: array[0..0] of TRGBQuad;
                                           {tabla de colores usada por el mapa de bits}
end;
```

*bmiHeader*: Un registro de tipo *TBitmapInfoHeader* que contiene información sobre las dimensiones y formato de color del mapa de bits DIB. El registro *TBitmapInfoHeader* se define como:

```
TBitmapInfoHeader = packed record
    biSize: DWORD;           {tamaño del registro en bytes}
    biWidth: Longint;         {ancho del mapa de bits en píxeles}
    biHeight: Longint;        {altura del mapa de bits en píxeles}
    biPlanes: Word;           {número de planos de color}
    biBitCount: Word;         {bits por pixel necesarios para describir un
                                color}
    biCompression: DWORD;     {opción de compresión}
    biSizeImage: DWORD;        {tamaño de la imagen en bytes}
    biXPelsPerMeter: Longint;  {píxeles horizontales por unidad de medida
                                del dispositivo de destino}
    biYPelsPerMeter: Longint;  {píxeles verticales por unidad de medida
                                del dispositivo de destino}
    biClrUsed: DWORD;          {cantidad de índices de color usados}
    biClrImportant: DWORD;     {cantidad de índices de color importantes}
end;
```

*biSize*: El tamaño del registro *TBitmapInfoHeader* en bytes. A este campo se le debe asignar *SizeOf(TBitmapInfoHeader)*.

*biWidth*: Especifica el ancho del mapa de bits en píxeles.

*biHeight*: Especifica la altura del mapa de bits en píxeles. Si este valor es positivo, el mapa de bits DIB estará orientado de abajo a arriba, con el origen en la esquina inferior izquierda. Si es negativo, el DIB estará orientado de arriba a abajo con el origen en la esquina superior izquierda, como un mapa de bits corriente.

*biPlanes*: Especifica el número de planos de color en uso.

*biBitCount*: La cantidad de bits necesarios para describir el color de un píxel (por ejemplo, 8 bits para imágenes de 256 colores, 24 bits para imágenes de 16,7 millones de colores, etc.). Este campo puede ser un valor de la Tabla 5-6.

*biCompression*: Opción que indica el tipo de compresión usado para los DIB orientados de abajo a arriba (los DIB orientados de arriba a abajo no pueden utilizar compresión). Este campo puede ser un valor de la Tabla 5-7.

*biSizeImage*: Especifica el tamaño de la imagen en bytes. A este campo se le puede asignar cero en el caso de DIBs que utilicen la opción *BI\_RGB* en el campo *biCompression*. Aunque al campo *biWidth* se le puede asignar cualquier valor, cada línea de barrido del DIB debe estar alineada a frontera de doble palabra. Para hallar el valor correcto para este campo que producirá que las líneas de barrido del DIB estén alineadas a frontera de doble palabra, use la siguiente fórmula:

$$((((biBitCount * biWidth) + 31) \text{ div } 32) * 4) * \text{ABS}(biHeight)$$

Los bits extras serán rellenados con ceros y no serán utilizados.

*biXPelsPerMeter*: Especifica los píxeles horizontales por medida de resolución del dispositivo de destino indicado en el parámetro *DC*. Este valor puede ser usado para seleccionar de los recursos de la aplicación el mapa de bits que mejor se corresponda con las características del dispositivo de visualización actual.

*biYPelsPerMeter*: Especifica los píxeles verticales por medida de resolución del dispositivo de destino indicado en el parámetro *DC*.

*biClrUsed*: Especifica el número de índices de color de la tabla de colores que son usados por el mapa de bits. Si este campo es cero, el mapa de bits utiliza la máxima cantidad de colores indicados por el campo *biBitCount* para el modo de compresión indicado en el campo *biCompression*. Si el DIB es un mapa de bits empaquetado, lo cual significa que el *array* de bits que describe la imagen sigue directamente al registro *TBitmapInfo* y por lo tanto todos los datos que definen el mapa de bits son contiguos y se referencian mediante un único puntero, entonces a este campo hay que asignarle cero o el tamaño actual de la tabla de colores. Si al parámetro *p3* se le asigna *DIB\_PAL\_COLORS* y el mapa de bits DIB es un mapa de bits empaquetado, a este campo hay que asignarle un número par, de manera que los valores del mapa de bits DIB comiencen en una frontera de doble palabra.

*biClrImportant*: Especifica el número de entradas en la tabla de colores que son consideradas importantes para mostrar el mapa de bits correctamente. Los colores en el *array bmiColors* deben estar ordenados por orden de importancia, con los más importantes en las primeras posiciones. A este campo se le puede asignar cero, en cuyo caso todos los colores son considerados importantes.

*bmiColors*: Un *array* de registros *TRGBQuad* o valores de doble palabra que definen la tabla de color del mapa de bits. El registro *TRGBQuad* se define como:

TRGBQuad = **packed record**

rgbBlue: Byte;	{intensidad del color azul}
rgbGreen: Byte;	{intensidad del color verde}
rgbRed: Byte;	{intensidad del color rojo}
rgbReserved: Byte;	{valor reservado}

**end;**

*rgbBlue*: Especifica la intensidad del color azul.

*rgbGreen*: Especifica la intensidad del color verde.

*rgbRed*: Especifica la intensidad del color rojo.

*rgbReserved*: Este campo es reservado y se le debe asignar cero.

*p3*: Un valor que indica el tipo de información almacenada en el campo *bmiColors* del registro *TBitmapInfo* al que apunta el parámetro *p2*. Este parámetro puede tomar un valor de la Tabla 5-8.

*p4*: Un puntero a una variable que recibe un puntero a los valores de los bits del mapa de bits DIB.

*p5*: Un manejador opcional de un objeto de mapeado de fichero, creado mediante una llamada a la función *CreateFileMapping*. Este objeto de mapeado de fichero es usado para crear el mapa de bits DIB. Los valores de los bits del DIB serán ubicados en el desplazamiento indicado por el parámetro *p6* en el objeto de mapeado de fichero. Este objeto de mapeado de fichero puede ser recuperado posteriormente llamando a la función *GetObject*, utilizando el manejador *HBITMAP* devuelto por *CreateDIBSection*. El desarrollador tiene que cerrar manualmente el objeto de mapeado de fichero una vez que el mapa de bits es eliminado. Si este parámetro es cero, Windows reserva memoria para el mapa de bits DIB, el parámetro *p6* es ignorado y el manejador de mapeado de fichero devuelto por *GetObject* será cero.

*p6*: Especifica el desplazamiento a partir del inicio del objeto de mapeado de fichero referido por el parámetro *p5* en el que se sitúan los valores de los bits del mapa de bits DIB. Los valores de los bits del mapa de bits deben estar alineados a frontera de doble palabra, de manera que este parámetro tiene que ser un múltiplo de 4. Si al parámetro *p5* se le asigna cero, este parámetro es ignorado.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de un nuevo mapa de bits independiente del dispositivo y la variable indicada por el parámetro *p4* contiene un puntero a los valores de los bits del mapa de bits. Si la función falla, devuelve cero y la variable indicada por el parámetro *p4* contiene **nil**. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateFileMapping*, *DeleteObject*, *GetDIBColorTable*, *GetObject*, *SetDIBits*, *SetDIBitsToDevice*, *SetDIBColorTable*, *StretchDIBits*

*Ejemplo***Listado 5-9: Creando un mapa de bits independiente del dispositivo**

```

procedure TForm1.Button1Click(Sender: TObject);
var
    Dib: HBITMAP;           // almacena un manejador del DIB
    DibInfo: PBitmapInfo;    // puntero al registro de información del mapa de bits
    BitsPtr: PByte;          // almacena un puntero a los bits del mapa de bits
    ReferenceDC: HDC;         // manejador del contexto de dispositivo de referencia
    iCount: Integer;          // contador de bucle

    SystemPalette: array[0..255] of TPaletteEntry;
                        // necesario para convertir la paleta del sistema
                        // en paletas compatibles DIB
begin
    {Reserva la memoria necesaria para el registro de información del mapa de bits}
    GetMem(DibInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));

    {Inicializa la información del mapa de bits}
    DibInfo^.bmiHeader.biWidth      := 64;    // crea un DIB de 64 X 64 píxeles,
    DibInfo^.bmiHeader.biHeight     := -64;   // orientado de arriba a abajo
    DibInfo^.bmiHeader.biPlanes     := 1;
    DibInfo^.bmiHeader.biBitCount   := 8;     // 256 colores
    DibInfo^.bmiHeader.biCompression := BI_RGB; // sin compresión
    DibInfo^.bmiHeader.biSizeImage   := 0;     // Windows determina el tamaño
    DibInfo^.bmiHeader.biXPelsPerMeter := 0;
    DibInfo^.bmiHeader.biYPelsPerMeter := 0;
    DibInfo^.bmiHeader.biClrUsed     := 0;
    DibInfo^.bmiHeader.biClrImportant := 0;
    DibInfo^.bmiHeader.biSize        := SizeOf(TBitmapInfoHeader);

    {Recupera la paleta actual del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256, SystemPalette);

    {La paleta del sistema es devuelta como un array de registros TPaletteEntry,
     que almacena los colores de la paleta en formato (Red, Green, Blue). Sin
     embargo, el campo bmiColors del registro TBitmapInfo necesita un array de
     registros TRGBQuad, que almacena los colores en formato (Blue, Green, Red).
     Por eso tenemos que convertir los registros TPaletteEntry en registros TRGBQuad}
    for iCount := 0 to 255 do
    begin
        DibInfo^.bmiColors[iCount].rgbBlue      := SystemPalette[iCount].peBlue;
        DibInfo^.bmiColors[iCount].rgbRed       := SystemPalette[iCount].peRed;
        DibInfo^.bmiColors[iCount].rgbGreen     := SystemPalette[iCount].peGreen;
        DibInfo^.bmiColors[iCount].rgbReserved := 0;
    end;

    {Crea un contexto de dispositivo en memoria}
    ReferenceDC := CreateCompatibleDC(0);

    {Crea el DIB sobre el contexto de dispositivo en memoria, a partir de la
     información del mapa de bits inicializada}
    Dib := CreateDIBSection(ReferenceDC, DibInfo^, DIB_RGB_COLORS,

```

```

        Pointer(BitsPtr), 0, 0);

{Elimina el contexto de dispositivo de referencia}
DeleteDC(ReferenceDC);

{Rellena los bits de la imagen DIB con bandas de color que se alternan}
FillMemory(BitsPtr, 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+8*64), 8*64, $05);
FillMemory(Pointer(LongInt(BitsPtr)+2*(8*64)), 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+3*(8*64)), 8*64, $05);
FillMemory(Pointer(LongInt(BitsPtr)+4*(8*64)), 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+5*(8*64)), 8*64, $05);
FillMemory(Pointer(LongInt(BitsPtr)+6*(8*64)), 8*64, $03);
FillMemory(Pointer(LongInt(BitsPtr)+7*(8*64)), 8*64, $05);

{Dibuja el DIB sobre el formulario}
SetDIBitsToDevice(Form1.Canvas.Handle, 30, 5, 64, 64, 0, 0, 0, 64, BitsPtr,
    DibInfo^, DIB_RGB_COLORS);

{Dibuja el DIB de nuevo, pero esta vez lo agranda al doble de su tamaño}
StretchDIBits(Form1.Canvas.Handle, 105, 5, 128, 128, 0, 0, 64, 64, BitsPtr,
    DibInfo^, DIB_RGB_COLORS, SRCCOPY);

{Eliminamos el DIB y también la memoria reservada para el registro}
DeleteObject(Dib);
FreeMem(DibInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));
end;
```

Figura 5-10:  
El DIB, en su  
tamaño  
original y el  
doble

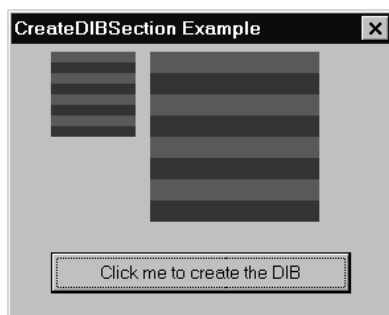


Tabla 5-6: Valores del campo p2.bmiHeader.biBitCount de CreateDIBSection

Valor	Descripción
1	Este mapa de bits tiene dos colores como máximo y el array bmiColors contiene sólo dos entradas. Cada píxel en la imagen del mapa de bits está representado por un bit simple. Si el bit es 0, el píxel es dibujado usando el color especificado en la primera entrada del array bmiColors. Si el bit es 1, el píxel es dibujado usando el color de la segunda entrada del array bmiColors.

Valor	Descripción
4	Este mapa de bits tiene un máximo de 16 colores y el array <code>bmiColors</code> puede contener hasta 16 entradas. Cada byte que compone el mapa de bits representa 2 píxeles. Los 4 bits inferiores en el byte representan el índice en la paleta de colores para el primer píxel y los 4 bits superiores representan el índice de color para el segundo píxel.
8	Este mapa de bits tiene un máximo de 256 colores y el <i>array</i> <code>bmiColors</code> puede contener hasta 256 entradas. Cada byte que compone el mapa de bits representa un píxel, y especifica el índice del color dentro del array <code>bmiColors</code> de 256 entradas.
16	Este mapa de bits tiene un máximo de 65.536 colores. Si el campo <code>biCompression</code> del registro <code>TBitmapInfoHeader</code> contiene el valor <code>BI_RGB</code> , al campo <code>bmiColors</code> se le asigna <code>nil</code> . En este caso, cada palabra (16 bits) en los valores de bits del mapa de bits representa un píxel. Moviéndose del bit menos significativo al más significativo, los últimos 5 bits de la palabra especifican la intensidad de azul del píxel, los siguientes 5 bits especifican la intensidad de verde y los siguientes 5 bits especifican la intensidad de rojo. El bit más significativo de la palabra no se utiliza. Si al campo <code>biCompression</code> del registro <code>TBitmapInfoHeader</code> se le asigna <code>BI_BITFIELDS</code> , el campo <code>bmiColors</code> contendrá 3 dobles palabras que representan una máscara de bits. Estas máscaras de bits son aplicadas al valor de la palabra para cada píxel, usando el operador booleano <code>and</code> , para recuperar las intensidades de los colores rojo, verde y azul respectivamente para ese píxel. Bajo Windows NT, el conjunto de bits activos en cada máscara de doble palabra tiene que ser contiguo y no debe solaparse con los bits de las otras máscaras. En este caso, el desarrollador no tiene que usar todos los bits que describen el píxel. Bajo Windows 95/98, sólo se permiten los siguientes valores de máscaras de doble palabra: un formato 5-5-5, donde la máscara azul es \$0000001F, la verde es \$000003E0 y la roja es \$00007C00, o un formato 5-6-5, donde la máscara azul es \$0000001F, la verde es \$000007E0 y la roja es \$0000F800.
24	Este mapa de bits tiene un máximo de 16,7 millones de colores y el campo <code>bmiColors</code> contiene <code>nil</code> . Cada píxel en la imagen del mapa de bits está representado por 3 bytes. Estos 3 bytes indican la intensidad relativa de los colores azul, verde y rojo del píxel.

Valor	Descripción
32	Este mapa de bits tiene un máximo de 4,3 billones de colores aproximadamente. Si al campo <code>biCompression</code> del registro <code>TBitmapInfoHeader</code> se le asigna <code>BI_RGB</code> , el campo <code>bmiColors</code> se establece a <b>nil</b> . En este caso, cada doble palabra en los valores de bits del mapa de bits representa un píxel. Moviéndose del bit menos significativo al más significativo, el último byte de la doble palabra especifica la intensidad del azul del píxel, el próximo byte especifica la intensidad del verde, y el siguiente byte especifica la intensidad del rojo. El byte más significativo de la doble palabra no es utilizado. Si al campo <code>biCompression</code> del registro <code>TBitmapInfoHeader</code> se le asigna <code>BI_BITFIELDS</code> , el campo <code>bmiColors</code> contendrá 3 dobles palabras que representan una máscara de bits. Estas máscaras de bits son aplicadas al valor de doble palabra de cada píxel usando el operador booleano <b>and</b> para recuperar la intensidad de los colores rojo, verde y azul, respectivamente para ese píxel. Bajo Windows NT, el conjunto de bits activos en cada máscara de doble palabra tiene que ser contiguo y no debe solaparse con los bits de las otras máscaras. En este caso, el desarrollador no tiene que usar todos los bits que describen el píxel. Bajo Windows 95/98, sólo se permite una máscara azul de \$000000FF, una verde de \$0000FF00 y una roja de \$00FF0000.

Tabla 5-7: Valores del campo `p2.bmiHeader.biCompression` de `CreateDIBSection`

Valor	Descripción
<code>BI_RGB</code>	No se utiliza compresión.
<code>BI_RLE8</code>	Se utiliza un algoritmo de compresión según longitud de repeticiones ( <i>run-length encoding</i> ) para los mapas de bits de 256 colores (el formato de color es de 8 bits por píxel). Este formato de compresión consiste en una secuencia de pares de bytes. El primer byte de cada par es un byte de conteo que especifica cuántas veces hay que repetir el siguiente byte cuando se dibuja la imagen. El segundo byte es un índice a la tabla de colores.
<code>BI_RLE4</code>	Se utiliza un algoritmo de compresión según longitud de repeticiones ( <i>run-length encoding</i> ) para los mapas de bits de 16 colores (el formato de color es de 4 bits por píxel). Este formato de compresión consiste en una secuencia de pares de bytes. El primer byte de cada par es un byte de conteo que especifica cuántas veces hay que repetir el siguiente byte cuando se dibuja la imagen. El segundo byte especifica dos índices a la tabla de colores: el primero en los 4 bits más significativos y el segundo en los 4 bits menos significativos.
<code>BI_BITFIELDS</code>	Este formato es válido sólo para mapas de bits de colores de 16 y 32 bits por píxel. El mapa de bits no está comprimido y la tabla de colores consiste en 3 máscaras de colores de doble palabra para las intensidades de rojo, verde y azul. Estas máscaras, combinadas con los bits que describen cada píxel individual mediante el operador booleano <b>and</b> , determinan la intensidad del rojo, verde y azul de cada píxel.

Tabla 5-8: Valores del parámetro p3 de CreateDIBSection

Valor	Descripción
DIB_PAL_COLORS	El campo bmiColors del registro TBitmapInfo es un array de índices de 16 bits en la paleta lógica actualmente activa del contexto de dispositivo especificado. Este valor no debe ser usado si el mapa de bits va a ser guardado en disco.
DIB_RGB_COLORS	El campo bmiColors del registro TBitmapInfo es un array de valores de colores RGB literales.

**CreateEnhMetaFile****Windows.Pas****Sintaxis**

```

CreateEnhMetaFile(
    DC: HDC;                {manejador de un contexto de dispositivo de referencia}
    p2: PChar;              {puntero a un nombre de fichero}
    p3: PRect;              {puntero a un rectángulo límite}
    p4: PChar;              {puntero a una cadena de descripción}
): HDC;                    {devuelve un manejador de contexto de dispositivo de
                           metaarchivo}

```

**Descripción**

Esta función crea un contexto de dispositivo de metaarchivo mejorado. Este contexto de dispositivo puede ser usado con cualquier función del GDI para dibujar en el metaarchivo mejorado. Un manejador del metaarchivo se obtiene llamando a la función *CloseEnhMetaFile* y el metaarchivo es dibujado usando la función *PlayEnhMetaFile*.

**Parámetros**

*DC*: Manejador de un contexto de dispositivo usado como referencia para el nuevo contexto de dispositivo del metaarchivo mejorado. Este contexto de dispositivo de referencia es usado para registrar la resolución y unidades del dispositivo sobre el cual el metaarchivo aparecerá originalmente. Si el parámetro es cero, el dispositivo de visualización actual es usado como referencia. Esta información es usada para escalar el metaarchivo cuando es dibujado.

*p2*: Puntero a una cadena terminada en nulo, que describe el nombre del fichero en el cual se almacenará el metaarchivo mejorado. La extensión de este nombre de fichero es habitualmente .EMF. Si este parámetro es **nil**, el metaarchivo mejorado sólo existirá en memoria y será borrado con la llamada a la función *DeleteEnhMetaFile*.

*p3*: Puntero a un rectángulo que describe las dimensiones de la imagen almacenada en el metaarchivo mejorado. Estas dimensiones se expresan en términos de centésimas de milímetro (por ejemplo, un valor de 3 equivale a 0,03 milímetros). Si este parámetro es **nil**, las dimensiones del rectángulo más pequeño alrededor de la imagen del

metafichero serán automáticamente calculadas. Esta información es usada para escalar el metafichero cuando es dibujado.

*p4*: Puntero a una cadena terminada en nulo que contiene una descripción del metafichero y su contenido. Típicamente, esta descripción consiste en el nombre de la aplicación seguido de un carácter nulo, seguido por el título del metafichero y terminando con dos caracteres nulos (por ejemplo, “CreateEnhMetaFile Example Program” + *Chr(0)* + “Example metafile” + *Chr(0)* + *Chr(0)*). Este parámetro puede ser **nil**, en cuyo caso no habrá cadena de descripción almacenada en el metafichero. Bajo Windows 95/98, la longitud máxima para la descripción del metafichero mejorado es de 16.384 bytes.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de un contexto de dispositivo de metafichero mejorado, que puede ser usado en llamadas a cualquier función del GDI. En caso contrario, devuelve cero.

#### Véase además

*CloseEnhMetaFile*, *CopyEnhMetaFile*, *DeleteEnhMetaFile*,  
*GetEnhMetaFileDescription*, *GetEnhMetaFileHeader*, *PlayEnhMetaFile*

#### Ejemplo

##### Listado 5-10: Creando un metafichero mejorado

```
procedure TForm1.Button1Click(Sender: TObject);
var
    {Aquí se almacena información importante sobre las dimensiones de la pantalla,
     que es usada cuando se crea el rectángulo de referencia}
    WidthInMM,
    HeightInMM,
    WidthInPixels,
    HeightInPixels: Integer;

    {Almacena la relación de milímetros por píxel}
    MMPerPixelHorz,
    MMPerPixelVer: Integer;

    ReferenceRect: Trect;    // Rectángulo de referencia
    MetafileDC: HDC;        // Manejador del contexto de dispositivo del metafichero
    TheMetafile: HENHMETAFILE; // manejador de metafichero
    TheBrush, OldBrush: HBRUSH; // manejadores a las brochas usadas
begin
    {La función CreateEnhMetaFile asume que las dimensiones del rectángulo de
     referencia están en términos de 0,01 milímetros. Por eso las siguientes líneas
     son necesarias para obtener relación de milímetros por píxel. Esto puede ser
     usado para crear un rectángulo de referencia con las dimensiones apropiadas.}

    {Recupera el tamaño de la pantalla, en milímetros}
    WidthInMM := GetDeviceCaps(Form1.Canvas.Handle, HORZSIZE);
    HeightInMM := GetDeviceCaps(Form1.Canvas.Handle, VERTSIZE);
```

```

{Recupera el tamaño de la pantalla, en píxeles}
WidthInPixels:=GetDeviceCaps(Form1.Canvas.Handle, HORZRES);
HeightInPixels:=GetDeviceCaps(Form1.Canvas.Handle, VERTRES);

{Calcula la relación de milímetros por píxel. Las medidas en milímetros tienen
 que ser multiplicadas por 100 para obtener las unidades de medida apropiadas
 que espera la función CreateEnhMetaFile (donde 1 equivale a 0,01 milímetros.)}
MMPerPixelHorz := (WidthInMM * 100) div WidthInPixels;
MMPerPixelVer := (HeightInMM * 100) div HeightInPixels;

{Crea nuestro rectángulo de referencia para el metafile}
ReferenceRect.Top := 0;
ReferenceRect.Left := 0;
ReferenceRect.Right := Image1.Width * MMPerPixelHorz;
ReferenceRect.Bottom := Image1.Height * MMPerPixelVer;

{Crea un metafile que será guardado en disco}
MetafileDC := CreateEnhMetaFile(Form1.Canvas.Handle, 'Example.emf',
                                @ReferenceRect,
                                'CreateEnhMetaFile Example Program' + Chr(0) +
                                'Example Metafile' + Chr(0) + Chr(0));

{Muestra un texto en el metafile}
TextOut(MetafileDC, 15, 15, 'This is an enhanced metafile.', 29);

{Crea una brocha diagonal sombreada y la selecciona en el metafile}
TheBrush := CreateHatchBrush(HS_DIAGCROSS, clRed);
OldBrush := SelectObject(MetafileDC, TheBrush);

{Dibuja un rectángulo relleno}
Rectangle(MetafileDC, 15, 50, 250, 250);

{Elimina la brocha actual}
SelectObject(MetafileDC, OldBrush);
DeleteObject(TheBrush);

{Crea una brocha sombreada horizontal y la selecciona en el metafile}
TheBrush := CreateHatchBrush(HS_CROSS, clBlue);
OldBrush := SelectObject(MetafileDC, TheBrush);

{Dibuja una elipse rellena}
Ellipse(MetafileDC, 15, 50, 250, 250);

{Elimina la brocha actual}
SelectObject(MetafileDC, OldBrush);
DeleteObject(TheBrush);

{Cierra el metafile, guardándolo en disco y recuperando un manejador}
TheMetafile:=CloseEnhMetaFile(MetafileDC);

{Dibuja el metafile dentro del área de dibujo de Image1}
PlayEnhMetaFile(Image1.Canvas.Handle, TheMetafile, Image1.Canvas.Cliprect);

{Hemos terminado con el metafile, eliminamos su manejador}

```

```
DeleteEnhMetaFile(TheMetafile);
end;
```

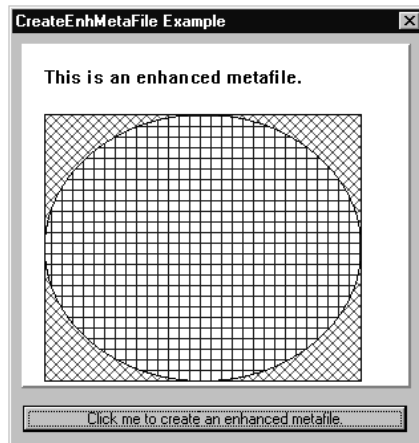


Figura 5-11:  
El nuevo  
metaarchivo

## **DeleteEnhMetaFile**      **Windows.Pas**

### **Sintaxis**

```
DeleteEnhMetaFile(
    p1: HENHMETAFILE                    {manejador de un metaarchivo mejorado}
): BOOL;                                {devuelve TRUE o FALSE}
```

### **Descripción**

Esta función elimina el metaarchivo asociado al manejador dado. Si este metaarchivo está almacenado en memoria, se elimina el metaarchivo y se libera la memoria asociada. Si este manejador identifica un metaarchivo almacenado en disco, el manejador y la memoria asociada son liberados, pero el fichero no es destruido.

### **Parámetros**

*p1*: Manejador del metaarchivo mejorado que será borrado.

### **Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

### **Véase además**

*CopyEnhMetaFile*, *CreateEnhMetaFile*, *GetEnhMetaFile*

### **Ejemplo**

Vea el Listado 5-10 bajo *CreateEnhMetaFile*.

**EnumEnhMetaFile****Windows.Pas****Sintaxis**

```
EnumEnhMetaFile(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: HENHMETAFILE;        {manejador del metaarchivo mejorado a enumerar}
    p3: TFNEnhMFEnumProc;    {puntero a función de respuesta definida por aplicación}
    p4: Pointer;              {puntero a datos definidos por la aplicación}
    const p5: TRect           {puntero a un registro TRect}
): BOOL;                    {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recorre todos los registros del metaarchivo especificado, pasando cada uno de ellos a una función de respuesta definida por la aplicación. Esta función de respuesta procesa los registros según sea necesario, y la enumeración continúa hasta que todos los registros hayan sido procesados o la función de respuesta devuelva cero.

**Parámetros**

*DC*: Manejador del contexto de dispositivo en el cual el metaarchivo puede ser ejecutado. Este parámetro es pasado directamente a la función de respuesta y se le puede asignar cero si la función de respuesta no ejecuta los registros del metaarchivo.

*p2*: Manejador del metaarchivo mejorado cuyos registros serán enumerados.

*p3*: La dirección de la función de respuesta definida por la aplicación.

*p4*: Puntero a datos definidos por la aplicación. Este parámetro está previsto sólo para propósitos específicos de la aplicación y su valor es pasado directamente a la función de respuesta definida por la aplicación.

*p5*: Puntero a un registro *TRect* que contiene las coordenadas superior izquierda e inferior derecha del rectángulo que contiene la imagen del metaarchivo, expresadas en unidades lógicas. Los puntos a lo largo del borde de este rectángulo están incluidos en la imagen. Si el parámetro *DC* contiene cero, este parámetro es ignorado.

**Valor que devuelve**

Si la función tiene éxito y la función de respuesta enumeró todos los registros del metaarchivo mejorado, devuelve TRUE. Si *EnumEnhMetaFile* falla, o la función de respuesta no enumeró todos los registros del metaarchivo mejorado, devuelve FALSE.

**Sintaxis de la función de respuesta**

```
EnumerateEnhMetafileProc(
    DisplaySurface: HDC;                {manejador de contexto de dispositivo}
    var MetafileTable: THandleTable;    {puntero a una tabla de manejadores del
                                         metaarchivo}
    var MetafileRecord: TEnhMetaRecord; {puntero a un registro del metaarchivo}
    ObjectCount: Integer;               {cantidad de objetos con manejadores}
```

```

var Data: Longint           {puntero a datos definidos por la aplicación}
): Integer;                 {devuelve un valor entero}

```

### Descripción

Esta función recibe un puntero a un registro de metaфichero por cada registro almacenado en el metaфichero mejorado que está siendo enumerado. Puede ejecutar cualquier acción deseada.

### Parámetros

*DisplaySurface*: Manejador del contexto de dispositivo en el cual el registro del metaфichero puede ser ejecutado. Si los registros del metaфichero no van a ser ejecutados por la función de respuesta, este parámetro puede ser cero.

*MetafileTable*: Puntero a un *array* de valores de tipo *HGDIOBJ*. Este *array* contiene manejadores de objetos gráficos, tales como plumas y brochas, que conforman el metaфichero. La primera entrada en el *array* es un manejador del propio metaфichero mejorado.

*MetafileRecord*: Puntero a un registro de tipo *TEnhMetaRecord*. Este registro define el elemento actual del metaфichero que está siendo enumerado. El registro *TEnhMetaRecord* se define de la siguiente forma:

```

TEnhMetaRecord = packed record
    iType: DWORD;           {tipo de registro}
    nSize: DWORD;           {tamaño del registro}
    dParm: array[0..0] of DWORD; {array de parámetros}
end;

```

*iType*: Indica el tipo de registro e indirectamente la función del GDI que creó el registro. Esta es una constante de la forma *EMR\_XXX* y todos los tipos de registros están listados en el fichero **Windows.Pas**.

*nSize*: El tamaño del registro en bytes.

*dParm*: Un *array* de parámetros usados por la función del GDI identificada por el campo *iType*.

*ObjectCount*: Un valor entero que indica el número de objetos gráficos del GDI con manejadores en la tabla de manejadores a la que apunta el parámetro *MetafileTable*.

*Data*: Puntero a datos definidos por la aplicación. Estos datos están previstos sólo para usos específicos de la aplicación.

### Valor que devuelve

La función de respuesta debe devolver un valor distinto de cero para continuar la enumeración; en caso contrario, debe devolver cero.

### Véase además

*GetEnhMetaFile*, *PlayEnhMetaFile*, *PlayEnhMetaFileRecord*

*Ejemplo***Listado 5-II: Cambiando las brochas en un metaarchivo mejorado**

```
{función de respuesta para enumerar los registros del metaarchivo mejorado}
function EnumerateEnhMetafile(DisplaySurface: HDC;
                               var MetafileTable: THandleTable;
                               var MetafileRecord: TEnhMetaRecord;
                               ObjectCount: Integer;
                               var Data: Longint): Integer; stdcall;

implementation

procedure TForm1.FileListBox1Click(Sender: TObject);
var
    TheMetafile: HENHMETAFILE; // almacena un metaarchivo mejorado
begin
    {Abre y recupera un manejador del metaarchivo seleccionado}
    TheMetafile := GetEnhMetaFile(PChar(FileListBox1.FileName));

    {Borra la última imagen}
    Image1.Canvas.FillRect(Image1.Canvas.ClipRect);

    {Enumera los registros en el metaarchivo}
    EnumEnhMetaFile(Image1.Canvas.Handle, TheMetafile, @EnumerateEnhMetafile,
                    nil, Image1.BoundsRect);
end;

{Esta función se dispara para todos los registros almacenados en el metaarchivo}
function EnumerateEnhMetafile(DisplaySurface: HDC;
                               var MetafileTable: THandleTable;
                               var MetafileRecord: TEnhMetaRecord;
                               ObjectCount: Integer;
                               var Data: Longint): Integer;

var
    NewBrush: HBRUSH; // almacena una nueva brocha
    BrushInfo: TLogBrush; // define una nueva brocha
begin
    {Si el metaarchivo está tratando de crear una brocha...}
    if MetafileRecord.iType = EMR_CREATEBRUSHINDIRECT then
        begin
            {...lo intercepta y crea nuestra propia brocha}
            BrushInfo.lbStyle := BS_SOLID;
            BrushInfo.lbColor := clRed;
            BrushInfo.lbHatch := 0;
            NewBrush := CreateBrushIndirect(BrushInfo);

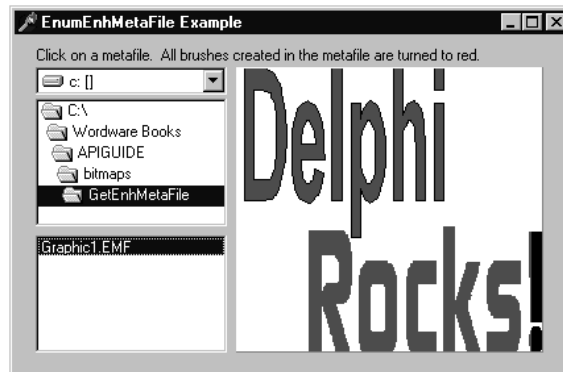
            {Selecciona esta brocha en el contexto de dispositivo donde el metaarchivo
             está siendo ejecutado. Esto reemplazará todas las brochas en el metaarchivo
             con una brocha roja sólida}
            SelectObject(DisplaySurface, NewBrush);
        end
    else
        {Si no es un registro de brocha creada, lo ejecutará}
```

```

    PlayEnhMetaFileRecord(DisplaySurface, MetafileTable, MetafileRecord,
                          ObjectCount);

    Result:=1; // continua la enumeración
end;
```

Figura 5-12:  
Todas las  
brochas en el  
metaarchivo  
mejorado  
fueron  
cambiadas



## GetBitmapBits Windows.Pas

### Sintaxis

```

GetBitmapBits(
    Bitmap: HBITMAP;           {manejador de mapa de bits}
    Count: Longint;            {cantidad de bytes en el array Bits}
    Bits: Pointer              {puntero a un array de bytes}
): Longint;                   {devuelve la cantidad de bytes recuperados del bitmap}
```

### Descripción

Esta función copia la información de color del mapa de bits especificado en un *buffer*. La función *GetBitmapBits* se mantiene en el API Win32 por motivos de compatibilidad. Las aplicaciones basadas en Win32 deben usar la función *GetDIBits*.

### Parámetros

*Bitmap*: Manejador del mapa de bits del cual la información de color es recuperada.

*Count*: Indica la cantidad de bytes del *array* al que apunta el parámetro *Bits*.

*Bits*: Puntero a un *array* de bytes que recibe la información de color del mapa de bits.

### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de bytes recuperados del mapa de bits; en caso contrario, devuelve cero.

### Véase además

*CreateBitmap*, *GetDIBits*, *SetBitmapBits*, *SetDIBits*

*Ejemplo***Listado 5-12: Recuperando datos del color del mapa de bits**

```

procedure TForm1.Button1Click(Sender: TObject);
type
  TBitmapBits = array[0..0] of Byte;
var
  BitmapBits: ^TBitmapBits;           // almacena los bytes del mapa de bits
  LoopRow, LoopCol: Integer;          // variables de control de bucles
begin
  {Asigna a la rejilla de cadenas la dimensiones del mapa de bits}
  StringGrid1.ColCount := Image1.Picture.Bitmap.Width;
  StringGrid1.RowCount := Image1.Picture.Bitmap.Height;

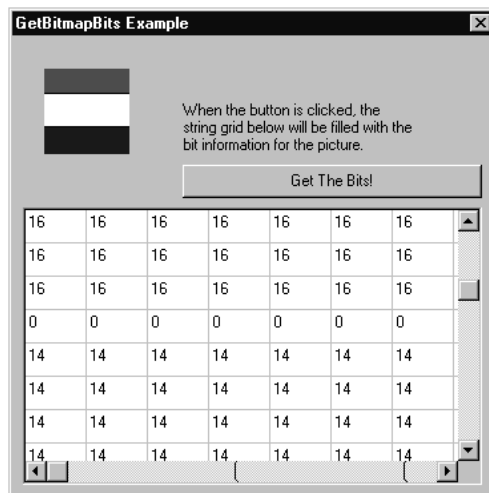
  {Reserva dinámicamente el espacio necesario para los datos de color del bitmap}
  GetMem(BitmapBits, StringGrid1.RowCount * StringGrid1.ColCount);

  {Recupera los datos de color del mapa de bits}
  GetBitmapBits(Image1.Picture.Bitmap.Handle, StringGrid1.RowCount *
    StringGrid1.ColCount, BitmapBits);

  {Muestra los valores que definen el mapa de bits en la rejilla de cadena. Debido
  a que se trata de un mapa de bits de 256 colores, estos valores representan
  índices en la paleta de colores del mapa de bits.}
  for LoopRow := 0 to Image1.Height-1 do
    for LoopCol := 0 to Image1.Width-1 do
      StringGrid1.Cells[LoopCol, LoopRow] :=
        IntToStr(BitmapBits[LoopRow * Image1.Width + LoopCol]);

  {Libera la memoria}
  FreeMem(BitmapBits);
end;

```



*Figura 5-13:  
Los bits del  
mapa de bits*

**GetBitmapDimensionEx Windows.Pas****Sintaxis**

```
GetBitmapDimensionEx(
  p1: HBITMAP;           {manejador de mapa de bits}
  var p2: TSize           {dirección de registro TSize}
): BOOL;                {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recupera las dimensiones preferidas del mapa de bits especificado, asignadas anteriormente por una llamada a la función *SetBitmapDimensionEx*. Si esta función no ha sido llamada, el registro *TSize* devuelto por *GetBitmapDimensionEx* contendrá cero en todos los campos.

**Parámetros**

*p1*: Manejador del mapa de bits cuyas dimensiones preferidas serán recuperadas.

*p2*: Puntero a un registro *TSize*. El registro *TSize* describe el ancho y la altura de un rectángulo y se define como:

*TSize* = **record**

```
  cx: Longint;           {ancho preferido}
  cy: Longint;           {altura preferida}
```

**end;**

*cx*: El ancho preferido del mapa de bits. Cada unidad representa 0.1 milímetros.

*cy*: La altura preferida del mapa de bits. Cada unidad representa 0.1 milímetros.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*SetBitmapDimensionEx*

**Ejemplo**

Vea el Listado 5-5 bajo *CreateBitmap*.

**GetDIBits Windows.Pas****Sintaxis**

```
GetDIBits(
  DC: HDC;               {manejador de contexto de dispositivo}
  Bitmap: HBITMAP;       {manejador de un mapa de bits corriente}
  StartScan: UINT;       {línea de barrido inicial}
```

```

NumScans: UINT;           {número total de líneas de barrido}
Bits: Pointer;            {puntero a los valores de los bits del mapa de bits DIB}
var BitsInfo: TBitmapInfo; {puntero a la información del mapa de bits DIB}
Usage: UINT               {opciones de tipo de color}
): Integer;               {devuelve el número de líneas de barrido copiadas}

```

### Descripción

Esta función crea un mapa de bits independiente del dispositivo a partir de la imagen almacenada en un mapa de bits dependiente del dispositivo, recuperando los valores de los bits del mapa de bits dependiente del dispositivo especificado y almacenándolos en un *buffer* en el formato definido por el registro *TBitmapInfo*, al que apunta el parámetro *BitsInfo*. El parámetro *Bitmap* puede también especificar el manejador de un mapa de bits independiente del dispositivo, en cuyo caso esta función puede ser usada para crear una copia del DIB en el formato que se desee, especificado en el registro *TBitmapInfo*. Si el formato de color del DIB solicitado no se corresponde con el formato de color del mapa de bits especificado, se generará una paleta de colores para el DIB, usando los colores por defecto para el formato de color solicitado. Si el parámetro *BitsInfo* indica un formato de colores de 16 bits por píxel o superior para el DIB, entonces no se generará una tabla de colores.

### Parámetros

*DC*: Manejador de un contexto de dispositivo. El mapa de bits dependiente del dispositivo especificado por el parámetro *Bitmap* utiliza la paleta actual de este contexto de dispositivo para su información de color.

*Bitmap*: Manejador del mapa de bits dependiente del dispositivo cuyos valores de bits son copiados.

*StartScan*: Especifica la línea de barrido de inicio a recuperar del mapa de bits dependiente del dispositivo.

*NumScans*: Especifica la cantidad de líneas de barrido a recuperar del mapa de bits dependiente del dispositivo.

*Bits*: Puntero a un *buffer* que recibe los valores de los bits del mapa de bits. La aplicación es responsable de reservar suficiente memoria para almacenar la imagen del mapa de bits, y de liberar esta memoria cuando no sea necesaria. Los seis campos del registro *TBitmapInfoHeader* contenido a su vez en el registro *TBitmapInfo* al que apunta el parámetro *BitsInfo* tienen que ser inicializados para indicar las dimensiones y formato de color de los bits del DIB solicitado. Si este parámetro es **nil**, la función rellena el registro *TBitmapInfo* al que apunta el parámetro *BitsInfo* con las dimensiones y el formato de color del mapa de bits dependiente del dispositivo especificado por el parámetro *Bitmap*. En este caso, al campo *biSize* del registro *TBitmapInfoHeader* hay que asignarle *SizeOf(TBitmapInfoHeader)* o la función fallará. Adicionalmente, si el campo *biBitCount* es cero, el registro *TBitmapInfo* es rellenado sin la tabla de colores del mapa de bits. Esta funcionalidad puede utilizarse para solicitar los atributos de un mapa de bits.

*BitsInfo*: Puntero al registro *TBitmapInfo* que describe el formato deseado para el DIB, incluyendo la información sobre sus dimensiones y tabla de colores. El registro *TBitmapInfo* se define de la siguiente forma:

```
TBitmapInfo = packed record
    bmiHeader: TBitmapInfoHeader;    {información de encabezamiento}
    bmiColors: array[0..0] of TRGBQuad; {tabla de colores usada por el mapa de
                                         bits}
end;
```

El registro *TBitmapInfoHeader* se define como:

```
TBitmapInfoHeader = packed record
    biSize: DWORD;    {tamaño del registro en bytes}
    biWidth: Longint; {ancho del mapa de bits en píxeles}
    biHeight: Longint; {altura del mapa de bits en píxeles}
    biPlanes: Word;    {número de planos de colores}
    biBitCount: Word;  {bits necesarios para describir un color}
    biCompression: DWORD; {opciones de compresión}
    biSizeImage: DWORD; {tamaño de la imagen en bytes}
    biXPelsPerMeter: Longint; {píxeles horizontales por medida del
                               dispositivo de destino}
    biYPelsPerMeter: Longint; {píxeles verticales por medida del
                               dispositivo de destino}
    biClrUsed: DWORD;    {número de índices de colores usados}
    biClrImportant: DWORD; {número de índices de colores importantes}
end;
```

El registro *TRGBQuad* se define como:

```
TRGBQuad = packed record
    rgbBlue: Byte;    {intensidad del color azul}
    rgbGreen: Byte;   {intensidad del color verde}
    rgbRed: Byte;     {intensidad del color rojo}
    rgbReserved: Byte; {valor reservado}
end;
```

Para ver una explicación de estos registros, consulte la función *CreateDIBSection*.

*Usage*: Valor que indica el tipo de información sobre el color, almacenado en el campo *bmiColors* del registro *TBitmapInfo* al que apunta el parámetro *BitsInfo*. Este parámetro puede tomar un valor de la Tabla 5-9.

#### Valor que devuelve

Si la función tiene éxito y el parámetro *Bits* no es **nil**, devuelve la cantidad de líneas de barrido copiadas del mapa de bits dependiente del dispositivo. Si la función tiene éxito y el parámetro *Bits* es **nil**, el registro *TBitmapInfo* al que apunta el parámetro *BitsInfo*

es inicializado con las dimensiones y formato del mapa de bits dependiente del dispositivo y la función devuelve la cantidad total de líneas de barrido en el mapa de bits dependiente del dispositivo. Si la función falla, devuelve cero.

Véase además

*CreateDIBitmap, CreateDIBSection, GetBitmapBits, SetDIBits*

*Ejemplo*

#### Listado 5-13: Creando un DIB a partir de un mapa de bits dependiente del dispositivo

```

procedure TForm1.Button1Click(Sender: TObject);
var
    TheBitmap: HBITMAP;           // manejador de un mapa de bits corriente
    RegularBitmapInfo: Windows.TBitmap; // registro con información de un mapa de
                                     // bits de Windows
    BitmapInfo: PBitmapInfo;      // puntero a registro con información
                                     // de un DIB
    BitmapBits: Pointer;          // puntero a valores de un DIB
begin
    {Obtiene un manejador de un mapa de bits del sistema}
    TheBitmap:=LoadBitmap(0, MakeIntResource(OBM_CHECKBOXES));

    {Rellena un registro TBitmap de Windows}
    GetObject(TheBitmap, SizeOf(Windows.TBitmap), @RegularBitmapInfo);

    {Reserva memoria para el encabezamiento del mapa de bits DIB}
    GetMem(BitmapInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));

    {Inicializa la información del mapa de bits}
    BitmapInfo^.bmiHeader.biWidth      := RegularBitmapInfo.bmWidth;
    BitmapInfo^.bmiHeader.biHeight     := RegularBitmapInfo.bmHeight;
    BitmapInfo^.bmiHeader.biPlanes     := 1;
    BitmapInfo^.bmiHeader.biBitCount   := 8;      // 256 colores
    BitmapInfo^.bmiHeader.biCompression := BI_RGB; // sin compresión
    BitmapInfo^.bmiHeader.biSizeImage   := 0;      // Windows determina el tamaño
    BitmapInfo^.bmiHeader.biXPelsPerMeter := 0;
    BitmapInfo^.bmiHeader.biYPelsPerMeter := 0;
    BitmapInfo^.bmiHeader.biClrUsed     := 0;
    BitmapInfo^.bmiHeader.biClrImportant := 0;
    BitmapInfo^.bmiHeader.biSize        := SizeOf(TBitmapInfoHeader);

    {Reserva suficiente memoria para almacenar los bits del mapa de bits}
    GetMem(BitmapBits, RegularBitmapInfo.bmWidth * RegularBitmapInfo.bmHeight);
    {Recupera los bits de un mapa de bits corriente en formato DIB}
    GetDIBits(Form1.Canvas.Handle, TheBitmap, 0, RegularBitmapInfo.bmHeight,
        BitmapBits, BitmapInfo^, 0);

    {Muestra este nuevo mapa de bits DIB}
    SetDIBitsToDevice(Form1.Canvas.Handle,
        (Form1.Width div 2) - (BitmapInfo^.bmiHeader.biWidth div 2), 25,
        BitmapInfo^.bmiHeader.biWidth,
        BitmapInfo^.bmiHeader.biHeight, 0, 0, 0,

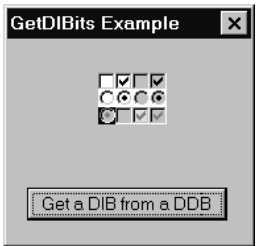
```

```
        BitmapInfo^.bmiHeader.biHeight, BitmapBits, BitmapInfo^,
        DIB_RGB_COLORS);

{Elimina el mapa de bits corriente}
DeleteObject(TheBitmap);

{Libera la memoria}
FreeMem(BitmapInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));
```

Figura 5-14:  
DIB creado a  
partir de un  
mapa de bits  
dependiente  
del dispositivo



```
FreeMem(BitmapBits, RegularBitmapInfo.bmWidth * RegularBitmapInfo.bmHeight);
end;
```

Tabla 5-9: Valores del parámetro Usage de GetDIBits

Valor	Descripción
DIB_PAL_COLORS	El campo bmiColors del registro TBitmapInfo es un array de índices de 16 bits en la paleta lógica actual del contexto de dispositivo especificado. Este valor no debe ser usado si el mapa de bits va a ser guardado en disco.
DIB_RGB_COLORS	El campo bmiColors del registro TBitmapInfo es un array de valores de colores RGB literales.

GetEnhMetaFile

Windows.Pas

Sintaxis

```
GetEnhMetaFile(
    p1: PChar                {nombre de fichero de metaфichero mejorado}
): HENHMETAFILE;           {devuelve un manejador de metaфichero mejorado}
```

Descripción

Esta función crea un metaфichero mejorado y devuelve su manejador, utilizando la información de metaфichero mejorado almacenada en el fichero especificado. Cuando la aplicación deje de necesitar el metaфichero mejorado, deberá eliminarlo llamando a la función *DeleteObject*. Esta función sólo abre metaфicheros en formato mejorado.

**Parámetros**

*p1*: Cadena de caracteres terminada en nulo que contiene el nombre de fichero del metaarchivo mejorado que será abierto.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de metaarchivo mejorado; en caso contrario, devuelve cero.

**Véase además**

*CreateEnhMetaFile*, *DeleteEnhMetaFile*, *GetEnhMetaFileHeader*,  
*GetEnhMetaFileDescription*

**Ejemplo****Listado 5-14: Abriendo metaarchivos mejorados**

```
procedure TForm1.FileListBox1DbClick(Sender: TObject);
var
    TheMetafile: HENHMETAFILE;           // manejador del metaarchivo original
    CopyMetafile: HENHMETAFILE;          // manejador del metaarchivo copiado
    MetafileInfo: TEnhMetaHeader;         // encabezamiento del metaarchivo
    MetafileDescription: PChar;           // almacena la descripción del metaarchivo
    DescriptionSize: UINT;                 // almacena el tamaño de la descripción
    CorrectedRect: TRect;                  // relación de aspecto de un rectángulo
    ScaleVert,                             // se utilizan para calcular
    ScaleHorz,                             // la relación de aspecto corregida
    ScaleLeast: Real;
begin
    {Abre y recupera un manejador del metaarchivo seleccionado}
    TheMetafile := GetEnhMetaFile(PChar(FileListBox1.FileName));

    {Recupera el tamaño de la cadena de descripción}
    DescriptionSize := GetEnhMetaFileDescription(TheMetafile, 0, nil);

    {Reserva dinámicamente un buffer para almacenar la descripción}
    MetafileDescription := StrAlloc(DescriptionSize + 1);

    {Recupera la cadena de descripción del metaarchivo, si existe}
    GetEnhMetaFileDescription(TheMetafile, DescriptionSize, MetafileDescription);

    {Recupera la información del encabezamiento del metaarchivo}
    GetEnhMetaFileHeader(TheMetafile, SizeOf(MetafileInfo), @MetafileInfo);

    {Busca la menor relación entre el tamaño del rectángulo límite del metaarchivo
    y el rectángulo TImage}
    ScaleVert:=Image1.Height / (MetafileInfo.rc1Bounds.Bottom -
                                MetafileInfo.rc1Bounds.Top);
    ScaleHorz:=Image1.Width / (MetafileInfo.rc1Bounds.Right -
                                MetafileInfo.rc1Bounds.Left);

    {Busca la menor relación}
    if ScaleVert < ScaleHorz then
        ScaleLeast := ScaleVert
```

```

else
    ScaleLeast := ScaleHorz;

{Determina el nuevo rectángulo límite usando este factor de compresión}
CorrectedRect.Left := Trunc(MetafileInfo.rclBounds.Left * ScaleLeast);
CorrectedRect.Top := Trunc(MetafileInfo.rclBounds.Top * ScaleLeast);
CorrectedRect.Right := Trunc(MetafileInfo.rclBounds.Right * ScaleLeast);
CorrectedRect.Bottom := Trunc(MetafileInfo.rclBounds.Bottom * ScaleLeast);

{Ajusta el nuevo rectángulo límite para comenzar en la esquina superior izq.}
CorrectedRect.Left := 0;
CorrectedRect.Top := 0;
CorrectedRect.Right := CorrectedRect.Right - CorrectedRect.Left;
CorrectedRect.Bottom := CorrectedRect.Bottom - CorrectedRect.Top;

{Comienza mostrando la información del metafichero}
with ListBox1.Items do begin
    Clear;
    Add('Description -');
    if DescriptionSize > 0 then
    begin
        {La primera parte de la descripción es el nombre del programa usado
        para crear el metafichero, seguido de un carácter nulo}
        Add(string(MetafileDescription));

        {La segunda parte contiene el nombre del metafichero, seguido de dos
        caracteres nulos}
        Add(string(PChar(MetafileDescription + StrLen(MetafileDescription) + 1)));
    end
    else
        Add('No Description found. ');
    Add('Type: ' + IntToStr(MetafileInfo.iType));
    Add('Size: ' + IntToStr(MetafileInfo.nSize));
    Add('Bounding Rectangle -');
    Add('    Left: ' + IntToStr(MetafileInfo.rclBounds.Left));
    Add('    Top: ' + IntToStr(MetafileInfo.rclBounds.Top));
    Add('    Right: ' + IntToStr(MetafileInfo.rclBounds.Right));
    Add('    Bottom: ' + IntToStr(MetafileInfo.rclBounds.Bottom));
    Add('Frame Rectangle - (1 = .01 millimeters)');
    Add('    Left: ' + IntToStr(MetafileInfo.rclFrame.Left));
    Add('    Top: ' + IntToStr(MetafileInfo.rclFrame.Top));
    Add('    Right: ' + IntToStr(MetafileInfo.rclFrame.Right));
    Add('    Bottom: ' + IntToStr(MetafileInfo.rclFrame.Bottom));
    Add('Signature: ' + IntToStr(MetafileInfo.dSignature));
    Add('Version: ' + IntToStr(MetafileInfo.nVersion));
    Add('Bytes: ' + IntToStr(MetafileInfo.nBytes));
    Add('Records: ' + IntToStr(MetafileInfo.nRecords));
    Add('Handles: ' + IntToStr(MetafileInfo.nHandles));
    Add('Reserved: ' + IntToStr(MetafileInfo.sReserved));
    Add('Description Size: ' + IntToStr(MetafileInfo.nDescription));
    Add('Description Offset: ' + IntToStr(MetafileInfo.offDescription));
    Add('Palette Entries: ' + IntToStr(MetafileInfo.nPalEntries));
    Add('Reference Resolution, Pixels - ');
    Add('    Horizontal: ' + IntToStr(MetafileInfo.szlDevice.cx));
    Add('    Vertical: ' + IntToStr(MetafileInfo.szlDevice.cy));

```

end;

```
{Borra cualquier imagen previa}
Image1.Canvas.Fillrect(Image1.Canvas.Cliprect);
Image2.Canvas.Fillrect(Image2.Canvas.Cliprect);
{Muestra el metaarchivo como aparece originalmente}
PlayEnhMetaFile(Image1.Canvas.Handle, TheMetafile, CorrectedRect);

{Hace una copia del metaarchivo original en memoria}
CopyMetafile := CopyEnhMetaFile(TheMetafile, nil);

{Muestra este metaarchivo copiado}
PlayEnhMetaFile(Image2.Canvas.Handle, CopyMetafile, Image1.Canvas.Cliprect);

{Elimina los manejadores de ambos metaarchivos pues ya no son necesarios}
DeleteEnhMetaFile(TheMetafile);
```

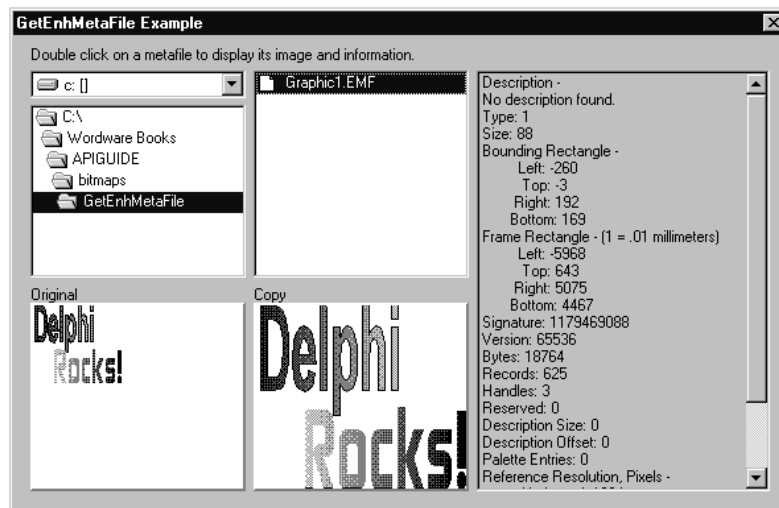


Figura 5-15:  
Un  
metaarchivo  
mejorado

```
DeleteEnhMetaFile(CopyMetafile);
```

```
{Libera la memoria reservada para la descripción}
StrDispose(MetafileDescription);
```

end;

### **GetEnhMetaFileDescription**      **Windows.Pas**

#### **Sintaxis**

```
GetEnhMetaFileDescription(
  p1: HENHMETAFILE;      {manejador de metaarchivo}
  p2: UINT;                {tamaño del buffer al que apunta el parámetro p3}
  p3: PChar                {puntero a buffer}
): UINT;                    {devuelve la longitud de la descripción}
```

**Descripción**

Esta función extrae la cadena de descripción de un metaarchivo mejorado, copiándola en el *buffer* especificado. Esta descripción es opcional, de manera que algunos metaarchivos mejorados pueden no tenerla. La cadena de descripción contiene dos cadenas individuales, separadas por un carácter nulo y terminada por dos caracteres nulos (por ejemplo, “CreateEnhMetaFile Example Program” + *Chr(0)* + “Example metafile” + *Chr(0)* + *Chr(0)*). Normalmente, la primera cadena contiene el nombre del paquete gráfico que creó el metaarchivo mejorado y la segunda cadena contiene el título de la imagen del metaarchivo mejorado. Consulte la función *CreateEnhMetaFile* para más información acerca de cómo incluir una descripción en un metaarchivo mejorado. Bajo Windows 95/98, la longitud máxima de una cadena de descripción es de 16.384 bytes.

**Parámetros**

*p1*: Manejador del metaarchivo cuya cadena de descripción será recuperada.

*p2*: Especifica el tamaño del *buffer* de texto al que apunta el parámetro *p3*, en caracteres. Si la longitud de la cadena de descripción es superior a este valor será truncada.

*p3*: Puntero al *buffer* de texto que recibe la cadena de descripción. Este parámetro puede ser **nil**.

**Valor que devuelve**

Si la función tiene éxito, devuelve el número de caracteres copiados en el *buffer*. Si la función tiene éxito y el parámetro *p3* tiene valor **nil**, simplemente devuelve la longitud de la cadena de descripción, en caracteres. Si la cadena de descripción no existe, devuelve cero. Si la función falla, devuelve *GDI\_ERROR*.

**Véase además**

*CreateEnhMetaFile*

**Ejemplo**

Vea el Listado 5-14 bajo *GetEnhMetaFile*.

**GetEnhMetaFileHeader****Windows.Pas****Sintaxis**

```
GetEnhMetaFileHeader(
  p1: HENHMETAFILE;      {manejador de metaarchivo mejorado}
  p2: UINT;               {tamaño del buffer al que apunta el parámetro p3}
  p3: PEnhMetaHeader      {puntero a un registro TEnhMetaHeader}
): UINT;                 {devuelve la cantidad de bytes copiados}
```

**Descripción**

Esta función recupera un registro que contiene la información de encabezamiento del metaarchivo mejorado especificado. La información del encabezamiento describe completamente el metaarchivo mejorado, incluyendo elementos tales como su paleta de colores, sus dimensiones y su tamaño.

**Parámetros**

*p1*: Manejador de un metaarchivo mejorado cuya información de encabezamiento será recuperada.

*p2*: Especifica el tamaño del *buffer* al que apunta el parámetro *p3*, en bytes. A este parámetro se le debe asignar *SizeOf(TEnhMetaHeader)*.

*p3*: Un puntero a un registro *TEnhMetaHeader* que recibe la información sobre el metaarchivo mejorado especificado. Este parámetro puede ser **nil**. El registro *TEnhMetaHeader* se define de la siguiente forma:

*TEnhMetaHeader* = **packed record**

<i>iType</i> : DWORD;	{identificador del tipo de registro}
<i>nSize</i> : DWORD;	{tamaño del registro, en bytes}
<i>rclBounds</i> : TRect;	{las dimensiones del rectángulo límite}
<i>rclFrame</i> : TRect;	{las dimensiones de la imagen rectangular}
<i>dSignature</i> : DWORD;	{firma del metaarchivo mejorado}
<i>nVersion</i> : DWORD;	{versión del metaarchivo mejorado}
<i>nBytes</i> : DWORD;	{tamaño del metaarchivo mejorado, en bytes}
<i>nRecords</i> : DWORD;	{cantidad de registros en el metaarchivo mejorado}
<i>nHandles</i> : Word;	{cantidad de manejadores en tabla de manejadores}
<i>sReserved</i> : Word;	{valor reservado}
<i>nDescription</i> : DWORD;	{longitud de la cadena de descripción}
<i>offDescription</i> : DWORD;	{desplazamiento de la cadena de descripción}
<i>nPalEntries</i> : DWORD;	{número de entradas en la paleta de colores}
<i>szlDevice</i> : TSize;	{resolución de dispositivo de referencia, en píxeles}
<i>szlMillimeters</i> : TSize;	{resolución de dispositivo de referencia, en mm.}

**end;**

*iType*: Siempre se le asigna el identificador de registro de metaarchivo mejorado *EMR\_HEADER*.

*nSize*: Especifica el tamaño del registro *TEnhMetaHeader*, en bytes.

*rclBounds*: Un registro *TRect* que contiene las coordenadas, en unidades del dispositivo, del rectángulo más pequeño que puede contener la imagen almacenada en el metaarchivo mejorado. Estas dimensiones son suministradas por el GDI.

*rclFrame*: Un registro *TRect* que contiene las coordenadas, en centésimas de milímetro, del rectángulo que encierra la imagen almacenada en el metaarchivo mejorado. Esta coordenadas son suministradas por la función que originalmente creó el metaarchivo mejorado.

*dSignature*: Siempre se le asigna la constante de firma de metaфicheros mejorados *ENHMETA\_SIGNATURE*.

*nVersion*: La versión del metaфichero. La versión más actual, en el momento que se escribía este libro, era \$10000.

*nBytes*: El tamaño del metaфichero mejorado en bytes.

*nRecords*: La cantidad de registros contenidos en el metaфichero.

*nHandles*: La cantidad de manejadores almacenados en la tabla de manejadores del metaфichero mejorado. Nota: el índice 0 de esta tabla está reservado.

*sReserved*: Este campo es reservado y se le asigna cero.

*nDescription*: La longitud de la cadena de descripción opcional del metaфichero mejorado. Si el metaфichero mejorado no contiene una cadena de descripción, a este campo se le asigna cero.

*offDescription*: El desplazamiento a partir del inicio del registro *TEnhMetaHeader*, al *array* que contiene los caracteres de la cadena de descripción opcional del metaфichero mejorado. Si el metaфichero mejorado no contiene una cadena de descripción a este campo se le asigna cero.

*nPalEntries*: La cantidad de entradas en la paleta de colores del metaфichero mejorado. Si el metaфichero mejorado no contiene una paleta de colores a este campo se le asigna cero.

*szlDevice*: Un registro *TSize* que contiene la resolución horizontal y vertical del dispositivo de referencia para el metaфichero mejorado, en píxeles.

*szlMillimeters*: Un registro *TSize* que contiene la resolución horizontal y vertical del dispositivo de referencia para el metaфichero mejorado, en milímetros.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de bytes que fueron copiados al registro *TEnhMetaHeader* al que apunta el parámetro *p3*. Si la función tiene éxito y al parámetro *p3* se le asigna **nil**, devuelve el tamaño del *buffer* necesario para almacenar la información del encabezamiento. Si la función falla, devuelve cero.

#### Véase además

*CreateEnhMetaFile*, *GetEnhMetaFile*, *GetEnhMetaFileDescription*, *PlayEnhMetaFile*

#### Ejemplo

Vea el Listado 5-14 bajo *GetEnhMetaFile*.

## GetStretchBltMode

## Windows.Pas

### Sintaxis

```
GetStretchBltMode(
    DC: HDC           {manejador de contexto de dispositivo}
): Integer;          {devuelve el modo de escalamiento del mapa de bits}
```

**Descripción**

Esta función recupera el modo actual de escalamiento de mapas de bits del contexto de dispositivo especificado. Este modo determina cómo se añaden o eliminan filas y columnas de un mapa de bits cuando se llama a la función *StretchBlt*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo modo de escalamiento será recuperado.

**Valor que devuelve**

Si la función tiene éxito, devuelve el modo actual de escalamiento del mapa de bits. Este puede ser un valor de la Tabla 5-10. Si la función falla, devuelve cero.

**Véase además**

*SetStretchBltMode*, *StretchBlt*

**Ejemplo**

Vea el Listado 5-15 bajo *LoadBitmap*.

**Tabla 5-10: Valores que devuelve *GetStretchBltMode***

Valor	Descripción
BLACKONWHITE	Ejecuta una operación booleana <b>and</b> usando los valores de los colores para píxeles existentes y eliminados. Si el mapa de bits es monocromático, este modo mantiene los píxeles negros a expensas de los blancos.
COLORONCOLOR	Borra píxeles sin hacer ningún intento de conservar información sobre ellos.
HALFTONE	Mapea píxeles del mapa de bits de origen en bloques de píxeles sobre el mapa de bits de destino. El color del píxel de destino es el promedio de los colores de los píxeles de origen. Este modo requiere más tiempo de procesamiento que otros, pero produce imágenes de mejor calidad. Si este modo es utilizado, la aplicación tiene que llamar a la función <i>SetBrushOrgEx</i> para reinicializar la brocha original, u ocurrirá un desalineamiento de la brocha.
STRETCH_ANDSCANS	Equivalente a BLACKONWHITE.
STRETCH_DELETESCANS	Equivalente a COLORONCOLOR.
STRETCH_HALFTONE	Equivalente a HALFTONE.
STRETCH_ORSCANS	Equivalente a WHITEONBLACK.
WHITEONBLACK	Ejecuta una operación booleana <b>or</b> usando los valores de los colores para píxeles eliminados y existentes. Si el mapa de bits es monocromático, este modo mantiene los píxeles blancos a expensas de los negros.

**LoadBitmap**      **Windows.Pas****Sintaxis**

```
LoadBitmap(
  hInstance: HINST;           {manejador de instancia}
  lpBitmapName: PAnsiChar     {nombre de recurso de mapa de bits}
): HBITMAP;                   {devuelve un manejador de mapa de bits}
```

**Descripción**

Esta función carga un mapa de bits de los recursos del fichero ejecutable, devolviendo su manejador. Cuando la aplicación termina de utilizar el mapa de bits, deberá eliminarlo llamando a la función *DeleteObject*. Esta función asume que el mapa de bits tendrá sólo 16 colores. Utilice la función *LoadResource* para cargar mapas de bits con mayor resolución de colores.

**Parámetros**

*hInstance*: Manejador de instancia de un módulo cuyo fichero ejecutable contiene los recursos del mapa de bits a cargar.

*lpBitmapName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso de mapa de bits a cargar. La función *MakeIntResource* puede ser utilizada con un identificador de recurso para suministrar un valor para este parámetro. Para cargar alguno de los recursos de mapa de bits predefinidos utilizados por el Win32 API, asígnele al parámetro *hInstance* el valor cero (0) y utilice la función *MakeIntResource* con uno de los valores de la Tabla 5-11 para este parámetro.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del mapa de bits cargado del recurso del fichero ejecutable; en caso contrario, devuelve cero.

**Véase además**

*BitBlt*, *CreateBitmap*, *CreateBitmapIndirect*, *CreateCompatibleBitmap*,  
*CreateDIBitmap*, *CreateDIBSection*, *DeleteObject*, *LoadResource*, *StretchBlt*

**Ejemplo****Listado 5-15: Cargando un mapa de bits predefinido**

```
procedure TForm1.ComboBox1Change(Sender: TObject);
var
  TheBitmap: HBITMAP;           // almacena el mapa de bits
  BitmapInfo: Windows.TBitmap; // almacena la información sobre el mapa de bits
  OffscreenDC: HDC;             // manejador de un contexto de dispositivo
                                   // fuera de pantalla

{Esto define todos los mapas de bits del sistema accesibles en Windows}
type
  TBitmapTypes = array[0..25] of Integer;
```

```

const
  BitmapTypes: TBitmapTypes = (OBM_CLOSE, OBM_UPARROW, OBM_DNARROW, OBM_RGARROW,
                                OBM_LFARROW, OBM_REDUCE, OBM_ZOOM, OBM_RESTORE,
                                OBM_REDUCED, OBM_ZOOMD, OBM_RESTORED, OBM_UPARROWD,
                                OBM_DNARROWD, OBM_RGARROWD, OBM_LFARROWD,
                                OBM_MNARROW, OBM_COMBO, OBM_UPARROWI, OBM_DNARROWI,
                                OBM_RGARROWI, OBM_LFARROWI, OBM_BTFSIZE,
                                OBM_CHECK, OBM_CHECKBOXES, OBM_BTNCORNERS,
                                OBM_SIZE);

begin
  {Borra las últimas imágenes}
  Image1.Canvas.Brush.Color:=clBtnFace;
  Image2.Canvas.Brush.Color:=clBtnFace;
  Image1.Canvas.FillRect(Image1.Canvas.Cliprect);
  Image2.Canvas.FillRect(Image2.Canvas.Cliprect);

  {Carga el mapa de bits seleccionado}
  TheBitmap := LoadBitmap(0, MakeIntResource(BitmapTypes[ComboBox1.ItemIndex]));

  {Crea contexto de dispositivo fuera de pantalla y selecciona mapa de bits en él}
  OffscreenDC := CreateCompatibleDC(0);
  SelectObject(OffscreenDC, TheBitmap);

  {Rellena un registro de un mapa de bits}
  GetObject(TheBitmap, SizeOf(Windows.TBitmap), @BitmapInfo);

  {Dibuja el mapa de bits en Image1}
  BitBlt(Image1.Canvas.Handle, 45, 45, Image1.Width, Image1.Height, OffscreenDC,
        0,0,SRCCOPY);

  {Verifica que el modo de escalamiento en Image2 es el que se quiere}
  if GetStretchBltMode(Image2.Canvas.Handle) <> COLORONCOLOR then
    SetStretchBltMode(Image2.Canvas.Handle, COLORONCOLOR);

  {Dibuja el mapa de bits en Image2, agrandándolo para llenar la imagen}
  StretchBlt(Image2.Canvas.Handle, 0, 0, Image2.Width, Image2.Height,
    OffscreenDC, 0, 0, BitmapInfo.bmWidth, BitmapInfo.bmHeight,
    SRCCOPY);

  {Elimina el mapa de bits}

```

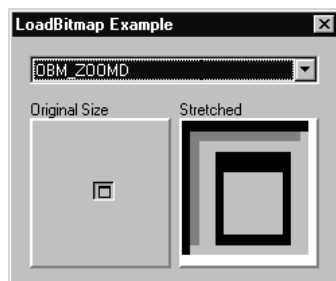


Figura 5-16:  
Un recurso de  
mapa de bits  
predefinido

```

DeleteObject(TheBitmap);

```

```
{Elimina el contexto de dispositivo fuera de pantalla}
DeleteDC(OffscreenDC);
end;
```

**Tabla 5-II: Valores del parámetro lpBitmapName de LoadBitmap**

Valor	Descripción
OBM_BTNCORNERS	Carga el recurso del mapa de bits para marcador de esquina del sistema.
OBM_BTFSIZE	Carga el recurso del mapa de bits para el botón de dimensionamiento.
OBM_CHECK	Carga el recurso del mapa de bits para la marca de verificación por defecto.
OBM_CHECKBOXES	Carga la colección de símbolos del sistema de las casillas de verificación.
OBM_CLOSE	Carga el recurso de iconos del menú del sistema.
OBM_COMBO	Carga el recurso del mapa de bits para la flecha de despliegue de la caja de combinación.
OBM_DNARROW	Carga el recurso del mapa de bits para la flecha hacia abajo de la barra de desplazamiento, en estado normal.
OBM_DNARROWD	Carga el recurso del mapa de bits para la flecha hacia abajo de la barra de desplazamiento, en estado pulsado.
OBM_DNARROWI	Carga el recurso del mapa de bits para la flecha hacia abajo de la barra de desplazamiento, en estado deshabilitado.
OBM_LFARROW	Carga el recurso del mapa de bits para la flecha hacia la izquierda de la barra de desplazamiento, en estado normal.
OBM_LFARROWD	Carga el recurso del mapa de bits para la flecha hacia la izquierda de la barra de desplazamiento, en estado pulsado.
OBM_LFARROWI	Carga el recurso del mapa de bits para la flecha hacia la izquierda de la barra de desplazamiento, en estado deshabilitado.
OBM_MNARROW	Carga el recurso del mapa de bits usado para indicar una opción de menú que contiene un submenú.
OBM_REDUCE	Carga el recurso del mapa de bits para un botón de minimizar, en estado normal.
OBM_REDUCEd	Carga el recurso del mapa de bits para un botón de minimizar, en estado pulsado.
OBM_RESTORE	Carga el recurso del mapa de bits para un botón de restaurar, en estado normal.
OBM_RESTOREd	Carga el recurso del mapa de bits para un botón de restaurar, en estado pulsado.
OBM_RGARROW	Carga el recurso del mapa de bits para la flecha hacia la derecha de la barra de desplazamiento, en estado normal.
OBM_RGARROWD	Carga el recurso del mapa de bits para la flecha hacia la derecha de la barra de desplazamiento, en estado pulsado.

Valor	Descripción
OBM_RGARROWI	Carga el recurso del mapa de bits para la flecha hacia la derecha de la barra de desplazamiento, en estado deshabilitado.
OBM_SIZE	Carga el recurso del mapa de bits para la esquina de redimensionamiento.
OBM_UPARROW	Carga el recurso del mapa de bits para la flecha hacia arriba de la barra de desplazamiento, en estado normal.
OBM_UPARROWD	Carga el recurso del mapa de bits para la flecha hacia arriba de la barra de desplazamiento, en estado pulsado.
OBM_UPARROWI	Carga el recurso del mapa de bits para la flecha hacia arriba de la barra de desplazamiento, en estado deshabilitado.
OBM_ZOOM	Carga el recurso del mapa de bits para un botón de maximizar, en estado normal.
OBM_ZOOMD	Carga el recurso del mapa de bits para un botón de maximizar, en estado pulsado.

## LoadImage Windows.Pas

### Sintaxis

```
LoadImage(
    hInst: HINST;           {manejador de la instancia que contiene la imagen}
    ImageName: PChar;       {nombre de la imagen}
    ImageType: UINT;        {indicador de tipo de imagen}
    X: Integer;             {ancho de la nueva imagen}
    Y: Integer;             {altura de la nueva imagen}
    Flags: UINT             {opciones de carga de la imagen}
): THandle;               {devuelve un manejador de la imagen cargada}
```

### Descripción

Esta función carga un icono, un cursor, un metaarchivo mejorado o un mapa de bits desde un fichero o desde recursos ejecutables. La imagen puede ser redimensionada como se desee y numerosas opciones afectan a la imagen final cargada.

### Parámetros

*hInst*: Manejador de instancia del módulo que contiene la imagen que será cargada.

*ImageName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre de la imagen que será cargada. Si el parámetro *Flags* incluye la opción *LR\_LOADFROMFILE*, este parámetro contendrá un puntero a una cadena terminada en nulo que especifica el nombre de fichero de la imagen a cargar.

*ImageType*: Un valor que indica el tipo de imagen que será cargada. Este parámetro puede tomar un valor de la Tabla 5-12.

*X*: Indica el ancho deseado de la imagen, en píxeles. Si a este parámetro se le asigna cero y el parámetro *Flags* no incluye la opción *LR\_DEFAULTSIZE*, al ancho de la imagen cargada se le asigna el ancho del recurso original. Si a este parámetro se le asigna cero y el parámetro *Flags* contiene la opción *LR\_DEFAULTSIZE*, al ancho se le asigna el valor devuelto por la llamada a *GetSystemMetrics(SM\_CXICON)* o *GetSystemMetrics(SM\_CXCURSOR)*, si la imagen cargada es un icono o un cursor.

*Y*: Indica la altura deseada de la imagen, en píxeles. Si a este parámetro se le asigna cero y el parámetro *Flags* no incluye la opción *LR\_DEFAULTSIZE*, a la altura de la imagen cargada se le asigna la altura del recurso original. Si este parámetro es cero y el parámetro *Flags* contiene la opción *LR\_DEFAULTSIZE*, a la altura se le asigna el valor devuelto por *GetSystemMetrics(SM\_CYICON)* o *GetSystemMetrics(SM\_CYCURSOR)*, si la imagen cargada es un icono o un cursor.

*Flags*: Un valor que indica acciones adicionales que se ejecutan cuando la imagen es cargada. Este parámetro puede tomar uno o más valores de la Tabla 5-13.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la imagen cargada; en caso contrario, devuelve cero.

#### Véase además

*CopyImage*, *GetSystemMetrics*, *LoadBitmap*, *LoadCursor*, *LoadIcon*

#### Ejemplo

##### Listado 5-16: Cargando imágenes de mapas de bits desde archivos

```
procedure TForm1.FileListBox1Click(Sender: TObject);
var
  TheBitmap: THandle;           // imagen de mapa de bits a cargar
  BitmapInfo: Windows.TBitmap; // almacena la información del mapa de bits
  TheOffscreenDC: HDC;         // manejador de contexto de dispositivo en memoria
begin
  {Crea un contexto de dispositivo en memoria}
  TheOffscreenDC := CreateCompatibleDC(0);

  {Carga el fichero del mapa de bits especificado}
  TheBitmap := LoadImage(0, PChar(FileListBox1.FileName), IMAGE_BITMAP, 0, 0,
    LR_LOADFROMFILE);

  {Recupera información sobre el mapa de bits (el ancho y la altura serán usados)}
  GetObject(TheBitmap, SizeOf(Windows.TBitmap), @BitmapInfo);

  {Selecciona el mapa de bits en el contexto de dispositivo en memoria}
  SelectObject(TheOffscreenDC, TheBitmap);
  {Copia la imagen en Image1 en su tamaño original}
  BitBlt(Image1.Canvas.Handle, 0, 0, Image1.Width, Image1.Height, TheOffscreenDC,
    0, 0, SRCCOPY);

  {Copia la imagen en Image2, y la comprime hasta que se adapte}
```

```

StretchBlt(Image2.Canvas.Handle, 0, 0, Image2.Width, Image2.Height,
TheOffscreenDC, 0, 0,
BitmapInfo.bmWidth, BitmapInfo.bmHeight, SRCCOPY);

{Actualiza la imagen en la pantalla}
Image1.Refresh;
Image2.Refresh;

```

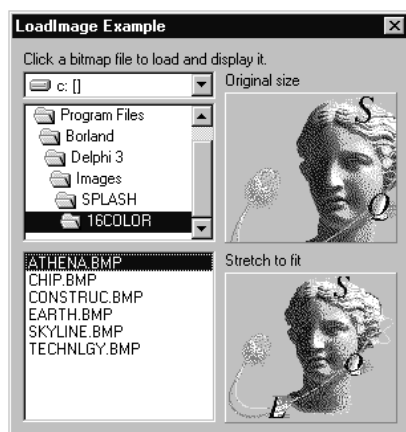


Figura 5-17:  
La imagen del  
mapa de bits  
cargado

```

{Elimina la imagen cargada y el contexto de dispositivo fuera de pantalla}
DeleteDC(TheOffscreenDC);
DeleteObject(TheBitmap);
end;

```

Tabla 5-12: Valores del parámetro ImageType de LoadImage

Valor	Descripción
IMAGE_BITMAP	La imagen es un mapa de bits.
IMAGE_CURSOR	La imagen es un cursor.
IMAGE_ENHMETAFILE	La imagen es un metaarchivo mejorado.
IMAGE_ICON	La imagen es un icono.

Tabla 5-13: Valores del parámetro Flags de LoadImage

Valor	Descripción
LR_CREATEDIBSECTION	Si el parámetro ImageType contiene el valor IMAGE_BITMAP, esta función devuelve un manejador de un mapa de bits de sección DIB.
LR_DEFAULTCOLOR	Carga la imagen en su formato de colores definido. Esta es la opción por defecto.

Valor	Descripción
LR_DEFAULTSIZE	Solamente para una imagen de icono o cursor. Esta opción hace que la función cargue la imagen usando el ancho y altura de éstos por defecto, según indica la función <code>GetSystemMetrics</code> .
LR_LOADFROMFILE	Indica que la cadena terminada en nulo a la que apunta el parámetro <code>ImageName</code> contiene un nombre de fichero y la imagen debe ser cargada desde disco.
LR_LOADMAP3DCOLORS	Busca en los píxeles de la imagen cargada y reemplaza los píxeles gris oscuro ( <code>RGB(128,128,128)</code> ) con el color del sistema <code>COLOR_3DSHADOW</code> , los píxeles grises ( <code>RGB(192,192,192)</code> ) con el color del sistema <code>COLOR_3DFACE</code> y los píxeles gris claros ( <code>RGB(223,223,223)</code> ) con el color del sistema <code>COLOR_3DLIGHT</code> .
LR_LOADTRANSPARENT	Recupera el valor del primer píxel en la imagen, y reemplaza todos los píxeles del mismo color en la imagen con el color del sistema <code>COLOR_WINDOW</code> . Esto tiene el mismo efecto que realizar un “blit” de la imagen con el área de dibujo usando la función <code>BrushCopy</code> . Si la opción <code>LR_LOADMAP3DCOLORS</code> ha sido incluida, <code>LR_LOADTRANSPARENT</code> se hace precedente, pero reemplaza el color indicado del píxel con el color del sistema <code>COLOR_3DFACE</code> .
LR_MONOCHROME	Crea una versión en blanco y negro de la imagen original.
LR_SHARED	Para recursos, esta opción hace que la función devuelva el mismo manejador para recursos idénticos cargados varias veces. Sin esta opción, <code>LoadImage</code> devuelve un manejador diferente cada vez que el mismo recurso es cargado. No especifique esta opción para imágenes cargadas desde ficheros o para imágenes que cambiarán después de cargadas.

## PatBlt

## Windows.Pas

### Sintaxis

```
PatBlt(
    DC: HDC;           {manejador de contexto de dispositivo}
    X: Integer;         {coordenada horizontal de inicio del rectángulo a rellenar}
    Y: Integer;         {coordenada vertical de inicio del rectángulo a rellenar}
    Width: Integer;     {ancho del rectángulo a rellenar}
    Height: Integer;    {altura del rectángulo a rellenar}
    ROP: DWORD          {operación de barrido}
): BOOL;              {devuelve TRUE o FALSE}
```

**Descripción**

Esta función rellena un rectángulo usando la brocha actualmente seleccionada en el contexto de dispositivo especificado, combinando los colores de la brocha y el destino usando la operación de barrido especificada. Algunos dispositivos pueden no soportar la función *PatBlt*; utilice la función *GetDeviceCaps* para determinar si el dispositivo de destino soporta transferencias de bloques de bits.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el cual el rectángulo será dibujado.

*X*: La coordenada horizontal de la esquina superior izquierda del rectángulo que será relleno, en unidades lógicas.

*Y*: La coordenada vertical de la esquina superior izquierda del rectángulo que será relleno, en unidades lógicas.

*Width*: El ancho del rectángulo que será relleno, en unidades lógicas.

*Height*: La altura del rectángulo que será relleno, en unidades lógicas.

*Rop*: El código de la operación de barrido. Este valor determina cómo los píxeles de la brocha usada para pintar el rectángulo serán combinados con los píxeles actuales en el contexto de dispositivo y puede ser un valor de la Tabla 5-14.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetDeviceCaps*, *CreateBrush*, *CreatePatternBrush*

**Ejemplo****Listado 5-17: Rellenando un fondo**

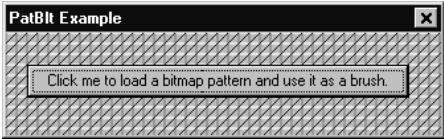
```
procedure TForm1.Button1Click(Sender: TObject);
var
    BitmapPattern: HBITMAP;    // almacena el patrón de la brocha del mapa de bits
    PatternBrush: HBRUSH;      // almacena el manejador de la brocha patrón
    OldBrush: HBRUSH;          // almacena la brocha original
begin
    {Carga un mapa de bits desde el fichero de recursos}
    BitmapPattern := LoadBitmap(hInstance, 'BRUSHPATTERN');

    {Lo usa para crear una brocha patrón}
    PatternBrush := CreatePatternBrush(BitmapPattern);

    {Selecciona esta nueva brocha en el contexto de dispositivo del formulario}
    OldBrush := SelectObject(Canvas.Handle, PatternBrush);
```

```
{Pinta un rectángulo relleno con el patrón}
PatBlt(Canvas.Handle, 0, 0, Width, Height, PATINVERT);
```

Figura 5-18:  
El patrón de la  
brocha  
resultante



```
{Reemplaza el manejador de la brocha original}
SelectObject(Canvas.Handle, OldBrush);

{Ni la brocha patrón ni el mapa de bits son ya necesarios y los eliminamos}
DeleteObject(PatternBrush);
DeleteObject(BitmapPattern);
end;
```

Tabla 5-14: Valores del parámetro ROP de PatBlt

Valor	Descripción
BLACKNESS	Rellena los píxeles en el rectángulo especificado en el destino con el color indicado en el índice 0 de la paleta física. Por defecto este color es negro.
DSTINVERT	Invierte el color de los píxeles en el rectángulo destino.
PATCOPY	Copia el patrón contenido en la brocha seleccionada en el contexto de dispositivo de destino directamente en el destino.
PATINVERT	Combina los colores de los píxeles del patrón contenido en la brocha seleccionada en el contexto de dispositivo de destino con los colores de los píxeles en el destino, usando el operador booleano <b>xor</b> .
WHITENESS	Rellena los píxeles en el rectángulo especificado en el destino con el color indicado en el índice 255 de la paleta física. Por defecto este color es blanco.

**PlayEnhMetaFile**

**Windows.Pas**

*Sintaxis*

```
PlayEnhMetaFile(
  DC: HDC;                {manejador de contexto de dispositivo}
  p2: HENHMETAFILE;        {manejador de metaфichero mejorado}
  const p3: TRect           {puntero a registro TRect}
); BOOL;                   {devuelve TRUE o FALSE}
```

*Descripción*

Esta función reproduce el metaфichero mejorado identificado por el parámetro *p2* sobre el contexto de dispositivo especificado. El metaфichero puede ser recortado definiendo

la región de recorte del contexto de dispositivo antes de ejecutar el metaarchivo. Si el metaarchivo mejorado contiene una paleta de colores, la aplicación puede mantener la consistencia de colores creando y activando una paleta de colores en el contexto de dispositivo antes de ejecutar el metaarchivo. Utilice la función *GetEnhMetaFilePaletteEntries* para recuperar la paleta de colores del metaarchivo mejorado. Un metaarchivo mejorado puede ser incrustado en otro metaarchivo mejorado creado anteriormente, utilizando esta función para ejecutar el primer metaarchivo en el contexto de dispositivo del segundo. El estado del contexto de dispositivo especificado es preservado por esta función. Si un objeto fue creado, pero no borrado cuando el metaarchivo original fue creado, esta función elimina ese objeto “errante” después de que el metaarchivo es ejecutado.

#### Parámetros

*DC*: Manejador del contexto de dispositivo sobre el cual el metaarchivo mejorado será reproducido (dibujado).

*p2*: Manejador del metaarchivo mejorado a dibujar.

*p3*: Puntero a un registro TRect. El metaarchivo mejorado será dibujado dentro de las coordenadas especificadas por este registro. Estas coordenadas se especifican en unidades lógicas. El campo *rclFrame* del encabezamiento del metaarchivo mejorado es usado para mapear el metaarchivo dentro de las coordenadas de rectángulo.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateEnhMetaFile*, *GetEnhMetaFile*, *GetEnhMetaFileHeader*, *GetEnhMetaFilePaletteEntries*, *PlayEnhMetaFileRecord*

#### Ejemplo

Vea el Listado 5-10 bajo *CreateEnhMetaFile*.

### **PlayEnhMetaFileRecord**

### **Windows.Pas**

#### Sintaxis

<b>PlayEnhMetaFileRecord</b> (	
<i>DC</i> : HDC;	{manejador de contexto de dispositivo}
<b>var</b> <i>p2</i> : THandleTable;	{puntero a tabla de manejadores de metaarchivos}
<b>const</b> <i>p3</i> : TEnhMetaRecord;	{puntero a un registro de metaarchivo}
<i>p4</i> : UINT	{cantidad de manejadores en la tabla de manejadores del metaarchivo}
);	{devuelve TRUE o FALSE}

**Descripción**

Esta función ejecuta las funciones del GDI identificadas por el registro del metaфichero mejorado. *PlayEnhMetaFileRecord* está prevista para ser usada conjuntamente con la función *EnumEnhMetaFile* para procesar y ejecutar un metaфichero mejorado registro por registro. Los parámetros *DC*, *p2* y *p3* tienen que corresponderse exactamente con el contexto de dispositivo, la tabla de manejadores y el contador de manejadores de la tabla de manejadores del metaфichero pasados a la función de respuesta que usa la función *EnumEnhMetaFile*. Si el registro pasado en el parámetro *p3* no es reconocido, es ignorado y la función devuelve TRUE.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el cual el metaфichero mejorado está siendo ejecutado.

*p2*: Puntero a una tabla de manejadores de objetos GDI. Estos objetos definen la imagen del metaфichero mejorado.

*p3*: Puntero al registro *TEnhMetaRecord* que define la entrada del metaфichero mejorado que será ejecutada. El registro *TEnhMetaRecord* se define como:

*TEnhMetaRecord* = **packed record**

<i>iType</i> : DWORD;	{identificador de registro del metaфichero}
<i>nSize</i> : DWORD;	{tamaño del registro en bytes}
<i>dParm</i> : <b>array</b> [0..0] of DWORD;	{array de parámetros}

**end;**

Consulte la función de respuesta asociada a *EnumEnhMetaFile* para ver una explicación de esta estructura.

*p4*: La cantidad de manejadores almacenados en la tabla de manejadores del metaфichero mejorado.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*EnumEnhMetaFile*, *PlayEnhMetaFile*

**Ejemplo**

Vea el Listado 5-11 bajo *EnumEnhMetaFile*.

**SetBitmapBits Windows.Pas****Sintaxis**

```
SetBitmapBits(
    p1: HBITMAP;           {manejador de mapa de bits}
```

p2: DWORD;	{cantidad de bytes en el <i>array</i> de bits}
bits: Pointer	{puntero a <i>array</i> de bytes}
): Longint;	{devuelve la cantidad de bytes usados para componer el mapa de bits}

### Descripción

Esta función compone la imagen del mapa de bits especificado, partiendo de los valores almacenados en el *array bits*. La función *SetBitmapBits* está incluida en el API Win32 por razones de compatibilidad. Las aplicaciones basadas en Win32 deben usar la función *SetDIBits*.

### Parámetros

*p1*: Manejador del mapa de bits cuya imagen será compuesta a partir de los valores almacenados en el *array* al que apunta el parámetro *bits*.

*p2*: Indica la cantidad de bytes del *array* al que apunta el parámetro *bits*.

*bits*: Puntero a un *array* de bytes que contiene los datos de la imagen para el mapa de bits.

### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de bytes usados para componer los bits del mapa de bits; en caso contrario, devuelve cero.

### Véase además

*CreateBitmap*, *DestroyBitmap*, *GetBitmapBits*, *SetDIBits*

### Ejemplo

#### Listado 5-18: Componiendo los bits del mapa de bits

{Este ejemplo funcionará apropiadamente sólo con un controlador de vídeo de 256 colores.}

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
```

```
    BitmapBits: array[0..9999] of byte;    // almacena los bits del mapa de bits
```

```
    BitmapImage: TBitmap;                  // la imagen del mapa de bits
```

```
    Loop: Integer;                          // contador de bucle
```

```
const
```

```
    Started: Boolean = FALSE;               // controla el bucle más externo
```

```
begin
```

```
    {Cambia la variable de control del bucle}
```

```
    Started := not Started;
```

```
    {Cambia el texto del botón para reflejar su nuevo estado}
```

```
    if Started then
```

```
        Button1.Caption := 'Stop'
```

```
    else
```

```
        Button1.Caption := 'Start';
```

```
{Crea un mapa de bits de 100 x 100 píxeles}
BitmapImage := TBitmap.Create;
BitmapImage.Height := 100;
BitmapImage.Width := 100;

{Lo fuerza a ser un mapa de bits dependiente del dispositivo}
BitmapImage.HandleType := bmDDB;

{Este bucle continúa hasta que el botón sea presionado de nuevo}
while Started do
begin
  {Rellena la información de los bits del mapa de bits con el color blanco}
  FillChar(BitmapBits, SizeOf(BitmapBits), 255);

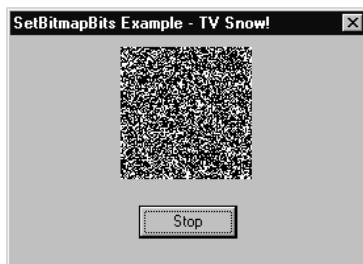
  {Pone 10000 píxeles escogidos aleatoriamente en color negro}
  for Loop := 0 to 1000 do
  begin
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
    BitmapBits[Random(100)*100+Random(100)] := 0;
  end;

  {Asigna los nuevos bits en el mapa de bits}
  SetBitmapBits(BitmapImage.Handle, 10000, @BitmapBits);

  {Copia el mapa de bits al área de dibujo del formulario}
  BitBlt(Form1.Canvas.Handle, 84, 8, 100, 100, BitmapImage.Canvas.Handle, 0,
    0, SRCCOPY);

  {Esto es necesario para que Windows opere adecuadamente}
```

Figura 5-19:  
Usando  
SetBitmapBits  
para producir  
un efecto  
nieve de TV



```
Application.ProcessMessages;
end;
```

```
{Libera nuestro mapa de bits}
BitmapImage.Free
```

end;

### **SetBitmapDimensionEx**

**Windows.Pas**

#### **Sintaxis**

```
SetBitmapDimensionEx(
  hBitmap: HBITMAP;      {manejador de mapa de bits}
  Width: Integer;         {ancho preferido del mapa de bits}
  Height: Integer;        {altura preferida del mapa de bits}
  Size: PSize              {puntero a registro TSize}
): BOOL;                  {devuelve TRUE o FALSE}
```

#### **Descripción**

Esta función asigna el ancho y altura preferidos del mapa de bits especificado, en términos de 0.1 milímetros. Estas dimensiones son para uso específico de la aplicación, no afectan a la apariencia de la imagen del mapa de bits y no son usadas por Windows. Una vez asignadas, estas dimensiones pueden ser recuperadas usando la función *GetBitmapDimensionEx*.

#### **Parámetros**

*hBitmap*: Manejador del mapa de bits cuyas dimensiones preferidas van a ser asignadas. Este no puede ser un manejador de un mapa de bits devuelto por la función *CreateDIBSection*.

*Width*: Un entero que especifica el ancho preferido del mapa de bits en términos de 0.1 milímetros.

*Height*: Un entero que especifica la altura preferida del mapa de bits en términos de 0.1 milímetros.

*Size*: Puntero a un registro TSize que recibirá las dimensiones previamente asignadas. Este parámetro puede ser **nil**.

#### **Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### **Véase además**

*GetBitmapDimensionEx*

#### **Ejemplo**

Vea el Listado 5-5 bajo *CreateBitmap*.

**SetDIBits****Windows.Pas****Sintaxis**

```

SetDIBits(
    DC: HDC;                {manejador de contexto de dispositivo}
    Bitmap: HBitmap;         {manejador de mapa de bits corriente}
    StartScan: UINT;         {línea de barrido inicial}
    NumScans: UINT;          {cantidad de líneas de barrido}
    Bits: Pointer;           {puntero a los valores de los bits del mapa de bits DIB}
    var BitsInfo: TBitmapInfo; {puntero a registro de información del mapa de bits DIB}
    Usage: UINT              {opciones de tipo de color}
); Integer;                 {devuelve la cantidad de líneas de barrido copiadas}

```

**Descripción**

Esta función copia los valores de los bits del área especificada del DIB a la que apunta el parámetro *Bits*, directamente en el mapa de bits dependiente del dispositivo indicado por el parámetro *Bitmap*. Observe que la velocidad de copia óptima del mapa de bits se obtiene cuando los bits del mapa de bits DIB especifican índices en la paleta del sistema.

**Parámetros**

*DC*: Manejador de un contexto de dispositivo. Si la opción *DIB\_PAL\_COLORS* es especificada en el parámetro *Usage*, los valores de los bits copiados desde el DIB usan los colores en la paleta actualmente activada de este contexto de dispositivo. Si la opción *DIB\_PAL\_COLORS* no es especificada, este parámetro es ignorado.

*Bitmap*: Manejador del mapa de bits cuyos valores de bits están siendo copiados.

*StartScan*: Especifica la línea de barrido inicial para comenzar la copia desde la imagen DIB a la que apunta el parámetro *Bits*.

*NumScans*: Especifica la cantidad de líneas de barrido a copiar al mapa de bits dependiente del dispositivo desde la imagen a la que apunta el parámetro *Bits*.

*Bits*: Puntero a la imagen que representa el DIB, en forma de un *array* de bytes.

*BitsInfo*: Puntero a un registro *TBitmapInfo* que describe el DIB, que incluye información sobre sus dimensiones y tabla de colores. El registro *TBitmapInfo* se define como:

```

TBitmapInfo = packed record
    bmiHeader: TBitmapInfoHeader;    {información de encabezamiento}
    bmiColors: array[0..0] of TRGBQuad;
                                         {tabla de colores usada por el mapa de bits}
end;

```

El registro *TBitmapInfoHeader* se define como:

TBitmapInfoHeader = **packed record**

biSize: DWORD;	{tamaño del registro en bytes}
biWidth: Longint;	{ancho del mapa de bits en píxeles}
biHeight: Longint;	{altura del mapa de bits en píxeles}
biPlanes: Word;	{número de planos de colores}
biBitCount: Word;	{bits necesarios para describir un color}
biCompression: DWORD;	{opciones de compresión}
biSizeImage: DWORD;	{tamaño de la imagen en bytes}
biXPelsPerMeter: Longint;	{píxeles horizontales por medida del dispositivo de destino}
biYPelsPerMeter: Longint;	{píxeles verticales por medida del dispositivo de destino}
biClrUsed: DWORD;	{número de índices de colores usados}
biClrImportant: DWORD;	{número de índices de colores importantes}

**end;**

El registro *TRGBQuad* se define como:

TRGBQuad = **packed record**

rgbBlue: Byte;	{intensidad del color azul}
rgbGreen: Byte;	{intensidad del color verde}
rgbRed: Byte;	{intensidad del color rojo}
rgbReserved: Byte;	{valor reservado}

**end;**

Para ver una explicación de estos registros, consulte la función *CreateDIBSection*.

*Usage:* Valor que indica el tipo de información sobre el color almacenada en el campo *bmiColors* del registro *TBitmapInfo* al que apunta el parámetro *BitsInfo*. Este parámetro puede tomar un valor de la Tabla 5-15.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de líneas de barrido que fueron copiadas al mapa de bits dependiente del dispositivo; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*BitBlt*, *CreateBitmap*, *GetDIBits*, *SetBitmapBits*

#### Ejemplo

##### Listado 5-19: Componiendo la imagen de un DDB desde un DIB

{Este ejemplo sólo funcionará apropiadamente con un controlador de vídeo de 256 colores.}

**procedure** TForm1.Button1Click(Sender: TObject);

```

var
TheBitmap: HBitmap;      // manejador de un mapa de bits corriente
OffScreen: HDC;          // contexto de dispositivo fuera de pantalla
Dib: HBITMAP;            // manejador de mapa de bits DIB
DibInfo: PBitmapInfo;    // puntero al registro de información del mapa de bits
BitsPtr: PByte;          // puntero a los bits del mapa de bits
ReferenceDC: HDC;        // manejador del contexto de dispositivo de referencia
iLoop: Integer;          // contador de bucle

SystemPalette: array[0..255] of TPaletteEntry;
                // necesario para convertir la paleta del sistema
                // en paletas compatibles DIB

const
  Started: Boolean = FALSE; // controla el bucle más externo
begin
  {Cambia la variable de control del bucle}
  Started := not Started;

  {Cambia el texto del botón para reflejar el nuevo estado}
  if Started then
    Button1.Caption := 'Stop'
  else
    Button1.Caption := 'Start';

  {Crea un mapa de bits de 128 x 128 píxeles}
  TheBitmap := CreateBitmap(128, 128, 1, 8, nil);

  {Crea un contexto de dispositivo fuera de pantalla compatible con la pantalla}
  OffScreen := CreateCompatibleDC(0);

  {Selecciona el nuevo mapa de bits en este contexto de dispositivo }
  SelectObject(OffScreen, TheBitmap);

  {Reserva la memoria necesaria para el registro de información del mapa de bits}
  GetMem(DibInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));

  {Inicializa la información del mapa de bits}
  DibInfo^.bmiHeader.biWidth      := 128;    // Imagen de 128 x 128 píxeles
  DibInfo^.bmiHeader.biHeight     := -128;   // orientado de arriba a abajo
  DibInfo^.bmiHeader.biPlanes     := 1;
  DibInfo^.bmiHeader.biBitCount   := 8;      // 256 colores
  DibInfo^.bmiHeader.biCompression := BI_RGB; // sin compresión
  DibInfo^.bmiHeader.biSizeImage   := 0;      // Windows determina el tamaño
  DibInfo^.bmiHeader.biXPelsPerMeter := 0;
  DibInfo^.bmiHeader.biYPelsPerMeter := 0;
  DibInfo^.bmiHeader.biClrUsed     := 0;
  DibInfo^.bmiHeader.biClrImportant := 0;
  DibInfo^.bmiHeader.biSize        := SizeOf(TBitmapInfoHeader);

  {Recupera la paleta del sistema actual}
  GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256, SystemPalette);

  {La paleta del sistema es devuelta como un array de registros TPaletteEntry,

```

```

que almacena los colores de la paleta en formato (Red, Green, Blue). Sin
embargo, el campo bmiColors del registro TBitmapInfo necesita un array de
registros TRGBQuad, que almacena los colores en formato (Blue, Green, Red).
Por eso hay que convertir los registros TPaletteEntry en registros TRGBQuad}
for iLoop := 0 to 255 do
begin
  DibInfo^.bmiColors[iLoop].rgbBlue      := SystemPalette[iLoop].peBlue;
  DibInfo^.bmiColors[iLoop].rgbRed       := SystemPalette[iLoop].peRed;
  DibInfo^.bmiColors[iLoop].rgbGreen     := SystemPalette[iLoop].peGreen;
  DibInfo^.bmiColors[iLoop].rgbReserved := 0;
end;

{Crea un contexto de dispositivo en memoria}
ReferenceDC := CreateCompatibleDC(0);

{Crea un DIB sobre el contexto de dispositivo en memoria, utilizando la
información del mapa de bits}
Dib := CreateDIBSection(ReferenceDC, DibInfo^, DIB_RGB_COLORS,
  Pointer(BitsPtr), 0, 0);

{Borra el contexto de dispositivo de referencia}
DeleteDC(ReferenceDC);

{Este bucle continúa hasta que el botón es presionado de nuevo}
while Started do
begin
  {Rellena la información de los bits del mapa de bits con blanco}
  FillMemory(BitsPtr, 128*128, $FF);

  {Pone 10000 píxeles escogidos aleatoriamente en color negro}
  for Loop := 0 to 1000 do
  begin
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
    PByte(Longint(BitsPtr)+Random(128)*128+Random(128))^ := 0;
  end;

  {Copia los valores de bits del DIB directamente en el mapa de bits del DDB}
  SetDIBits(Form1.Canvas.Handle, TheBitmap, 0, 128, BitsPtr, DibInfo^,
    DIB_RGB_COLORS);

  {Copia el mapa de bits al área de dibujo del formulario}
  BitBlt(Form1.Canvas.Handle, (Form1.Width div 2) - 64, 8, 128, 128,
    Offscreen, 0, 0, SRCCOPY);

```

```

    {Esto es necesario para que Windows opere correctamente}
    Application.ProcessMessages;
end;

{Destruye el contexto de dispositivo fuera de pantalla}
DeleteDC(Offscreen);

{Libera los mapas de bits}
DeleteObject(TheBitmap); DeleteObject(Dib);
FreeMem(DibInfo, SizeOf(TBitmapInfo) + 256 * SizeOf(TRGBQuad));
end;
```

Figura 5-20:  
La imagen  
DDB fue  
asignada  
desde bits  
almacenados  
en un DIB

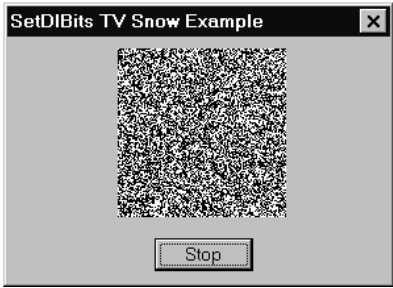


Tabla 5-15: Valores del parámetro Usage de SetDIBits

Valor	Descripción
DIB_PAL_COLORS	El campo bmiColors del registro TBitmapInfo es un array de índices de 16 bits en la paleta actual del contexto de dispositivo especificado. Este valor no debe ser usado si el mapa de bits va a ser guardado en disco.
DIB_RGB_COLORS	El campo bmiColors del registro TBitmapInfo es un array de valores de colores RGB literales.

SetDIBitsToDevice

Windows.Pas

Sintaxis

SetDIBitsToDevice(	
DC: HDC;	{manejador de contexto de dispositivo}
DestX: Integer;	{coordenada horizontal del rectángulo de destino}
DestY: Integer;	{coordenada vertical del rectángulo de destino}
Width: DWORD;	{ancho del DIB}
Height: DWORD;	{altura del DIB}
SrcX: Integer;	{coordenada horizontal del rectángulo de origen}
SrcY: Integer;	{coordenada vertical del rectángulo de origen}
n StartScan: UINT;	{línea de barrido inicial}
NumScans: UINT;	{cantidad total de líneas de barrido}
Bits: Pointer;	{puntero a los valores de los bits del mapa de bits}

```

var BitsInfo: TBitmapInfo; {puntero al registro de datos del mapa de bits DIB}
Usage: UINT                {opciones de tipo de color}
): Integer;                {devuelve el número de líneas de barrido copiadas}

```

### Descripción

Esta función copia píxeles desde la sección especificada de la imagen DIB al contexto de dispositivo de destino. En el caso de mapas de bits independientes del dispositivo de gran tamaño, la operación de copia se puede realizar por partes, llamando repetidamente a *SetDIBitsToDevice*, pasándole cada vez una porción diferente del DIB a través de los parámetros *nStartScan* y *NumScans*. Observe que la velocidad óptima de copia de un mapa de bits se obtiene cuando los bits del mapa de bits DIB especifican índices dentro de la paleta del sistema. Esta función fallará cuando sea llamada por un proceso que corra en segundo plano mientras que un proceso MS-DOS esté ejecutándose a pantalla completa en primer plano.

### Parámetros

*DC*: El contexto de dispositivo sobre el cual la imagen DIB es copiada y mostrada.

*DestX*: La coordenada horizontal de la esquina superior izquierda del rectángulo de destino en el contexto de dispositivo de destino, en unidades lógicas.

*DestY*: La coordenada vertical de la esquina superior izquierda del rectángulo de destino en el contexto de dispositivo de destino, en unidades lógicas.

*Width*: El ancho de la imagen DIB, en unidades lógicas.

*Height*: la altura de la imagen DIB, en unidades lógicas

*SrcX*: La coordenada horizontal de la esquina inferior izquierda del DIB, en unidades lógicas.

*SrcY*: La coordenada vertical de la esquina inferior izquierda del DIB, en unidades lógicas.

*nStartScan*: Especifica la línea de barrido de inicio de la imagen DIB (a la que apunta el parámetro *Bits*) a partir de la cual se realizará la copia.

*NumScans*: Especifica la cantidad de líneas de barrido a copiar al destino, desde la imagen a la que apunta el parámetro *Bits*.

*Bits*: Puntero a la imagen DIB, en forma de un *array* de bytes.

*BitsInfo*: Puntero a un registro *TBitmapInfo* que describe las características del DIB, incluyendo información sobre sus dimensiones y tabla de colores. El registro *TBitmapInfo* se define de la siguiente forma:

TBitmapInfo = **packed record**

bmiHeader: TBitmapInfoHeader; {información de encabezamiento}

bmiColors: **array**[0..0] **of** TRGBQuad;

{tabla de colores usada por el mapa de bits}

**end;**

El registro *TBitmapInfoHeader* se define como:

```
TBitmapInfoHeader = packed record
    biSize: DWORD;           {tamaño del registro en bytes}
    biWidth: Longint;        {ancho del mapa de bits en píxeles}
    biHeight: Longint;       {altura del mapa de bits en píxeles}
    biPlanes: Word;          {número de planos de colores}
    biBitCount: Word;        {bits necesarios para describir un color}
    biCompression: DWORD;    {opciones de compresión}
    biSizeImage: DWORD;      {tamaño de la imagen en bytes}
    biXPelsPerMeter: Longint; {píxeles horizontales por medida del
                             dispositivo de destino}
    biYPelsPerMeter: Longint; {píxeles verticales por medida del
                             dispositivo de destino}
    biClrUsed: DWORD;        {número de índices de colores usados}
    biClrImportant: DWORD;   {número de índices de colores importantes}
end;
```

El registro *TRGBQuad* se define como:

```
TRGBQuad = packed record
    rgbBlue: Byte;           {intensidad del color azul}
    rgbGreen: Byte;          {intensidad del color verde}
    rgbRed: Byte;            {intensidad del color rojo}
    rgbReserved: Byte;       {valor reservado}
end;
```

Para ver una explicación de estos registros, consulte la función *CreateDIBSection*.

*Usage*: Valor que indica el tipo de información sobre el color almacenada en el campo *bmiColors* del registro *TBitmapInfo* al que apunta el parámetro *BitsInfo*. Este parámetro puede tomar un valor de la Tabla 5-16.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de líneas de barrido que fueron copiadas al contexto de dispositivo destino; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*SetDIBits*, *StretchDIBits*

#### Ejemplo

Vea el Listado 5-9 bajo *CreateDIBSection*.

Tabla 5-16: Valores del parámetro Usage de SetDIBitsToDevice

Valor	Descripción
DIB_PAL_COLORS	El campo bmiColors del registro TBitmapInfo es un array de índices de 16 bits en la paleta actual del contexto de dispositivo especificado. Este valor no debe ser usado si el mapa de bits va a ser guardado en disco.
DIB_RGB_COLORS	El campo bmiColors del registro TBitmapInfo es un array de valores de colores RGB literales.

**SetStretchBltMode****Windows.Pas***Sintaxis*

```
SetStretchBltMode(
    DC: HDC;           {manejador de contexto de dispositivo}
    p2: Integer         {opción del modo de escalado del mapa de bits}
): Integer;           {devuelve el modo de escalado anterior}
```

*Descripción*

Esta función asigna el modo de escalado de mapas de bits del contexto de dispositivo especificado. Este modo define cómo las filas y columnas de un mapa de bits son añadidas o eliminadas cuando la función *StretchBlt* sea utilizada.

*Parámetros*

*DC*: Manejador del contexto de dispositivo cuyo modo de escalado de mapas de bits va a ser modificado.

*p2*: El identificador del nuevo modo de escalado de mapas de bits. Este parámetro puede tomar un valor de la Tabla 5-17. El controlador de vídeo puede soportar modos de escalado adicionales.

*Valor que devuelve*

Si la función tiene éxito, devuelve el modo de escalado anterior; en caso contrario, devuelve cero.

*Véase además*

*GetStretchBltMode*, *SetBrushOrgEx*, *StretchBlt*

*Ejemplo*

Vea el Listado 5-15 bajo *LoadBitmap*.

Tabla 5-17: Valores del parámetro p2 de SetStretchBltMode

Valor	Descripción
BLACKONWHITE	Ejecuta una operación booleana <b>and</b> usando los valores de color para píxeles existentes y eliminados. Si el mapa de bits es monocromático, este modo mantiene los píxeles negros a expensas de los blancos.
COLORONCOLOR	Borra píxeles sin hacer ningún intento de mantener información sobre ellos.
HALFTONE	Mapea píxeles del mapa de bits de origen en bloques de píxeles sobre el mapa de bits de destino. El color del píxel destino es el promedio de los colores de los píxeles de origen. Este modo requiere más tiempo de procesamiento que otros, pero produce imágenes de mejor calidad. Si este modo es utilizado, la aplicación tiene que llamar a la función <i>SetBrushOrgEx</i> para reinicializar la brocha original, u ocurrirá un desalineamiento de la brocha.
STRETCH_ANDSCANS	Equivalente a BLACKONWHITE.
STRETCH_DELETESCANS	Equivalente a COLORONCOLOR.
STRETCH_HALFTONE	Equivalente a HALFTONE.
STRETCH_ORSCANS	Equivalente a WHITEONBLACK.
WHITEONBLACK	Ejecuta una operación booleana <b>or</b> usando los valores de los colores para píxeles eliminados y existentes. Si el mapa de bits es monocromático, este modo mantiene los píxeles blancos a expensas de los negros.

## StretchBlt Windows.Pas

### Sintaxis

```
StretchBlt(
    DestDC: HDC;           {manejador del contexto de dispositivo de destino}
    X: Integer;             {coordenada horizontal del rectángulo de destino}
    Y: Integer;             {coordenada vertical del rectángulo de destino}
    Width: Integer;         {ancho del rectángulo de destino}
    Height: Integer;        {altura del rectángulo de destino}
    SrcDC: HDC;             {manejador del contexto de dispositivo de origen}
    XSrc: Integer;          {coordenada horizontal del rectángulo de origen}
    YSrc: Integer;          {coordenada vertical del rectángulo de origen}
    SrcWidth: Integer;      {ancho del rectángulo de origen}
    SrcHeight: Integer;     {altura del rectángulo de origen}
    ROP: DWORD              {código de la operación de barrido}
): BOOL;                  {devuelve TRUE o FALSE}
```

**Descripción**

Esta función copia un rectángulo de píxeles del mapa de bits del contexto de dispositivo de origen especificado, en el mapa de bits del contexto de dispositivo de destino. El área del mapa de bits copiado puede ser ampliada o reducida como se desee. El modo de escalamiento del contexto de dispositivo de destino, que se asigna mediante la función *SetStretchBltMode*, determina cómo el mapa de bits será ampliado o reducido. Si los formatos de colores de los contextos de dispositivos de origen y destino difieren, esta función convierte el formato de colores del contexto de dispositivo de origen al formato de colores del contexto de dispositivo de destino. Si la operación de barrido especificada indica se deben mezclar colores de los contextos de origen y de destino, la mezcla tiene lugar después de que el mapa de bits de origen es ampliado o reducido. Tenga en cuenta que si los signos del ancho o altura del origen y el destino son diferentes, *StretchBlt* creará una imagen espejo del mapa de bits resultante.

**Parámetros**

*DestDC*: Manejador del contexto de dispositivo al cual son copiados los píxeles.

*X*: La coordenada horizontal de la esquina superior izquierda del rectángulo de destino en el contexto de dispositivo de destino, en unidades lógicas.

*Y*: La coordenada vertical de la esquina superior izquierda del rectángulo de destino en el contexto de dispositivo de destino, en unidades lógicas.

*Width*: El ancho del rectángulo de destino, en unidades lógicas.

*Height*: La altura del rectángulo de destino, en unidades lógicas.

*SrcDC*: Manejador del contexto de dispositivo desde el cual los píxeles son copiados. No puede ser el manejador de un contexto de dispositivo de metaarchivo.

*XSrc*: La coordenada horizontal de la esquina superior izquierda del rectángulo de origen en el contexto de dispositivo de origen, en unidades lógicas.

*YSrc*: La coordenada vertical de la esquina superior izquierda del rectángulo de origen en el contexto de dispositivo de origen, en unidades lógicas.

*SrcWidth*: El ancho del rectángulo de origen, en unidades lógicas.

*SrcHeight*: La altura del rectángulo de origen, en unidades lógicas.

*ROP*: Un código de operación de barrido que determina cómo son combinados los colores de los píxeles en la imagen de origen, con los colores de los píxeles en el contexto de dispositivo de destino. Este parámetro puede tomar un valor de la Tabla 5-18.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*BitBlt, GetDC, CreateCompatibleDC, CreateBitmap, LoadBitmap, SetStretchBltMode*

#### Ejemplo

Vea el Listado 5-16 bajo *LoadImage* y otros ejemplos a lo largo de este libro.

**Tabla 5-18: Valores del parámetro ROP de StretchBlt**

Valor	Descripción
BLACKNESS	Rellena los píxeles del rectángulo especificado en el destino con el color especificado en el índice 0 de la paleta física. Por defecto este color es el negro.
DSTINVERT	Invierte los colores de los píxeles del rectángulo especificado en el destino.
MERGECOPY	Combina los colores de los píxeles del rectángulo de origen con los colores de los píxeles del patrón de la brocha seleccionada en el contexto de dispositivo de destino, usando el operador booleano <b>and</b> .
MERGEPAINT	Invierte los colores de los píxeles del rectángulo de origen y los combina con los colores de los píxeles del rectángulo destino, usando el operador booleano <b>or</b> .
NOTSRCCOPY	Invierte los colores de los píxeles del rectángulo de origen y los copia en el rectángulo de destino.
NOTSRCERASE	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>or</b> y luego invierte el color resultante.
PATCOPY	Copia el patrón de la brocha seleccionada en el contexto de dispositivo de destino directamente en el destino.
PATINVERT	Combina los colores de los píxeles del patrón de la brocha seleccionada en el contexto de dispositivo de destino, con los colores de los píxeles en el destino, usando el operador booleano <b>xor</b> .
PATPAINT	Combina los colores del patrón de la brocha seleccionada en el contexto de dispositivo de destino con los colores invertidos de los píxeles del rectángulo de origen, usando el operador booleano <b>or</b> , y entonces combina el resultado con los colores de los píxeles del rectángulo de destino, usando el operador booleano <b>or</b> .
SRCAND	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>and</b> .
SRCCOPY	Copia los colores de los píxeles del rectángulo de origen directamente en el rectángulo de destino.
SRCERASE	Combina los colores de los píxeles del rectángulo de origen con los colores invertidos del rectángulo de destino, usando el operador booleano <b>and</b> .

Valor	Descripción
SRCINVERT	Combina los colores de los píxeles de los rectángulos de origen y destino, usando el operador booleano <b>xor</b> .
SRCPAINT	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>or</b> .
WHITENESS	Rellena los píxeles del rectángulo especificado en el destino con el color especificado en el índice 255 de la paleta física. Por defecto este color es el blanco.

### StretchDIBits      Windows.Pas

#### Sintaxis

```

StretchDIBits(
    DC: HDC;                {manejador de contexto de dispositivo}
    DestX: Integer;          {coordenada horizontal del rectángulo de destino}
    DestY: Integer;          {coordenada vertical del rectángulo de destino}
    DestWidth: Integer;      {ancho del rectángulo de destino}
    DestHeight: Integer;     {altura del rectángulo de destino}
    SrcX: Integer;           {coordenada horizontal del rectángulo de origen}
    SrcY: Integer;           {coordenada vertical del rectángulo de origen}
    SrcWidth: Integer;       {ancho del rectángulo de origen}
    SrcHeight: Integer;      {altura del rectángulo de origen}
    Bits: Pointer;           {puntero a los valores de los bits del mapa de bits}
    var BitsInfo: TBitmapInfo; {puntero al registro del DIB}
    Usage: UINT;             {opciones de tipo de color}
    ROP: DWORD               {código de la operación de barrido}
); Integer;                 {devuelve el número de líneas de barrido copiadas}

```

#### Descripción

Esta función copia píxeles del área rectangular especificada de la imagen DIB en el área rectangular especificada del contexto de dispositivo de destino. El área del mapa de bits copiado puede ser ampliada o reducida como se desee. El modo de escalamiento del contexto de dispositivo de destino, que se asigna mediante la función *SetStretchBltMode*, determina cómo el mapa de bits será ampliado o reducido. Observe que la velocidad de copia óptima del mapa de bits se obtiene cuando los bits del mapa de bits DIB especifican índices en la paleta del sistema. Tenga en cuenta que si los signos del ancho o altura del origen y el destino son diferentes, *StretchDIBits* creará una imagen espejo del mapa de bits resultante. Esta función hará una copia fiable de la imagen del mapa de bits sobre un contexto de dispositivo de impresora.

#### Parámetros

*DC*: Manejador del contexto de dispositivo sobre el cual la imagen DIB es copiada y mostrada.

*DestX*: La coordenada horizontal de la esquina superior izquierda del rectángulo de destino en el contexto de dispositivo de destino, en unidades lógicas.

*DestY*: La coordenada vertical de la esquina superior izquierda del rectángulo de destino en el contexto de dispositivo de destino, en unidades lógicas.

*DestWidth*: El ancho del rectángulo de destino, en unidades lógicas.

*DestHeight*: La altura del rectángulo de destino, en unidades lógicas.

*SrcX*: La coordenada horizontal de la esquina superior izquierda del rectángulo de origen en el contexto de dispositivo de origen, en unidades lógicas.

*SrcY*: La coordenada vertical de la esquina superior izquierda del rectángulo de origen en el contexto de dispositivo de origen, en unidades lógicas.

*SrcWidth*: El ancho del rectángulo de origen, en unidades lógicas.

*SrcHeight*: la altura del rectángulo de origen, en unidades lógicas.

*Bits*: Puntero a la imagen que representa el DIB, en forma de un *array* de bytes.

*BitsInfo*: Puntero a un registro *TBitmapInfo* que describe la imagen DIB, incluyendo información sobre sus dimensiones y tabla de colores. El registro *TBitmapInfo* se define de la siguiente forma:

*TBitmapInfo* = **packed record**

<i>bmiHeader</i> : <i>TBitmapInfoHeader</i> ;	{información de encabezamiento}
<i>bmiColors</i> : <b>array</b> [0..0] <b>of</b> <i>TRGBQuad</i> ;	{tabla de colores usada por el mapa de bits}

**end;**

El registro *TBitmapInfoHeader* se define como:

*TBitmapInfoHeader* = **packed record**

<i>biSize</i> : <i>DWORD</i> ;	{tamaño del registro en bytes}
<i>biWidth</i> : <i>Longint</i> ;	{ancho del mapa de bits en píxeles}
<i>biHeight</i> : <i>Longint</i> ;	{altura del mapa de bits en píxeles}
<i>biPlanes</i> : <i>Word</i> ;	{número de planos de colores}
<i>biBitCount</i> : <i>Word</i> ;	{bits necesarios para describir un color}
<i>biCompression</i> : <i>DWORD</i> ;	{opciones de compresión}
<i>biSizeImage</i> : <i>DWORD</i> ;	{tamaño de la imagen en bytes}
<i>biXPelsPerMeter</i> : <i>Longint</i> ;	{píxeles horizontales por medida del dispositivo de destino}
<i>biYPelsPerMeter</i> : <i>Longint</i> ;	{píxeles verticales por medida del dispositivo de destino}
<i>biClrUsed</i> : <i>DWORD</i> ;	{número de índices de colores usados}
<i>biClrImportant</i> : <i>DWORD</i> ;	{número de índices de colores importantes}

**end;**

El registro *TRGBQuad* se define como:

```

TRGBQuad = packed record
    rgbBlue: Byte;           {intensidad del color azul}
    rgbGreen: Byte;          {intensidad del color verde}
    rgbRed: Byte;            {intensidad del color rojo}
    rgbReserved: Byte;       {valor reservado}
end;

```

Para ver una explicación de estos registros, consulte la función *CreateDIBSection*.

*Usage*: Valor que indica el tipo de información sobre el color almacenada en el campo *bmiColors* del registro *TBitmapInfo* al que apunta el parámetro *BitsInfo*. Este parámetro puede tomar un valor de la Tabla 5-19.

*ROP*: Un código de operación de barrido que determina cómo son combinados los colores de los píxeles en la imagen de origen, con los colores de los píxeles en el contexto de dispositivo de destino. Este parámetro puede tomar un valor de la Tabla 5-20.

#### Valor que devuelve

Si la función tiene éxito, devuelve el número de líneas de barrido que fueron copiadas al contexto de dispositivo destino; en caso contrario, devuelve *GDI\_ERROR*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*SetDIBits*, *SetDIBitsToDevice*, *SetStretchBltMode*

#### Ejemplo

Vea el Listado 5-9 bajo *CreateDIBSection*.

**Tabla 5-19: Valores del parámetro Usage de StretchDIBits**

Valor	Descripción
DIB_PAL_COLORS	El campo <i>bmiColors</i> del registro <i>TBitmapInfo</i> es un array de índices de 16 bits en la paleta actual del contexto de dispositivo especificado. Este valor no debe ser usado si el mapa de bits va a ser guardado en disco.
DIB_RGB_COLORS	El campo <i>bmiColors</i> del registro <i>TBitmapInfo</i> es un array de valores de colores RGB literales.

**Tabla 5-20: Valores del parámetro ROP de StretchDIBits**

Valor	Descripción
BLACKNESS	Rellena los píxeles del rectángulo especificado en el destino con el color especificado en el índice 0 de la paleta física. Por defecto este color es el negro.

Valor	Descripción
DSTINVERT	Invierte los colores de los píxeles del rectángulo especificado en el destino.
MERGECOPY	Combina los colores de los píxeles del rectángulo de origen con los colores de los píxeles del patrón de la brocha seleccionada en el contexto de dispositivo de destino, usando el operador booleano <b>and</b> .
MERGEPAINT	Invierte los colores de los píxeles del rectángulo de origen y los combina con los colores de los píxeles del rectángulo destino, usando el operador booleano <b>or</b> .
NOTSRCCOPY	Invierte los colores de los píxeles del rectángulo de origen y los copia en el rectángulo de destino.
NOTSRCERASE	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>or</b> y luego invierte el color resultante.
PATCOPY	Copia el patrón de la brocha seleccionada en el contexto de dispositivo de destino directamente en el destino.
PATINVERT	Combina los colores de los píxeles del patrón de la brocha seleccionada en el contexto de dispositivo de destino, con los colores de los píxeles en el destino, usando el operador booleano <b>xor</b> .
PATPAINT	Combina los colores del patrón de la brocha seleccionada en el contexto de dispositivo de destino con los colores invertidos de los píxeles del rectángulo de origen, usando el operador booleano <b>or</b> , y entonces combina el resultado con los colores de los píxeles del rectángulo de destino, usando el operador booleano <b>or</b> .
SRCAND	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>and</b> .
SRCCOPY	Copia los colores de los píxeles del rectángulo de origen directamente en el rectángulo destino.
SRCERASE	Combina los colores de los píxeles del rectángulo de origen con los colores invertidos del rectángulo destino, usando el operador booleano <b>and</b> .
SRCINVERT	Combina los colores de los píxeles de los rectángulos de origen y destino, usando el operador booleano <b>xor</b> .
SRCPAINT	Combina los colores de los píxeles de los rectángulos de origen y destino usando el operador booleano <b>or</b> .
WHITENESS	Rellena los píxeles del rectángulo especificado en el destino con el color especificado en el índice 255 de la paleta física. Por defecto este color es el blanco.



## Capítulo 6

# Funciones de iconos, cursores y cursores de edición

Windows, siendo el entorno gráfico que es, muestra la información de varios modos, más obviamente mediante simples gráficos. Diversas imágenes se utilizan para indicar el tipo de fichero que está siendo visualizado en el Explorador o qué tipo de acciones están disponibles al usuario, dependiendo de la posición del cursor del ratón. Las funciones de Windows que tienen que ver con la creación y manipulación de imágenes de iconos, cursores y cursores de edición ofrecen al desarrollador una amplia variedad de modos de comunicar información específica o acciones disponibles.

## Cursores de edición

El *cursor de edición* (*caret*) es la imagen pequeña e intermitente que se utiliza para indicar qué ventana tiene en un momento dado el foco del teclado y puede aceptar entrada de texto. Debido a que solamente una ventana puede tener el foco cada vez, sólo hay un cursor de edición en el sistema. En Delphi, el cursor de edición por defecto que utilizan los componentes que aceptan texto es una línea delgada y vertical semejante a una 'I' mayúscula, y Delphi encapsula tan completamente sus funciones que el desarrollador probablemente nunca tendrá que vérselas con ellas. Sin embargo, si se desea hacer uso de un cursor de edición diferente, las funciones de cursor de edición de Windows permiten al desarrollador especificar su forma, un ancho y una altura o basarlo en un mapa de bits. Consulte la descripción de la función *CreateCaret* para ver un ejemplo de creación de un cursor de edición basado en un mapa de bits. El siguiente ejemplo muestra cómo crear un cursor de edición que sea una caja negra sólida.

### Listado 6-1: Creando un cursor de edición que sea una caja negra sólida

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {Tenemos que seleccionar la ventana en la que queremos escribir. Cuando esta
  ventana recibe el foco manualmente (por ejemplo, debido al uso de la tecla
  Tab), Delphi asigna automáticamente el cursor de edición apropiado}
```

```

Memo1.SetFocus;

{Oculta el cursor de edición actual}
HideCaret(0);

{Destruye el cursor de edición actual}
DestroyCaret;

{Crea el nuevo cursor de edición (con forma de caja negra sólida)}
CreateCaret(Memo1.Handle, 0, 10, 12);

{Muestra la imagen del nuevo cursor de edición}
ShowCaret(0);
end;

```



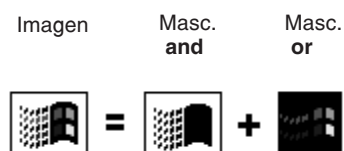
Figura 6-1:  
El cursor de  
edición en  
forma de caja  
negra

## Máscaras de iconos y cursores

Windows no tiene funciones nativas del API que copien un mapa de bits a un destino interpretando algunos de sus píxeles como transparentes. Sin embargo, los iconos y cursores tienen formas irregulares que, cuando son dibujadas en la pantalla, permiten al fondo mostrarse a través de ellos. Esto se consigue mediante operaciones booleanas de barrido y máscaras.

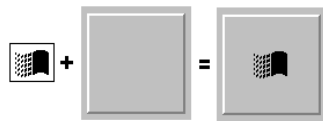
Cada icono y cursor está compuesto de dos imágenes de mapas de bits, conocidas como máscaras **and** y **or**. Estas imágenes se combinan mediante operaciones booleanas de barrido, en dos pasos, con la imagen del fondo de destino, para crear la imagen final, que muestra píxeles transparentes.

Figura 6-2: El  
icono y la  
imagen del  
cursor son la  
composición  
de máscaras  
**and** y **or**



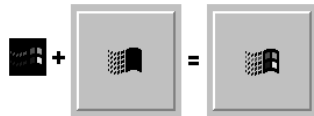
Primero, la máscara **and** es combinada con la imagen del fondo sobre el dispositivo de contexto de destino, usando el operador booleano **and**. Los píxeles blancos de la máscara **and** preservarán aquellos píxeles en el destino, mientras los píxeles negros de la máscara **and** cambiarán los píxeles en el destino a negro labrando de este modo un espacio para la imagen final, como en la siguiente imagen:

Figura 6-3:  
El primer  
paso  
combina la  
máscara **and**  
con el  
destino



Una vez que la máscara **and** es combinada con el destino, la máscara **or** es combinada con el fondo resultante de la operación anterior utilizando el operador booleano **or**. Los píxeles negros de la máscara **or** preservarán los píxeles correspondientes en el destino, mientras que los píxeles coloreados, que coincidirán con los píxeles negros creados en el primer paso, aparecerán tal cual se muestran en la máscara **or**, creándose de este modo una imagen final con la ilusión de píxeles transparentes, como la siguiente:

Figura 6-4:  
El segundo  
paso  
combina la  
máscara **or**  
con el  
destino



Es este método de mezclar mapas de bits el que permite a los iconos y cursores tener formas irregulares, mostrando el fondo a través de áreas transparentes. La misma técnica puede ser usada para mostrar cualquier mapa de bits de forma transparente, utilizando la función *BitBlt* y los códigos de operaciones de barrido *SRCAND* y *SRCPAINT*. La función *GetIconInfo* puede ser usada para recuperar las máscaras **and** y **or**, tanto de iconos como de cursores.

## Conversión de iconos en mapas de bits

La naturaleza complementaria de las funciones de Windows para manipular mapas de bits, cursores e iconos ofrecen al desarrollador los mecanismos para convertir iconos o cursores en mapas de bits y viceversa. Mediante algunas simples llamadas al API, combinadas con la potencia y facilidad de uso de Delphi, un desarrollador puede crear aplicaciones que utilicen mapas de bits, iconos y cursores de manera intercambiable. El siguiente ejemplo muestra un método por el cual un mapa de bits puede ser convertido en un icono o un icono convertido en un mapa de bits.

### Listado 6-2: Convirtiendo iconos en mapas de bits y viceversa

```
var
  Form1: TForm1;
  CurIcon: TIcon;           // almacena un icono
```

```
CurBitmap: TBitmap; // almacena un mapa de bits
```

#### implementation

```
{ $R *.DFM }
```

```
procedure TForm1.FileListBox1DbClick(Sender: TObject);
begin
    {Abrir el fichero seleccionado como un icono
     (Esto convierte automáticamente mapas de bits en iconos)}
    CurIcon.Handle := ExtractIcon(hInstance, PChar(FileListBox1.FileName), 0);

    {Habilitar el botón de guardar imagen}
    Button1.Enabled := TRUE;

    {Borrar el cuadro de dibujo}
    PaintBox1.Canvas.Brush.Color := clBtnFace;
    PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);

    {Si el usuario quiere convertir iconos en mapas de bits...}
    if RadioButton1.Checked then
        begin
            {...borrar la imagen actual del mapa de bits}
            CurBitmap.Canvas.Brush.Color := clBtnFace;
            CurBitmap.Canvas.FillRect(PaintBox1.ClientRect);

            {Dibujar el icono sobre el mapa de bits}
            DrawIcon(CurBitmap.Canvas.Handle, 0, 0, CurIcon.Handle);

            {Mostrar la imagen del mapa de bits}
            PaintBox1.Canvas.Draw((PaintBox1.Width div 2) - 16,
                                  (PaintBox1.Height div 2) - 16, CurBitmap);
        end
    else
        {Mostrar el icono}
        DrawIcon(PaintBox1.Canvas.Handle, (PaintBox1.Width div 2) - 16,
                  (PaintBox1.Height div 2) - 16, CurIcon.Handle);
    end;

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    {Si el usuario quiere convertir iconos en mapas de bits...}
    if Sender = RadioButton1 then
        begin
            {...filtrar únicamente los iconos}
            FileListBox1.Mask := '*.ico';

            {En consecuencia, inicializar el diálogo de guardar imágenes}
            SavePictureDialog1.Filter := 'Bitmaps (*.bmp)|*.bmp';
            SavePictureDialog1.DefaultExt := '*.bmp';
        end
    else
        begin
            {De lo contrario, filtrar sólo los mapas de bits}
```

```

FileListBox1.Mask := '*.bmp';

{En consecuencia, inicializar el diálogo de guardar imágenes}
SavePictureDialog1.Filter := 'Icons (*.ico)|*.ico';
SavePictureDialog1.DefaultExt := '*.ico';
end;

{Borrar la imagen actual de cuadro de dibujo}
PaintBox1.Canvas.Brush.Color := clBtnFace;
PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);

{Deshabilitar el botón de guardar imagen hasta que el usuario seleccione
un fichero a convertir}
Button1.Enabled := FALSE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Crear el icono que contendrá el mapa de bits convertido}
    CurIcon := TIcon.Create;

    {Crear e inicializar el mapa de bits que contendrá iconos convertidos}
    CurBitmap := TBitmap.Create;
    CurBitmap.Width := GetSystemMetrics(SM_CXICON);
    CurBitmap.Height := GetSystemMetrics(SM_CYICON);

    {Inicializar el diálogo de guardar imágenes para guardar mapas de bits}
    SavePictureDialog1.Filter := 'Bitmaps (*.bmp)|*.bmp';
    SavePictureDialog1.DefaultExt := '*.bmp';
end;

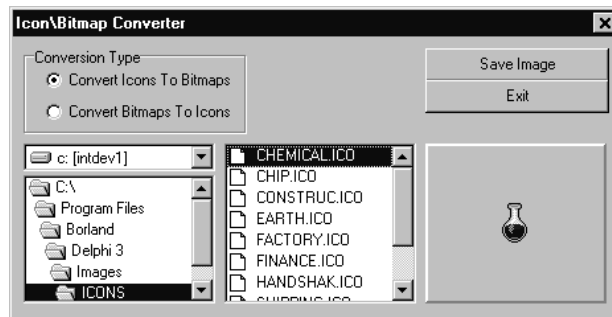
procedure TForm1.Button1Click(Sender: TObject);
begin
    {Borra el último nombre de fichero especificado}
    SavePictureDialog1.FileName := '';

    {Mostrar cuadro de diálogo}
    if SavePictureDialog1.Execute then
        if RadioButton1.Checked then
            {Como se indica, guardar el fichero como mapa de bits...}
            CurBitmap.SaveToFile(SavePictureDialog1.FileName)
        else
            {...o icono}
            CurIcon.SaveToFile(SavePictureDialog1.FileName);
    end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Liberar los recursos}
    CurIcon.Free;
    CurBitmap.Free;
end;

```

Figura 6-5:  
Convirtiendo  
un icono



## Funciones de iconos, cursores y cursores de edición

En este capítulo se describen las siguientes funciones de iconos, cursores y cursores de edición:

**Tabla 6-1: Funciones de iconos, cursores y cursores de edición**

Función	Descripción
CopyIcon	Crea una copia de un icono existente.
CreateCaret	Crea un nuevo cursor de edición.
CreateCursor	Crea un nuevo cursor.
CreateIcon	Crea un nuevo icono.
CreateIconFromResource	Crea un nuevo icono o cursor desde un recurso.
CreateIconFromResourceEx	Crea un nuevo icono o cursor desde un recurso con un ancho y altura específicos.
CreateIconIndirect	Crea un icono o cursor desde un registro de datos.
DestroyCaret	Destruye un cursor de edición.
DestroyCursor	Destruye un cursor.
DestroyIcon	Destruye un icono.
DrawIcon	Dibuja un icono en una ubicación específica.
DrawIconEx	Dibuja un icono o cursor en una ubicación específica.
ExtractAssociatedIcon	Recupera un manejador de icono del fichero ejecutable asociado con un fichero específico.
ExtractIcon	Recupera un manejador de icono desde un fichero específico.
ExtractIconEx	Recupera un manejador de los iconos pequeño y grande de un fichero específico.
GetCursor	Recupera un manejador del cursor actual.
GetIconInfo	Recupera información sobre un icono o cursor.
HideCaret	Oculto un cursor de edición.
LoadCursor	Carga un cursor desde los recursos de un ejecutable.

LoadCursorFromFile	Carga un cursor desde un fichero.
LoadIcon	Carga un icono desde los recursos de un ejecutable.
LookupIconIdFromDirectory	Busca en los datos de recursos un icono o cursor compatible.
LookupIconIdFromDirectoryEx	Busca en los datos de recursos un icono o cursor con la altura y ancho especificado.
SetCursor	Establece el cursor especificado como cursor activo.
SetSystemCursor	Asigna el cursor especificado como un cursor del sistema.
ShowCaret	Muestra el cursor de edición si estaba oculto.
ShowCursor	Muestra u oculta el cursor.

## CopyIcon Windows.Pas

### Sintaxis

```
CopyIcon(
    hIcon: HICON          {manejador del icono a copiar}
): HICON;                {devuelve un manejador de icono}
```

### Descripción

Esta función hace un duplicado del icono especificado, devolviendo su manejador. Esto puede ser usado para copiar iconos pertenecientes a otros módulos.

### Parámetros

*hIcon*: Manejador del icono a copiar.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de una copia exacta del icono especificado; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*DrawIcon*, *DrawIconEx*

### Ejemplo

#### Listado 6-3: Copiando el icono de la aplicación

```
procedure TForm1.Button1Click(Sender: TObject);
var
    IconCopy: HICON;      // almacena un manejador del icono duplicado
begin
    {hace una copia del icono de la aplicación...}
    IconCopy := CopyIcon(Application.Icon.Handle);
```

```

    {...y lo muestra}
    DrawIcon(PaintBox1.Canvas.Handle, (PaintBox1.Width div 2) - 16,
            (PaintBox1.Height div 2) - 16, IconCopy);
end;

```



Figura 6-6:  
El icono de la  
aplicación  
duplicado

## CreateCaret Windows.Pas

### Sintaxis

```

CreateCaret(
    hWnd: HWND;           {manejador de la ventana propietaria}
    hBmp: HBITMAP;        {manejador de mapa de bits}
    nWidth: Integer;       {ancho del cursor de edición}
    nHeight: Integer;      {altura del cursor de edición}
): BOOL;                 {devuelve TRUE o FALSE}

```

### Descripción

Esta función crea una nueva imagen para el cursor de edición. El cursor de edición es asignado a la ventana *hWnd* y puede ser una línea, un bloque o un mapa de bits. Si un mapa de bits es especificado, las dimensiones del mapa de bits determinan el ancho y la altura del nuevo cursor de edición. En caso contrario, el ancho y la altura estarán en términos de unidades lógicas y las dimensiones exactas dependen del modo de mapeado activo. El desarrollador puede recuperar el ancho y altura predeterminados de los cursores de edición llamando a la función *GetSystemMetrics*, utilizando las constantes *SM\_CXBORDER* y *SM\_CYBORDER*. La función *CreateCaret* destruye automáticamente la imagen anterior del cursor de edición, y el nuevo cursor de edición no será visible hasta que se haga una llamada a la función *ShowCaret*.

### Parámetros

*hWnd*: Manejador de la ventana que poseerá el cursor de edición.

*hBmp*: Manejador del mapa de bits a utilizar como cursor de edición. Si este parámetro es cero, el cursor de edición será un rectángulo sólido. Si a este parámetro (*hBmp*) se le asigna 1, el cursor de edición será gris.

*nWidth*: El ancho del cursor de edición, en unidades lógicas. Si a este parámetro se le asigna cero, el ancho de borde de ventanas predefinido de Windows será usado como ancho del cursor. Si al parámetro *hBmp* se le asigna un manejador de un mapa de bits, el mapa de bits determina el ancho del cursor y este parámetro es ignorado.

*nHeight*: La altura del cursor de edición, en unidades lógicas. Si a este parámetro se le asigna cero, la altura de borde de ventanas predefinido de Windows será usada como altura del cursor. Si al parámetro *hBmp* se le asigna un manejador de un mapa de bits, el mapa de bits determina la altura del cursor y este parámetro es ignorado.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateBitmap*, *CreateDIBitmap*, *DestroyCaret*, *GetSystemMetrics*, *HideCaret*, *LoadBitmap*, *LoadImage*, *ShowCaret*

#### Ejemplo

##### Listado 6-4: Creando un nuevo cursor de edición

```
procedure TForm1.Button1Click(Sender: TObject);
var
    TheCaretBitmap: HBitmap;    // manejador de bitmap para cursor de edición
begin
    {Carga el mapa de bits del cursor de edición desde un fichero externo}
    TheCaretBitmap := LoadImage(0, 'NewCaret.bmp', IMAGE_BITMAP, 0, 0,
                                LR_DEFAULTSIZE or LR_LOADFROMFILE);

    {Debemos enfocar la ventana en la que queremos escribir. Cuando esta ventana
    reciba el foco manualmente (por ejemplo, mediante la tecla Tab), Delphi
    automáticamente reasignará el cursor de edición adecuado}
    Memo1.SetFocus;

    {Oculta el cursor de edición actual}
    HideCaret(0);

    {Destruye el cursor de edición actual}
    DestroyCaret;

    {Crea el nuevo cursor de edición desde el mapa de bits cargado}
    CreateCaret(Memo1.Handle, TheCaretBitmap, 0, 0);

    {Muestra la nueva imagen del cursor de edición}
    ShowCaret(0);
end;
```

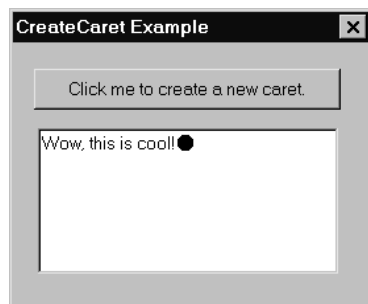


Figura 6-7:  
El nuevo  
cursor de  
edición

## CreateCursor Windows.Pas

### Sintaxis

```
CreateCursor(
  hInst: HINST;           {manejador de la instancia actual de la aplicación}
  xHotSpot: Integer;       {posición horizontal del 'punto caliente' del cursor}
  yHotSpot: Integer;       {posición vertical del 'punto caliente' del cursor}
  nWidth: Integer;         {ancho del cursor en píxeles}
  nHeight: Integer;        {altura del cursor in píxeles}
  pvANDPlane: Pointer;     {puntero a los datos de la máscara and del cursor}
  pvXORPlane: Pointer      {puntero a los datos de la máscara xor del cursor}
): HCURSOR;               {devuelve un manejador del cursor}
```

### Descripción

Esta función crea un nuevo cursor con las dimensiones, imagen y 'punto caliente' especificados. Este cursor puede ser añadido al *array* de cursores de pantalla de Delphi para hacerlo persistente.

### Parámetros

*hInst*: Manejador de la instancia actual de la aplicación.

*xHotSpot*: La coordenada horizontal del 'punto caliente' del cursor.

*yHotSpot*: La coordenada vertical del 'punto caliente' del cursor.

*nWidth*: El ancho del cursor en píxeles. Utilice *GetSystemMetrics(SM\_CXCURSOR)* para determinar el ancho de cursor soportado por el controlador de vídeo.

*nHeight*: La altura del cursor en píxeles. Utilice *GetSystemMetrics(SM\_CYCURSOR)* para determinar la altura de cursor soportada por el controlador de vídeo.

*pvANDPlane*: Puntero a un *array* de bytes que contienen los valores de los bits para la máscara **and** del cursor. Este *array* contiene información con el formato de un mapa de bits monocromático dependiente del dispositivo.

*pvXORPlane*: Puntero a un *array* de bytes que contienen los valores de los bits para la máscara **xor** del cursor. Este *array* contiene información con el formato de un mapa de bits monocromático dependiente del dispositivo.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del nuevo cursor; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateIcon*, *DestroyCursor*, *GetCursor*, *GetSystemMetrics*, *SetCursor*

#### Ejemplo

##### Listado 6-5: Creando un nuevo cursor

```
var
  Form1: TForm1;
  OldCursor: HCURSOR; // mantiene el cursor anterior
  NewCursor: HCURSOR; // manejador del nuevo cursor

implementation

procedure TForm1.Button1Click(Sender: TObject);
var
  MaskSize: Integer; // almacena el tamaño calculado del cursor
  AndMask, // arrays de bits del cursor
  XorMask: ^Byte;
  AndImage, // mapas de bits usados para definir la forma del cursor
  XorImage: TBitmap;
begin
  {Calcula el tamaño de los arrays de bits del cursor}
  MaskSize := (GetSystemMetrics(SM_CXICON) div 8) * GetSystemMetrics(SM_CYICON);

  {Crea el mapa de bits usado para definir la forma de la máscara and}
  AndImage := TBitmap.Create;
  with AndImage do
  begin
    {Estamos creando un cursor blanco y negro}
    Monochrome := TRUE;

    {Asigna las dimensiones del cursor}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {Crea la forma de una 'X'}
    Canvas.Brush.Color := clWhite;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Canvas.ClipRect);
    Canvas.MoveTo(0,0);
    Canvas.LineTo(Width,Height);
    Canvas.MoveTo(Width,0);
```

```

    Canvas.LineTo(0,Height);
end;

{Crea el mapa de bits usado para definir la forma de la máscara xor}
XorImage := TBitmap.Create;
with XorImage do
begin
    {Estamos creando un cursor blanco y negro}
    Monochrome := TRUE;

    {Asigna las dimensiones del cursor}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {Rellena el mapa de bits de negro}
    Canvas.Brush.Color := clBlack;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Canvas.ClipRect);
end;

{Reserva la memoria para los arrays}
GetMem(AndMask, MaskSize);
GetMem(XorMask, MaskSize);

{Transfiere las imágenes a los arrays}
GetBitmapBits(AndImage.Handle, MaskSize, AndMask);
GetBitmapBits(XorImage.Handle, MaskSize, XorMask);

{Crea un nuevo cursor basado en las imágenes transferidas a los arrays}
NewCursor := CreateCursor(hInstance, 0, 0, GetSystemMetrics(SM_CXICON),
    GetSystemMetrics(SM_CYICON), AndMask, XorMask);

{Si el cursor de la clase de ventana no es cero, SetCursor tendrá éxito, pero el
cursor será reinicializado al cursor de la clase tan pronto como sea movido el
ratón. Por eso tenemos que asignar cero al cursor de la clase del botón y al
formulario}
SetClassLong(Form1.Handle, GCL_HCURSOR, 0);
SetClassLong(Button1.Handle, GCL_HCURSOR, 0);

{Asigna la nueva forma del cursor}
SetCursor(NewCursor);

{Los mapas de bits temporales ya no son necesarios}
AndImage.Free;
XorImage.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Recupera y guarda un manejador del cursor original}
    OldCursor := GetCursor;
end;

```

```

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Restaura el cursor original}
    SetCursor(OldCursor);

    {Elimina el nuevo cursor}
    DestroyCursor(NewCursor);
end;

```

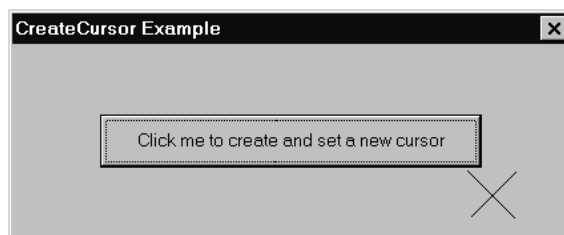


Figura 6-8:  
El nuevo  
cursor

## CreateIcon Windows.Pas

### Sintaxis

```

CreateIcon(
    hInstance: HINST;           {manejador de la instancia de la aplicación}
    nWidth: Integer;            {ancho del icono}
    nHeight: Integer;           {altura del icono}
    cPlanes: Byte;              {cantidad de planos de colores}
    cBitsPixel: Byte;           {el número de bits que describen una máscara de píxel
                                xor}
    lpbANDbits: Pointer;        {puntero a los datos de la máscara and}
    lpbXORbits: Pointer;        {puntero a los datos de la máscara xor}
): HICON;                      {devuelve un manejador de un icono}

```

### Descripción

Crea dinámicamente un nuevo icono con las dimensiones e imagen especificadas.

### Parámetros

*hInstance*: Manejador de la instancia de la aplicación que crea el icono.

*nWidth*: El ancho del icono en píxeles. A este parámetro hay que asignarle el valor devuelto por *GetSystemMetrics(SM\_CXICON)*.

*nHeight*: La altura del icono en píxeles. A este parámetro hay que asignarle el valor devuelto por *GetSystemMetrics(SM\_CYICON)*.

*cPlanes*: El número de planos de colores usados en la máscara **xor** del icono.

*cBitsPixel*: El número de bits necesarios para describir el color de un píxel (por ejemplo, 8 bits para imágenes de 256 colores, 24 bits para imágenes de 16,7 millones de colores, etc.).

*lpbANDbits*: Puntero a un *array* de bytes que contiene la imagen de la máscara **and** del icono. La información de la imagen contenida en este *array* tiene que describir un mapa de bits monocromático.

*lpbXORbits*: Puntero a un *array* de bytes que contiene la imagen de la máscara **xor** del icono. La información de imagen contenida en este *array* puede describir un mapa de bits monocromático o un mapa de bits a colores dependiente del dispositivo.

#### Valor que devuelve

Si la función tiene éxito, devuelve el manejador de un nuevo icono; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateIconFromResource*, *CreateIconFromResourceEx*, *CreateIconIndirect*, *DrawIcon*, *DrawIconEx*, *GetSystemMetrics*, *LoadIcon*

#### Ejemplo

##### Listado 6-6: Creando un icono en tiempo de ejecución

```
procedure TForm1.Button1Click(Sender: TObject);
var
    AndMaskSize,           // almacena el tamaño calculado del icono
    XorMaskSize: Integer;

    AndMask,               // arrays de bits del icono
    XorMask: ^Byte;

    AndImage,              // mapas de bits usados para definir la forma del icono
    XorImage: TBitmap;
begin
    {Calcula el tamaño de los arrays de bits del icono}
    XorMaskSize := GetSystemMetrics(SM_CXICON) * GetSystemMetrics(SM_CYICON);

    {La máscara AND es un mapa de bits monocromático. De esta manera, cada bit
    representa un píxel; se divide el ancho entre 8 para obtener el número correcto
    de bytes}
    AndMaskSize := (GetSystemMetrics(SM_CXICON) div 8) * GetSystemMetrics(SM_CYICON);

    {Crea el mapa de bits usado para definir la forma de la máscara XOR}
    XorImage := TBitmap.Create;
    with XorImage do
    begin
        {Asigna las dimensiones a partir de las indicadas por el sistema}
        Width := GetSystemMetrics(SM_CXICON);
        Height := GetSystemMetrics(SM_CYICON);

        {Rellena el fondo con negro}
        Canvas.Brush.Color := clBlack;
        Canvas.FillRect(Canvas.ClipRect);
```

```

    {Dibuja una caja roja}
    Canvas.Brush.Color := clRed;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Rect(5, 5, GetSystemMetrics(SM_CXICON) - 5,
        GetSystemMetrics(SM_CYICON) - 5));
end;

{Crea el mapa de bits usado para definir la forma de la máscara AND}
AndImage := TBitmap.Create;
with AndImage do
begin
    {La máscara AND siempre es en blanco y negro}
    Monochrome := TRUE;

    {Asigna las dimensiones a partir de las indicadas por el sistema}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {Rellena el fondo con blanco}
    Canvas.Brush.Color := clWhite;
    Canvas.FillRect(Canvas.ClipRect);

    {Dibuja una caja negra del mismo tamaño que la caja roja en la máscara XOR}
    Canvas.Brush.Color := clBlack;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Rect(5, 5, GetSystemMetrics(SM_CXICON) - 5,
        GetSystemMetrics(SM_CYICON)-5));
end;

{Reserva la memoria para los arrays de bits}
GetMem(AndMask, AndMaskSize);
GetMem(XorMask, XorMaskSize);

{Transfiere las imágenes en los mapas de bits a los arrays}
GetBitmapBits(AndImage.Handle, AndMaskSize, AndMask);
GetBitmapBits(XorImage.Handle, XorMaskSize, XorMask);

{Crea el nuevo icono}
NewIcon := CreateIcon(hInstance, GetSystemMetrics(SM_CXICON),
    GetSystemMetrics(SM_CYICON), 1, 8, AndMask, XorMask);

{Asigna al icono de la aplicación el nuevo icono}
Application.Icon.Handle := NewIcon;

{Muestra el icono sobre el formulario}
DrawIcon(PaintBox1.Canvas.Handle,
    PaintBox1.Width div 2-(GetSystemMetrics(SM_CXICON) div 2),
    PaintBox1.Height div 2-(GetSystemMetrics(SM_CYICON) div 2), NewIcon);

{Elimina los mapas de bits temporales}
AndImage.Free;
XorImage.Free;
end;

```



Figura 6-9:  
El nuevo icono

### ***CreatelconFromResource***

### ***Windows.Pas***

#### ***Sintaxis***

```

CreatelconFromResource(
  pResBits: PByte;           {puntero a un icono o a los bits que componen la imagen}
  dwResSize: DWORD;          {cantidad de bytes a los que apunta pResBits}
  fIcon: BOOL;               {indica si se trata de un icono o de un cursor}
  dwVer: DWORD               {número de versión del formato}
): HICON;                   {devuelve un manejador de icono o cursor}

```

#### ***Descripción***

Esta función crea un nuevo icono o un nuevo cursor a partir de los bits de un recurso que definen la imagen del icono o cursor.

#### ***Parámetros***

*pResBits*: Puntero a un *buffer* que contiene los bits de recurso del icono o cursor. El valor devuelto por las funciones *LoadResource* o *LookupIconIdFromDirectory* puede ser usado como valor de este parámetro.

*dwResSize*: El tamaño del *buffer* al que apunta el parámetro *pResBits*, en bytes.

*fIcon*: Opción que indica si se desea crear un icono o un cursor. El valor TRUE hace que la función cree un icono; FALSE hace que la función cree un cursor.

*dwVer*: Especifica el número de la versión del formato de icono o cursor. Las aplicaciones Win32 deben asignar a este parámetro el valor \$30000.

#### ***Valor que devuelve***

Si la función tiene éxito, devuelve un manejador de icono o cursor; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*CreateIcon, CreateIconFromResourceEx, CreateIconIndirect, LoadResource, LookupIconIdFromDirectory, LookupIconIdFromDirectoryEx*

Ejemplo

#### Listado 6-7: Creando un icono desde la información de recurso

```

procedure TForm1.Button1Click(Sender: TObject);
var
    IconBits: HGLOBAL;           // manejador de la imagen del icono
    IconBitsPtr: Pointer;        // puntero a la imagen de un icono
    ResHandle: HRSRC;            // manejador de la información de recurso del icono
    ResId: Integer;              // almacena el id del recurso para el icono
    TheIcon: HICON;              // manejador del icono creado
begin
    {Recupera un manejador del recurso del icono}
    ResHandle := FindResource(0, 'TARGET', RT_GROUP_ICON);

    {Recupera un manejador de la imagen del recurso del icono}
    IconBits := LoadResource(0, ResHandle);

    {Recupera un puntero a la imagen del icono}
    IconBitsPtr := LockResource(IconBits);

    {Busca el icono adecuado para el dispositivo de visualización actual}
    ResId := LookupIconIdFromDirectory(IconBitsPtr, TRUE);

    {Recupera un manejador de este icono}
    ResHandle := FindResource(0, MakeIntResource(ResId), RT_ICON);

    {Carga la imagen del recurso del icono}
    IconBits := LoadResource(0, ResHandle);

    {Recupera un puntero a la imagen del icono}
    IconBitsPtr := LockResource(IconBits);

    {Crea un nuevo icono desde la información correcta de recurso del icono}
    TheIcon := CreateIconFromResource(IconBitsPtr, SizeOfResource(0, ResHandle),
                                     TRUE, $30000);

    {Muestra el icono}
    DrawIcon(PaintBox1.Canvas.Handle,
             PaintBox1.Width div 2-(GetSystemMetrics(SM_CXICON) div 2),
             PaintBox1.Height div 2-(GetSystemMetrics(SM_CYICON) div 2), TheIcon);
end;

```

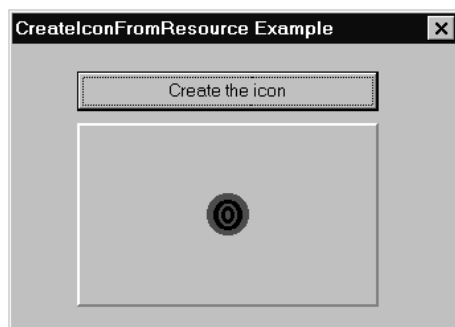


Figura 6-10:  
El nuevo icono

### CreateIconFromResourceEx

### Windows.Pas

#### Sintaxis

```
CreateIconFromResourceEx(
  pResBits: PByte;           {puntero a los bits de un icono o cursor}
  dwResSize: DWORD;          {cantidad de bytes a los que apunta pResBits}
  fIcon: BOOL;               {indica si se trata de un icono o de un cursor}
  dwVer: DWORD;              {número de versión del formato}
  cxDesired: Integer;        {ancho preferido del icono o cursor}
  cyDesired: Integer;        {altura preferida del icono o cursor}
  Flags: UINT;               {opción de color}
): HICON;                   {devuelve un manejador de icono o cursor}
```

#### Descripción

Esta función crea un nuevo icono o cursor a partir de los bits del recurso especificado que definen la imagen del icono o cursor. A diferencia de la función *CreateIconFromResource*, esta función permite al desarrollador determinar las dimensiones y el formato de colores del icono o cursor.

#### Parámetros

*pResBits*: Puntero a un *buffer* que contiene los bits del recurso del icono o cursor. El valor devuelto por las funciones *LoadResource* o *LookupIconIdFromDirectory* puede ser utilizado como valor de este parámetro.

*dwResSize*: El tamaño del *buffer* al que apunta el parámetro *pResBits*, en bytes.

*fIcon*: Opción que indica si se desea crear un icono o un cursor. El valor TRUE hace que la función cree un icono; FALSE hace que la función cree un cursor.

*dwVer*: Especifica el número de la versión del formato de icono o cursor. Las aplicaciones Win32 deben asignar a este parámetro el valor \$30000.

*cxDesired*: Especifica el ancho preferido del icono o cursor, en píxeles. Si este parámetro es cero, la función utiliza el valor devuelto por *GetSystemMetrics(SM\_CXICON)*.

*cyDesired*: Especifica la altura preferida del icono o cursor, en píxeles. Si este parámetro es cero, la función utiliza el valor devuelto por *GetSystemMetrics(SM\_CYICON)*.

*Flags*: Un valor que indica el formato de color para el icono o cursor. Este parámetro puede ser un valor de la Tabla 6-2.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de un icono o cursor; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateIcon*, *CreateIconFromResource*, *CreateIconIndirect*, *LoadResource*, *LookupIconIdFromDirectory*, *LookupIconIdFromDirectoryEx*

#### Ejemplo

##### Listado 6-8: Más opciones para la creación de un icono desde la información de recurso

```
procedure TForm1.Button1Click(Sender: TObject);
var
  IconBits: HGLOBAL;           // manejador de la imagen de un icono
  IconBitsPtr: Pointer;        // puntero a la imagen del icono
  ResHandle: HRSRC;            // manejador de la información de recurso del icono
  ResId: Integer;              // almacena el id del recurso del icono
  TheIcon: HICON;              // manejador del icono creado
begin
  {Recupera un manejador del recurso del icono}
  ResHandle := FindResource(0, 'TARGET', RT_GROUP_ICON);

  {Recupera un manejador de la imagen del recurso del icono}
  IconBits := LoadResource(0, ResHandle);

  {Recupera un puntero a la imagen del icono}
  IconBitsPtr := LockResource(IconBits);

  {Busca el icono más adecuado para el dispositivo de visualización actual}
  ResId := LookupIconIdFromDirectoryEx(IconBitsPtr, TRUE,
                                       0, 0, LR_DEFAULTCOLOR);

  {Recupera un manejador para este icono}
  ResHandle := FindResource(0, MakeIntResource(ResId), RT_ICON);

  {Carga la imagen del recurso del icono}
  IconBits := LoadResource(0, ResHandle);

  {Recupera un puntero a la imagen del icono}
  IconBitsPtr := LockResource(IconBits);

  {Crea un nuevo icono desde la información del recurso del icono}
  TheIcon := CreateIconFromResourceEx(IconBitsPtr, SizeOfResource(0, ResHandle),
```

```

TRUE, $30000, 0, 0, LR_DEFAULTCOLOR);
    {Muestra el icono}
    DrawIcon(PaintBox1.Canvas.Handle,
        PaintBox1.Width div 2 - (GetSystemMetrics(SM_CXICON) div 2),
        PaintBox1.Height div 2 - (GetSystemMetrics(SM_CYICON) div 2), TheIcon);
end;
```

**Tabla 6-2: Opciones de color de CreateIconFromResourceEx**

Valor	Descripción
LR_DEFAULTCOLOR	Crea un cursor o icono en colores, usando los colores predefinidos del sistema.
LR_MONOCHROME	Crea un cursor o icono monocromático.

## **CreateIconIndirect** **Windows.Pas**

### Sintaxis

```

CreateIconIndirect(
    var pIconInfo: TIconInfo {puntero a registro de información del icono}
): HICON;                  {devuelve un manejador de icono}
```

### Descripción

Esta función crea dinámicamente un nuevo icono con las dimensiones e imágenes definidas en la variable *pIconInfo*. Después de crear el icono, la aplicación tiene que gestionar los mapas de bits usados en la definición del icono y eliminarlos cuando dejen de ser necesarios. Los iconos creados mediante esta función tienen que ser eliminados utilizando la función *DestroyIcon*.

### Parámetros

*pIconInfo*: Puntero a un registro *TIconInfo* que describe la imagen del icono. El registro *TIconInfo* se define como:

```

TIconInfo = packed record
    flcon: BOOL;                {indica información de icono o cursor}
    xHotspot: DWORD;            {coordenada horizontal del ‘punto caliente’}
    yHotspot: DWORD;            {coordenada vertical del ‘punto caliente’}
    hbmMask: HBITMAP;           {manejador de mapa de bits}
    hbmColor: HBITMAP;          {manejador de mapa de bits}
end;
```

Consulte la función *GetIconInfo* para ver una descripción de los campos del registro.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del nuevo icono; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*CreateIcon, CreateIconFromResource, CreateIconFromResourceEx, DrawIcon, DrawIconEx, DestroyIcon, LoadIcon*

Ejemplo

#### Listado 6-9: Creando un icono indirectamente

```
var
  Form1: TForm1;
  NewIcon: HICON; // almacena el nuevo icono

implementation

procedure TForm1.Button1Click(Sender: TObject);
var
  AndImage,           // mapas de bits usados para definir la forma del icono
  XorImage: TBitmap;
  IconInfo: TIconInfo; // el registro de información del icono
begin
  {Crea el mapa de bits usado para definir la forma de la máscara XOR}
  XorImage := TBitmap.Create;
  with XorImage do
  begin
    {Asigna las dimensiones predefinidas del sistema}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {Rellena el fondo con negro}
    Canvas.Brush.Color := clBlack;
    Canvas.FillRect(Canvas.ClipRect);

    {Dibuja una caja roja}
    Canvas.Brush.Color := clRed;
    Canvas.Pen.Color := clBlack;
    Canvas.FillRect(Rect(5, 5, GetSystemMetrics(SM_CXICON) - 5,
      GetSystemMetrics(SM_CYICON) - 5));
  end;

  {Crea el mapa de bits usado para definir la forma de la máscara AND}
  AndImage := TBitmap.Create;
  with AndImage do
  begin
    {La máscara AND es siempre en blanco y negro}
    Monochrome := TRUE;

    {Asigna las dimensiones}
    Width := GetSystemMetrics(SM_CXICON);
    Height := GetSystemMetrics(SM_CYICON);

    {Rellena el fondo con blanco}
    Canvas.Brush.Color := clWhite;
    Canvas.FillRect(Canvas.ClipRect);
```

```

        {Dibuja una caja negra del mismo tamaño que la roja en la máscara XOR}
        Canvas.Brush.Color := clBlack;
        Canvas.Pen.Color := clBlack;
        Canvas.FillRect(Rect(5, 5, GetSystemMetrics(SM_CXICON) - 5,
            GetSystemMetrics(SM_CYICON) - 5));
    end;

    {Inicializa el registro para definir el nuevo icono}
    IconInfo.fIcon := TRUE;
    IconInfo.xHotspot := 0;
    IconInfo.yHotspot := 0;
    IconInfo.hbmMask := AndImage.Handle;
    IconInfo.hbmColor := XorImage.Handle;

    {Crea un nuevo icono basado en la información del registro del icono}
    NewIcon := CreateIconIndirect(IconInfo);

    {Hace que el icono de la aplicación apunte a este nuevo icono}
    Application.Icon.Handle := NewIcon;

    {Muestra el icono sobre el formulario}
    DrawIcon(PaintBox1.Canvas.Handle,
        PaintBox1.Width div 2 - (GetSystemMetrics(SM_CXICON) div 2),
        PaintBox1.Height div 2 - (GetSystemMetrics(SM_CYICON) div 2), NewIcon);

    {Se eliminan los mapas de bits temporales}
    AndImage.Free;
    XorImage.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Elimina el nuevo icono}
    DestroyIcon(NewIcon);
end;

```

## **DestroyCaret**      **Windows.Pas**

### **Sintaxis**

DestroyCaret: BOOL;      {devuelve TRUE o FALSE}

### **Descripción**

Esta función elimina la figura actual del cursor de edición, lo libera de la ventana y lo quita de la pantalla. Si un mapa de bits fue usado para definir la forma del cursor de edición, el mapa de bits no es liberado. *DestroyCaret* fallará si la ventana que posee el cursor de edición no pertenece a la tarea actual.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateCaret*, *HideCaret*, *ShowCaret*

**Ejemplo**

Vea el Listado 6-4 bajo *CreateCaret*.

**DestroyCursor      Windows.Pas****Sintaxis**

```
DestroyCursor(
    hCursor: HICON           {manejador del cursor que se desea destruir}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función destruye el cursor identificado por el manejador del cursor especificado y libera su memoria. Esta función sólo debe ser utilizada para destruir cursores creados con la función *CreateCursor*.

**Parámetros**

*hCursor*: Manejador del cursor que será destruido. Este manejador de cursor no deberá estar en uso cuando la función sea llamada.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateCursor*

**Ejemplo**

Vea el Listado 6-5 bajo *CreateCursor*.

**DestroyIcon**      **Windows.Pas****Sintaxis**

```

DestroyIcon(
    hIcon: HICON           {manejador del icono que se desea destruir}
): BOOL;                  {devuelve TRUE o FALSE}

```

**Descripción**

Esta función destruye el icono identificado por el manejador del icono especificado y libera su memoria. Esta función debe ser utilizada solamente para destruir iconos creados con la función *CreateIconIndirect*.

**Parámetros**

*hIcon*: Un manejador del icono que será destruido. Este manejador del icono no deberá estar en uso cuando la función sea llamada.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateIconIndirect*

**Ejemplo**

Vea el Listado 6-9 bajo *CreateIconIndirect*.

**DrawIcon**      **Windows.Pas****Sintaxis**

```

DrawIcon(
    hDC: HDC;              {manejador de contexto de dispositivo}
    X: Integer;            {coordenada horizontal del icono o cursor}
    Y: Integer;            {coordenada vertical del icono o cursor}
    hIcon: HICON           {manejador del icono o cursor a dibujar}
): BOOL;                  {devuelve TRUE o FALSE}

```

**Descripción**

Esta función dibuja un icono o cursor (incluyendo cursores animados), sobre el contexto de dispositivo especificado.

**Parámetros**

*hDC*: Manejador del contexto de dispositivo sobre el cual el icono o cursor será dibujado.

*X*: Indica la posición horizontal de la esquina superior izquierda del icono o cursor dentro del contexto de dispositivo especificado, de acuerdo al modo de mapeado actual.

*Y*: Indica la posición vertical de la esquina superior izquierda del icono o cursor dentro del contexto de dispositivo especificado, de acuerdo al modo de mapeado actual.

*hIcon*: Manejador del icono o cursor que será dibujado.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*DrawIconEx*, *LoadCursor*, *LoadIcon*, *LoadImage*

**Ejemplo**

Vea el Listado 6-6 bajo *CreateIcon*.

**DrawIconEx Windows.Pas****Sintaxis**

```
DrawIconEx(
    hdc: HDC;                {manejador de contexto de dispositivo}
    xLeft: Integer;          {coordenada horizontal para mostrar el icono}
    yTop: Integer;           {coordenada vertical para mostrar el icono}
    hIcon: HICON;            {manejador del icono a mostrar}
    cxWidth: Integer;        {ancho del icono}
    cyWidth: Integer;        {altura del icono}
    iStepIfAniCur: UINT;    {índice de cuadro para un cursor animado}
    hbrFlickerFreeDraw: HBRUSH; {manejador de brocha}
    diFlags: UINT            {opción de modo de presentación del icono}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función dibuja un icono o cursor (incluyendo cursores animados), sobre el contexto de dispositivo especificado. El icono o cursor puede ser ampliado o reducido como se desee.

**Parámetros**

*hdc*: Manejador del contexto de dispositivo sobre el cual será dibujado el icono o cursor.

*xLeft*: Indica la posición horizontal de la esquina superior izquierda del icono o cursor dentro del contexto de dispositivo especificado, de acuerdo al modo de mapeado actual.

*yTop*: Indica la posición vertical de la esquina superior izquierda del icono o cursor dentro del contexto de dispositivo especificado, de acuerdo al modo de mapeado actual.

*hIcon*: Manejador del icono o cursor que será dibujado.

*cxWidth*: Especifica el ancho con que debe ser dibujado el icono o cursor, en unidades lógicas. Si este parámetro es cero y al parámetro *diFlags* se le asigna *DI\_DEFAULTSIZE*, la función utiliza los valores predefinidos del sistema *SM\_CXICON* o *SM\_CXCURSOR* para el ancho. Si este parámetro es cero y al parámetro *diFlags* no se le asigna *DI\_DEFAULTSIZE*, la función utiliza el ancho actual del icono o cursor.

*cyWidth*: Especifica la altura con que debe ser dibujado el icono o cursor, en unidades lógicas. Si este parámetro es cero y al parámetro *diFlags* se le asigna *DI\_DEFAULTSIZE*, la función utiliza los valores predefinidos del sistema *SM\_CYICON* o *SM\_CYCURSOR* para la altura. Si este parámetro es cero y al parámetro *diFlags* no se le asigna *DI\_DEFAULTSIZE*, la función utiliza la altura actual del icono o cursor.

*iStepIfAniCur*: Especifica qué cuadro del cursor animado se desea dibujar. Si el parámetro *hIcon* no corresponde a un manejador de un icono animado, este parámetro es ignorado.

*hbrFlickerFreeDraw*: Manejador de una brocha. Si el manejador de la brocha es válido, la función crea un mapa de bits fuera de pantalla usando la brocha para el color del fondo, dibuja el icono o cursor en este mapa de bits fuera de pantalla y entonces lo copia sobre el contexto de dispositivo especificado por el parámetro *hdc*. Esto elimina cualquier parpadeo cuando el icono o cursor sea mostrado. Si este parámetro es cero, el icono o cursor es dibujado directamente en el contexto de dispositivo especificado.

*diFlags*: Indicador que controla el comportamiento de dibujo. Este parámetro puede tomar uno o más valores de la Tabla 6-3.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*DrawIcon*, *LoadCursor*, *LoadIcon*, *LoadImage*

**Ejemplo**

Vea el Listado 6-14 bajo *LoadCursorFromFile*.

**Tabla 6-3: Valores del parámetro diFlags de DrawIconEx**

Valor	Descripción
DI_COMPAT	Dibuja la imagen del cursor por defecto para el cursor estándar especificado, incluso si el cursor ha sido reemplazado mediante una llamada a la función <i>SetSystemCursor</i> .
DI_DEFAULTSIZE	Dibuja el icono o cursor con las medidas predefinidas por el sistema para el ancho y altura de iconos y cursores.
DI_IMAGE	Dibuja sólo la máscara <b>or</b> del icono o cursor.
DI_MASK	Dibuja sólo máscara <b>and</b> del icono o cursor.
DI_NORMAL	Combina los valores DI_IMAGE y DI_MASK para dibujar el cursor o icono en la forma en que son mostrados normalmente.

**ExtractAssociatedIcon****ShellAPI.Pas****Sintaxis**

```
ExtractAssociatedIcon(
    hInst: HINST;           {manejador de instancia de la aplicación}
    lpIconPath: PChar;      {puntero a cadena que contiene el nombre del fichero}
    var lpIcon: Word;       {índice del icono}
): HICON;                 {devuelve un manejador de icono}
```

**Descripción**

Esta función devuelve el manejador de un icono extraído del fichero especificado por el parámetro *lpIconPath*. Si este parámetro no apunta a un fichero ejecutable, el icono es extraído del fichero ejecutable asociado con el fichero especificado. Si este fichero no tiene una aplicación asociada, la función devuelve el manejador de un icono predeterminado asignado por Windows. Adicionalmente, si el nombre de fichero especifica un fichero de mapa de bits o cursor, esta función creará un icono utilizando la imagen del fichero y devolverá su manejador.

**Parámetros**

*hInst*: Manejador de instancia de la aplicación.

*lpIconPath*: Puntero a una cadena terminada en nulo que contiene el nombre de fichero del cual extraer el icono.

*lpIcon*: Puntero a una variable que contiene el índice del icono a extraer. El índice está basado en cero, de manera que un valor cero recuperará el primer icono del fichero.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de icono; en caso contrario, devuelve cero.

**Véase además**

*DrawIcon, DrawIconEx, ExtractIcon, LoadIcon*

**Ejemplo****Listado 6-10: Extrayendo iconos asociados con un fichero**

```

procedure TForm1.FileListBox1Click(Sender: TObject);
var
    IconIndex: Word;
    TheIcon: TIcon;
begin
    {Extrae el primer icono encontrado}
    IconIndex := 0;

    {Crea el objeto de icono temporal}
    TheIcon := TIcon.Create;

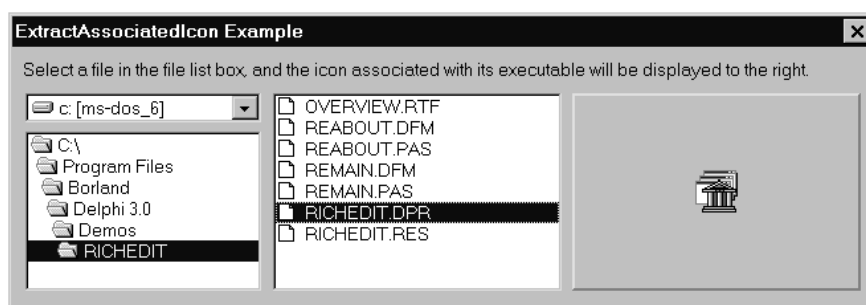
    {Extrae el icono del ejecutable asociado con el fichero seleccionado}
    TheIcon.Handle := ExtractAssociatedIcon(hInstance, PChar(FileListBox1.FileName),
                                           IconIndex);

    {Copia el icono en el objeto imagen}
    Image1.Picture.Assign(TheIcon);

    {Libera el objeto de icono temporal}
    TheIcon.Free;
end;

```

**Figura 6-11:**  
Un icono  
asociado con  
un fichero

**ExtractIcon****ShellAPI.Pas****Sintaxis**

ExtractIcon(

<code>hInst: HINST;</code>	<code>{manejador de instancia de la aplicación}</code>
<code>lpzExeFileName: PChar;</code>	<code>{puntero a cadena que contiene nombre de fichero}</code>
<code>nIconIndex: UINT</code>	<code>{índice de icono}</code>
<code>): HICON;</code>	<code>{devuelve un manejador de icono}</code>

### Descripción

Esta función devuelve un manejador de un icono extraído de un fichero ejecutable, DLL, o fichero de icono. Adicionalmente, si el nombre de fichero especifica un fichero de mapa de bits o cursor, la función creará un icono utilizando la imagen y devolverá su manejador. Esta función puede ser usada para convertir mapas de bits o cursores en iconos.

### Parámetros

*hInst*: Manejador de instancia de la aplicación.

*lpzExeFileName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre de fichero desde el cual extraer el icono.

*nIconIndex*: El índice del icono a recuperar. El índice es de base cero, de manera que un valor 0 recuperará el primer icono del fichero, si existe alguno. Si este valor es -1, el valor devuelto será la cantidad total de iconos almacenados en el fichero.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de icono. Si la función falla o no hay iconos almacenados en el fichero, devuelve cero.

### Véase además

*DrawIcon*, *DrawIconEx*, *ExtractAssociatedIcon*, *LoadIcon*

### Ejemplo

#### Listado 6-II: Extrayendo el icono de un fichero

```
procedure TForm1.FileListBox1DBClick(Sender: TObject);
var
    TheIcon: TIcon;           // almacenará el icono devuelto
    NumIcons: Integer;        // almacena la cantidad de iconos
begin
    {Determina el número de iconos almacenados en el fichero}
    NumIcons:=ExtractIcon(hInstance, PChar(FileListBox1.FileName), -1);
    {Muestra la cantidad}
    Label2.Caption := IntToStr(NumIcons) + ' icon(s) in this file';

    {Crea un objeto icono}
    TheIcon := TIcon.Create;

    {Busca el icono en la aplicación seleccionada, si existe alguno}
    TheIcon.Handle := ExtractIcon(hInstance, PChar(FileListBox1.FileName), 0);

    {Muestra el icono. Si no existe icono, el mostrado actualmente será borrado}
```

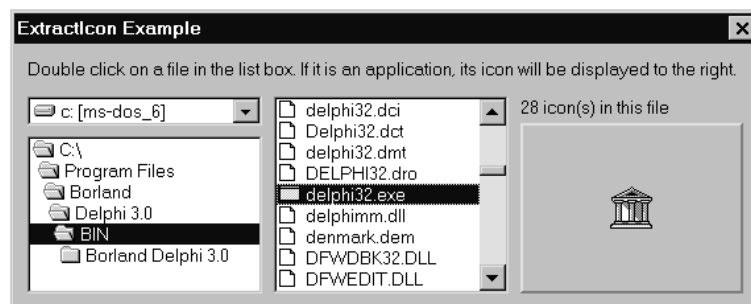
```

Image1.Picture.Assign(TheIcon);

{Libera el objeto icono}
TheIcon.Free;
end;

```

Figura 6-12:  
El icono  
extraído



## **ExtractIconEx     ShellAPI.Pas**

### **Sintaxis**

```

ExtractIconEx(
    lpszFile: PChar;           {puntero a una cadena con el nombre de fichero}
    nIconIndex: Integer;       {índice de icono}
    var phiconLarge: HICON;     {puntero a manejador de icono grande}
    var phiconSmall: HICON;     {puntero a manejador de icono pequeño}
    nIcons: UINT               {cantidad de iconos a recuperar}
): UINT;                     {la cantidad de iconos que contiene el fichero}

```

### **Descripción**

Esta función devuelve un manejador de los iconos grande y pequeño almacenados en un ejecutable, una DLL o un fichero de iconos.

### **Parámetros**

*lpszFile*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre de fichero de un ejecutable, una DLL o un fichero de iconos, desde el cual extraer los iconos.

*nIconIndex*: El índice del icono a recuperar. El índice es basado en cero, de manera que un valor cero recuperará el primer icono grande o pequeño en el fichero, si alguno existe. Si este valor es -1, el valor devuelto será la cantidad total de iconos almacenados en el fichero.

*phiconLarge*: Puntero a un manejador de icono. Si la función tiene éxito, a través de este parámetro se devolverá el manejador de un icono grande extraído del fichero dado.

*phiconSmall*: Puntero a un manejador de icono. Si la función tiene éxito, a través de este parámetro se devolverá el manejador de un icono pequeño extraído del fichero dado.

*nIcons*: Valor que indica la cantidad de iconos a extraer.

#### Valor que devuelve

Si la función tiene éxito y el valor de *nIconIndex* es -1, devuelve la cantidad total de iconos almacenados en el fichero. La función considera un icono grande y su icono pequeño correspondiente como uno solo (por ejemplo, si hay 3 iconos grandes y 3 iconos pequeños en un fichero, esta función devolverá 3). Si la función extrae iconos, devuelve el número total de iconos grandes y pequeños extraídos (por ejemplo, si extrae un icono grande y otro pequeño, devuelve 2). Si la función falla o no hay iconos almacenados en el fichero indicado, devuelve cero.

#### Véase además

*DrawIcon*, *DrawIconEx*, *ExtractIcon*, *LoadIcon*

#### Ejemplo

##### Listado 6-12: Extrayendo iconos grandes y pequeños

```
var
  Form1: TForm1;
  LargeIconsBitmap: TBitmap; // almacena las imágenes de los iconos
  SmallIconsBitmap: TBitmap;

implementation

{$R *.DFM}

procedure TForm1.FileListBox1DbClick(Sender: TObject);
var
  NumIcons: Integer;          // almacena la cantidad de iconos
  LIcon: HICON;               // almacena los manejadores de los iconos extraídos
  SIcon: HICON;
  LoopCount: Integer;         // contador de bucle
begin
  {Determina el número de iconos almacenados en el fichero}
  NumIcons := ExtractIconEx(PChar(FileListBox1.FileName), -1, LIcon, SIcon, 0);

  {Muestra esta cantidad}
  Label4.Caption := 'Total Number of Icons: ' + IntToStr(NumIcons);

  {Redimensiona las imágenes y borra las áreas de dibujo de los mapas de bits
   fuera de pantalla. Añadimos un 1 al ancho en caso de que no haya iconos. Esto
   evita que la altura de estos objetos sea reiniciada a 1.}
  Image1.Width := NumIcons * 40 + 1;
  Image2.Width := NumIcons * 40 + 1;
  LargeIconsBitmap.Width := NumIcons * 40 + 1;
  LargeIconsBitmap.Canvas.FillRect(LargeIconsBitmap.Canvas.ClipRect);
  SmallIconsBitmap.Width := NumIcons * 40 + 1;
```

```

SmallIconsBitmap.Canvas.FillRect(SmallIconsBitmap.Canvas.ClipRect);

{Extrae cada icono grande y pequeño del fichero}
for LoopCount:=0 to NumIcons-1 do
begin
  {Busca el icono en la aplicación seleccionada, si existe}
  ExtractIconEx(PChar(FileListBox1.FileName), LoopCount, LIcon, SIcon, 1);

  {Muestra el icono grande}
  DrawIcon(LargeIconsBitmap.Canvas.Handle, (LoopCount*40) + 4, 2, LIcon);

  {Dibuja el icono pequeño a las dimensiones correctas}
  DrawIconEx(SmallIconsBitmap.Canvas.Handle, (LoopCount * 40) + 4, 2, SIcon,
    GetSystemMetrics(SM_CXSMICON), GetSystemMetrics(SM_CYSMICON),
    0, 0, DI_NORMAL);
end;

{Asigna los mapas de bits fuera de pantalla a las imágenes a mostrar}
Image1.Picture.Bitmap.Assign(LargeIconsBitmap);
Image2.Picture.Bitmap.Assign(SmallIconsBitmap);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {Crea los mapas de bits fuera de pantalla para contener imágenes de los iconos}
  LargeIconsBitmap := TBitmap.Create;
  LargeIconsBitmap.Height := 53;
  LargeIconsBitmap.Width := 40;

  SmallIconsBitmap:=TBitmap.Create;
  SmallIconsBitmap.Height := 53;

```

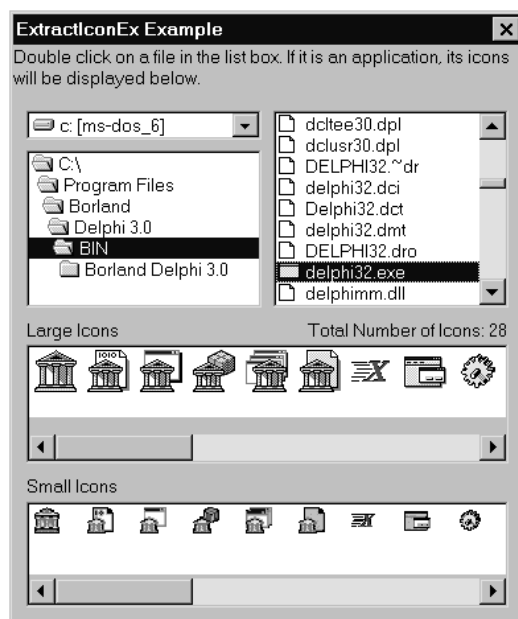


Figura 6-13:  
Los iconos  
grandes y  
pequeños  
extraídos de  
un fichero

```

        SmallIconsBitmap.Width := 40;
    end;

    procedure TForm1.FormDestroy(Sender: TObject);
    begin
        {Libera los mapas de bits fuera de pantalla}
        LargeIconsBitmap.Free;
        SmallIconsBitmap.Free;
    end;

```

## GetCursor Windows.Pas

### Sintaxis

GetCursor: HCURSOR; {devuelve un manejador de cursor}

### Descripción

Esta función recupera un manejador del cursor actual.

### Valor que devuelve

Si tiene éxito, devuelve un manejador del cursor actual; en caso contrario, cero.

### Véase además

*CreateCursor, SetCursor*

### Ejemplo

Vea el Listado 6-5 bajo *CreateCursor*.

## GetIconInfo Windows.Pas

### Sintaxis

```

GetIconInfo(
    hIcon: HICON;           {manejador de un icono o cursor}
    var piconinfo: TIconInfo {puntero a un registro de información de icono}
): BOOL;                  {devuelve TRUE o FALSE}

```

### Descripción

Esta función recupera información sobre un icono o cursor, incluyendo el 'punto caliente' y las imágenes de máscara.

### Parámetros

*hIcon*: Manejador de un icono o cursor cuya información será recuperada. Para recuperar información sobre un icono o cursor estándar, a este parámetro se le puede asignar un valor de la Tabla 6-4. Consulte la función *LoadCursor* para obtener una explicación detallada de los valores del cursor.

*piconinfo*: Puntero a un registro *TIconInfo*. Este registro es rellenado con la información recuperada del icono o cursor especificado cuando la función retorna. El registro *TIconInfo* se define de la siguiente forma:

*TIconInfo* = **packed record**

<i>flcon</i> : BOOL;	{indica si se trata de un icono o cursor}
<i>xHotspot</i> : DWORD;	{coordenada horizontal del ‘punto caliente’}
<i>yHotspot</i> : DWORD;	{coordenada vertical del ‘punto caliente’}
<i>hbmMask</i> : HBITMAP;	{manejador de un mapa de bits}
<i>hbmColor</i> : HBITMAP;	{manejador de un mapa de bits}

**end;**

*flcon*: Valor que indica si el registro contiene información sobre un icono o un cursor. Si este campo contiene TRUE, el registro contiene información sobre un icono; en caso contrario, contiene información sobre un cursor.

*xHotspot*: Especifica la coordenada horizontal del ‘punto caliente’ del cursor. Si el registro contiene información sobre un icono, el ‘punto caliente’ está siempre en el centro del icono y este campo es ignorado.

*yHotspot*: Especifica la coordenada vertical del ‘punto caliente’ del cursor. Si el registro contiene información sobre un icono, el ‘punto caliente’ está siempre en el centro del icono y este campo es ignorado.

*hbmMask*: Manejador del mapa de bits de la máscara **and** del icono o cursor. Si el registro contiene información sobre un icono o cursor en blanco y negro, la máscara **and** es formateada de manera que la mitad superior contenga la máscara **and** y la mitad inferior la máscara **or**. En este caso, el campo *hbmColor* puede contener cero. La aplicación deberá eliminar este mapa de bits cuando deje de ser necesario.

*hbmColor*: Manejador del mapa de bits de la máscara **or** del icono o cursor. Para cursores animados, éste será el primer cuadro de imagen del cursor. La aplicación deberá eliminar este mapa de bits cuando deje de ser necesario.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateIcon*, *CreateIconFromResource*, *CreateIconIndirect*, *DestroyIcon*, *DrawIcon*, *DrawIconEx*, *LoadIcon*

#### Ejemplo

##### Listado 6-13: Recuperando información sobre iconos y cursores del sistema

```
procedure TForm1.ComboX1Change(Sender: TObject);
var
    TheIcon: HICON;           // almacena un icono
```

```

TheCursor: HCURSOR;           // almacena un cursor

TheIconInfo: TIconInfo;       // almacena información sobre el cursor o el icono

{Esto especifica los iconos y cursores del sistema}
type
  TIconTypes = array[0..8] of PAnsiChar;
  TCursorTypes = array[0..12] of PAnsiChar;

const
  IconTypes: TIconTypes = (IDI_APPLICATION, IDI_ASTERISK, IDI_ERROR,
                           IDI_EXCLAMATION, IDI_HAND, IDI_INFORMATION,
                           IDI_QUESTION, IDI_WARNING, IDI_WINLOGO);
  CursorTypes: TCursorTypes = (IDC_ARROW, IDC_IBEAM, IDC_WAIT, IDC_CROSS,
                                IDC_UPARROW, IDC_SIZENWSE, IDC_SIZENESW,
                                IDC_SIZEWE, IDC_SIZENS, IDC_SIZEALL,
                                IDC_NO, IDC_APPSTARTING, IDC_HELP);

begin
  {Borra la última imagen}
  Image1.Canvas.Brush.Color := clBtnFace;
  Image1.Canvas.Fillrect(Image1.Canvas.Cliprect);
  if TheIconInfo.hbmMask <> 0 then DeleteObject(TheIconInfo.hbmMask);
  if TheIconInfo.hbmColor <> 0 then DeleteObject(TheIconInfo.hbmColor);

  {Si hemos seleccionado iconos, obtiene un icono...}
  if RadioButton1.Checked then
    begin
      {Carga el icono de sistema seleccionado}
      TheIcon := LoadIcon(0, IconTypes[ComboBox1.ItemIndex]);

      {Rellena el registro para este icono}
      GetIconInfo(TheIcon, TheIconInfo);

      {Ahora dibuja el icono sobre el área de dibujo de Image1}
      DrawIconEx(Image1.Canvas.Handle, 25, 25, TheIcon, 0, 0, 0,
                  Image1.Canvas.Brush.Handle, DI_DEFAULTSIZE or DI_NORMAL);
    end
  else
    {...en caso contrario, obtiene un cursor}
    begin
      {Carga el cursor de sistema seleccionado}
      TheCursor := LoadCursor(0, CursorTypes[ComboBox2.ItemIndex]);

      {Rellena el registro para este cursor}
      GetIconInfo(TheCursor, TheIconInfo);

      {Ahora dibuja el cursor en el área de dibujo de Image1}
      DrawIconEx(Image1.Canvas.Handle, 25, 25, TheCursor, 0, 0, 0,
                  Image1.Canvas.Brush.Handle, DI_DEFAULTSIZE or DI_NORMAL);
    end;

  {Limpia el cuadro de lista}
  ListBox1.Items.Clear;

```

```

{Rellena el cuadro de lista con la información sobre el icono o cursor}
if TheIconInfo.fIcon then
    ListBox1.Items.Add('This is an icon')
else
    ListBox1.Items.Add('This is a cursor');

{Muestra el punto caliente}
ListBox1.Items.Add('X Hotspot: ' + IntToStr(TheIconInfo.xHotspot));
ListBox1.Items.Add('Y Hotspot: ' + IntToStr(TheIconInfo.yHotspot));
{Muestra las máscaras AND y OR para este cursor o icono}
Image2.Picture.Bitmap.Handle := TheIconInfo.hbmMask;
Image3.Picture.Bitmap.Handle := TheIconInfo.hbmColor;
end;

```

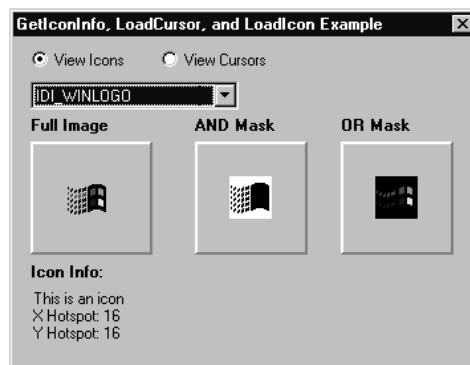


Figura 6-14:  
La información  
sobre el icono  
o cursor

Tabla 6-4: Valores del parámetro hIcon de GetIconInfo

Valor	Descripción
IDI_APPLICATION	El icono por defecto de la aplicación.
IDI_ASTERISK	El icono de información del sistema.
IDI_ERROR	El icono de error del sistema.
IDI_EXCLAMATION	El icono de exclamación del sistema.
IDI_HAND	Equivalente a IDI_ERROR.
IDI_INFORMATION	Equivalente a IDI_ASTERISK.
IDI_QUESTION	El icono de interrogación del sistema.
IDI_WARNING	Equivalente a IDI_EXCLAMATION.
IDI_WINLOGO	El icono del logotipo de Windows.
IDC_ARROW	El cursor predeterminado (flecha).
IDC_IBEAM	El cursor identificado por la clave IBeam.
IDC_WAIT	El cursor identificado por la clave Wait.
IDC_CROSS	El cursor identificado por la clave Crosshair.
IDC_UPARROW	El cursor identificado por la clave UpArrow.
IDC_SIZENWSE	El cursor identificado por la clave SizeNWSE.

Valor	Descripción
IDC_SIZEESW	El cursor identificado por la clave SizeNESW.
IDC_SIZEWE	El cursor identificado por la clave SizeWE.
IDC_SIZEWS	El cursor identificado por la clave SizeNS.
IDC_SIZEALL	El cursor identificado por la clave SizeAll.
IDC_NO	El cursor identificado por la clave No.
IDC_APPSTARTING	El cursor identificado por la clave AppStarting.
IDC_HELP	El cursor identificado por la clave Help.

### **HideCaret**      **Windows.Pas**

#### **Sintaxis**

```
HideCaret(
    hWnd: HWND      {manejador de la ventana que posee el cursor de edición}
); BOOL;            {devuelve TRUE o FALSE}
```

#### **Descripción**

Esta función oculta el cursor de edición de la pantalla, pero no lo destruye ni pierde el punto de inserción. El ocultamiento del cursor de edición es acumulativo. Por cada vez que la función *HideCaret* sea llamada, deberá realizarse una llamada a la función *ShowCaret* para que el cursor de edición sea mostrado de nuevo.

#### **Parámetros**

*hWnd*: Manejador de la ventana que posee el cursor de edición. Si a este parámetro se le asigna FALSE, la función busca todas las ventanas pertenecientes a la tarea actual. Si ninguna ventana de la tarea actual posee el cursor de edición, la función *HideCaret* falla.

#### **Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### **Véase además**

*CreateCaret*, *DestroyCaret*, *ShowCaret*

#### **Ejemplo**

Vea el Listado 6-4 bajo *CreateCaret*.

### **LoadCursor**      **Windows.Pas**

#### **Sintaxis**

```
LoadCursor(
```

```

hInstance: HINST;           {manejador de instancia}
lpCursorName: PAnsiChar     {identificador o nombre del recurso del cursor}
): HCURSOR;                 {devuelve un manejador de cursor}

```

### Descripción

Esta función recupera un manejador de cursor desde los recursos de cursor almacenados en el fichero ejecutable asociado con el manejador de instancia dado. Si el cursor no está actualmente cargado, esta función cargará el recurso de cursor especificado y devolverá su manejador; en caso contrario, devolverá un manejador del cursor existente.

### Parámetros

*hInstance*: Manejador de instancia del módulo cuyo fichero ejecutable contiene el recurso de cursor que será cargado.

*lpCursorName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso de cursor a cargar. La función *MakeIntResource* puede ser usada con un identificador de recurso para ofrecer un valor para este parámetro. Para cargar uno de los cursores definidos por el usuario, se deberá asignar cero (0) al parámetro *hInstance* y asignar a este parámetro a un valor de la Tabla 6-5. Los cursores definidos por el usuario se especifican en la configuración del ratón en el Panel de Control, y son almacenados en el registro de Windows bajo la entrada *HKEY\_CURRENT\_USER\Control Panel\Cursors*.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del cursor cargado desde los recursos del fichero ejecutable; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*GetCursor*, *GetIconInfo*, *LoadImage*, *SetCursor*, *ShowCursor*

### Ejemplo

Vea el Listado 6-13 bajo *GetIconInfo*.

**Tabla 6-5: Valores del parámetro lpCursorName de LoadCursor**

Valor	Descripción
IDC_APPSTARTING	El cursor identificado por la clave AppStarting.
IDC_ARROW	El cursor identificado por la clave Arrow.
IDC_CROSS	El cursor identificado por la clave Crosshair.
IDC_HELP	El cursor identificado por la clave Help.
IDC_IBEAM	El cursor identificado por la clave IBeam.
IDC_NO	El cursor identificado por la clave No.
IDC_SIZEALL	El cursor identificado por la clave SizeAll.

Valor	Descripción
IDC_SIZESEW	El cursor identificado por la clave SizeSEW.
IDC_SIZESE	El cursor identificado por la clave SizeSE.
IDC_SIZESEWSE	El cursor identificado por la clave SizeSEWSE.
IDC_SIZESE	El cursor identificado por la clave SizeSE.
IDC_UPARROW	El cursor identificado por la clave UpArrow.
IDC_WAIT	El cursor identificado por la clave Wait.

## LoadCursorFromFile Windows.Pas

### Sintaxis

```
LoadCursorFromFile(
  lpFileName: PAnsiChar      {nombre del fichero de cursor}
): HCURSOR;                  {devuelve un manejador de cursor}
```

### Descripción

Esta función crea un cursor a partir de los datos almacenados en el fichero especificado, devolviendo un manejador del nuevo cursor. El fichero puede ser un fichero de cursor normal (\*.cur), o puede contener un cursor animado (\*.ani).

### Parámetros

*lpFileName*: Cadena de caracteres terminada en nulo que identifica el fichero de cursor usado para crear el cursor.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del nuevo cursor; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*LoadCursor*, *SetCursor*, *SetSystemCursor*

### Ejemplo

#### Listado 6-14: Cargando un cursor desde un fichero

```
procedure TForm1.FileListBox1Click(Sender: TObject);
var
  TheCursor: HCURSOR; // manejador de un cursor cargado desde un fichero
begin
  {Borra la última imagen}
  Image1.Canvas.Brush.Color := clBtnFace;
  Image1.Canvas.Fillrect(Image1.Canvas.Cliprect);

  {Carga el cursor desde el fichero seleccionado}
```

```

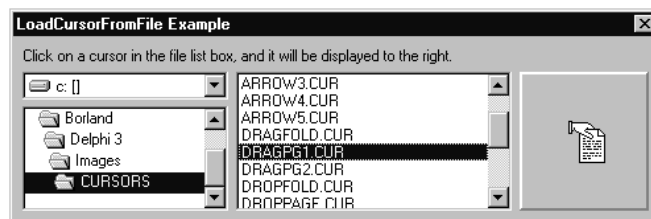
TheCursor := LoadCursorFromFile(PChar(FileListBox1.FileName));

{Dibuja el cursor en el área de dibujo de Image1}
DrawIconEx(Image1.Canvas.Handle, 35, 35, TheCursor, 0, 0, 0, 0,
           DI_DEFAULTSIZE or DI_NORMAL);

{El cursor no es necesario y lo eliminamos}
DeleteObject(TheCursor);
end;

```

Figura 6-15:  
El cursor  
cargado



## LoadIcon Windows.Pas

### Sintaxis

```

LoadIcon(
    hInstance: HINST;           {manejador de instancia}
    lpIconName: PChar           {nombre de recurso de icono}
): HICON;                       {devuelve un manejador de icono}

```

### Descripción

Esta función recupera un manejador de un icono desde los recursos almacenados en el fichero ejecutable asociado con el manejador de instancia especificado. Si el icono no está actualmente cargado, esta función cargará el recurso de icono especificado y devolverá su manejador; en caso contrario, devolverá un manejador del icono existente. El icono tiene que tener las dimensiones predeterminadas por el sistema para los iconos (que pueden obtenerse llamando a *GetSystemMetrics* con los valores *SM\_CXICON* y *SM\_CYICON*). Utilice la función *LoadImage* para cargar iconos de dimensiones diferentes.

### Parámetros

*hInstance*: Manejador del módulo de instancia cuyo fichero ejecutable contiene el recurso de icono que se desea cargar.

*lpIconName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso de icono a cargar. La función *MakeIntResource* puede ser usada con un identificador de recurso para ofrecer un valor para este parámetro. Para cargar uno de los iconos predefinidos usados por el API Win32, asigne cero al parámetro *hInstance* y asigne a este parámetro uno de los valores de la Tabla 6-6.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del icono cargado desde los recursos del fichero ejecutable; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateIcon*, *LoadImage*

#### Ejemplo

Vea el Listado 6-13 bajo *GetIconInfo*.

**Tabla 6-6: Valores del parámetro *lplconName* de *LoadIcon***

Valor	Descripción
IDI_APPLICATION	El icono por defecto de la aplicación.
IDI_ASTERISK	El icono de información del sistema.
IDI_ERROR	El icono de error del sistema.
IDI_EXCLAMATION	El icono de exclamación del sistema.
IDI_HAND	Equivalente a IDI_ERROR.
IDI_INFORMATION	Equivalente a IDI_ASTERISK.
IDI_QUESTION	El icono de interrogación del sistema.
IDI_WARNING	Equivalente a IDI_EXCLAMATION.
IDI_WINLOGO	El icono del logotipo de Windows.

#### **LookupIconIdFromDirectory**

#### **Windows.Pas**

#### Sintaxis

```
LookupIconIdFromDirectory(
    pResBits: PByte;           {puntero a los bits de un recurso de icono o cursor}
    fIcon: BOOL                {indicador de icono o cursor}
): Integer;                  {devuelve un identificador del recurso}
```

#### Descripción

Esta función busca en la información de recursos de iconos o cursores para encontrar el icono o cursor más apropiado para el dispositivo de visualización actual. Está prevista para usarse con ficheros de recursos que contengan imágenes de iconos y cursores en formatos dependientes del dispositivo e independientes del dispositivo. El valor devuelto por esta función puede ser usado con *MakeIntResource* y *FindResource* para ubicar el cursor o icono en los recursos del módulo.

**Parámetros**

*pResBits*: Puntero a los bits de un recurso de icono o cursor. Utilice el valor devuelto por la función *LockResource* para este parámetro.

*fIcon*: Un valor que indica si se desea un icono o un cursor. Un valor TRUE indica que se debe hallar un icono; FALSE indica un cursor.

**Valor que devuelve**

Si la función tiene éxito, devuelve un valor entero que es un identificador de recurso del icono o cursor más apropiado para el dispositivo de visualización actual; en caso contrario, devuelve cero.

**Véase además**

*CreateIconFromResource*, *FindResource*, *FindResourceEx*, *LoadCursor*, *LookupIconIdFromDirectoryEx*

**Ejemplo**

Vea el Listado 6-7 bajo *CreateIconFromResource*.

**LookupIconIdFromDirectoryEx****Windows.Pas****Sintaxis**

```
LookupIconIdFromDirectoryEx(
    pResBits: PByte;           {puntero a los bits de un recurso de icono o cursor}
    fIcon: BOOL                {indicador de icono o cursor}
    cxDesired: Integer;        {ancho preferido del icono o cursor}
    cyDesired: Integer;        {altura preferida del icono o cursor}
    Flags: UINT                {opción de color}
): Integer;                   {devuelve un identificador del recurso}
```

**Descripción**

Esta función busca en la información de recursos de iconos o cursores para encontrar el icono o cursor más apropiado para el dispositivo de visualización actual. Está prevista para usarse con ficheros de recursos que contengan imágenes de iconos y cursores en formatos dependientes del dispositivo e independientes del dispositivo. El valor devuelto por esta función puede ser usado con *MakeIntResource* y *FindResource* para ubicar el cursor o icono en los recursos del módulo. A diferencia de la función *LookupIconIdFromDirectory*, esta función permite al desarrollador especificar las dimensiones y formato de colores del icono o cursor.

**Parámetros**

*pResBits*: Puntero a los bits de un recurso de icono o cursor. Utilice el valor devuelto por la función *LockResource* para este parámetro.

*fIcon*: Un valor que indica si se desea un icono o un cursor. Un valor TRUE indica que se debe hallar un icono; FALSE indica un cursor.

*cxDesired*: Especifica el ancho preferido para el icono o cursor en píxeles. Si este parámetro es cero, la función usa el valor devuelto por *GetSystemMetrics(SM\_CXICON)*.

*cyDesired*: Especifica la altura preferida para el icono o cursor en píxeles. Si este parámetro es cero, la función usa el valor devuelto por *GetSystemMetrics(SM\_CYICON)*.

*Flags*: Un valor que indica el formato de colores para el icono o cursor. Este parámetro puede tomar un valor de la Tabla 6-7.

#### Valor que devuelve

Si la función tiene éxito, devuelve un valor entero que es un identificador de recurso para el icono o cursor más apropiado para el dispositivo de visualización actual; en caso contrario, devuelve cero.

#### Véase además

*CreateIconFromResourceEx*, *FindResource*, *FindResourceEx*, *LoadCursor*, *LookupIconIdFromDirectory*

#### Ejemplo

Vea el Listado 6-8 bajo *CreateIconFromResourceEx*.

**Tabla 6-7: Valores del parámetro *Flags* de *LookupIconIdFromDirectoryEx***

Valor	Descripción
LR_DEFAULTCOLOR	Crea un cursor o icono a colores, usando los colores por defecto del sistema.
LR_MONOCHROME	Crea un cursor o icono monocromático.

## SetCursor

## Windows.Pas

### Sintaxis

```
SetCursor(
    hCursor: HICON           {manejador de cursor}
); HCURSOR;                {devuelve manejador del cursor anterior}
```

### Descripción

Esta función asigna a la forma del cursor del ratón el cursor asociado con el manejador de cursor especificado. Un nuevo cursor es asignado sólo si es diferente del actual. Cuando se usa la función *SetCursor* para cambiar el cursor, el cursor de la clase de ventana a la que pertenecen la ventana de la aplicación y sus ventanas hijas tiene que

ser puesto a cero. Si el cursor de la clase de ventana de una ventana no es cero, Windows restaura la forma del cursor de la clase cada vez que el ratón es movido sobre esa ventana en particular. Utilice la función *ShowCursor* para incrementar el contador interno con vistas a mostrar el nuevo cursor.

#### Parámetros

*hCursor*: Manejador del cursor que reemplaza la forma del cursor actual del ratón. Este manejador de cursor debe haber sido obtenido de una llamada a las funciones *CreateCursor*, *LoadCursor* o *LoadImage*. Adicionalmente, el ancho y la altura del cursor tienen que corresponderse con los devueltos por la función *GetSystemMetrics*, y su profundidad de color tiene que ser igual o menor que la profundidad de color del dispositivo de visualización actual.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del cursor anterior, si existe; en caso contrario, devuelve cero.

#### Véase además

*CreateCursor*, *GetCursor*, *ShowCursor*

#### Ejemplo

Vea el Listado 6-5 bajo *CreateCursor*.

### SetSystemCursor

### Windows.Pas

#### Sintaxis

```
SetSystemCursor(
    hcur: HICON;           {manejador del nuevo cursor}
    id: DWORD              {identificador de cursor del sistema}
): BOOL;                 {devuelve TRUE o FALSE}
```

#### Descripción

Esta función reemplaza la imagen del cursor del sistema especificado, con la imagen del cursor identificado por el parámetro *hcur*. El registro de Windows no es actualizado con esta nueva selección de cursor, y el cursor original del sistema será restaurado la próxima vez que Windows sea iniciado.

#### Parámetros

*hcur*: Manejador del cursor que reemplazará al cursor del sistema especificado.

*id*: Un identificador de cursor del sistema. Determina el cursor del sistema que será reemplazado por el cursor especificado por el parámetro *hcur*. Este parámetro puede tomar un valor de la Tabla 6-8.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*CreateCursor*, *GetCursor*, *LoadCursor*, *LoadCursorFromFile*, *SetCursor*

### Ejemplo

#### Listado 6-15: Asignando un nuevo cursor del sistema

```
var
  Form1: TForm1;
  CurSysCursor: HCURSOR; // almacena el cursor actual del sistema

procedure TForm1.Button1Click(Sender: TObject);
begin
  {Guarda un manejador del cursor actual del sistema}
  CurSysCursor := GetCursor;

  {Asigna un nuevo cursor del sistema}
  SetSystemCursor(Screen.Cursors[crHandPoint], OCR_NORMAL);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {Restaura el cursor del sistema anterior}
  SetSystemCursor(CurSysCursor, OCR_NORMAL);
end;
```

Figura 6-16:  
El nuevo  
cursor del  
sistema

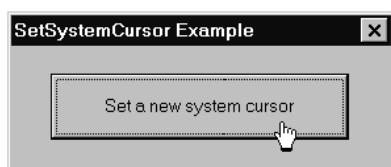


Tabla 6-8: Valores del parámetro id de *SetSystemCursor*

Valor	Descripción
OCR_APPSTARTING	El cursor del reloj de arena pequeño con flecha.
OCR_CROSS	El cursor de punto de mira.
OCR_IBEAM	El cursor de insertar texto.
OCR_NO	El cursor internacional de ausencia de símbolo.
OCR_NORMAL	El cursor de flecha normal.
OCR_SIZEALL	El cursor de redimensionamiento en todas direcciones.
OCR_SIZENESW	El cursor de redimensionamiento de noreste a suroeste.

Valor	Descripción
OCR_SIZENS	El cursor de redimensionamiento vertical.
OCR_SIZENWSE	El cursor de redimensionamiento de noroeste a sureste.
OCR_SIZEWE	El cursor de redimensionamiento horizontal.
OCR_UP	El cursor de flecha hacia arriba.
OCR_WAIT	El cursor de reloj de arena.

**ShowCaret****Windows.Pas****Sintaxis**

```
ShowCaret(
    hWnd: HWND           {manejador de ventana}
): BOOL;                {devuelve TRUE o FALSE}
```

**Descripción**

Muestra el cursor de edición en la pantalla en el punto de inserción actual. El cursor de edición aparece sólo si la ventana especificada lo posee, si tiene forma no vacía y si la función *HideCaret* no ha sido llamada dos o más veces secuencialmente. El ocultamiento del cursor de edición es acumulativo. Por cada vez que la función *HideCaret* sea llamada, deberá realizarse una llamada a la función *ShowCaret* para que el cursor de edición sea mostrado de nuevo.

**Parámetros**

*hWnd*: Manejador de la ventana propietaria del cursor de edición. Si a este parámetro se le asigna FALSE, la función busca en todas las ventanas pertenecientes a la tarea actual. Si ninguna ventana de la tarea actual posee el cursor de edición, la función *ShowCaret* falla.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateCaret*, *DestroyCaret*, *HideCaret*

**Ejemplo**

Vea el Listado 6-4 bajo *CreateCaret*.

**ShowCursor**      **Windows.Pas****Sintaxis**

```
ShowCursor(
  bShow: BOOL           {opción de visibilidad del cursor}
): Integer;             {devuelve el contador de muestra del cursor}
```

**Descripción**

Esta función muestra u oculta el cursor, incrementando o decrementando un contador interno asociado al dispositivo de visualización. Cuando este contador interno se hace menor que cero (0), el cursor del ratón es ocultado. Si hay un ratón instalado en el sistema, el valor inicial del contador es 0; en caso contrario, su valor inicial es -1.

**Parámetros**

*bShow*: Un valor booleano que indica si el contador interno del dispositivo de visualización debe ser incrementado o decrementado. Un valor TRUE incrementa el contador; FALSE lo decrementa.

**Valor que devuelve**

Si la función tiene éxito, devuelve el nuevo valor del contador interno del dispositivo de visualización. Si la función falla, devuelve cero. Un valor de retorno igual a cero no significa necesariamente que la función falló; para determinar un posible fallo, deberá hacerse una comparación con valores devueltos previamente por la función.

**Véase además**

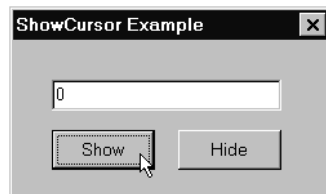
*GetCursor*, *GetCursorPos*, *SetCursor*, *SetCursorPos*

**Ejemplo****Listado 6-16: Ocultando y mostrando el cursor**

```
procedure TForm1.Button1Click(Sender: TObject);
const
  RefCount: Integer = 0; // almacena la cantidad de llamadas
begin
  {Si se ha pulsado el botón 'Show' (Mostrar)...}
  if TButton(Sender).Caption = 'Show' then begin
    {...muestra el cursor e incrementa la cuenta de referencia}
    ShowCursor(TRUE);
    Inc(RefCount);
  end
  {Si se ha pulsado el botón 'Hide' (Ocultar)...}
  else begin
    {...oculta el cursor y decrementa la cuenta de referencia}
    ShowCursor(FALSE);
    Inc(RefCount, -1);
  end;
```

```
{Muestra valor actual del contador}  
Edit1.Text := IntToStr(RefCount);  
end;
```

*Figura 6-17:  
El contador  
asociado al  
cursor*



## Capítulo 7

# Funciones de paleta

El color es una parte muy importante de la interfaz de usuario. Permite comunicar datos de una forma muy intuitiva y efectiva y enriquece el atractivo estético general de la aplicación. Los sistemas informáticos actuales incorporan *hardware* de vídeo que puede soportar millones de colores. Si bien es común que los usuarios dispongan de controladores de vídeo de más de 256 colores, es una buena práctica desarrollar las aplicaciones para 256 colores, que es la profundidad de color más comúnmente soportada en los ordenadores de hoy, por increíble que parezca. Adicionalmente, la mayoría de los juegos a pantalla completa se desarrollan para controladores de vídeo de 256 colores, para lograr una mayor velocidad y poder realizar efectos especiales de animación sin necesidad de disponer de hardware especializado. Como resultado, la mayoría de las funciones de este capítulo serán inútiles cuando los controladores de vídeo sean configurados con una profundidad de más de 256 colores.

Las funciones de este capítulo muestran el trabajo con colores y paletas de colores en modo de vídeo de 256 colores. La mayoría de los ejemplos en este capítulo sólo funcionarán en caso de que el controlador de vídeo sean configurado para 256 colores.

## El administrador de paleta de Windows

Windows mantiene una paleta del sistema que contiene todos los colores que pueden ser mostrados. Las 10 primeras y 10 últimas entradas están reservadas para el sistema y contienen los 20 colores estáticos que son usados para los elementos específicos de las ventanas, tales como la barra de título, los bordes o los biseles de los elementos 3-D de la interfaz. Los restantes 236 colores pueden ser asignados por la aplicación, aunque ésta no puede cambiar los colores en la paleta del sistema directamente.

Para modificar los colores de la paleta de sistema, una aplicación tiene que crear una paleta lógica, usando la función *CreatePalette*. Una vez que la paleta es creada, tiene que ser seleccionada en un contexto de dispositivo específico utilizando la función *SelectPalette*. Esta paleta lógica es mezclada con la paleta del sistema (*realizada*) llamando a la función *RealizePalette*. Cuando la función *RealizePalette* es llamada, el administrador de paleta de Windows analiza cada entrada de la paleta lógica y la mapea a una entrada existente en la paleta del sistema. Si no se encuentra una entrada

correspondiente, el color de la entrada lógica es colocado en la primera entrada no utilizada en la paleta del sistema. Si no se encuentra una entrada correspondiente y no quedan entradas libres en la paleta del sistema, el administrador de paleta de Windows mapea el color a la entrada que se corresponda más aproximadamente. El proceso continúa hasta que todos los colores en la paleta lógica hayan sido mezclados con los colores de la paleta del sistema. Si el contexto de dispositivo especificado identifica a una ventana activa, la paleta lógica es tratada como una paleta de primer plano y el administrador de paleta de Windows marca todas las entradas no estáticas en la paleta del sistema como no usadas cuando la función *RealizePalette* es llamada. Esto permite potencialmente a la paleta lógica reemplazar todos los colores no estáticos de la paleta del sistema. En caso contrario, la paleta lógica es tratada como una paleta de fondo y se podrán asignar los espacios no usados de la paleta sólo si queda alguno después de que la paleta de primer plano sea realizada.

### **Paletas de identidad**

Muchas aplicaciones, como los juegos o las utilidades de manipulación de imágenes, tienen que tener un control explícito sobre dónde son mapeadas sus entradas de paleta en la paleta del sistema. En tales situaciones, la aplicación puede crear lo que se conoce como una *paleta de identidad*. Para crear una paleta de identidad, comience por preparar una paleta lógica de 256 entradas. Utilice la función *GetSystemPaletteEntries* para recuperar las 10 primeras y 10 últimas entradas de la paleta del sistema y colocarlas en las 10 primeras y 10 últimas entradas de la paleta lógica. A continuación, asigne a los 236 colores restantes de la paleta lógica los colores deseados, utilizando la opción *PC\_NOCOLLAPSE* en el campo *peFlags* del registro *TPaletteEntry* para cada entrada de la paleta. Cuando esta paleta sea realizada, los primeros y últimos diez colores mapearán directamente los colores estáticos del sistema, y los 236 restantes serán colocados en las entradas no usadas sin ningún tipo de mapeado a los colores existentes.

### **Especificadores de colores**

Los colores dentro de una paleta se identifican mediante *especificadores de colores*. Un especificador de color es un valor de 32 bits que puede indicar la intensidad relativa de los componentes rojo, verde y azul del color en los 3 bytes menos significativos, o puede contener un índice de paleta específico en el byte menos significativo. El byte más significativo determina cómo los otros 3 bytes serán interpretados, según se indica en la Tabla 7-1. Un especificador de color puede ser utilizado en cualquier función de paleta donde se requiera un valor de tipo *COLORREF*, *TColor*, o *DWORD*.

Tabla 7-I: Valores del byte más significativo del especificador de color

Valor	Descripción
\$00	Indica que los 3 bytes menos significativos contienen los valores relativos de la intensidad de los componentes de color azul, verde y rojo. El especificador de color será: \$00bbgrr Este especificador de color producirá un color obtenido mediante la combinación de los veinte colores estáticos que genere la aproximación más cercana al color requerido.
\$01	Indica que el byte menos significativo especifica un índice de la paleta lógica actualmente realizada. El especificador de color será: \$010000nn Este especificador de color producirá el color almacenado en el índice indicado de la paleta lógica activa (realizada).
\$02	Indica que los 3 bytes menos significativos contienen los valores relativos de la intensidad de los componentes azul, verde y rojo del color. El especificador de color será: \$02bbgrr Este especificador de color producirá el color de la paleta lógica activa (realizada) que más se aproxime al color requerido.

El siguiente ejemplo muestra el uso de los 3 tipos de especificadores de colores para indicar varios colores.

#### Listado 7-I: Usando los especificadores de colores

```
{iOJO! Delphi importa incorrectamente la función GetSystemPaletteEntries.
Esta es la declaración correcta, que nos permite utilizar toda la funcionalidad
de esta función del API}
function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: UINT;
                                PaletteEntries: Pointer): UINT; stdcall;

var
    Form1: TForm1;
    FormPalette: HPALETTE;           // manejador de paleta lógica

implementation

{$R *.DFM}

{Enlaza la función GetSystemPaletteEntries}
function GetSystemPaletteEntries; external gdi32 name 'GetSystemPaletteEntries';

procedure TForm1.FormCreate(Sender: TObject);
var
    ThePalette: PLogPalette; // registro de definición de paleta lógica
    iLoop: Integer;          // contador de bucle
begin
```

```

{Reservar memoria suficiente para almacenar los colores en las primeras 10
posiciones de la paleta del sistema, más los 236 nuestros. Esta memoria es
temporal, y no será necesaria después de que la paleta sea creada.}
GetMem(ThePalette, SizeOf(TLogPalette) + 246 * SizeOf(TPaletteEntry));

{Inicializar el número de versión de la paleta}
ThePalette^.palVersion := $300;

{Tendremos un total de 246 entradas en nuestra paleta}
ThePalette^.palNumEntries := 246;

{Obtener las 10 primeras entradas en la paleta del sistema}
GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 10, @(ThePalette^.palPalEntry));

{Sólo queremos 236 nuevas entradas en la paleta y que comiencen inmediatamente
después de las 10 primeras de la paleta del sistema. Recuperando las 10 primeras
entradas de la paleta del sistema, cuando realicemos nuestra nueva paleta, las
10 primeras entradas en la paleta lógica serán mapeadas a las 10 primeras en la
paleta del sistema y le seguirán nuestras entradas.}
for iLoop := 0 to 235 do
begin
    {Crear una paleta de gradiente rojo}
    ThePalette^.palPalEntry[iLoop+10].peRed := Trunc(255-((255 / 235)*iLoop));
    ThePalette^.palPalEntry[iLoop+10].peGreen := 0;
    ThePalette^.palPalEntry[iLoop+10].peBlue := 0;
    {No hace corresponder esta entrada de paleta con ninguna otra}
    ThePalette^.palPalEntry[iLoop+10].peFlags := PC_NOCOLLAPSE;
end;

{Crear la paleta}
FormPalette := CreatePalette(ThePalette^);

{Liberar la memoria temporal}
FreeMem(ThePalette, SizeOf(TLogPalette) + 246 * SizeOf(TPaletteEntry));
end;

function TForm1.GetPalette: HPALETTE;
begin
    {Cuando alguien solicita la paleta del formulario, le entregamos la nueva paleta
    lógica}
    Result := FormPalette;
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    {Seleccionar la nueva paleta lógica en el contexto del dispositivo}
    SelectPalette(Canvas.Handle, FormPalette, FALSE);

    {Mapear la paleta lógica a la paleta del sistema}
    RealizePalette(Canvas.Handle);

    {Mostrar los atributos del color solicitado}
    Label1.Caption := 'Red: ' + IntToStr(ScrollBar1.Position);

```

```

{Mostrar el color combinado}
Canvas.Brush.Color := RGB(ScrollBar1.Position, 0, 0);
Canvas.Rectangle(24, 80, 113, 145);
Label5.Caption := IntToHex(Canvas.Brush.Color, 8);

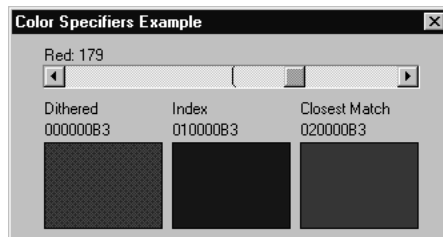
{Mostrar el color en el índice específico}
Canvas.Brush.Color := $01000000 or RGB(ScrollBar1.Position, 0, 0);
Canvas.Rectangle(120, 80, 209, 145);
Label6.Caption := IntToHex(Canvas.Brush.Color, 8);

{Mostrar el color correspondiente más próximo}
Canvas.Brush.Color := $02000000 or RGB(ScrollBar1.Position, 0, 0);
Canvas.Rectangle(216, 80, 305, 145);
Label7.Caption := IntToHex(Canvas.Brush.Color, 8);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Eliminar la paleta lógica, ya que no es necesaria}
    DeleteObject(FormPalette);
end;

```

Figura 7-1:  
Las diferentes  
formas de  
especificación  
de colores



## Funciones de paleta

Este capítulo describe las siguientes funciones de paleta:

Tabla 7-2: Funciones de paleta

Función	Descripción
AnimatePalette	Reemplaza entradas en una paleta lógica.
CreateHalftonePalette	Crea una paleta de tonos intermedios.
CreatePalette	Crea una paleta lógica.
GetBValue	Recupera la intensidad del azul de un especificador de color.
GetDIBColorTable	Recupera la paleta de colores de un mapa de bits independiente del dispositivo.
GetEnhMetaFilePaletteEntries	Recupera la paleta de colores de un metaarchivo mejorado.

<b>Función</b>	<b>Descripción</b>
GetGValue	Recupera la intensidad del verde de un especificador de color.
GetNearestColor	Recupera el especificador de color de la paleta del sistema que más se acerca al color indicado.
GetNearestPaletteIndex	Recupera el índice de la paleta lógica activa (realizada) que más se acerca al color indicado.
GetPaletteEntries	Recupera un rango de entradas de paleta de una paleta lógica.
GetRValue	Recupera la intensidad del rojo de un especificador de color.
GetSysColor	Recupera el especificador de color para el sistema de colores indicado.
GetSystemPaletteEntries	Recupera un rango de entradas de paleta de la paleta del sistema.
GetSystemPaletteUse	Indica si la paleta del sistema contiene colores estáticos.
PaletteIndex	Recupera un especificador de color de un índice específico de la paleta lógica actualmente activa.
PaletteRGB	Recupera el especificador de color de la paleta lógica actualmente activa que más se acerca al color indicado.
RealizePalette	Realiza una paleta lógica en la paleta del sistema.
ResizePalette	Modifica el tamaño de la paleta lógica.
RGB	Recupera un especificador de color para el color requerido que será generado mediante la combinación de los 20 colores estáticos.
SelectPalette	Selecciona una paleta lógica en un contexto de dispositivo.
SetDIBColorTable	Asigna la paleta de colores de un mapa de bits independiente del dispositivo.
SetPaletteEntries	Asigna un rango de entradas de paleta en una paleta lógica.
SetSysColor	Asigna un color de sistema al especificador de color indicado.
SetSystemPaletteUse	Cambia el número de colores estáticos usados por la paleta del sistema.

### **AnimatePalette**    **Windows.Pas**

#### **Sintaxis**

```
AnimatePalette(
    p1: HPALETTE;                    {manejador de una paleta lógica}
```

p2: UINT;	{primera entrada que será reemplazada}
p3: UINT;	{cantidad de entradas que serán reemplazadas}
p4: PPaletteEntry	{puntero a un <i>array</i> de entradas de paleta}
): BOOL;	{devuelve TRUE o FALSE}

**Descripción**

Esta función reemplaza entradas de colores en la paleta lógica especificada, con las nuevas entradas a las que apunta el parámetro *p4*. Sólo serán reemplazadas aquellas entradas de la paleta lógica especificada que tengan asignado *PC\_RESERVED* en el campo *peFlags*.

**Parámetros**

*p1*: Manejador de la paleta lógica cuyas entradas serán reemplazadas.

*p2*: Especifica el índice de la paleta lógica a partir de donde comenzará la sustitución de entradas.

*p3*: Especifica la cantidad de entradas de paleta que serán reemplazadas.

*p4*: Puntero a un *array* de registros *TPaletteEntry* que reemplazarán las entradas especificadas en la paleta lógica. El registro *TPaletteEntry* se define como:

*TPaletteEntry* = **packed record**

peRed: Byte;	{intensidad del color rojo}
peGreen: Byte;	{intensidad del color verde}
peBlue: Byte;	{intensidad del color azul}
peFlags: Byte;	{opciones de uso de entrada de paleta}

**end;**

Consulte la función *CreatePalette* para ver una descripción de este registro.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreatePalette*, *SelectPalette*

**Ejemplo****Listado 7-2: Animación del gradiente de color**

{iOJO! Delphi importa incorrectamente la función *GetSystemPaletteEntries*. Esta es la declaración que brinda toda la funcionalidad accesible mediante esta función del API}

```
function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: UINT;
                                PaletteEntries: Pointer): UINT; stdcall;
```

```

var
    Form1: TForm1;
    FormPalette: HPALETTE;           // almacena la nueva paleta lógica
    AnimPalette: array[0..31] of TPaletteEntry; // entradas de paleta

implementation

{Enlace en la función GetSystemPaletteEntries}
function GetSystemPaletteEntries; external gdi32 name 'GetSystemPaletteEntries';

function TForm1.GetPalette: HPALETTE;
begin
    {Cada vez que al formulario se le pide su paleta devuelve la nueva paleta lógica}
    Result := FormPalette;
end;

procedure TForm1.CreateThePalette;
var
    ThePalette: PLogPalette; // almacena un registro de paleta lógica
    iLoop: Integer;          // contador de bucle
begin
    {Reserva memoria suficiente para almacenar los colores de las 10 primeras
    entradas de la paleta del sistema, más 32 nuestros}
    GetMem(ThePalette, SizeOf(TLogPalette) + 42 * SizeOf(TPaletteEntry));

    {Obtiene las primeras 10 entradas de la paleta del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 10,
        @(ThePalette^.palPalEntry));

    {Inicializa información de la paleta}
    ThePalette^.palVersion := $300;
    ThePalette^.palNumEntries := 42;

    {Crea una paleta de gradiente rojo}
    for iLoop := 0 to 31 do
    begin
        ThePalette^.palPalEntry[iLoop + 10].peRed := 255 - ((255 div 32) * iLoop);
        ThePalette^.palPalEntry[iLoop + 10].peGreen := 0;
        ThePalette^.palPalEntry[iLoop + 10].peBlue := 0;
        ThePalette^.palPalEntry[iLoop + 10].peFlags := PC_RESERVED;

        {Esto seguirá el movimiento de entradas de paleta, por lo que le hacemos
        corresponder con la paleta lógica}
        AnimPalette[iLoop] := ThePalette^.palPalEntry[iLoop+10];
    end;

    {Crea la nueva paleta lógica}
    FormPalette := CreatePalette(ThePalette^);

    {Libera el registro de la paleta, que ya no es necesario}
    FreeMem(ThePalette, SizeOf(TLogPalette) + 42 * SizeOf(TPaletteEntry));
end;

procedure TForm1.FormCreate(Sender: TObject);

```

```

begin
    {Crea la nueva paleta lógica}
    CreateThePalette;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Comienza la animación de la paleta}
    Timer1.Enabled := TRUE;
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    iLoop: Integer; // contador de bucle
begin
    {Selecciona y activa la nueva paleta de gradiente rojo}
    SelectPalette(Canvas.Handle, FormPalette, FALSE);
    RealizePalette(Canvas.Handle);

    {Dibuja una serie de rectángulos que muestran el gradiente rojo}
    for iLoop := 0 to 31 do
    begin
        Canvas.Brush.Color := PaletteIndex(iLoop + 10);
        Canvas.FillRect(Rect(30, iLoop * 10 + 56, 290, (iLoop * 10) + 76));
    end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var
    PaletteEntry: TPALETTEENTRY; // almacenamiento temporal de entradas de paleta
    iLoop: Integer; // contador de bucle
begin
    {El array AnimPalette comienza como un duplicado exacto de la paleta lógica.
    Rotamos las entradas de paleta desde el principio hasta el final del array,
    guardando antes la primera entrada en el array...}
    PaletteEntry := AnimPalette[0];

    {...moviendo todas las demás entradas una posición hacia el inicio...}
    for iLoop := 0 to 30 do
        AnimPalette[iLoop] := AnimPalette[iLoop + 1];

    {...y colocando la antigua primera entrada en la última entrada del array}
    AnimPalette[31] := PaletteEntry;

    {Ahora colocamos el nuevo array de entradas de paleta en la zona exacta de la
    paleta lógica donde asignamos el gradiente rojo. Los resultados visuales
    serán muy agradables.}
    AnimatePalette(FormPalette, 10, 42, @AnimPalette);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {No necesitamos ya la paleta lógica y la eliminamos}
    DeleteObject(FormPalette);

```

```
end;
```

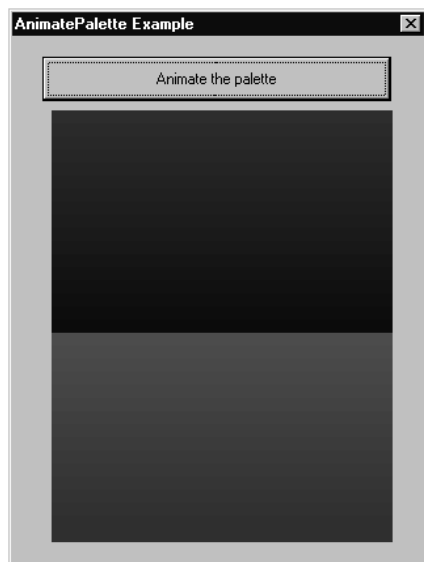


Figura 7-2:  
Un gradiente  
de color en  
movimiento

### **CreateHalftonePalette** Windows.Pas

#### **Sintaxis**

```
CreateHalftonePalette(  
    DC: HDC                                {manejador de contexto de dispositivo}  
): HPALETTE;                             {devuelve un manejador de paleta lógica}
```

#### **Descripción**

Esta función crea una paleta lógica para el contexto de dispositivo especificado. Una paleta de medios tonos debe ser creada cuando una aplicación utiliza la función *SetStretchBltMode* con la opción de modo de escalamiento *HALFTONE*. Esta paleta debe ser seleccionada y activada en el contexto de dispositivo antes de que las funciones *StretchBlt* o *StretchDIBits* sean llamadas. Cuando la paleta ya no sea necesaria, deberá ser eliminada usando la función *DeleteObject*.

#### **Parámetros**

*DC*: Manejador de un contexto de dispositivo usado como referencia para los colores de la paleta de medios tonos.

#### **Valor que devuelve**

Si la función tiene éxito, devuelve el manejador de la nueva paleta lógica; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*CreatePalette, DeleteObject, RealizePalette, SelectPalette, SetStretchBltMode, StretchDIBits, StretchBlt*

Ejemplo

### Listado 7-3: Creando una paleta de medios tonos

```
{iOJO! Delphi importa incorrectamente la función GetPaletteEntries.
Aquí está la declaración correcta, que brinda toda la funcionalidad
accesible mediante esta función del API}
function GetPaletteEntries(Palette: HPALETTE; StartIndex, NumEntries: UINT;
    PaletteEntries: Pointer): UINT; stdcall;

var
    Form1: TForm1;
    SysPalette: array[0..255] of TPaletteEntry; // entradas de paleta
    FormPalette: HPALETTE; // manejador de paleta lógica

implementation

{$R *.DFM}

{Enlace de la función GetPaletteEntries}
function GetPaletteEntries; external gdi32 name 'GetPaletteEntries';

function TForm1.GetPalette: HPALETTE;
begin
    {Cuando al formulario se le pide que devuelva su paleta, devuelve la paleta
    lógica creada por la aplicación}
    Result := FormPalette;
end;

procedure TForm1.SetThePalette;
begin
    {Crea una paleta de colores de medios tonos}
    FormPalette := CreateHalftonePalette(Form1.Canvas.Handle);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    iLoop: Integer; // variable de control de bucle general
begin
    {Crea la paleta de medios tonos}
    SetThePalette;

    {Selecciona y activa la paleta de medios tonos en el contexto de dispositivo
    del formulario}
    SelectPalette(Canvas.Handle, FormPalette, FALSE);
    RealizePalette(Canvas.Handle);

    {Recupera los valores de los colores de la paleta de medios tonos}
    GetPaletteEntries(FormPalette, 0, 256, @SysPalette);
```

```

{Muestra los valores de los colores de la paleta de medios tonos}
for iLoop := 0 to 255 do
begin
  ListBox1.Items.Add('Palette slot: ' + IntToStr(iLoop));
  ListBox1.Items.Add('    Red: ' + IntToStr(SysPalette[iLoop].peRed));
  ListBox1.Items.Add('   Green: ' + IntToStr(SysPalette[iLoop].peGreen));
  ListBox1.Items.Add('    Blue: ' + IntToStr(SysPalette[iLoop].peBlue));

  case SysPalette[iLoop].peFlags of
    PC_EXPLICIT:  ListBox1.Items.Add('  Flags: PC_EXPLICIT');
    PC_NOCOLLAPSE: ListBox1.Items.Add('  Flags: PC_NOCOLLAPSE');
    PC_RESERVED:  ListBox1.Items.Add('  Flags: PC_RESERVED');
    else ListBox1.Items.Add('    Flags: NULL');
  end;
end;
end;
end;

```

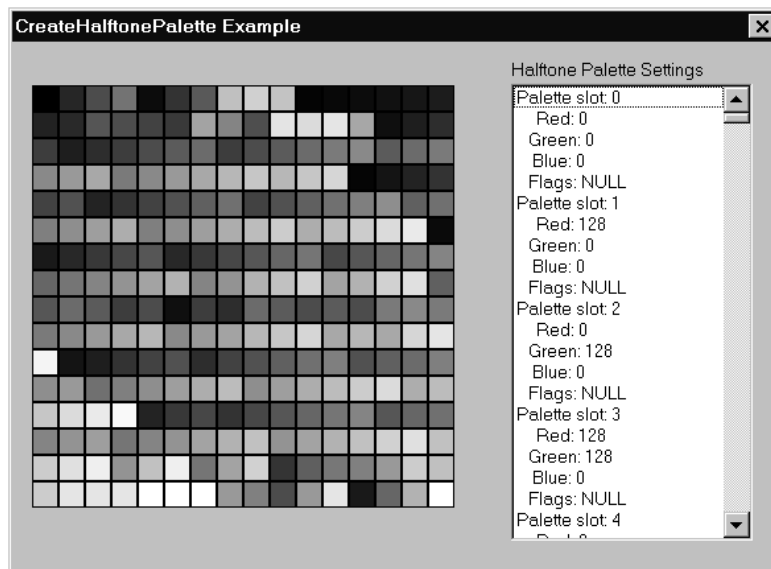


Figura 7-3:  
La paleta de  
medios tonos

```

procedure TForm1.FormPaint(Sender: TObject);
var
  iOutLoop, iInLoop: Integer; // variables de control de bucles
begin
  {Selecciona y activa la paleta de medios tonos en el contexto de dispositivo
  del formulario}
  SelectPalette(Canvas.Handle, FormPalette, FALSE);
  RealizePalette(Canvas.Handle);

  {Dibuja una serie de rectángulos para mostrar la paleta de medios tonos}
  for iOutLoop := 0 to 15 do
    for iInLoop := 0 to 15 do
      begin
        Canvas.Brush.Color := PaletteIndex((iOutLoop * 16) + iInLoop);

```

```

        Canvas.Rectangle((iInLoop * 20) + 15, (iOutLoop * 20) + 32,
                        (iInLoop * 20) + 35, (iOutLoop * 20) + 52);
    end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {No necesitamos ya la paleta de medios tonos y la eliminamos}
    DeleteObject(FormPalette);
end;

```

## CreatePalette Windows.Pas

### Sintaxis

```

CreatePalette(
    const LogPalette: TLogPalette {puntero a registro TLogPalette}
): HPalette; {devuelve un manejador de paleta lógica}

```

### Descripción

Esta función crea una paleta lógica a partir de la descripción de la paleta de colores contenida en el registro de tipo *TLogPalette* al que apunta el parámetro *LogPalette*. Esta nueva paleta puede ser utilizada en las funciones *SelectPalette* y *RealizePalette*. Cuando la paleta ya no sea necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

### Parámetros

*LogPalette*: Puntero al registro *TLogPalette* que contiene información sobre el formato y los colores de la paleta deseada. El registro *TLogPalette* se define como:

```

TLogPalette = packed record
    palVersion: Word; {número de versión de la paleta}
    palNumEntries: Word; {cantidad de entradas de la paleta}
    palPalEntry: array[0..0] of TPaletteEntry; {entradas de la paleta}
end;

```

*palVersion*: Especifica el número de versión de la paleta de Windows. En el momento en que se escribe este libro, a este campo se le debe asignar siempre el valor \$300.

*palNumEntries*: Especifica la cantidad de entradas deseadas en la paleta lógica.

*palPalEntry*: Un *array* de registros de tipo *TPaletteEntry*. Estos registros definen la intensidad de los colores rojo, verde y azul, y el tratamiento de la entrada de paleta para cada entrada en la paleta lógica de colores. Los colores en este *array* deben colocarse en orden descendente de importancia, ya que las primeras entradas son las que más probablemente entren a la paleta del sistema. El registro *TPaletteEntry* se define como:

TPaletteEntry = **packed record**

```

    peRed: Byte;           {intensidad del color rojo}
    peGreen: Byte;         {intensidad del color verde}
    peBlue: Byte;          {intensidad del color azul}
    peFlags: Byte;         {opción de tratamiento de entrada de paleta}

```

**end;**

*peRed*: Especifica la intensidad del componente rojo.

*peGreen*: Especifica la intensidad del componente verde.

*peBlue*: Especifica la intensidad del componente azul.

*peFlags*: Opción que establece cómo esta entrada de paleta será usada. Este campo puede tomar un valor Tabla 7-3.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la nueva paleta lógica; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*DeleteObject*, *RealizePalette*, *SelectPalette*

#### Ejemplo

##### Listado 7-4: Creando una nueva paleta

{iOJO! Delphi importa incorrectamente la función *GetPaletteEntries*. Aquí está la declaración correcta, que brinda toda la funcionalidad accesible mediante esta función del API}

```

function GetPaletteEntries(Palette: HPALETTE; StartIndex, NumEntries: UINT;
    PaletteEntries: Pointer): UINT; stdcall;

```

{iOJO! Delphi importa incorrectamente la función *GetSystemPaletteEntries*. Aquí está la declaración correcta, que brinda toda la funcionalidad accesible mediante esta función del API}

```

function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: UINT;
    PaletteEntries: Pointer): UINT; stdcall;

```

**var**

```

    Form1: TForm1;
    FormPalette: HPALETTE;           // manejador de una paleta lógica

```

**implementation**

```

{$R *.DFM}

```

{Enlace de la función *GetPaletteEntries*}

```

function GetPaletteEntries; external gdi32 name 'GetPaletteEntries';

```

{Enlace de la función *GetSystemPaletteEntries*}

```

function GetSystemPaletteEntries; external gdi32 name 'GetSystemPaletteEntries';

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Crea la nueva paleta}
    CreateThePalette;

    {Muestra información sobre las nuevas entradas de paleta}
    DisplayPaletteEntries;
end;

function TForm1.GetPalette: HPALETTE;
begin
    {Cuando se pide la paleta del formulario, éste devuelve la nueva paleta lógica}
    Result := FormPalette;
end;

procedure TForm1.CreateThePalette;
var
    ThePalette: PLogPalette; // registro de definición de una paleta lógica
    iLoop: Integer;          // contador de bucle
begin
    {Reserva suficiente memoria para almacenar los colores en las primeras 10
    entradas de la paleta del sistema, más los 32 nuestros. Esta memoria es
    temporal y no será necesaria después que la paleta haya sido creada}
    GetMem(ThePalette, SizeOf(TLogPalette) + 42 * SizeOf(TPaletteEntry));

    {Inicializa el número de versión de la paleta}
    ThePalette^.palVersion := $300;

    {Tendremos un total de 42 entradas en nuestra paleta}
    ThePalette^.palNumEntries := 42;

    {Obtiene las primeras 10 entradas de la paleta del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 10,
        @(ThePalette^.palPalEntry));

    {Sólo tenemos 32 nuevas entradas de paleta y queremos que comiencen
    inmediatamente después de las primeras 10 de la paleta del sistema.
    Recuperando las 10 primeras entradas de la paleta del sistema, cuando activemos
    nuestra nueva paleta, las primeras 10 entradas de la paleta lógica serán
    mapeadas a las primeras 10 entradas de la paleta del sistema y nuestras
    entradas de paleta irán a continuación}
    for iLoop := 0 to 31 do
    begin
        {Crea una paleta de gradiente rojo}
        ThePalette^.palPalEntry[iLoop+10].peRed   := 255 - ((255 div 32) * iLoop);
        ThePalette^.palPalEntry[iLoop+10].peGreen := 0;
        ThePalette^.palPalEntry[iLoop+10].peBlue  := 0;
        {No hace corresponder esta entrada de paleta con ninguna otra}
        ThePalette^.palPalEntry[iLoop+10].peFlags := PC_NOCOLLAPSE;
    end;

```

```

    {Crea la paleta}
    FormPalette := CreatePalette(ThePalette^);

    {Libera la memoria temporal}
    FreeMem(ThePalette, SizeOf(TLogPalette) + 42 * SizeOf(TPaletteEntry));
end;

procedure TForm1.DisplayPaletteEntries;
var
    PaletteEntries: array[0..31] of TPaletteEntry; // entradas de paleta
    iLoop: Integer;
begin
    {Obtiene nuestras 32 entradas de la paleta lógica}
    GetPaletteEntries(FormPalette, 10, 32, @PaletteEntries);

    {Muestra la información de la entrada de paleta}
    ListBox1.Items.Clear;
    for iLoop := 0 to 31 do
        begin
            ListBox1.Items.Add('Palette slot: ' + IntToStr(iLoop+10));
            ListBox1.Items.Add('    Red: ' + IntToStr(PaletteEntries[iLoop].peRed));
            ListBox1.Items.Add('    Green: ' + IntToStr(PaletteEntries[iLoop].peGreen));
            ListBox1.Items.Add('    Blue: ' + IntToStr(PaletteEntries[iLoop].peBlue));

            case PaletteEntries[iLoop].peFlags of
                PC_EXPLICIT: ListBox1.Items.Add('    Flags: PC_EXPLICIT');
                PC_NOCOLLAPSE: ListBox1.Items.Add('    Flags: PC_NOCOLLAPSE');
                PC_RESERVED: ListBox1.Items.Add('    Flags: PC_RESERVED');
                else ListBox1.Items.Add('    Flags: NULL');
            end;
        end;
    end;
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    OldPalette: HPALETTE; // manejador de la paleta anterior
    iLoop: Integer; // variable de control de bucle
begin
    {Selecciona nuestra nueva paleta lógica en el contexto de dispositivo}
    OldPalette := SelectPalette(Canvas.Handle, FormPalette, FALSE);

    {Mapea nuestra paleta lógica en la paleta del sistema}
    RealizePalette(Canvas.Handle);

    {Muestra el gradiente rojo}
    for iLoop := 0 to 31 do
        begin
            Canvas.Brush.Color := $01000000 or iLoop + 10;
            Canvas.FillRect(Rect(10, iLoop * 10 + 16, 260, (iLoop * 10) + 36));
        end;

    {Selecciona la paleta previa en el contexto de dispositivo}
    SelectPalette(Canvas.Handle, OldPalette, FALSE);
end;

```

```

procedure TForm1.Button1Click(Sender: TObject);
var
  PaletteEntries: array[0..31] of TPaletteEntry; // entradas de paleta
  iLoop: Integer;
begin
  {Instala un gradiente verde}
  for iLoop := 0 to 31 do
  begin
    PaletteEntries[iLoop].peRed   := 0;
    PaletteEntries[iLoop].peGreen := 255 - ((255 div 32) * iLoop);
    PaletteEntries[iLoop].peBlue  := 0;
    PaletteEntries[iLoop].peFlags := PC_NOCOLLAPSE;
  end;

  {Reinicia la paleta lógica a este gradiente verde}
  SetPaletteEntries(FormPalette, 10, 32, PaletteEntries);

  {Muestra las nuevas entradas de la paleta}
  DisplayPaletteEntries;

  {Redibuja el formulario}
  Invalidate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {No necesitamos ya la paleta lógica y la eliminamos}
  DeleteObject(FormPalette);
end;

```

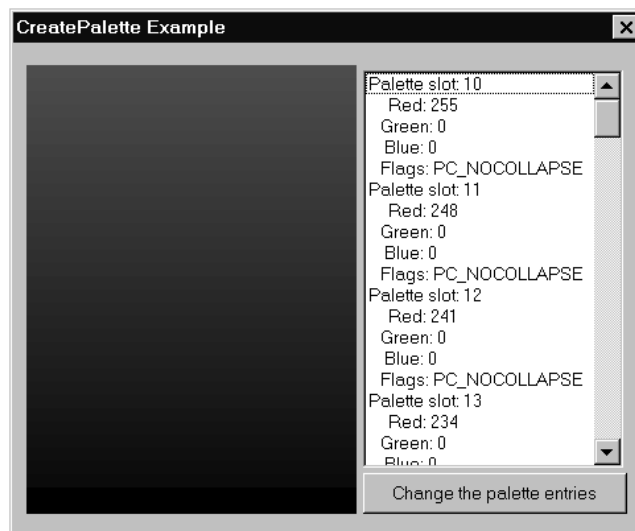


Figura 7-4:  
La paleta  
gradiente

Tabla 7-3: Valores del campo `LogPalette.palPalEntry.peFlags` de `CreatePalette`

Valor	Descripción
PC_EXPLICIT	Indica que el byte menos significativo de la entrada de la paleta lógica designa un índice de la paleta del sistema.
PC_NOCOLLAPSE	Indica que esta entrada de color será ubicada en una entrada de la paleta del sistema no usada. No será mapeada en otra entrada de color si una entrada duplicada ya existe en la paleta lógica. Si no queda ninguna entrada sin usar de la paleta del sistema, el color es mapeado normalmente como si ningún valor <code>peFlags</code> hubiera sido especificado. Otros colores pueden ser mapeados a esta entrada de color de la manera normal.
PC_RESERVED	Indica que esta entrada de color es usada para animación de paleta. Será ubicada en una entrada sin usar de la paleta del sistema y no será mapeada en otra entrada de color si un duplicado ya existe en la paleta lógica. Otros colores no pueden ser mapeados a esta entrada de color si son idénticos. Si no quedan entradas sin usar en la paleta del sistema, esta entrada de color no será ubicada en dicha paleta y por esta razón será inaccesible.

**GetBValue**      **Windows.Pas****Sintaxis**

```
GetBValue(
    rgb: DWORD           {especificador de color de 32 bits}
): Byte;                {devuelve la intensidad del azul}
```

**Descripción**

Esta función recupera el valor de la intensidad del color azul del especificador de color de 32 bits indicado.

**Parámetros**

*rgb*: El especificador de color de 32 bits cuya intensidad del color azul es recuperada.

**Valor que devuelve**

Si esta función tiene éxito, devuelve la intensidad del color azul del especificador de color de 32 bits. Este valor está en el rango de 0 a 255. Si la función falla, devuelve cero.

**Véase además**

*GetGValue*, *GetRValue*, *PaletteIndex*, *PaletteRGB*, *RGB*

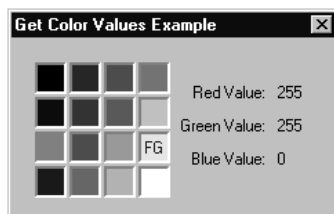
**Ejemplo****Listado 7-5: Mostrando la intensidad de los colores rojo, verde y azul**

```

procedure TForm1.ColorGrid1Change(Sender: TObject);
begin
    {Muestra la intensidad del rojo, verde y azul del color seleccionado}
    Label14.Caption := IntToStr(GetRValue(ColorGrid1.ForegroundColor));
    Label15.Caption := IntToStr(GetGValue(ColorGrid1.ForegroundColor));
    Label16.Caption := IntToStr(GetBValue(ColorGrid1.ForegroundColor));
end;

```

Figura 7-5:  
Los valores de  
intensidad de  
los colores  
básicos

**GetDIBColorTable****Windows.Pas****Sintaxis**

```

GetDIBColorTable(
    DC: HDC;           {manejador de contexto de dispositivo con DIB seleccionado}
    p2: UINT;          {índice de paleta de la primera entrada a recuperar}
    p3: UINT;          {cantidad de entradas de paleta a recuperar}
    var RGBQuadStructs {puntero a buffer que recibe un array de TRGBQuad}
): UINT;              {devuelve la cantidad de entradas de paleta recuperadas}

```

**Descripción**

Esta función recupera un rango específico de entradas de paleta de colores desde el mapa de bits independiente de dispositivo actualmente seleccionado en el contexto de dispositivo identificado por el parámetro *DC*. Estas entradas están en la forma de un *array* de registros *TRGBQuad*, almacenado en el *buffer* al que apunta el parámetro *RGBQuadStructs*. Esta función es útil sólo para DIBs que tienen un formato de color de 1, 4, u 8 bits por píxel, debido a que los DIBs con un formato de color superior a 8 bits por píxel no contienen una paleta de colores.

**Parámetros**

*DC*: Manejador de un contexto de dispositivo. El DIB desde el cual las entradas de la paleta de colores van a ser recuperados tiene que estar seleccionado en este contexto de dispositivo.

*p2*: Especifica el índice (basado en cero) de la primera entrada de la paleta a recuperar.

*p3*: Especifica la cantidad de entradas de paleta a recuperar.

*RGBQuadStructs*: Puntero a un *buffer* que recibe un *array* de registros de tipo *TRGBQuad* que describen los valores de los colores de las entradas de paleta del DIB especificado. Este *buffer* tiene que ser lo suficientemente grande para almacenar el número de registros *TRGBQuad* indicado en el parámetro *p3*. El registro *TRGBQuad* se define como:

```
TRGBQuad = packed record
    rgbBlue: Byte;           {intensidad del color azul}
    rgbGreen: Byte;          {intensidad del color verde}
    rgbRed: Byte;            {intensidad del color rojo}
    rgbReserved: Byte;       {valor reservado}
end;
```

Consulte la función *CreateDIBSection* para ver una descripción de este registro.

#### Valor que devuelve

Si la función tiene éxito, el *buffer RGBQuadStructs* es rellenado con un *array* de registros *TRGBQuad* y la función devuelve la cantidad de entradas de paleta recuperadas. Si la función falla, el *buffer RGBQuadStructs* será vaciado y la función devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreatePalette*, *CreateDIBSection*, *SelectObject*, *SetDIBColorTable*, *SetPaletteEntries*

#### Ejemplo

##### Listado 7-6: Recuperando una paleta de colores de un mapa de bits independiente del dispositivo

```
var
    Form1: TForm1;
    FormPalette: HPALETTE;    // almacena nuestra paleta lógica genérica

implementation

function TForm1.GetPalette: HPALETTE;
begin
    {Cuando al formulario se le pide la paleta de colores, devuelve la paleta
     creada por la aplicación}
    Result := FormPalette;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    ThePalette: PLogPalette;    // un registro de datos de una paleta lógica
begin
    {Reserva suficiente memoria para crear una paleta lógica de 256 colores}
```

```

GetMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));

{Inicializa los campos apropiados}
ThePalette^.palVersion := $300;
ThePalette^.palNumEntries := 256;

{Esta paleta es usada para transferir los colores del DIB seleccionado en el
contexto de dispositivo del formulario. Por eso, debido a que los colores de la
paleta estarán basados en la paleta de colores almacenada en el DIB
seleccionado, no necesitamos inicializar esta nueva paleta de colores}
FormPalette := CreatePalette(ThePalette^);

{Tenemos una paleta lógica de colores, borramos la memoria innecesaria}
FreeMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Borra nuestra paleta lógica genérica}
    DeleteObject(FormPalette);
end;

procedure TForm1.FileListBox1Click(Sender: TObject);
var
    InLoop, OutLoop: Integer;           // variables de control de bucles
    NumColors: Integer;                 // cantidad de colores en la
                                        // tabla de colores del DIB
    PalEntries: array[0..255] of TPaletteEntry; // almacena información sobre colores
                                        // en un formato de paleta lógica
    Colors: array[0..255] of TRGBQuad; // almacena las entradas de paleta
                                        // del DIB recuperado
    TheDC: HDC;                         // manejador contexto de dispositivo
    OldBrush, NewBrush: HBRUSH;         // manejadores de brochas usadas
                                        // para mostrar la paleta del DIB
begin
    {La clase TBitmap de Delphi encapsula un DIB. Por eso se puede cargar un DIB
    en el objeto Image1}
    Image1.Picture.Bitmap.LoadFromFile(FileListBox1.FileName);

    {El DIB cargado es automáticamente seleccionado en el contexto de dispositivo de
    Image1. Se recupera la paleta de colores del DIB}
    NumColors := GetDIBColorTable(Image1.Canvas.Handle, 0, 256, Colors);

    {Los campos del registro TRGBQuad están en orden inverso con respecto a los
    campos del registro TPaletteEntry. Necesitamos crear un array de registros
    TPaletteEntry para que la paleta del formulario pueda ser asignada a la del
    DIB. Por eso tenemos que convertir el tipo TRGBQuad al formato TPaletteEntry.}
    for InLoop := 0 to NumColors-1 do
        begin
            PalEntries[InLoop].peRed := Colors[InLoop].rgbRed;
            PalEntries[InLoop].peGreen := Colors[InLoop].rgbGreen;
            PalEntries[InLoop].peBlue := Colors[InLoop].rgbBlue;
            PalEntries[InLoop].peFlags := PC_NOCOLLAPSE;
        end;

```

```

{Asigna la paleta del formulario para que se corresponda con la del DIB}
SetPaletteEntries(FormPalette, 10, NumColors, PalEntries);

{Selecciona y activa esta nueva paleta}
TheDC := GetDC(Form1.Handle);
SelectPalette(TheDC, FormPalette, FALSE);
RealizePalette(TheDC);

{Inicializa el cuadro de lista para mostrar la información de la paleta}
ListBox1.Items.Clear;
ListBox1.Items.Add('Number of colors: ' + IntToStr(NumColors));

{Dibuja una rejilla con celdas coloreadas que representan la paleta del DIB
seleccionado. Debido a que sólo 236 colores pueden ser asignados a la paleta
del sistema, este código no mostrará los primeros y últimos 10 colores de la
paleta lógica}
for OutLoop := 0 to 15 do
  for InLoop := 0 to 15 do
    begin
      if ((OutLoop*16)+InLoop > NumColors-1) or ((OutLoop*16)+InLoop > 245) then
        {Si hay menos de 236 colores o se trata de los últimos 10 colores,
        muestra un cuadrado negro sólido}
        NewBrush := CreateSolidBrush(clBlack)
      else
        begin
          {Muestra los valores de los colores para este índice de paleta}
          ListBox1.Items.Add('Index: ' + IntToStr((OutLoop*16) + InLoop));
          ListBox1.Items.Add('    Red: ' + IntToStr(Colors[
            (OutLoop*16) + InLoop].rgbRed));
          ListBox1.Items.Add('  Green: ' + IntToStr(Colors[
            (OutLoop*16) + InLoop].rgbGreen));
          ListBox1.Items.Add('    Blue: ' + IntToStr(Colors[
            (OutLoop*16) + InLoop].rgbBlue));

          {Crea una brocha coloreada para este índice de paleta específico,
          saltándose los primeros 10 colores del sistema}
          NewBrush := CreateSolidBrush($01000000 or (OutLoop*16) + InLoop+10);
        end;

        {Selecciona la nueva brocha en el contexto de dispositivo}
        OldBrush := SelectObject(TheDC, NewBrush);

        {Dibuja un rectángulo usando el color específico del índice de la paleta}
        Rectangle(TheDC, (InLoop*15) + 15, (OutLoop*15) + 256,
          (InLoop*15) + 30, (OutLoop*15) + 271);

        {Elimina la brocha coloreada}
        SelectObject(TheDC, OldBrush);
        DeleteObject(NewBrush);
      end;
    end;

  {Libera el contexto de dispositivo}
  ReleaseDC(Form1.Handle, TheDC);
end;

```

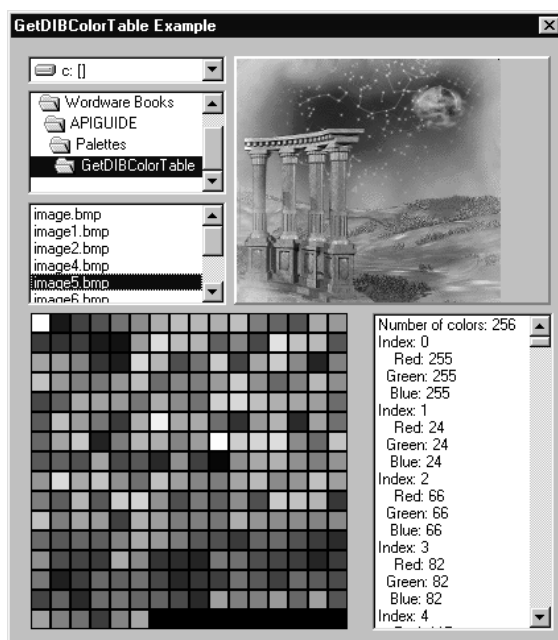


Figura 7-6:  
Mostrando la  
paleta de  
colores de un  
DIB

### GetEnhMetaFilePaletteEntries

### Windows.Pas

#### Sintaxis

```
GetEnhMetaFilePaletteEntries(
    p1: HENHMETAFILE;           {manejador de un metafichero mejorado}
    p2: UINT;                    {cantidad de entradas de paleta a recuperar}
    p3: Pointer                  {puntero a buffer que recibe las entradas de paleta}
): UINT;                        {devuelve la cantidad de entradas copiadas}
```

#### Descripción

Esta función recupera la cantidad especificada de entradas de paleta del metafichero mejorado indicado por el parámetro *p1*, si éste contiene una paleta. Las entradas recuperadas siempre comienzan en el índice 0 de la tabla de entradas de paleta del metafichero mejorado. Una vez que las entradas de paleta son recuperadas, el desarrollador puede usar las funciones *CreatePalette*, *SelectPalette* y *RealizePalette* antes de llamar a la función *PlayEnhMetaFile* para mostrar el metafichero mejorado en sus colores originales.

#### Parámetros

*p1*: Manejador de un metafichero mejorado cuyas entradas de paleta serán recuperadas.  
*p2*: Especifica la cantidad de entradas de la paleta a recuperar.

*p3*: Puntero a un *buffer* que recibe un *array* de registros de tipo *TPaletteEntry* que describen los valores de los colores de las entradas de paleta del metaarchivo. Este *buffer* tiene que ser lo suficientemente grande para almacenar el número de registros *TPaletteEntry* indicado por el parámetro *p2*. Si a este parámetro se le asigna **nil** y el metaarchivo mejorado especificado contiene una paleta de colores, la función devuelve la cantidad de entradas en la paleta del metaarchivo. El registro *TPaletteEntry* se define como:

*TPaletteEntry* = **packed record**

```

    peRed: Byte;           {intensidad del color rojo}
    peGreen: Byte;         {intensidad del color verde}
    peBlue: Byte;          {intensidad del color azul}
    peFlags: Byte;         {opciones de tratamiento de entradas de paleta}

```

**end;**

Consulte la función *CreatePalette* para ver una descripción de esta estructura de datos.

#### Valor que devuelve

Si la función tiene éxito y el metaarchivo mejorado especificado contiene una paleta, el *buffer* al que apunta el parámetro *p3* es rellenado con un *array* de registros de tipo *TPaletteEntry* y la función devuelve la cantidad de entradas de paleta recuperadas. Si la función tiene éxito y el metaarchivo no contiene una paleta, devuelve cero. Si la función falla, devuelve *GDI\_ERROR*.

#### Véase además

*CreatePalette*, *GetEnhMetaFile*, *PlayEnhMetaFile*, *RealizePalette*, *SelectPalette*

#### Ejemplo

##### Listado 7-7: Recuperando una paleta de colores de un metaarchivo mejorado

```

procedure TForm1.FileListBox1Click(Sender: TObject);
var
    TheMetafile: HENHMETAFILE;           // manejador de un metaarchivo mejorado
    MetafilePalette: PLogPalette;         // registro de paleta lógica
    MetafilePaletteHandle: HPALETTE;      // manejador de una paleta
    NumPaletteEntries: Integer;           // cantidad de entradas en la paleta
    iLoop: Integer;                       // variable de control de bucle
begin
    {Borra la última imagen en el panel de dibujo}
    PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);

    {Abre el metaarchivo seleccionado}
    TheMetafile := GetEnhMetaFile(PChar(FileListBox1.FileName));

    {Recupera el número de entradas en esta paleta de metaarchivo}
    NumPaletteEntries := GetEnhMetaFilePaletteEntries(TheMetafile, 0, nil);

    {Reserva memoria para crear una paleta lógica lo suficientemente grande para
    todas las entradas de paleta del metaarchivo}

```

```

GetMem(MetafilePalette, SizeOf(TLogPalette) +
      NumPaletteEntries * SizeOf(TPaletteEntry));

{Si hay entradas en el metaarchivo, las recupera}
if NumPaletteEntries>0 then
  GetEnhMetaFilePaletteEntries(TheMetafile, NumPaletteEntries,
    @(MetafilePalette^.palPalEntry));

{Inicializa la información apropiada de la paleta lógica}
MetafilePalette^.palVersion := $300;
MetafilePalette^.palNumEntries := NumPaletteEntries;

{Crea una paleta lógica basada en la paleta del metaarchivo mejorado}
MetafilePaletteHandle := CreatePalette(MetafilePalette^);

{Selecciona y activa esta paleta en el área de dibujo del formulario}
SelectPalette(Form1.Canvas.Handle, MetafilePaletteHandle, FALSE);
RealizePalette(Form1.Canvas.Handle);

{Ahora muestra el metaarchivo mejorado. Este metaarchivo se ejecutará usando
sus colores originales}
PlayEnhMetaFile(PaintBox1.Canvas.Handle, TheMetafile, PaintBox1.BoundsRect);

{Inicializa el cuadro de lista para mostrar las entradas de la paleta}
ListBox1.Items.Clear;
ListBox1.Items.Add('Palette Entries: ' + IntToStr(NumPaletteEntries));

{Muestra las entradas de paletas del metaarchivo mejorado}
for iLoop := 0 to NumPaletteEntries-1 do
begin
  ListBox1.Items.Add('Palette slot: ' + IntToStr(iLoop));
  ListBox1.Items.Add('      Red: ' + IntToStr(
    MetafilePalette^.palPalEntry[iLoop].peRed));
  ListBox1.Items.Add('      Green: ' + IntToStr(
    MetafilePalette^.palPalEntry[iLoop].peGreen));
  ListBox1.Items.Add('      Blue: ' + IntToStr(
    MetafilePalette^.palPalEntry[iLoop].peBlue));

  case MetafilePalette^.palPalEntry[iLoop].peFlags of
    PC_EXPLICIT: ListBox1.Items.Add('      Flags: PC_EXPLICIT');
    PC_NOCOLLAPSE: ListBox1.Items.Add('      Flags: PC_NOCOLLAPSE');
    PC_RESERVED: ListBox1.Items.Add('      Flags: PC_RESERVED');
    else ListBox1.Items.Add('      Flags: NULL');
  end;
end;

{El registro de datos de la paleta ya no es necesario, liberamos su memoria}
FreeMem(MetafilePalette, SizeOf(TLogPalette) +
      NumPaletteEntries * SizeOf(TPaletteEntry));

{El metaarchivo mejorado ya no es necesario, lo eliminamos}
DeleteEnhMetaFile(TheMetafile);
DeleteObject(MetafilePaletteHandle);

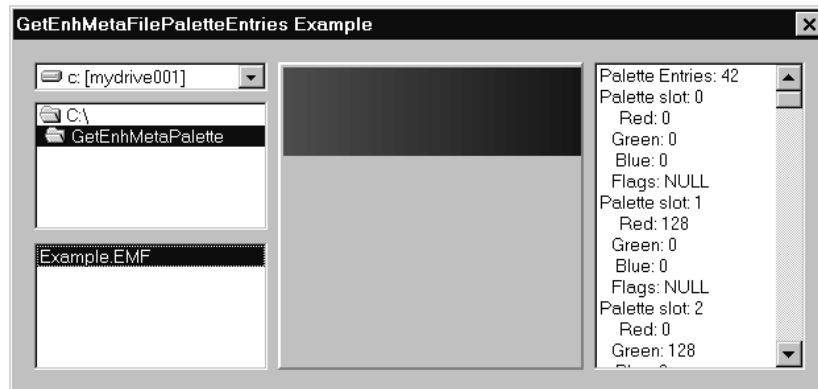
```

```

    {Borra la paleta lógica}
    DeleteObject(MetafilePaletteHandle);
end;

```

Figura 7-7:  
Mostrando un  
metafichero  
mejorado en  
sus colores  
originales



## GetGValue Windows.Pas

### Sintaxis

```

GetGValue(
    rgb: DWORD           {un especificador de color de 32 bits}
): Byte;               {devuelve la intensidad del verde}

```

### Descripción

Esta función recupera la intensidad del color verde del especificador de color de 32 bits indicado.

### Parámetros

*rgb*: El especificador de color de 32 bits cuya intensidad del color verde será recuperada.

### Valor que devuelve

Si esta función tiene éxito, devuelve la intensidad del color verde del especificador de color de 32 bits. Este valor está en el rango de 0 a 255. Si la función falla, devuelve cero.

### Véase además

*GetBValue*, *GetRValue*, *PaletteIndex*, *PaletteRGB*, *RGB*

### Ejemplo

Vea el Listado 7-5 bajo *GetBValue*.

**GetNearestColor****Windows.Pas****Sintaxis**

```

GetNearestColor(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: COLORREF             {especificador de color de 32 bits}
): COLORREF;               {devuelve un especificador de color de 32 bits}

```

**Descripción**

Esta función devuelve el especificador de color de 32 bits de la paleta de colores del sistema asociada al contexto de dispositivo con el color más cercano al especificador de color indicado por el parámetro *p2*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo en cuya paleta del sistema asociada se buscará el color más cercano al requerido.

*p2*: Especificador de color de 32 bits que representa el color requerido.

**Valor que devuelve**

Si la función tiene éxito, devuelve el especificador del color de la paleta del sistema más cercano al color solicitado; en caso contrario, devuelve *CLR\_INVALID*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetNearestPaletteIndex*, *RGB*

**Ejemplo****Listado 7-8: Hallando el color más cercano**

```

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    {Mostrar los atributos del color solicitado}
    Label1.Caption := 'Red: ' + IntToStr(ScrollBar1.Position);
    Label2.Caption := 'Green: ' + IntToStr(ScrollBar2.Position);
    Label3.Caption := 'Blue: ' + IntToStr(ScrollBar3.Position);

    {Colorear la primera forma con el color solicitado}
    Shape1.Brush.Color := RGB(ScrollBar1.Position,
                               ScrollBar2.Position,
                               ScrollBar3.Position);

    {Halla el color puro más cercano al solicitado en la paleta del sistema}
    Shape2.Brush.Color := GetNearestColor(Canvas.Handle, Shape1.Brush.Color);
end;

```

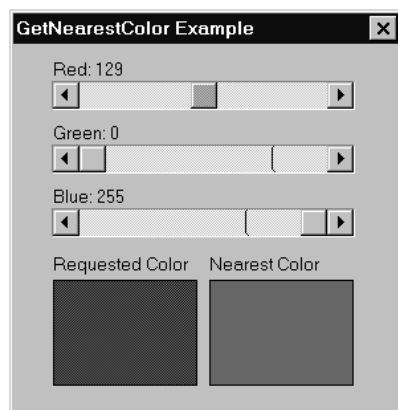


Figura 7-8:  
Mostrando el  
color más  
cercano

## GetNearestPaletteIndex

## Windows.Pas

### Sintaxis

```
GetNearestPaletteIndex(
    p1: HPALETTE;           {manejador de paleta lógica}
    p2: COLORREF             {valor de color RGB explícito de 32 bits}
): UINT;                   {devuelve un índice de base cero en la paleta lógica}
```

### Descripción

Esta función devuelve el índice de la entrada de color más cercano al color solicitado (especificado en el parámetro *p2*) de la paleta lógica indicada en el parámetro *p1*. Observe que si la paleta lógica contiene entradas de paleta que contienen el valor *PC\_EXPLICIT* en el campo *peFlags*, el valor devuelto es indefinido.

### Parámetros

*p1*: Manejador de la paleta lógica en la que se busca el color que más se acerca al color solicitado.

*p2*: Un valor de color RGB explícito de 32 bits que representa el color solicitado.

### Valor que devuelve

Si la función tiene éxito, devuelve el índice (de base cero) en la paleta lógica del color más cercano al solicitado. Si la función falla, devuelve *CLR\_INVALID*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*GetNearestColor*, *GetPaletteEntries*, *GetSystemPaletteEntries*, *PaletteRGB*, *RGB*

*Ejemplo***Listado 7-9: Hallando el índice de la paleta para el color más cercano al requerido**

```
{¡OJO! Delphi importa incorrectamente la función GetSystemPaletteEntries.
Aquí está la declaración correcta, que ofrece acceso a toda la funcionalidad
accesible con esta función del API}
function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: UINT;
                                PaletteEntries: Pointer): UINT; stdcall;

var
    Form1: TForm1;
    SysPalette: array[0..255] of TPaletteEntry; // almacena una copia de las
                                                // entradas de la paleta del sistema
    FormPalette: HPALETTE;                      // manejador de la paleta lógica

implementation

{Enlace de la función GetSystemPaletteEntries}
function GetSystemPaletteEntries; external gdi32 name 'GetSystemPaletteEntries';

function TForm1.GetPalette: HPalette;
begin
    {Cuando se solicita la paleta al formulario, éste devuelve la paleta creada
    por la aplicación}
    Result := FormPalette;
end;

procedure TForm1.SetThePalette;
var
    ThePalette: PLogPalette; // registro de paleta lógica
begin
    {Reserva memoria para una paleta de 256 entradas de colores}
    GetMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));

    {Recupera las entradas de paleta de la paleta del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256,
                           @(ThePalette^.palPalEntry));

    {Inicializa la información apropiada de la paleta lógica}
    ThePalette^.palVersion := $300;
    ThePalette^.palNumEntries := 256;

    {Crea un duplicado exacto de la actual paleta del sistema}
    FormPalette := CreatePalette(ThePalette^);

    {No necesitamos ya la memoria del registro de la paleta lógica y la liberamos}
    FreeMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    iLoop: Integer; // contador de bucle
begin
```

```

{Crea un duplicado exacto de la paleta de sistema actual}
SetThePalette;

{Recupera las entradas de colores de la paleta del sistema}
GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256, @SysPalette);

{Muestra la información de la entrada en la paleta}
for iLoop := 0 to 255 do
begin
  ListBox1.Items.Add('Palette slot: ' + IntToStr(iLoop));
  ListBox1.Items.Add('   Red: ' + IntToStr(SysPalette[iLoop].peRed));
  ListBox1.Items.Add('  Green: ' + IntToStr(SysPalette[iLoop].peGreen));
  ListBox1.Items.Add('   Blue: ' + IntToStr(SysPalette[iLoop].peBlue));

  case SysPalette[iLoop].peFlags of
    PC_EXPLICIT:  ListBox1.Items.Add('   Flags: PC_EXPLICIT');
    PC_NOCOLLAPSE: ListBox1.Items.Add('   Flags: PC_NOCOLLAPSE');
    PC_RESERVED:  ListBox1.Items.Add('   Flags: PC_RESERVED');
    else ListBox1.Items.Add('   Flags: NULL');
  end;
end;
end;

procedure TForm1.FormPaint(Sender: TObject);
var
  iOutLoop, iInLoop: Integer;
begin
  {Selecciona y activa el duplicado de la paleta del sistema en el contexto de
  dispositivo del formulario}
  SelectPalette(Canvas.Handle, FormPalette, FALSE);
  RealizePalette(Canvas.Handle);

  {Dibuja una rejilla de rectángulos para mostrar la paleta de colores}
  for iOutLoop := 0 to 15 do
    for iInLoop := 0 to 15 do
      begin
        Canvas.Brush.Color := PaletteIndex((iOutLoop*16) + iInLoop);
        Canvas.Rectangle((iInLoop*20) + 15, (iOutLoop*20) + 144,
          (iInLoop*20) + 35, (iOutLoop*20) + 164);
      end;
    end;
  end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  {No necesitamos ya la paleta lógica creada y la eliminamos}
  DeleteObject(FormPalette);
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
  {Muestra los atributos del color solicitado}
  Label1.Caption := 'Red: ' + IntToStr(ScrollBar1.Position);
  Label2.Caption := 'Green: ' + IntToStr(ScrollBar2.Position);
  Label3.Caption := 'Blue: ' + IntToStr(ScrollBar3.Position);

```

```

{Halla el índice del color más cercano}
Label8.Caption := IntToStr(GetNearestPaletteIndex(FormPalette, RGB(
                                                                    ScrollBar1.Position,
                                                                    ScrollBar2.Position,
                                                                    ScrollBar3.Position)));

{Muestra la entrada del color más cercano}
ListBox1.TopIndex := ListBox1.Items.IndexOf('Palette slot: ' + Label8.Caption);
end;

```

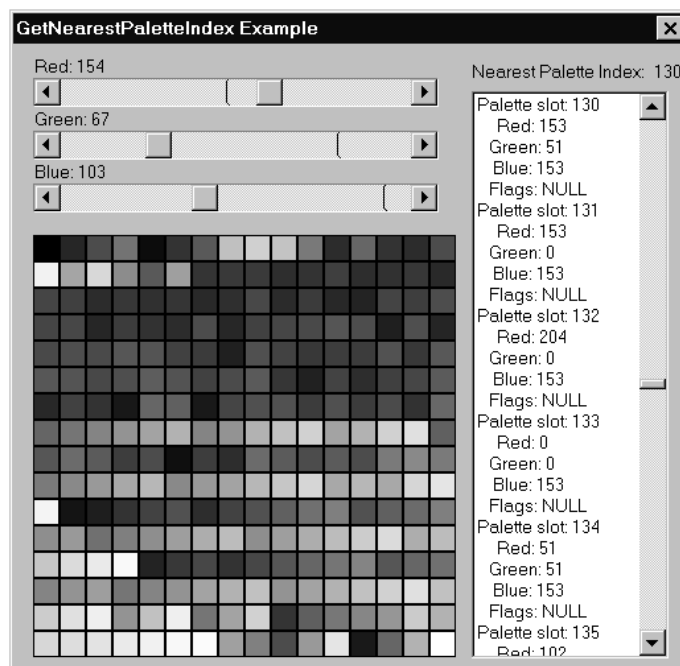


Figura 7-9:  
El índice de la  
paleta para el  
color más  
cercano

## GetPaletteEntries

## Windows.Pas

### Sintaxis

```

GetPaletteEntries(
    Palette: HPALETTE;           {manejador de paleta lógica}
    StartIndex: UINT;             {índice de inicio de la paleta}
    NumEntries: UINT;             {cantidad de entradas de paleta a recuperar}
    PaletteEntries: Pointer       {puntero a un array de registros TPaletteEntry}
): UINT;                         {devuelve la cantidad de entradas recuperadas}

```

**Descripción**

Esta función recupera un rango de entradas de paleta de la paleta lógica especificada. Estas entradas de paleta son devueltas en forma de un *array* de registros *TPaletteEntry* a través del *buffer* al que apunta el parámetro *PaletteEntries*.

**Parámetros**

*Palette*: Manejador de la paleta lógica del que las entradas de paleta son recuperadas.

*StartIndex*: El índice (de base cero) de la primera entrada a recuperar de la paleta lógica.

*NumEntries*: Especifica la cantidad de entradas a recuperar. Si la paleta lógica contiene menos entradas que las indicadas por este parámetro, el resto de las entradas en el *buffer* al que apunta el parámetro *PaletteEntries* no son alteradas.

*PaletteEntries*: Puntero a un *buffer* que recibe un *array* de registros *TPaletteEntry* que describen los valores de los colores recuperados de las entradas de la paleta lógica especificada. Este *buffer* tiene que ser lo suficientemente grande para almacenar la cantidad de registros *TPaletteEntry* indicados por el parámetro *NumEntries*. Si a este parámetro se le asigna **nil**, la función devuelve la cantidad de entradas en la paleta lógica especificada. El registro *TPaletteEntry* se define como:

*TPaletteEntry* = **packed record**

peRed: Byte;	{intensidad del color rojo}
peGreen: Byte;	{intensidad del color verde}
peBlue: Byte;	{intensidad del color azul}
peFlags: Byte;	{opciones de tratamiento de entradas de paleta}

**end;**

Consulte la función *CreatePalette* para ver una descripción de este registro.

**Valor que devuelve**

Si la función tiene éxito, el *buffer* al que apunta el parámetro *PaletteEntries* es rellenado con un *array* de registros *TPaletteEntry* y la función devuelve la cantidad de entradas de paleta recuperadas desde la paleta lógica. Si la función tiene éxito y al parámetro *PaletteEntries* se le asigna **nil**, la función devuelve la cantidad de entradas en la paleta lógica especificada. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreatePalette*, *GetSystemPaletteEntries*, *RealizePalette*, *SelectPalette*, *SetPaletteEntries*

**Ejemplo**

Vea el Listado 7-4 bajo *CreatePalette* o el Listado 7-3 bajo *CreateHalfTonePalette*.

**GetRValue**      **Windows.Pas****Sintaxis**

```
GetRValue(
    rgb: DWORD           {especificador de color de 32 bits}
): Byte;                {devuelve la intensidad del rojo}
```

**Descripción**

Esta función recupera el valor de la intensidad del color rojo del especificador de color de 32 bits indicado.

**Parámetros**

*rgb*: El especificador de color de 32 bits cuya intensidad del color rojo es recuperada.

**Valor que devuelve**

Si esta función tiene éxito, devuelve la intensidad del color rojo del especificador de color de 32 bits. Este valor está en el rango de 0 a 255. Si la función falla, devuelve cero.

**Véase además**

*GetBValue*, *GetGValue*, *PaletteIndex*, *PaletteRGB*, *RGB*

**Ejemplo**

Vea el Listado 7-5 bajo *GetBValue*.

**GetSysColor**      **Windows.Pas****Sintaxis**

```
GetSysColor(
    nIndex: Integer       {indicador del elemento de visualización}
): DWORD;                {devuelve un valor de color RGB}
```

**Descripción**

Esta función recupera el valor RGB del color asociado al elemento de visualización especificado. El indicador de elemento de visualización *nIndex* identifica a una de las diferentes partes de una ventana, cada una de las cuales es asociada con uno de los 20 colores estáticos definidos por el sistema.

**Parámetros**

*nIndex*: Especifica un elemento de visualización cuyo valor de color RGB será recuperado. Este parámetro puede tomar un valor de la Tabla 7-4.

*Valor que devuelve*

Si la función tiene éxito, devuelve el valor del color explícito de 32 bits asociado al elemento de visualización especificado. Este color se corresponderá con uno de los 20 colores estáticos del sistema. Si la función falla, devuelve cero.

*Véase además*

*GetBValue, GetGValue, GetRValue, GetSystemPaletteEntries, PaletteRGB, RGB, SetSysColors*

*Ejemplo***Listado 7-10: Recuperando y estableciendo el sistema de colores del elemento de visualización**

```
{Array de constantes con todos los elementos de visualización accesibles}
const
    DisplayElements: array[0..24] of Integer = (COLOR_3DDKSHADOW,COLOR_3DLIGHT,
        COLOR_ACTIVEBORDER,COLOR_ACTIVECAPTION,
        COLOR_APPWORKSPACE,COLOR_BACKGROUND,
        COLOR_BTNFACE,COLOR_BTNHIGHLIGHT,
        COLOR_BTNSHADOW,COLOR_BTNTEXT,
        COLOR_CAPTIONTEXT,COLOR_GRAYTEXT,
        COLOR_HIGHLIGHT,COLOR_HIGHLIGHTTEXT,
        COLOR_INACTIVEBORDER,COLOR_INACTIVECAPTION,
        COLOR_INACTIVECAPTIONTEXT,COLOR_INFOBK,
        COLOR_INFOTEXT,COLOR_MENU,COLOR_MENUTEXT,
        COLOR_SCROLLBAR,COLOR_WINDOW,
        COLOR_WINDOWFRAME,COLOR_WINDOWTEXT);

implementation

procedure TForm1.ComboBox1Change(Sender: TObject);
var
    DisplayColor: TColor; // almacena el color de sistema del elemento
begin
    {Recupera el color del elemento de visualización seleccionado}
    DisplayColor := GetSysColor(DisplayElements[ComboBox1.ItemIndex]);

    {Ajusta la posición de la barra de desplazamiento}
    ScrollBar1.Position := GetRValue(DisplayColor);
    ScrollBar2.Position := GetGValue(DisplayColor);
    ScrollBar3.Position := GetBValue(DisplayColor);
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    {Muestra el color seleccionado en la figura}
    Shape1.Brush.Color := RGB(ScrollBar1.Position,
        ScrollBar2.Position,
        ScrollBar3.Position);

    {Muestra los valores de los componentes del color}
```

```

Label1.Caption := 'Red: ' + IntToStr(ScrollBar1.Position);
Label2.Caption := 'Green: ' + IntToStr(ScrollBar2.Position);
Label3.Caption := 'Blue: ' + IntToStr(ScrollBar3.Position);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Elements: array[0..0] of Integer; // almacena el elemento a modificar
  NewColor: array[0..0] of TColor; // almacena el nuevo color para el elemento
begin
  {Identifica el elemento de visualización a cambiar}
  Elements[0] := DisplayElements[ComboBox1.ItemIndex];

  {Identifica el nuevo color para el elemento seleccionado}
  NewColor[0] := RGB(ScrollBar1.Position, ScrollBar2.Position,
                    ScrollBar3.Position);

  {Cambia el color del elemento de visualización. La pantalla se redibujará.}
  SetSysColors(1, Elements, NewColor);
end;

```

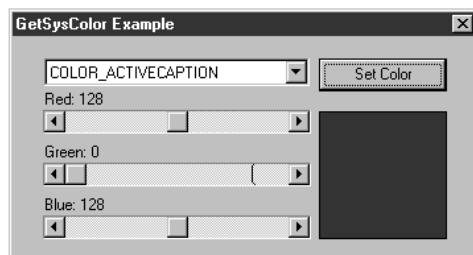


Figura 7-10:  
El nuevo color  
del sistema

Tabla 7-4: Valores del parámetro nIndex de GetSysColor

Valor	Descripción
COLOR_3DDKSHADOW	El color de la sombra oscura para elementos de visualización tridimensionales.
COLOR_3DLIGHT	El color del borde iluminado resaltado para elementos de visualización tridimensionales.
COLOR_ACTIVEBORDER	El color del borde de la ventana activa.
COLOR_ACTIVECAPTION	El color de la barra de título de la ventana activa.
COLOR_APPWORKSPACE	El color de fondo usado en la interfaz de aplicaciones multidocumentos (MDI).
COLOR_BACKGROUND	El color del escritorio.
COLOR_BTNFACE	El color de la superficie de los botones.
COLOR_BTNHIGHLIGHT	El color de los botones resaltados.
COLOR_BTNSHADOW	El color del borde sombreado de los botones.
COLOR_BTNTEXT	El color del texto de los botones.

Valor	Descripción
COLOR_CAPTIONTEXT	El color del texto usado en la barra de título, el cuadro de redimensionamiento y las cajas de flecha de las barras de desplazamiento.
COLOR_GRAYTEXT	El color del texto deshabilitado. Será puesto a cero si el controlador de vídeo no puede soportar gris sólido.
COLOR_HIGHLIGHT	El color usado para los elementos seleccionados en un control.
COLOR_HIGHLIGHTTEXT	El color usado para el texto de los elementos seleccionados en un control.
COLOR_INACTIVEBORDER	El color del borde de una ventana inactiva.
COLOR_INACTIVECAPTION	El color de la barra de título de una ventana inactiva.
COLOR_INACTIVECAPTIONTEXT	El color del texto en una barra de título inactiva.
COLOR_INFOBK	El color de fondo para los controles de las indicaciones.
COLOR_INFOTEXT	El color del texto para los controles de las indicaciones.
COLOR_MENU	El color de fondo de los menús.
COLOR_MENUTEXT	El color del texto de los menús.
COLOR_SCROLLBAR	El color del área “gris” de la barra de desplazamiento.
COLOR_WINDOW	El color de fondo de una ventana.
COLOR_WINDOWFRAME	El color del marco de una ventana.
COLOR_WINDOWTEXT	El color del texto de una ventana.

**GetSystemPaletteEntries****Windows.Pas****Sintaxis**

```

GetSystemPaletteEntries(
    DC: HDC;                {manejador de contexto de dispositivo}
    StartIndex: UINT;        {índice de inicio de la paleta}
    NumEntries: UINT;        {cantidad de entradas de la paleta a recuperar}
    PaletteEntries: Pointer  {puntero a un array de registros TPaletteEntry}
): UINT;                   {devuelve la cantidad de entradas recuperadas}

```

**Descripción**

Esta función recupera el rango especificado de entradas de paleta de la paleta del sistema asociada con el contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Manejador de un contexto de dispositivo. Las entradas de la paleta de colores son recuperadas de la paleta del sistema asociada con este contexto de dispositivo.

*StartIndex*: El índice basado en cero de la primera entrada a recuperar de la paleta del sistema.

*NumEntries*: Especifica la cantidad de entradas a recuperar.

*PaletteEntries*: Puntero a un *buffer* que recibe un *array* de registros *TPaletteEntry* que describen los valores de los colores recuperados desde las entradas de la paleta del sistema. Este *buffer* tiene que ser lo suficientemente grande para almacenar la cantidad de registros *TPaletteEntry* indicada por el parámetro *NumEntries*. Si a este parámetro se le asigna **nil**, la función devuelve la cantidad de entradas de paleta en la paleta del sistema. El registro *TPaletteEntry* se define como:

*TPaletteEntry* = **packed record**

peRed: Byte;	{intensidad del color rojo}
peGreen: Byte;	{intensidad del color verde}
peBlue: Byte;	{intensidad del color azul}
peFlags: Byte;	{opción de tratamiento de entrada de paleta}

**end;**

Consulte la función *CreatePalette* para ver una descripción de este registro.

#### Valor que devuelve

Si la función tiene éxito, el *buffer* al que apunta el parámetro *PaletteEntries* es rellenado con un *array* de registros *TPaletteEntry* y la función devuelve la cantidad de de entradas de paleta recuperadas de la paleta del sistema. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetPaletteEntries*, *GetSystemPaletteUse*, *RealizePalette*, *SelectPalette*, *SetSystemPaletteUse*

#### Ejemplo

##### Listado 7-II: Recuperando entradas de la paleta del sistema

```
{iOJO! Delphi importa incorrectamente la función GetSystemPaletteEntries.
Aquí está la declaración correcta, que ofrece toda la funcionalidad accesible
con esta función del API}
function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: UINT;
                                PaletteEntries: Pointer): UINT; stdcall;

var
    Form1: TForm1;
    SysPalette: array[0..255] of TPaletteEntry; // almacena la paleta del sistema
                                                    // recuperada
    FormPalette: HPALETTE;                       // manejador de paleta lógica

implementation
```

```

{Enlace de la función GetSystemPaletteEntries}
function GetSystemPaletteEntries; external gdi32 name 'GetSystemPaletteEntries';

function TForm1.GetPalette: HPALETTE;
begin
    {Cuando al formulario se le pide su paleta, éste devuelve la paleta lógica
    creada por la aplicación}
    Result := FormPalette;
end;

procedure TForm1.SetThePalette;
var
    ThePalette: PLogPalette; // un registro de descripción de paleta lógica
    iLoop: Integer;          // contador de bucle
begin
    {Reserva suficiente memoria para una paleta de 256 colores}
    GetMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));

    {Recupera todas las entradas de la paleta del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256,
        @(ThePalette^.palPalEntry));

    {Inicializa la información de la paleta lógica}
    ThePalette^.palVersion := $300;
    ThePalette^.palNumEntries := 256;

    {Crea una paleta lógica que es un duplicado exacto de la paleta del sistema}
    FormPalette := CreatePalette(ThePalette^);

    {No necesitamos ya la memoria de descripción de la paleta lógica y la liberamos}
    FreeMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    iLoop: Integer; // contador de bucle
begin
    {Crea la paleta lógica}
    SetThePalette;

    {Recupera el uso de la paleta del sistema actual...}
    GetSystemPaletteUse(Form1.Canvas.Handle);

    {...y la muestra}
    ListBox1.Items.Add('System Palette State:');
    case GetSystemPaletteUse(Form1.Canvas.Handle) of
        SYSPAL_NOSTATIC: ListBox1.Items.Add(' No static colors');
        SYSPAL_STATIC:   ListBox1.Items.Add(' Static colors');
        SYSPAL_ERROR:    ListBox1.Items.Add(' No color palette');
    end;

    {Recupera las entradas de la actual paleta del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 256, @SysPalette);

```

```

{Muestra los atributos de color para las 256 entradas en la paleta del sistema}
for iLoop := 0 to 255 do
begin
  ListBox1.Items.Add('Palette slot: ' + IntToStr(iLoop));
  ListBox1.Items.Add('    Red: ' + IntToStr(SysPalette[iLoop].peRed));
  ListBox1.Items.Add('  Green: ' + IntToStr(SysPalette[iLoop].peGreen));
  ListBox1.Items.Add('    Blue: ' + IntToStr(SysPalette[iLoop].peBlue));

  case SysPalette[iLoop].peFlags of
    PC_EXPLICIT:  ListBox1.Items.Add('  Flags: PC_EXPLICIT');
    PC_NOCOLLAPSE: ListBox1.Items.Add('  Flags: PC_NOCOLLAPSE');
    PC_RESERVED:  ListBox1.Items.Add('  Flags: PC_RESERVED');
  else ListBox1.Items.Add('  Flags: NULL');
  end;
end;
end;

```

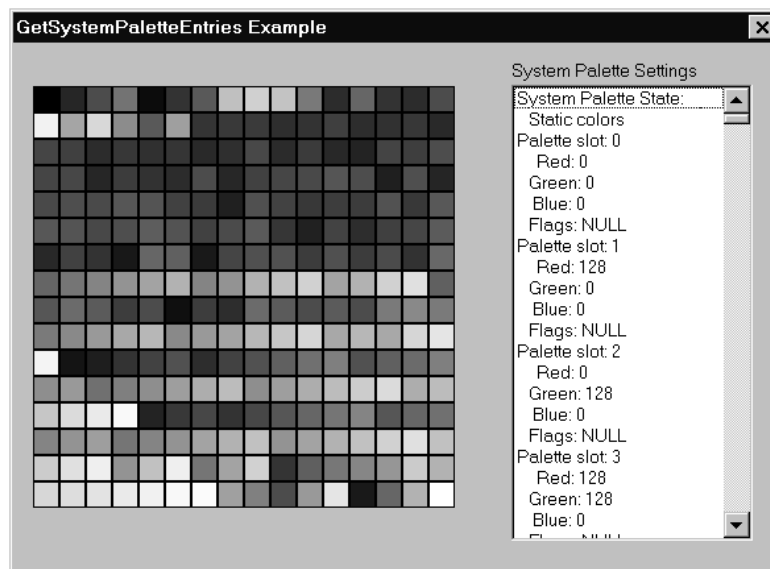


Figura 7-11:  
La paleta  
activa del  
sistema

```

procedure TForm1.FormPaint(Sender: TObject);
var
  iOutLoop, iInLoop: Integer; // variables de control de bucle
begin
  {Selecciona y activa la paleta lógica en el contexto de dispositivo del
  formulario}
  SelectPalette(Canvas.Handle, FormPalette, FALSE);
  RealizePalette(Canvas.Handle);

  {Dibuja una serie de rectángulos para mostrar la paleta del sistema}
  for iOutLoop := 0 to 15 do
    for iInLoop := 0 to 15 do
      begin
        Canvas.Brush.Color := PaletteIndex((iOutLoop*16) + iInLoop);

```

```

        Canvas.Rectangle((iInLoop*20) + 15, (iOutLoop*20) + 32,
                        (iInLoop*20) + 35, (iOutLoop*20) + 52);
    end;
end;

```

**GetSystemPaletteUse****Windows.Pas****Sintaxis**

```

GetSystemPaletteUse(
    DC: HDC                {manejador de contexto de dispositivo}
): UINT;                  {devuelve indicador del estado de la paleta del sistema}

```

**Descripción**

Esta función recupera un valor que indica el estado de utilización de los colores estáticos de la paleta del sistema asociada con el contexto de dispositivo asociado.

**Parámetros**

*DC*: Manejador al contexto de dispositivo cuyo estado de la paleta del sistema será recuperado. Debe ser un contexto de dispositivo que soporte paletas de colores.

**Valor que devuelve**

Si la función tiene éxito, devuelve el estado actual de la paleta del sistema asociada con el contexto de dispositivo asociado y puede ser un valor de la Tabla 7-5. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetSystemPaletteEntries*, *RealizePalette*, *SelectPalette*, *SetSystemPaletteUse*

**Ejemplo**

Vea el Listado 7-11 bajo *GetSystemPaletteEntries*.

**Tabla 7-5: Valores que devuelve GetSystemPaletteUse**

Valor	Descripción
SYSPAL_NOSTATIC	La paleta del sistema contiene sólo dos colores estáticos: blanco y negro.
SYSPAL_STATIC	La paleta del sistema contiene los colores estáticos por defecto. Estos colores no cambiarán cuando la paleta sea realizada. Este es el estado por defecto de la paleta del sistema.
SYSPAL_ERROR	El contexto de dispositivo especificado no es válido o no soporta paletas de colores.

**PaletteIndex**      **Windows.Pas****Sintaxis**

```

PaletteIndex(
    i: Word                                {índice de entrada de paleta lógica}
): COLORREF;                             {devuelve un especificador de color de 32 bits}

```

**Descripción**

Esta función convierte un índice de entrada de paleta explícito en un especificador de color de 32 bits. Este valor puede ser usado en funciones que esperan un especificador de color para indicar a las funciones el uso del color hallado en el índice de paleta lógica especificado. Esta función es equivalente a usar el operador booleano **or** con un índice de paleta explícito y el valor \$01000000 para especificar un índice de paleta lógica.

**Parámetros**

*i*: El índice de la entrada deseada en la paleta lógica.

**Valor que devuelve**

Si la función tiene éxito, devuelve un especificador de color de 32 bits correspondiente a un índice de la paleta lógica actualmente realizada; en caso contrario, devuelve cero.

**Véase además**

*CreatePalette, PaletteRGB, RGB*

**Ejemplo**

Vea el Listado 7-11 bajo *GetSystemPaletteEntries*.

**PaletteRGB**      **Windows.Pas****Sintaxis**

```

PaletteRGB(
    r: Byte;                               {intensidad del color rojo}
    g: Byte;                               {intensidad del color verde}
    b: Byte;                               {intensidad del color azul}
): COLORREF;                             {devuelve un especificador de color de 32 bits}

```

**Descripción**

Esta función devuelve un especificador de color de 32 bits que contiene el color más cercano al color solicitado de la paleta lógica actualmente realizada. Este número contiene un \$02 en el byte más significativo y los valores del rojo, el verde y el azul en los 3 bytes menos significativos de la entrada del color más cercano en la paleta lógica. Esta función es equivalente a la función *GetNearestColor*.

**Parámetros**

*r*: Especifica la intensidad del color rojo. Tiene que ser un valor entre 0 y 255.

*g*: Especifica la intensidad del color verde. Tiene que ser un valor entre 0 y 255.

*b*: Especifica la intensidad del color azul. Tiene que ser un valor entre 0 y 255.

**Valor que devuelve**

Si la función tiene éxito, devuelve un especificador de color de 32 bits para el color más cercano de la paleta lógica actualmente realizada; en caso contrario, devuelve cero.

**Véase además**

*CreatePalette*, *PaletteIndex*, *RGB*

**Ejemplo****Listado 7-12: Hallando el color más cercano**

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    {Colorea la figura con el color especificado difuminado}
    Shape1.Brush.Color := RGB(ScrollBar1.Position, ScrollBar2.Position,
                              ScrollBar3.Position);

    {Colorea la figura con el color en la paleta actual más cercano al solicitado}
    Shape2.Brush.Color := PaletteRGB(ScrollBar1.Position, ScrollBar2.Position,
                                      ScrollBar3.Position);
end;
```

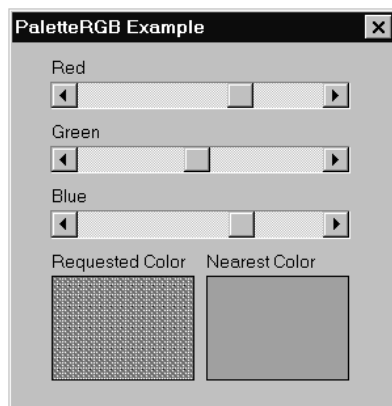


Figura 7-12:  
Busca un color  
correspondiente

**RealizePalette****Windows.Pas****Sintaxis**

```
RealizePalette(
    DC: HDC                                {manejador de contexto de dispositivo}
```

); UINT; {devuelve la cantidad de entradas de paleta realizadas}

### Descripción

Esta función mapea las entradas de la paleta lógica asociada con el contexto de dispositivo especificado en la paleta del sistema del dispositivo asociado con el contexto de dispositivo. Si el parámetro *DC* identifica un contexto de dispositivo en memoria, la paleta lógica es mapeada en la paleta de colores del mapa de bits seleccionado en el contexto de dispositivo. Si el parámetro *DC* identifica un contexto de dispositivo de visualización, la paleta lógica es mapeada en la paleta física del dispositivo. Una paleta lógica tiene que ser asociada con el contexto de dispositivo usando la función *SelectPalette* antes de llamar a *RealizePalette*.

El proceso de hacer corresponder los colores que tiene lugar cuando *RealizePalette* es llamada depende de si la paleta del contexto de dispositivo es una paleta de primer plano o de fondo. El estado de la paleta es determinado cuando la función *SelectPalette* es llamada. Si la paleta es una paleta de fondo, las entradas de la paleta lógica son mapeadas a las entradas más cercanas de la paleta del sistema y ninguna entrada de la paleta del sistema es reemplazada. Este es el comportamiento por defecto para las aplicaciones que se ejecutan en segundo plano. Si la paleta es una paleta de primer plano, las entradas de la paleta lógica son aún mapeadas a las entradas más cercanas en la paleta del sistema. Si una correspondencia exacta es hallada, esa entrada de la paleta lógica es mapeada a la correspondiente entrada en la paleta del sistema. Sin embargo, si no se encuentra una correspondencia exacta, a la primera entrada sin usar en la paleta del sistema se le asignan los valores de los colores de la entrada en la paleta lógica. Este comportamiento puede verse afectado por los valores del campo *peFlags* de la entrada de la paleta. Consulte la función *CreatePalette* para más detalles.

### Parámetros

*DC*: Manejador de un contexto de dispositivo cuya paleta lógica asociada es mapeada a la paleta del sistema del dispositivo que el contexto de dispositivo representa.

### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de entradas de la paleta lógica que fueron mapeadas a la paleta del sistema del contexto de dispositivo especificado. Si la función falla, devuelve *GDI\_ERROR*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*CreatePalette*, *SelectPalette*

### Ejemplo

Vea el Listado 7-4 bajo *CreatePalette*, y otros muchos ejemplos a lo largo de este capítulo.

**ResizePalette****Windows.Pas****Sintaxis**

```

ResizePalette(
  p1: HPALETTE;           {manejador de la paleta lógica que será redimensionada}
  p2: UINT                 {nuevo número de entradas en la paleta lógica}
): BOOL;                  {devuelve TRUE o FALSE}

```

**Descripción**

Esta función incrementa o decrementa la cantidad de entradas de paleta en la paleta lógica especificada. Si la paleta lógica es incrementada en tamaño, las nuevas entradas de la paleta son inicializadas a negro y a su campo *peFlags* se les asigna cero.

**Parámetros**

*p1*: Manejador de la paleta lógica cuya cantidad de entradas será modificada.

*p2*: La nueva cantidad de entradas que la paleta lógica contendrá.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreatePalette, RealizePalette, SelectPalette*

**Ejemplo****Listado 7-13: Redimensionando una paleta**

```

{¡OJO! Delphi importa incorrectamente la función GetSystemPaletteEntries.
Aquí está la declaración correcta, que brinda toda la funcionalidad accesible
con esta función del API}
function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: UINT;
                                PaletteEntries: Pointer): UINT; stdcall;

var
  Form1: TForm1;
  FormPalette: HPALETTE;           // manejador de paleta lógica

implementation

{Enlace de la función GetSystemPaletteEntries}
function GetSystemPaletteEntries; external gdi32 name 'GetSystemPaletteEntries';
function TForm1.GetPalette: HPALETTE;
begin
  {Cuando se pide al formulario su paleta, éste devuelve la nueva paleta lógica}
  Result := FormPalette;
end;

```

```

procedure TForm1.FormCreate(Sender: TObject);
var
    ThePalette: PLogPalette; // registro de definición de paleta lógica
    iLoop: Integer;          // contador de bucle
begin
    {Obtiene memoria para almacenar los colores de las primeras 10 entradas de
    la paleta además de las 22 nuestras. Esta memoria es temporal y no será
    necesaria una vez que se ha creado la paleta}
    GetMem(ThePalette, SizeOf(TLogPalette) + 32 * SizeOf(TPaletteEntry));

    {Inicializa el número de versión de la paleta}
    ThePalette^.palVersion := $300;

    {Tendremos un total de 32 entradas en la paleta}
    ThePalette^.palNumEntries := 32;

    {Obtiene las primeras 10 entradas de la paleta del sistema}
    GetSystemPaletteEntries(Form1.Canvas.Handle, 0, 10, @(ThePalette^.palPalEntry));

    {Sólo queremos 22 nuevas entradas y que se sitúen inmediatamente después de
    las 10 primeras de la paleta del sistema. Recuperando las 10 primeras entradas
    de la paleta del sistema, cuando activemos nuestra nueva paleta, las 10 primeras
    entradas de la paleta lógica serán mapeadas a las 10 primeras de la paleta de
    sistema y nuestras nuevas entradas de paleta se situarán a continuación}
    for iLoop := 0 to 21 do
    begin
        {Crea una paleta de gradiente rojo}
        ThePalette^.palPalEntry[iLoop+10].peRed   := 255 - ((255 div 32)*iLoop);
        ThePalette^.palPalEntry[iLoop+10].peGreen := 0;
        ThePalette^.palPalEntry[iLoop+10].peBlue  := 0;
        {No hace corresponder esta entrada de paleta con ninguna otra}
        ThePalette^.palPalEntry[iLoop+10].peFlags := PC_NOCOLLAPSE;
    end;

    {Crea la paleta}
    FormPalette := CreatePalette(ThePalette^);

    {Libera la memoria temporal}
    FreeMem(ThePalette, SizeOf(TLogPalette) + 42 * SizeOf(TPaletteEntry));
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    OldPalette: HPALETTE; // manejador de la paleta anterior
    iLoop: Integer;        // variable de control de bucle
begin
    {Selecciona nuestra nueva paleta lógica en el contexto de dispositivo}
    OldPalette := SelectPalette(Canvas.Handle, FormPalette, FALSE);

    {Mapea nuestra paleta lógica a la paleta del sistema}
    RealizePalette(Canvas.Handle);

    {Muestra el gradiente rojo}
    for iLoop := 0 to 31 do

```

```

begin
    Canvas.Brush.Color := $01000000 or iLoop + 10;
    Canvas.FillRect(Rect(10, iLoop*10 + 16, 260, iLoop*10 + 36));
end;

{Selecciona la paleta previa en el contexto de dispositivo}
SelectPalette(Canvas.Handle, OldPalette, FALSE);
end;

```

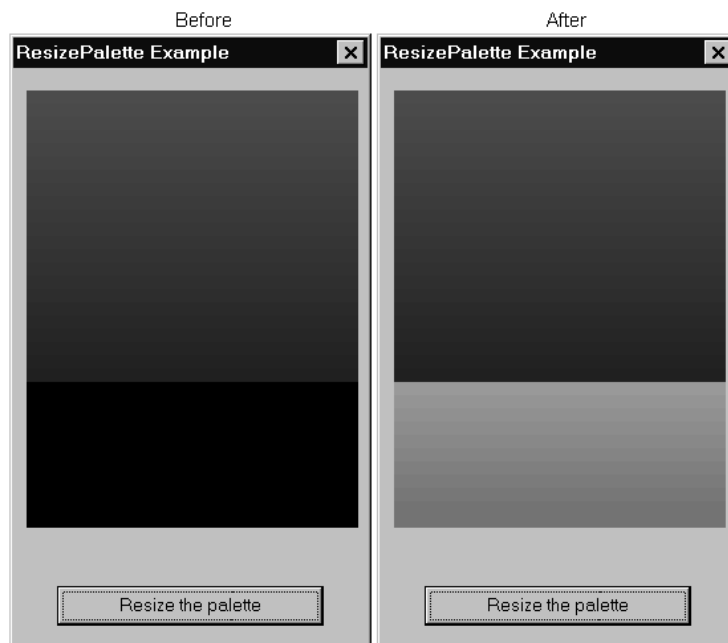


Figura 7-13:  
La paleta  
redimensionada

```

procedure TForm1.Button1Click(Sender: TObject);
var
    PaletteEntries: array[0..9] of TPaletteEntry; // almacena entradas de paleta
    iLoop: Integer;
begin
    ResizePalette(FormPalette, 42);

    {Muestra un gradiente verde}
    for iLoop := 0 to 9 do
    begin
        PaletteEntries[iLoop].peRed := 0;
        PaletteEntries[iLoop].peGreen := 255 - ((255 div 32) * iLoop);
        PaletteEntries[iLoop].peBlue := 0;
        PaletteEntries[iLoop].peFlags := PC_NOCOLLAPSE;
    end;

    {Reinicia la paleta lógica a este gradiente verde}
    SetPaletteEntries(FormPalette, 32, 10, PaletteEntries);

    {Redibuja el formulario}

```

```

        Invalidate;
    end;

    procedure TForm1.FormDestroy(Sender: TObject);
    begin
        {No necesitamos ya la paleta lógica y la eliminamos}
        DeleteObject(FormPalette);
    end;

```

## RGB Windows.Pas

### Sintaxis

```

RGB(
    r: Byte;           {intensidad del color rojo}
    g: Byte;           {intensidad del color verde}
    b: Byte;           {intensidad del color azul}
): COLORREF;         {devuelve un especificador de color de 32 bits}

```

### Descripción

Esta función devuelve un especificador de color de 32 bits que contiene un \$00 en el byte más significativo y los valores de los colores rojo, verde y azul en los 3 menos significativos. Windows muestra este color difuminando los 20 colores del sistema para obtener el color más cercano al solicitado.

### Parámetros

*r*: Especifica la intensidad del color rojo. Tiene que ser un valor entre 0 y 255.  
*g*: Especifica la intensidad del color verde. Tiene que ser un valor entre 0 y 255.  
*b*: Especifica la intensidad del color azul. Tiene que ser un valor entre 0 y 255.

### Valor que devuelve

Si la función tiene éxito, devuelve un especificador de color de 32 bits que indica que el color debe ser mostrado difuminando los 20 colores del sistema para obtener el color más cercano. Si la función falla, devuelve cero.

### Véase además

*CreatePalette*, *PaletteIndex*, *PaletteRGB*

### Ejemplo

Vea el Listado 7-8 bajo *GetNearestColor* y el Listado 7-12 bajo *PaletteRGB*.

## SelectPalette Windows.Pas

### Sintaxis

```

SelectPalette(

```

```

DC: HDC;                {manejador del contexto de dispositivo}
Palette: HPALETTE;      {manejador de paleta lógica}
ForceBackground: Bool   {indicador del modo de paleta}
): HPALETTE;            {devuelve un manejador de la paleta seleccionada
                        anteriormente}

```

### Descripción

Esta función asocia una paleta lógica con el contexto de dispositivo especificado. Una paleta lógica puede ser asociada con múltiples contextos de dispositivos, pero un cambio en la paleta lógica afectará a los contextos de dispositivos asociados con ella.

### Parámetros

*DC*: Manejador del contexto de dispositivo que será asociado con la paleta lógica especificada.

*Palette*: Manejador de una paleta lógica.

*ForceBackground*: Indica si la paleta lógica es una paleta de primer plano o de fondo. Un valor TRUE indica que la paleta es de fondo. Cuando la función *RealizePalette* es llamada, las entradas de la paleta lógica son mapeadas a las entradas correspondientes más cercanas en la paleta del sistema y ninguna entrada en la paleta del sistema es reemplazada. Este es el comportamiento por defecto para aplicaciones que se ejecutan en segundo plano. Un valor FALSE indica que la paleta es una paleta de primer plano. Cuando la función *RealizePalette* es llamada, las entradas de la paleta lógica son igualmente mapeadas a las entradas correspondientes más cercanas en la paleta del sistema. Si una correspondencia exacta es hallada, esa entrada de la paleta lógica es mapeada a la correspondiente entrada en la paleta del sistema. Sin embargo, si no se encuentra una correspondencia exacta, a la primera entrada, sin usar en la paleta del sistema, se le asignan los valores de los colores de la entrada en la paleta lógica. Este comportamiento puede verse afectado por los valores del campo *peFlags* de la entrada de la paleta. Consulte la función *CreatePalette* para más detalles.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la paleta lógica anteriormente seleccionada en el contexto de dispositivo; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*CreatePalette*, *RealizePalette*

### Ejemplo

Vea el Listado 7-4 bajo *CreatePalette*, y muchos otros ejemplos a lo largo de este capítulo.

**SetDIBColorTable****Windows.Pas****Sintaxis**

```
SetDIBColorTable(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: UINT;                {índice en la paleta de la primera entrada a modificar}
    p3: UINT;                {cantidad de entradas de paleta a modificar}
    var RGBQuadStructs      {puntero a un array de registros TRGBQuad}
): UINT;                    {devuelve la cantidad de entradas modificadas}
```

**Descripción**

Esta función reemplaza el rango especificado de entradas en la paleta de colores en mapa de bits independiente del dispositivo actualmente seleccionado en el contexto de dispositivo identificado por el parámetro *DC*, con las entradas a las que apunta el parámetro *RGBQuadStructs*. Esta función es útil sólo para DIBs que tengan un formato de color de 1, 4, u 8 bits por píxel, ya que los DIBs con formatos de color superiores a 8 bits por píxel no contienen una paleta de colores.

**Parámetros**

*DC*: Manejador de contexto de dispositivo. El DIB cuya paleta de colores va a ser modificada tiene que estar seleccionado en este contexto de dispositivo.

*p2*: Especifica el índice (con base cero) de la primera entrada en la paleta del colores del DIB que será modificada.

*p3*: Especifica la cantidad de entradas de paleta del DIB a modificar.

*RGBQuadStructs*: Puntero a un *array* de registros *TRGBQuad* que describen los valores de colores que reemplazan las entradas de paleta en el DIB especificado. Este *array* tiene que almacenar, al menos, la cantidad de registros *TRGBQuad* indicada por el parámetro *p3*. El registro *TRGBQuad* se define como:

TRGBQuad = **packed record**

```
    rgbBlue: Byte;           {intensidad del color azul}
    rgbGreen: Byte;          {intensidad del color verde}
    rgbRed: Byte;            {intensidad del color rojo}
    rgbReserved: Byte;       {valor reservado}
```

**end;**

Consulte la función *CreateDIBSection* para ver una descripción de este registro.

**Valor que devuelve**

Si la función tiene éxito, devuelve la cantidad de entradas reemplazadas en la paleta de colores del mapa de bits independiente del dispositivo; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*CreateDIBSection, CreatePalette, GetDIBColorTable*

Ejemplo

**Listado 7-14: Modificando la tabla de colores de un mapa de bits independiente del dispositivo**

```

procedure TForm1.Button1Click(Sender: TObject);
var
  Dib: TBitmap;                // mapa de bits DIB
  NewColors: array[0..63] of TRGBQuad; // nueva tabla de colores del DIB
  OffscreenDC: HDC;            // manejador de un DC fuera de pantalla
  OldBitmap: HBITMAP;          // manejador del mapa de bits del DC
  iLoop: Integer;              // un contador de bucle general
begin
  {Los objetos TBitmap de Delphi encapsulan un DIB. Creamos un mapa de bits y lo
  cargamos en una imagen de mapa de bits de prueba}
  Dib := TBitmap.Create;
  Dib.LoadFromFile('Gradient.bmp');

  {Crea un gradiente verde en el array NewColors}
  for iLoop := 0 to 63 do
    begin
      NewColors[iLoop].rgbBlue := 0;
      NewColors[iLoop].rgbGreen := (256 div 64)*iLoop;
      NewColors[iLoop].rgbRed := 0;
      NewColors[iLoop].rgbReserved := 0;
    end;

  {Crea un contexto de dispositivo fuera de pantalla compatible con el dispositivo
  de visualización}
  OffscreenDC := CreateCompatibleDC(0);

  {Selecciona el DIB en este contexto de dispositivo}
  OldBitmap := SelectObject(OffscreenDC, Dib.Handle);

  {Reemplaza su tabla de colores con los colores del nuevo gradiente verde}
  SetDIBColorTable(OffscreenDC, 0, 64, NewColors);

  {Selecciona el mapa de bits previo en el DC fuera de pantalla}
  SelectObject(OffscreenDC, OldBitmap);

  {El DC fuera de pantalla ya no es necesario y lo eliminamos}
  DeleteDC(OffscreenDC);

  {La tabla de colores del DIB ha sido modificada, así que lo guardamos en disco}
  Dib.SaveToFile('NewGrad.bmp');
  Dib.Free;

  {Carga y muestra el DIB con su nueva tabla de colores}
  Image1.Picture.Bitmap.LoadFromFile('NewGrad.bmp');
  Label1.Caption := 'After';

```

end;



Figura 7-14:  
Esta tabla de  
colores del  
DIB ha sido  
modificada

## SetPaletteEntries

## Windows.Pas

### Sintaxis

SetPaletteEntries( Palette: HPALETTE; StartIndex: UINT; NumEntries: UINT; <b>var</b> PaletteEntries );	UINT; {manejador de paleta lógica} {índice de inicio de la paleta} {cantidad de entradas de la paleta a reemplazar} {puntero a un <i>array</i> de registros <i>TPaletteEntry</i> } {devuelve la cantidad de entradas reemplazadas}
---	---

### Descripción

Esta función reemplaza un rango de entradas en la paleta de colores en la paleta lógica especificada, con los valores en el *array* de registros *TPaletteEntry* al que apunta el parámetro *PaletteEntries*. Una vez que las entradas de la paleta han sido reemplazadas, la función *RealizePalette* tiene que ser llamada para que los cambios tengan efecto.

### Parámetros

*Palette*: Manejador de la paleta lógica cuyas entradas de paleta van a ser reemplazadas.

*StartIndex*: El índice, basado en cero, de la primera entrada a reemplazar en la paleta lógica.

*NumEntries*: Especifica la cantidad de entradas a reemplazar.

*PaletteEntries*: Puntero a un *array* de registros de tipo *TPaletteEntry* que describen los valores de los colores que reemplazan las entradas en la paleta lógica especificada. Este *array* tiene que almacenar al menos la cantidad de registros *TPaletteEntry* que indica el parámetro *NumEntries*. El registro *TPaletteEntry* se define como:

```

TPaletteEntry = packed record
    peRed: Byte;           {intensidad del color rojo}
    peGreen: Byte;         {intensidad del color verde}
    peBlue: Byte;          {intensidad del color azul}
    peFlags: Byte;         {opciones de tratamiento de entrada de paleta}
end;

```

Consulte la función *CreatePalette* para ver una descripción de este registro.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de entradas de paleta reemplazadas en la paleta lógica especificada; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreatePalette*, *GetPaletteEntries*, *RealizePalette*, *SelectPalette*

#### Ejemplo

Vea el Listado 7-4 bajo *CreatePalette*.

## SetSysColors      Windows.Pas

### Sintaxis

```

SetSysColors(
    cElements: Integer;           {número de elementos de visualización a cambiar}
    const lpaElements;             {puntero a un array de valores de elementos}
    const lpaRgbValues             {puntero a un array de valores RGB}
): BOOL;                         {devuelve TRUE o FALSE}

```

### Descripción

Esta función reemplaza los valores de los colores RGB de los elementos de visualización del sistema especificados en el parámetro *lpaElements*, con los valores de colores RGB explícitos almacenados en el *array* al que apunta el parámetro *lpaRgbValues*. Un elemento de visualización identifica varias partes de una ventana y su color se asocia con uno de los 20 colores estáticos definidos por el sistema. Cuando el color de un elemento de visualización es cambiado, su color de sistema asociado es también cambiado. Los *arrays* a los que apuntan los parámetros *lpaElements* y *lpaRgbValues* deben contener la misma cantidad de entradas. Esta función hace que un mensaje *WM\_SYSCOLORCHANGE* sea enviado a todas las ventanas informándoles del cambio, y que Windows redibuje apropiadamente la parte afectada de esas ventanas. Estos nuevos colores del sistema estarán activos solamente durante la sesión actual de Windows y serán restaurados a sus valores previos cuando Windows sea reiniciado.

**Parámetros**

*cElements*: Especifica la cantidad de elementos de visualización en el *array* al que apunta el parámetro *lpaElements*.

*lpaElements*: Puntero a un *array* de enteros. Cada elemento en el *array* identifica un elemento de visualización cuyo color será cambiado y contiene uno de los valores de la Tabla 7-6.

*lpaRgbValues*: Puntero a un *array* de enteros largos. Cada elemento en este *array* se corresponde con un elemento en el *array* *lpaElements* en la misma posición y contiene los nuevos valores de colores RGB de 32 bits explícitos para este elemento de visualización.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetBValue*, *GetGValue*, *GetRValue*, *GetSysColor*, *GetSystemPaletteEntries*, *PaletteRGB*, *RGB*

**Ejemplo**

Vea el Listado 7-10 bajo *GetSysColor*.

**Tabla 7-6: Valores de los elementos del parámetro *lpaElements* de *SetSysColors***

Valor	Descripción
COLOR_3DDKSHADOW	El color de la sombra oscura para elementos de visualización tridimensionales.
COLOR_3DLIGHT	El borde iluminado para elementos de visualización tridimensionales.
COLOR_ACTIVEBORDER	El color del borde de la ventana activa.
COLOR_ACTIVECAPTION	El color de la barra de título de la ventana activa.
COLOR_APPWORKSPACE	El color de fondo usado en las aplicaciones de interfaz multidocumento (MDI).
COLOR_BACKGROUND	El color del escritorio.
COLOR_BTNFACE	El color de la superficie de los botones.
COLOR_BTNHIGHLIGHT	El color de los botones resaltados.
COLOR_BTNSHADOW	El color del borde sombreado de los botones.
COLOR_BTNTEXT	El color del texto de los botones.
COLOR_CAPTIONTEXT	El color del texto usado en barras de título, caja de redimensionamiento y caja de la flecha de las barras de desplazamiento.

Valor	Descripción
COLOR_GRAYTEXT	El color del texto deshabilitado. Este será puesto a cero si el controlador de vídeo no puede soportar gris sólido.
COLOR_HIGHLIGHT	El color usado para los elementos seleccionados en un control.
COLOR_HIGHLIGHTTEXT	El color usado para el texto de los elementos seleccionados en un control.
COLOR_INACTIVEBORDER	El color del borde de una ventana inactiva.
COLOR_INACTIVECAPTION	El color de la barra de título de una ventana inactiva.
COLOR_INACTIVECAPTIONTEXT	El color del texto de una barra de título inactiva.
COLOR_INFOBK	El color de fondo de los controles de indicaciones.
COLOR_INFOTEXT	El color del texto de los controles de indicaciones.
COLOR_MENU	El color del fondo de los menús.
COLOR_MENUTEXT	El color del texto usado en los menús.
COLOR_SCROLLBAR	El color del área “gris” de las barras de desplazamiento.
COLOR_WINDOW	El color del fondo de las ventanas.
COLOR_WINDOWFRAME	El color del marco de las ventanas.
COLOR_WINDOWTEXT	El color del texto usado en las ventanas.

### **SetSystemPaletteUse**      **Windows.Pas**

#### **Sintaxis**

```
SetSystemPaletteUse(
    DC: HDC                {manejador de contexto de dispositivo}
    p2: UINT               {nuevo estado de la paleta del sistema}
): UINT;                  {devuelve el estado anterior de la paleta del sistema}
```

#### **Descripción**

Esta función establece el estado de los colores estáticos de la paleta de sistema asociada con el contexto de dispositivo asociado. Por defecto, el sistema contiene 20 colores estáticos que no serán cambiados cuando cualquier paleta lógica sea realizada.

#### **Parámetros**

*DC*: Manejador del contexto de dispositivo para el cual el estado de la paleta del sistema asociada será modificado. Debe ser obligatoriamente un contexto de dispositivo que soporte paletas de colores.

*p2*: Valor que especifica el nuevo estado de los colores estáticos de la paleta del sistema. Puede ser un valor de la Tabla 7-7.

**Valor que devuelve**

Si la función tiene éxito, devuelve el estado anterior de la paleta del sistema asociada al contexto de dispositivo especificado, que será un valor de la Tabla 7-7. Si la función falla, devuelve *SYSPAL\_ERROR*, que indica que el contexto de dispositivo no era válido o no soportaba una paleta de colores. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetSysColor*, *GetSystemPaletteUse*, *RealizePalette*, *SelectPalette*, *SetPaletteEntries*, *SetSysColors*

**Ejemplo****Listado 7-15: Usando 254 colores en la paleta del sistema**

```
{¡OJO! Delphi importa incorrectamente la función GetSystemPaletteEntries.
Aquí está la declaración correcta, que brinda toda la funcionalidad accesible
con esta función del API}
function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: UINT;
                                PaletteEntries: Pointer): UINT; stdcall;

var
    Form1: TForm1;
    SysPalette: array[0..255] of TPaletteEntry; // almacena una copia de la paleta
    FormPalette: HPALETTE;                      // manejador de la nueva paleta

implementation

{Enlace de la función GetSystemPaletteEntries}
function GetSystemPaletteEntries; external gdi32 name 'GetSystemPaletteEntries';

procedure TForm1.SetThePalette;
var
    ThePalette: PLogPalette; // puntero a la descripción de la paleta lógica
    iLoop: Integer;          // contador de bucle
    TheDC: HDC;              // manejador de contexto de dispositivo
begin
    {Reserva suficiente memoria para una paleta lógica de 256 colores}
    GetMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));

    {Recupera un manejador del contexto de dispositivo del formulario}
    TheDC := GetDC(Form1.Handle);

    {Recupera las entradas actuales de la paleta del sistema}
    GetSystemPaletteEntries(TheDC, 0, 256, @(ThePalette^.palPalEntry));

    {Establece que la paleta del sistema sólo use 2 colores estáticos}
    SetSystemPaletteUse(TheDC, SYSPAL_NOSTATIC);

    {Inicializa la información de la paleta lógica apropiada}
    ThePalette^.palVersion := $300;
    ThePalette^.palNumEntries := 256;
```

```

{Crea una paleta gradiente en rojo de 254 colores}
for iLoop := 0 to 253 do
begin
    ThePalette^.palPalEntry[iLoop+1].peRed   := 255 - iLoop;
    ThePalette^.palPalEntry[iLoop+1].peGreen := 0;
    ThePalette^.palPalEntry[iLoop+1].peBlue  := 0;
    ThePalette^.palPalEntry[iLoop+1].peFlags := 0;
end;

{Crea la paleta de gradiente en rojo}
FormPalette := CreatePalette(ThePalette^);

{Selecciona y activa la paleta. Todos los colores del sistema, excepto el
 primero y el último, serán reemplazados con los colores de la nueva paleta
 lógica.}
SelectPalette(TheDC, FormPalette, FALSE);
RealizePalette(TheDC);

{Libera el registro de la paleta lógica y el contexto de dispositivo}
FreeMem(ThePalette, SizeOf(TLogPalette) + 256 * SizeOf(TPaletteEntry));
ReleaseDC(Form1.Handle, TheDC);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    iLoop: Integer;    // contador de bucle general
    TheDC: HDC;        // manejador de contexto de dispositivo
begin
    {Crea y activa la nueva paleta}
    SetThePalette;

    {Obtiene un manejador del contexto de dispositivo del formulario}
    TheDC := GetDC(Form1.Handle);

    {Muestra el estado actual de los colores estáticos de la paleta del sistema}
    ListBox1.Items.Add('System Palette State:');
    case GetSystemPaletteUse(TheDC) of
        SYSPAL_NOSTATIC: ListBox1.Items.Add(' No static colors');
        SYSPAL_STATIC:   ListBox1.Items.Add(' Static colors');
        SYSPAL_ERROR:    ListBox1.Items.Add(' No color palette');
    end;

    {Recupera las entradas de la paleta del sistema}
    GetSystemPaletteEntries(TheDC, 0, 256, @SysPalette);

    {Muestra los valores de los colores de la paleta del sistema}
    for iLoop := 0 to 255 do
    begin
        ListBox1.Items.Add('Palette slot: ' + IntToStr(iLoop));
        ListBox1.Items.Add('   Red: ' + IntToStr(SysPalette[iLoop].peRed));
        ListBox1.Items.Add('  Green: ' + IntToStr(SysPalette[iLoop].peGreen));
        ListBox1.Items.Add('   Blue: ' + IntToStr(SysPalette[iLoop].peBlue));
    end;

```

```

    case SysPalette[iLoop].peFlags of
        PC_EXPLICIT:  ListBox1.Items.Add('  Flags: PC_EXPLICIT');
        PC_NOCOLLAPSE: ListBox1.Items.Add('  Flags: PC_NOCOLLAPSE');
        PC_RESERVED:  ListBox1.Items.Add('  Flags: PC_RESERVED');
        else ListBox1.Items.Add('  Flags: NULL');
    end;
end;

{Borra el manejador del contexto de dispositivo}
ReleaseDC(Form1.Handle, TheDC);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    iOutLoop, iInLoop: Integer; // contadores de bucles
    TheDC: HDC;                // manejador de contexto de dispositivo
    OldBrush, NewBrush: HBRUSH; // manejadores de brochas
begin
    {Obtiene un manejador del contexto de dispositivo del formulario}
    TheDC := GetDC(Form1.Handle);

    {Selecciona y activa la nueva paleta lógica}
    SelectPalette(TheDC, FormPalette, FALSE);
    RealizePalette(TheDC);

    {Dibuja una rejilla de rectángulos que muestra la paleta lógica actual,
    la cual es un duplicado exacto de la paleta del sistema}
    for iOutLoop := 0 to 15 do
        for iInLoop := 0 to 15 do
            begin
                {Usa índices de paleta explícitos}
                NewBrush := CreateSolidBrush($01000000 or (iOutLoop*16) + iInLoop);
                OldBrush := SelectObject(TheDC, NewBrush);
                Rectangle(TheDC, (iInLoop*20) + 15, (iOutLoop*20) + 32,
                    (iInLoop*20) + 35, (iOutLoop*20) + 52);
                SelectObject(TheDC, OldBrush);
                DeleteObject(NewBrush);
            end;
        end;

    {Elimina el manejador del contexto de dispositivo}
    ReleaseDC(Form1.Handle, TheDC);
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
    TheDC: HDC; // manejador de contexto de dispositivo
begin
    {Reinicia los colores estáticos a los 20 colores estáticos por defecto}
    TheDC := GetDC(Form1.Handle);
    SetSystemPaletteUse(TheDC, SYSPAL_STATIC);
    ReleaseDC(Form1.Handle, TheDC);

    {Elimina la paleta lógica}
    DeleteObject(FormPalette);

```

```
end;
```

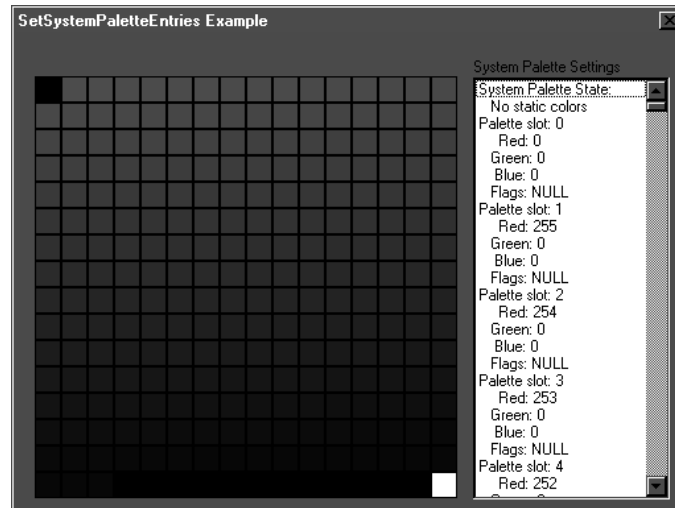


Figura 7-15:  
Una paleta  
roja

**Tabla 7-7: Valores del parámetro p2 de SetSystemPaletteUse**

Valor	Descripción
SYSPAL_NOSTATIC	La paleta del sistema contiene sólo dos colores estáticos: negro y blanco.
SYSPAL_STATIC	La paleta del sistema contiene los colores estáticos por defecto. Estos colores no cambiarán cuando una paleta sea realizada. Este es el estado por defecto de la paleta del sistema.

**Capítulo 8**

# Funciones de salida de texto

**8****Capítulo**

Dibujar texto en pantalla es la función más común realizada por Windows en casi todas las aplicaciones. Por eso las funciones del API para manipular y mostrar texto son numerosas y robustas. Si bien Delphi encapsula algunas de las funciones de salida de texto del API, los desarrolladores de Delphi pueden extender ampliamente las capacidades de dibujar texto de una aplicación utilizando las funciones descritas en este capítulo.

## Fuentes

Actualmente Windows soporta tres tipos de fuentes: de barrido (*raster*), vectoriales y *True Type*. Las diferencias entre los tipos de fuentes radica en el método mediante el cual el patrón de la fuente define la forma de un carácter. Un patrón para una fuente de barrido es un mapa de bits de un tamaño específico que contiene una imagen para cada carácter individual. Las fuentes vectoriales almacenan sus patrones como una serie de puntos de terminación usados para crear segmentos de líneas que definen el contorno de un carácter. Los patrones de las fuentes True Type son almacenados como una serie de líneas, curvas e indicaciones que son usados para dibujar el contorno del carácter. Debido al hecho de que las fuentes de barrido almacenan sus patrones como mapas de bits, estas fuentes pierden mucha resolución cuando son escaladas. Las fuentes vectoriales pueden generalmente ser escaladas hacia arriba o hacia abajo, pero comenzarán a perder su resolución cuando sean escaladas por debajo de su tamaño original y además serán dibujadas más lentamente. Sin embargo, las fuentes True Type pueden ser dibujadas relativamente rápido porque el subsistema GDI está optimizado para ello. Adicionalmente, las indicaciones almacenadas en las definiciones del patrón de las fuentes True Type ofrecen correcciones de escalamiento para las curvas y líneas del trazado del carácter, permitiendo a las fuentes True Type ser escaladas a cualquier tamaño sin pérdida de resolución.

## Familias de fuentes

Windows categoriza todas las fuentes en cinco familias. Una *familia de fuentes* es una colección de fuentes que comparten atributos similares de ancho de trazos y *serif*.

Cuando se está considerando elegir una fuente, las familias de fuentes permiten al desarrollador indicar el estilo general deseado, dejando a Windows la selección de la fuente. Por ejemplo, usando la función apropiada y especificando sólo una familia de fuentes, el desarrollador puede enumerar todas las fuentes de símbolos instaladas en el sistema. Las familias de fuentes también permiten al desarrollador crear una fuente lógica basada únicamente en características específicas, permitiendo así a Windows seleccionar la fuente más apropiada de la familia de fuentes indicada, basándose en las características señaladas.

Las cinco categorías de familias de fuentes definidas por Windows son:

**Tabla 8-1: Familias de fuentes**

Nombre Familia	Constante	Descripción
Decorative	FF_DECORATIVE	Indica una fuente decorativa, como la Old English.
Modern	FF_MODERN	Indica una fuente monoespaciada con ancho constante de trazos, con o sin serifs, como la Courier New.
Roman	FF_ROMAN	Indica una fuente proporcional con ancho variable de trazos, que contiene serifs, como la Times New Roman.
Script	FF_SCRIPT	Indica una fuente que simula la manuscrita, como la Brush Script.
Swiss	FF_SWISS	Indica una fuente proporcional con ancho variable de trazos, sin serifs, como la Arial.

## Conjuntos de caracteres

Por definición, una fuente define la imagen de cada carácter individual dentro de una colección de caracteres. Esta colección de caracteres es llamada un *conjunto de caracteres*. Cada conjunto de caracteres contiene símbolos, números, signos de puntuación, letras y otras imágenes imprimibles o visualizables de un lenguaje escrito, con cada carácter identificado por un número.

Hay cinco conjuntos de caracteres principales: Windows, Unicode, OEM, Symbol y específico del fabricante. El conjunto de caracteres de Windows es equivalente al conjunto de caracteres ANSI. El conjunto de caracteres Unicode es usado para lenguajes orientales que contienen cientos de símbolos en sus alfabetos y en la actualidad es el único conjunto de caracteres que usa dos bytes para identificar un carácter simple. El conjunto de caracteres OEM es generalmente equivalente al conjunto de caracteres de Windows, exceptuando que usualmente contiene caracteres en los rangos superiores e inferiores del espacio de caracteres imprimibles, que sólo pueden ser mostrados en una sesión de DOS a pantalla completa. El conjunto de caracteres Symbol contiene caracteres útiles para la representación de ecuaciones

matemáticas o científicas, o caracteres gráficos usados para ilustraciones. El conjunto de caracteres específico del fabricante generalmente ofrece caracteres que no están accesibles en los demás conjuntos de caracteres y se implementan más frecuentemente a nivel de impresora o dispositivo de salida.

En muchos casos, una fuente definirá un carácter por defecto. Cuando una cadena contiene un carácter que no está definido en el conjunto de caracteres de la fuente seleccionada en un contexto de dispositivo, el carácter por defecto sustituirá al carácter no definido cuando el texto sea mostrado. La mayoría de las fuentes True Type definen el carácter por defecto como un rectángulo no rellenado (“□”).

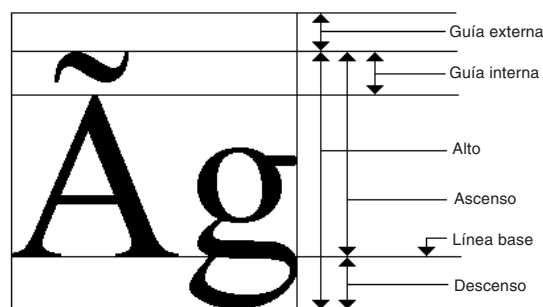
Para propósitos de cambio de línea y justificación, la mayoría de las fuentes definen un carácter de ruptura. El carácter de ruptura identifica el carácter más comúnmente usado para separar palabras en una línea de texto. La mayoría de las fuentes que usan el conjunto de caracteres de Windows definen el espacio (“ ”) como carácter de ruptura.

### Dimensiones de los caracteres

Los tamaños de las fuentes son habitualmente medidos en unidades llamadas *puntos*. Un punto equivale a 0,013837 pulgadas, comúnmente aproximado a 1/72 de pulgada. Observe que una pulgada lógica en Windows es aproximadamente un 30 o un 40 por ciento más larga que una pulgada física, con vistas a facilitar una mayor legibilidad de las fuentes en pantalla.

La imagen de un patrón de carácter está definida por dimensiones específicas, como se ilustra en la Figura 8-1. La *línea base* (*baseline*) de un patrón es una línea imaginaria

Figura 8-1:  
Las  
dimensiones  
de un patrón



que define la “base” sobre la cual un carácter se apoyará. El *descenso* (*descent*) es el espacio bajo la línea base que contiene las partes inferiores de ciertos caracteres como la “g” y la “y.” Las *guías internas* definen el espacio sobre el carácter donde residen los acentos y diacríticos. Las *guías externas* están fuera de la imagen del patrón y se utilizan solamente para el espaciado extra vertical entre líneas. El *ascenso* (*ascent*) se define como la distancia desde la línea base hasta la parte superior del espacio de la guía interna. La *altura* es la suma del ascenso y el descenso, y define el espacio vertical total que puede ser ocupado por los

datos de la imagen del patrón. Estas dimensiones de los caracteres pueden ser recuperadas llamando las funciones *GetOutlineTextMetrics* o *GetTextMetrics*. Las medidas recuperadas por estas funciones estarán en unidades lógicas, de modo que sus valores actuales dependerán del modo de mapeado del contexto de dispositivo especificado en la función llamada.

## La tabla de fuentes de Windows

Windows almacena una referencia a todas las fuentes no asociadas a un dispositivo en un *array* interno conocido como la *tabla de fuentes*. Cualquier fuente en esta tabla interna es accesible a las aplicaciones. Una aplicación puede añadir desde programa un recurso de fuente a esta tabla interna, llamando a la función *AddFontResource*. Una vez que esta función haya sido completada correctamente, la aplicación que está instalando la fuente debe avisar del cambio en la tabla de fuentes a todas las demás aplicaciones, enviando el mensaje *WM\_FONTCHANGE* con la función *SendMessage*, especificando el valor *HWND\_BROADCAST* en el parámetro *hWnd*. Cuando la aplicación haya terminado o la fuente ya no sea necesaria, deberá ser eliminada mediante una llamada a la función *RemoveFontResource*. Observe que la fuente no será eliminada de la tabla interna de fuentes hasta que todos los contextos de dispositivo la hayan deseleccionado, si la fuente fue seleccionada en algún contexto de dispositivo antes de la llamada a *RemoveFontResource*.

La función *AddFontResource* instala la fuente en la tabla interna sólo durante el transcurso de la ejecución de la aplicación o hasta que la fuente es totalmente liberada como se describe anteriormente. En versiones anteriores de Windows, la instalación permanente de una fuente requería que una aplicación modificara la sección *Fonts* en el fichero **Win.ini**. Bajo Windows 95/98 y NT 4.0, una aplicación puede instalar permanentemente una fuente simplemente copiando el fichero fuente en el directorio *Fonts* bajo el directorio *Windows*.

Cuando una aplicación llama a las funciones *CreateFont* o *CreateFontIndirect*, no se crea una nueva fuente. Estas funciones devuelven un manejador de una definición de fuente lógica la cual es usada por el mapeador de fuentes de Windows para seleccionar una fuente física apropiada de la tabla de fuentes. El mapeador de fuentes de Windows toma las características de la fuente deseada definidas en la fuente lógica y utiliza un algoritmo interno para compararlas con las características de las fuentes físicas instaladas en el sistema. Este algoritmo de mapeado de fuentes se ejecuta cuando la fuente lógica es seleccionada en el contexto de dispositivo mediante una llamada a la función *SelectObject* y su resultado es la selección de la fuente que más cercanamente se corresponde con las características deseadas. Por consiguiente, la fuente devuelta por el mapeador de fuentes puede no corresponderse exactamente con la fuente requerida.

## Incrustación de fuentes

Los procesadores de texto más avanzados permiten incrustar fuentes True Type en un documento. Incrustar una fuente True Type en un documento le permite a ese documento ser visualizado o editado por un procesador de texto en otro sistema en su formato original, si ese sistema no contiene la fuente específica instalada. Sin embargo, como las fuentes son propiedad y están bajo *copyright* de su desarrollador original, hay que tener ciertas precauciones cuando se incrusta una fuente.

El desarrollador de la fuente puede no permitir que la fuente sea incrustada. Una fuente puede permitir ser incrustada únicamente en un contexto *de-sólo-lectura*. Si un documento contiene una fuente incrustada *de-sólo-lectura*, el documento puede ser visualizado o impreso, pero no puede ser modificado ni la fuente ser desincrustada para instalarla permanentemente en el sistema. Algunas fuentes pueden tener licencia de *lectura-escritura*, lo que indica que la fuente puede ser incrustada en un documento e instalada permanentemente en el sistema de destino. Una aplicación puede determinar el estado de incrustación de una fuente usando la función *GetOutlineTextMetrics*. En cualquier caso, a menos de que se disponga de un permiso específico del creador de las fuentes, éstas pueden ser incrustadas solamente en un documento y no pueden ser incrustadas en una aplicación, ni una aplicación puede ser distribuida con documentos que contengan fuentes incrustadas.

Para incrustar una fuente en un documento, la aplicación tiene que recuperar los datos del fichero de fuente completo, llamando a la función *GetFontData* con el valor cero (0) en los parámetros *p2* y *p3*. Los datos de la fuente deben ser grabados entonces en el fichero de salida junto con el texto del documento, en el formato de fichero determinado por la aplicación. Tradicionalmente, las aplicaciones utilizan un formato de fichero que contiene el nombre de cada fuente incrustada en el documento y una indicación de la licencia *de-sólo-lectura* o de *lectura-escritura*. Observe que si una fuente *de-sólo-lectura* es incrustada en un documento, deberá ser encriptada, aunque el algoritmo de encriptación no tiene que ser demasiado complejo.

Cuando la aplicación abre un documento que contiene una fuente incrustada, debe primero determinar si la fuente permite una incrustación *de-sólo-lectura* o de *lectura-escritura*. Los datos de la fuente son entonces extraídos del fichero del documento y guardados en el disco mediante funciones de manipulación de ficheros, tales como *CreateFile* y *WriteFile*. Si la fuente es de *lectura-escritura*, puede ser escrita directamente a un fichero con extensión .TTF en el subdirectorio *Fonts* bajo el directorio *Windows*, para instalar la fuente permanentemente en el sistema. Si la fuente es *de-sólo-lectura*, tiene que ser encriptada y grabada en un fichero oculto. Este fichero oculto no debe tener extensión .TTF. Una vez que la fuente *de-sólo-lectura* es extraída y escrita en el disco, puede ser instalada en la tabla de fuentes interna de Windows mediante las funciones *CreateScalableFontResource* y *AddFontResource*, especificando un valor 1 para el parámetro *p1* de la función *CreateScalableFontResource*, para indicar una fuente *de-sólo-lectura*. Observe que las fuentes *de-sólo-lectura* no serán identificadas por las funciones *EnumFontFamilies* o

o *EnumFontFamiliesEx*. Cuando el documento que contiene la fuente incrustada *de-sólo-lectura* sea cerrado, el fichero .FOT creado por la función *CreateScalableFontResource* y el fichero creado al extraer la fuente *de-sólo-lectura* deberán ser eliminados y la fuente quitada de la tabla de fuentes de Windows, llamando a la función *RemoveFontResource*.

El siguiente ejemplo muestra cómo incrustar una fuente True Type en un documento de texto. El documento es guardado en disco en un formato propietario. Observe que el chequeo de la licencia *solo-de-lectura* ha sido omitido en aras de la claridad del código.

#### Listado 8-1: Incrustando fuentes True Type en un documento

```
{=====
  La fuente Ventilate usada en este ejemplo y su copyright © 1997 fueron
  generosamente donados por Brian J. Bonislowsky - Astigmatic One Eye. Usado con
  permiso.

  Astigmatic One Eye es un suministrador de shareware y freeware de fuentes de
  todo tipo. Para más información Visite http://www.comptechdev.com/cavop/aoe/

  Observe que este ejemplo hace uso de un documento que ya contiene una fuente
  True Type incrustada.
  =====}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SavedFile: THandle;    // almacena un manejador del fichero abierto
  TextSize: LongInt;     // almacena el tamaño del texto en el memo
  TheText: PChar;        // almacena el texto en el memo
  BytesWritten: DWORD;   // almacena el número de bytes escritos en el fichero
  FontData: Pointer;     // apunta a datos recuperados de la fuente
  FontDataSize: Integer; // almacena el tamaño de los datos de la fuente
  MemoDC: HDC;           // manejador de un contexto de dispositivo común
  OldFont: THandle;      // almacena la fuente previamente seleccionada en el DC
begin
  {Crea el fichero que contendrá el documento guardado y la fuente incrustada}
  SavedFile := CreateFile('ProprietaryFileFormat.PFF', GENERIC_WRITE, 0, nil,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL or
    FILE_FLAG_SEQUENTIAL_SCAN, 0);

  {Recupera el tamaño del texto en memo, añadiendo uno para el terminador nulo}
  TextSize := Length(Memo1.Text) + 1;

  {Recupera suficiente memoria para almacenar todo el texto}
  GetMem(TheText, TextSize);

  {Copia el texto a un buffer terminado en nulo}
  StrPCopy(TheText, Memo1.Text);

  {Asigna explícitamente el final del texto}
  TheText[TextSize] := #0;

  {Nuestro formato de fichero propietario es tal que los primeros 4 bytes del
```

```

    fichero contienen el número de bytes que tiene el texto del documento. Después
    de estos bytes, los próximos 4 bytes indican cuántos bytes ocupa la información
    de la fuente True Type embebida. Por eso, escribiremos los primeros cuatro
    bytes del documento como un entero que contiene el tamaño del texto y luego los
    bytes que contiene el texto del documento}
    WriteFile(SavedFile, TextSize, SizeOf(TextSize), BytesWritten, nil);
    WriteFile(SavedFile, TheText^, TextSize, BytesWritten, nil);

    {Para obtener los datos del fichero de fuente para su incrustación, la fuente
    debe ser seleccionada en un contexto de dispositivo. Recuperamos el contexto
    de dispositivo del memo, pero debido a que esto devuelve un DC común con
    atributos por defecto, tenemos que seleccionar la fuente del memo en el
    contexto de dispositivo recuperado.}
    MemoDC := GetDC(Memo1.Handle);
    OldFont := SelectObject(MemoDC, Memo1.Font.Handle);

    {En este punto, la fuente seleccionada debe ser chequeada para ver si permite
    ser incrustada. Si la fuente no permite ser incrustada, el documento debe ser
    simplemente guardado y se debe omitir el siguiente código. Si la fuente permite
    ser incrustada en un formato de-sólo-lectura, una vez que los datos son
    recuperados deberán ser encriptados antes de ser guardados en el fichero del
    documento}

    {Recupera el tamaño del buffer necesario para almacenar los datos de la fuente}
    FontDataSize := GetFontData(MemoDC, 0, 0, nil, 0);

    {Reserva la memoria necesaria}
    GetMem(FontData, FontDataSize);

    {Recupera el fichero de datos de la fuente completo}
    GetFontData(MemoDC, 0, 0, FontData, FontDataSize);

    {Ahora, escribe un entero que indica cuántos bytes contienen los datos de la
    fuente y luego escribe esos bytes que conforman los datos de la fuente}
    WriteFile(SavedFile, FontDataSize, SizeOf(FontDataSize), BytesWritten, nil);
    WriteFile(SavedFile, FontData^, FontDataSize, BytesWritten, nil);

    {Selecciona la fuente original en el contexto de dispositivo y borra el DC}
    SelectObject(MemoDC, OldFont);
    ReleaseDC(Memo1.Handle, MemoDC);

    {Descarga los buffers para forzar que el fichero sea escrito en el disco}
    FlushFileBuffers(SavedFile);

    {Cierra el manejador del fichero}
    CloseHandle(SavedFile);

    {El fichero ha sido guardado, se libera la memoria que ya no es necesaria}
    FreeMem(TheText, TextSize);
    FreeMem(FontData, FontDataSize);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
    SavedFile: THandle;           // manejador del fichero abierto

```

```

    TextSize: LongInt;           // el tamaño del texto en el memo
    TheText: PChar;             // almacena el texto en el memo
    BytesRead: DWORD;           // el número de bytes leídos del fichero
    BytesWritten: DWORD;        // el número de bytes escritos en el fichero
    FontData: Pointer;           // apunta a los datos de la fuente
    FontDataSize: Integer;       // tamaño de los datos de la fuente
    NewFontFile: THandle;        // manejador del fichero de la fuente
    CurDir: array[0..MAX_PATH] of Char; // almacena la ruta del directorio actual
begin
    {Abre el documento que contiene la fuente incrustada}
    SavedFile := CreateFile('ProprietaryFileFormat.PFF', GENERIC_READ, 0, nil,
                           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL or
                           FILE_FLAG_SEQUENTIAL_SCAN, 0);

    {Lee el número de bytes que ocupa el texto del documento}
    ReadFile(SavedFile, TextSize, SizeOf(TextSize), BytesRead, nil);

    {Reserva el tamaño de buffer necesario para almacenar el texto del documento}
    GetMem(TheText, TextSize);

    {Inicializa el buffer con caracteres nulos}
    FillMemory(TheText, TextSize, 0);

    {Asigna explícitamente al puntero del fichero la dirección siguiente a los
     cuatro primeros bytes, así la lectura comienza al principio del texto del
     documento}
    SetFilePointer(SavedFile, SizeOf(TextSize), nil, FILE_BEGIN);

    {Lee desde el fichero la cantidad indicada de bytes del texto}
    ReadFile(SavedFile, TheText^, TextSize, BytesRead, nil);

    {Asigna explícitamente al puntero del fichero la dirección siguiente al texto
     del documento. Este debe ahora estar apuntando al entero que indica el tamaño
     de los datos de la fuente incrustada}
    SetFilePointer(SavedFile, SizeOf(TextSize) + TextSize, nil, FILE_BEGIN);

    {Lee el tamaño de los datos de la fuente incrustada}
    ReadFile(SavedFile, FontDataSize, SizeOf(FontData), BytesRead, nil);

    {Recupera suficiente memoria para almacenar los datos de la fuente}
    GetMem(FontData, FontDataSize);

    {Asigna explícitamente al puntero del fichero la dirección siguiente a los
     cuatro bytes que contienen el tamaño de los datos de la fuente. Este debe
     estar ahora apuntando al principio de los datos de la fuente}
    SetFilePointer(SavedFile, SizeOf(TextSize) + TextSize + SizeOf(FontData),
                   nil, FILE_BEGIN);

    {Lee los datos de la fuente en el buffer de los datos fuente}
    ReadFile(SavedFile, FontData^, FontDataSize, BytesRead, nil);

    {Cierra el fichero del documento}
    CloseHandle(SavedFile);

```

{En este punto, la aplicación debe determinar, basándose en la información almacenada en el fichero del documento, si la fuente es *de-sólo-lectura* o de *lectura-escritura*. Si es de *lectura-escritura*, puede ser escrita directamente al subdirectorio Fonts bajo el directorio Windows. Si es *de-sólo-lectura*, debe ser escrita a un fichero oculto. En este ejemplo, escribiremos la fuente como un fichero TTF corriente en el directorio de la aplicación.}

```
{Crea el fichero que contendrá la información de la fuente}
NewFontFile := CreateFile('TempFont.TTF', GENERIC_WRITE, 0, nil,
                        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL or
                        FILE_FLAG_SEQUENTIAL_SCAN, 0);
```

```
{Escribe los datos de la fuente en el fichero}
WriteFile(NewFontFile, FontData^, FontDataSize, BytesWritten, nil);
```

```
{Descarga los buffers del fichero para asegurar que es escrito en el disco}
FlushFileBuffers(NewFontFile);
```

```
{Cierra el fichero de la fuente}
CloseHandle(NewFontFile);
```

```
{Recupera el directorio actual}
GetCurrentDirectory(MAX_PATH, @CurDir[0]);
```

```
{Debido a que la fuente fue escrita como un fichero TTF corriente, crea el
 fichero de recursos de la fuente, indicando que es de lectura-escritura}
CreateScalableFontResource(0, PChar(CurDir + '\TempFont.fot'),
                          PChar(CurDir + '\TempFont.ttf'), nil);
```

```
{Añade la fuente a la tabla interna de fuentes}
AddFontResource(PChar(CurDir + '\TempFont.fot'));
```

```
{Informa a otras aplicaciones que la tabla de fuentes ha cambiado}
SendMessage(HWND_BROADCAST, WM_FONTCHANGE, 0, 0);
```

```
{Asigna el documento recuperado al memo}
Memo1.Text := Copy(string(TheText), 0, StrLen(TheText));
```

```
{Libera el buffer de texto reservado}
FreeMem(TheText, TextSize);
```

```
{La fuente instalada fue 'Ventilate', se lo indicamos al memo}
Memo1.Font.Name := 'Ventilate';
Memo1.Font.Size := 16;
```

```
{Libera el buffer reservado para almacenar los datos de la fuente}
FreeMem(FontData, FontDataSize);
```

```
{Ahora que la fuente ha sido instalada, habilitar el botón de guardar documento}
Button1.Enabled := TRUE;
```

**end;**

**procedure** TForm1.FormDestroy(Sender: TObject);

**var**

```

CurDir: array[0..MAX_PATH] of Char; // almacena el directorio actual
begin
  {Recupera el directorio actual}
  GetCurrentDirectory(MAX_PATH, @CurDir[0]);

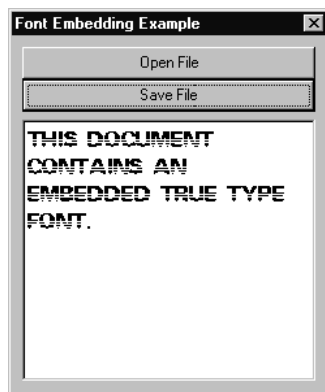
  {Borra la fuente de la tabla de fuentes interna}
  RemoveFontResource(PChar(CurDir + '\TempFont.fot'));

  {Informa a todas las aplicaciones del cambio en la tabla de fuentes}
  SendMessage(HWND_BROADCAST, WM_FONTCHANGE, 0, 0);

  {La aplicación (y el documento) están siendo cerrados, borramos el fichero de
  recursos de fuentes y el fichero de fuentes del disco, como si fuera una
  fuente de-sólo-lectura}
  DeleteFile(CurDir + '\TempFont.fot');
  DeleteFile(CurDir + '\TempFont.ttf');
end;

```

Figura 8-2:  
Este  
documento  
usa una fuente  
incrustada  
True Type



## Funciones de salida de texto

Las siguientes funciones de salida de texto serán tratadas en este capítulo:

**Tabla 8-2: Funciones de salida de texto**

Función	Descripción
AddFontResource	Añade el recurso de fuente contenido en el fichero especificado a la tabla de fuentes interna de Windows.
CreateFont	Crea una fuente lógica.
CreateFontIndirect	Crea una fuente lógica basada en la información especificada en un registro de datos.
CreateScalableFontResource	Crea un fichero de recursos de fuentes desde un fichero de fuentes True Type.
DrawText	Dibuja texto formateado sobre el contexto de dispositivo dentro del rectángulo especificado.

Función	Descripción
DrawTextEx	Dibuja texto formateado sobre el contexto de dispositivo dentro del rectángulo especificado, de acuerdo con el ancho de margen indicado.
EnumFontFamilies	Enumera las fuentes instaladas.
EnumFontFamiliesEx	Enumera las fuentes instaladas que se corresponden con las características de fuentes especificadas.
GetCharABCWidths	Recupera el ancho y espaciado de caracteres para fuentes True Type.
GetCharWidth	Recupera el ancho de caracteres.
GetFontData	Recupera la información del fichero de fuentes True Type.
GetGlyphOutline	Recupera un mapa de bits o delinea un carácter True Type.
GetKerningPairs	Recupera pares de kerning de caracteres.
GetOutlineTextMetrics	Recupera medidas de texto para fuentes True Type.
GetRasterizerCaps	Recupera información concerniente a la accesibilidad de fuentes True Type.
GetTabbedTextExtent	Recupera el ancho y alto de una cadena de caracteres que contiene tabulaciones.
GetTextAlign	Recupera la alineación del texto.
GetTextCharacterExtra	Recupera el espaciado entre caracteres.
GetTextColor	Recupera el color usado para dibujar texto.
GetTextExtentExPoint	Recupera la cantidad de caracteres en una cadena que se adecuará a un espacio especificado.
GetTextExtentPoint32	Recupera el ancho y altura de una cadena especificada.
GetTextFace	Recupera el nombre de una fuente seleccionada en un contexto de dispositivo.
GetTextMetrics	Recupera las medidas del texto para una fuente.
RemoveFontResource	Elimina un recurso de fuente de la tabla interna de fuentes de Windows.
SetTextAlign	Asigna el modo de alineación de texto.
SetTextCharacterExtra	Asigna el espaciado entre caracteres.
SetTextColor	Asigna el color usado para dibujar texto.
SetTextJustification	Asigna el modo de justificación de texto.
TabbedTextOut	Dibuja una cadena sobre un contexto de dispositivo, expandiendo las tabulaciones.
TextOut	Dibuja texto sobre un contexto de dispositivo.

**AddFontResource****Windows.Pas****Sintaxis**

```
AddFontResource(
    p1: PChar           {nombre de fichero de recursos de fuentes}
); Integer;             {devuelve la cantidad de fuentes añadidas}
```

**Descripción**

Esta función añade el recurso de fuentes contenido en el fichero de recursos de fuente especificado, a la tabla interna de fuentes del sistema, haciendo la fuente accesible a todas las aplicaciones. Si la fuente es añadida exitosamente a la tabla interna, la aplicación que añade la fuente debe informar a las demás aplicaciones del cambio. Esto es llevado a cabo enviando el mensaje *WM\_FONTCHANGE* mediante la función *SendMessage*, especificando *HWND\_BROADCAST* como valor para el parámetro *hWnd*. Cuando la fuente deje de ser necesaria, deberá ser borrada de la tabla interna de fuentes del sistema llamando a la función *RemoveFontResource*.

**Parámetros**

*p1*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso de fuentes a añadir. El fichero especificado puede contener recursos de fuentes (\*.FON), una fuente en bruto en mapa de bits (\*.FNT), información en bruto de fuente True Type (\*.TTF) o un recurso de fuentes True Type (\*.FOT).

**Valor que devuelve**

Si la función tiene éxito, devuelve el número de fuentes que fueron añadidas a la tabla interna de fuentes del sistema; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateScalableFontResource*, *GetFontData*, *RemoveFontResource*

**Ejemplo**

Vea el Listado 8-4 bajo *CreateScalableFontResource*.

**CreateFont****Windows.Pas****Sintaxis**

```
CreateFont(
    nHeight: Integer;    {altura de la fuente, en unidades lógicas}
    nWidth: Integer;     {ancho promedio de los caracteres, en unidades lógicas}
    nEscapement: Integer; {ángulo de inclinación}
    nOrientation: Integer; {ángulo de la línea base de carácter}
```

<code>fnWeight: Integer;</code>	{grosor de la negrita}
<code>fdwItalic: DWORD;</code>	{indicador de itálica}
<code>fdwUnderline: DWORD;</code>	{indicador del subrayado}
<code>fdwStrikeOut: DWORD;</code>	{indicador de tachado}
<code>fdwCharSet: DWORD;</code>	{conjunto de caracteres}
<code>fdwOutputPrecision: DWORD;</code>	{indicador de precisión de salida}
<code>fdwClipPrecision: DWORD;</code>	{opciones de precisión de recorte}
<code>fdwQuality: DWORD;</code>	{indicador de calidad de salida}
<code>fdwPitchAndFamily: DWORD;</code>	{opciones de pitch y familia de fuente}
<code>lpszFace: PChar</code>	{nombre del tipo de letra}
<code>); HFONT;</code>	{devuelve un manejador de la nueva fuente}

### Descripción

Esta función crea una fuente lógica que se corresponde con los atributos de fuente especificados. Esta fuente puede ser seleccionada en un contexto de dispositivo que soporte funciones de salida de texto. Cuando la fuente deje de ser necesaria, deberá ser eliminada llamando a la función *DeleteObject*.

### Parámetros

*nHeight*: Especifica la altura del carácter o de las celdas de caracteres de la fuente. La altura del carácter es la medida del valor de la altura de la celda del carácter, menos el valor de la guía interna. Este valor se expresa en unidades lógicas y dependerá del modo de mapeado actual. El mapeador de fuentes de Windows interpreta el valor del parámetro *nHeight* según se describe en la Tabla 8-3 y recuperará la fuente más grande disponible, hasta el tamaño especificado. Para el modo de mapeado *MM\_TEXT*, utilice la siguiente fórmula para expresar la altura de una fuente para un tamaño de punto específico:

```
nHeight := -MulDiv(PointSize, GetDeviceCaps(hDeviceContext, LOGPIXELSY), 72);
```

*nWidth*: Especifica el ancho promedio de los caracteres dentro de la fuente. Este valor se expresa en unidades lógicas y dependerá del modo de mapeado actual. Si a este parámetro se le asigna cero, el mapeador de fuentes de Windows elegirá una fuente apropiada, basándose en el valor absoluto de la diferencia entre la relación de aspecto del dispositivo actual y la relación de aspecto digitalizada de todas las fuentes apropiadas.

*nEscapement*: Especifica el ángulo entre la línea base de una línea de texto y el eje X, en décimas de grado. Bajo Windows NT, si al modo gráfico se le asigna *GM\_ADVANCED*, el ángulo de una línea de texto y el ángulo de cada carácter dentro de esa línea de texto pueden ser asignados independientemente. Si al modo gráfico se le asigna *GM\_COMPATIBLE*, el parámetro *nEscapement* especifica el ángulo para ambos, la línea de texto y los caracteres dentro de ella, y a los parámetros *nEscapement* y *nOrientation* se les debe asignar el mismo valor. Bajo Windows 95/98, el parámetro *nEscapement* siempre especifica el ángulo para ambos, la línea de texto y los caracteres

dentro de ella, y a los parámetros *nEscapement* y *nOrientation* se les debe asignar el mismo valor.

*nOrientation*: Especifica el ángulo entre la línea base de cada carácter individual y el eje X, en décimas de grado. Bajo Windows 95/98, el parámetro *nEscapement* siempre especifica el ángulo para ambos, la línea de texto y los caracteres dentro de esa línea de texto, y a los parámetros *nEscapement* y *nOrientation* se les debe asignar el mismo valor.

*fnWeight*: Especifica el grosor de la negrita de la fuente. El valor de este parámetro puede estar en el rango de 0 a 1000, o se le puede asignar un valor de la Tabla 8-4. Un valor cero indica el grosor por defecto de la negrita para la fuente especificada.

*fdwItalic*: Especifica el atributo de itálica para la fuente. Si a este parámetro se le asigna TRUE, la fuente será mostrada en *itálica*.

*fdwUnderline*: Especifica el atributo de subrayado para la fuente. Si a este parámetro se le asigna TRUE, la fuente será subrayada.

*fdwStrikeOut*: Especifica el atributo de tachado para la fuente. Si a este parámetro se le asigna TRUE, la fuente será ~~tachada~~.

*fdwCharSet*: Especifica el conjunto de caracteres que el mapeador de fuentes de Windows utiliza para elegir una fuente apropiada y al que se le puede asignar un valor de la Tabla 8-5. El nombre del tipo de letra especificado en el parámetro *lpszFace* tiene que ser una fuente que defina caracteres para el conjunto de caracteres especificado. Si a este parámetro se le asigna *DEFAULT\_CHARSET*, el tamaño de la fuente y el nombre del tipo de letra serán usados para encontrar una fuente apropiada. Sin embargo, si el nombre del tipo de letra no es encontrado, cualquier fuente de cualquier conjunto cuyas características se correspondan con las especificadas podrá ser usada, lo que puede producir resultados inesperados.

*fdwOutputPrecision*: Especifica cómo la fuente resultante tiene que corresponderse con los valores especificados para la altura, ancho, orientación de caracteres, inclinación, *pitch* y tipo de fuente. A este parámetro se le puede asignar un valor de la Tabla 8-6. Observe que las opciones *OUT\_DEVICE\_PRECIS*, *OUT\_RASTER\_PRECIS* y *OUT\_TT\_PRECIS* controlan el comportamiento del mapeador de fuentes de Windows cuando exista más de una fuente con el mismo nombre especificado por el parámetro *lpszFace*.

*fdwClipPrecision*: Especifica cómo son dibujados los caracteres que queden parcialmente fuera del área de recorte. A este parámetro se le puede asignar uno o más valores de la Tabla 8-7.

*fdwQuality*: Especifica cómo el mapeador de fuentes de Windows hace corresponder los atributos de fuente especificados con una fuente real. A este parámetro se le puede asignar un valor de la Tabla 8-8.

*fdwPitchAndFamily*: Las opciones de *pitch* y familia de fuente. Este parámetro puede contener una combinación de un valor de la tabla de opciones de *pitch* (Tabla 8-9) y un

valor de la tabla de opciones de familia de fuentes (Tabla 8-10). Los valores de estas tablas son combinados mediante el operador booleano **or**. El *pitch* describe cómo varía el ancho de los patrones de caracteres individuales, y la familia describe la apariencia general de la fuente. Si el nombre del tipo de letra especificado es inaccesible, la función devuelve la fuente más cercana que pertenezca a la familia especificada.

*lpzFace*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del tipo de letra de la fuente. El nombre del tipo de letra no puede exceder los 32 caracteres de largo, incluyendo el terminador nulo. Utilice la función *EnumFontFamilies* para recuperar una lista de todos los nombres de tipos de letra de fuentes instaladas. Si a este parámetro se le asigna **nil**, el mapeador de fuentes de Windows elegirá la primera fuente de la familia de fuentes especificada que se corresponda con los atributos especificados.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de la fuente lógica recién creada; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*CreateFontIndirect*, *DeleteObject*, *SelectObject*, *EnumFontFamilies*,  
*EnumFontFamiliesEx*

#### Ejemplo

##### Listado 8-2: Creando varias fuentes

```
procedure TForm1.FormPaint(Sender: TObject);
var
    NewFont, OldFont: HFont;    // almacena las fuentes antigua y nueva
begin
    {Pone el modo de fondo en transparente}
    SetBkMode(Form1.Canvas.Handle, TRANSPARENT);

    {Crea una fuente en negrita}
    NewFont := CreateFont(-MulDiv(16, GetDeviceCaps(Form1.Canvas.Handle,
        LOGPIXELSY), 72), 0, 0, 0, FW_BOLD, 0, 0, 0,
        DEFAULT_CHARSET, OUT_TT_ONLY_PRECIS,
        CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
        FF_DONTCARE, 'Arial');

    {Selecciona la fuente en el contexto de dispositivo del formulario}
    OldFont := SelectObject(Form1.Canvas.Handle, NewFont);

    {Muestra una línea de texto}
    TextOut(Form1.Canvas.Handle, 8, Label1.Top + Label1.Height, 'Delphi Rocks!',
        Length('Delphi Rocks!'));

    {Selecciona la antigua fuente en el contexto de dispositivo y elimina la nueva}
    SelectObject(Form1.Canvas.Handle, OldFont);
```

```

DeleteObject(NewFont);

{Crea una fuente tachada}
NewFont := CreateFont(-MulDiv(16, GetDeviceCaps(Form1.Canvas.Handle,
LOGPIXELSY), 72), 0, 0, 0, FW_DONTCARE, 0, 0, 1,
DEFAULT_CHARSET, OUT_TT_ONLY_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
FF_ROMAN, '');

{Selecciona la fuente en el contexto de dispositivo del formulario}
OldFont := SelectObject(Form1.Canvas.Handle, NewFont);

{Muestra una línea de texto}
TextOut(Form1.Canvas.Handle, 8, Label2.Top + Label2.Height, 'Delphi Rocks!',
Length('Delphi Rocks!'));

{Selecciona la antigua fuente en el contexto de dispositivo y elimina la nueva}
SelectObject(Form1.Canvas.Handle, OldFont);
DeleteObject(NewFont);

{Crea una fuente subrayada}
NewFont := CreateFont(-MulDiv(16, GetDeviceCaps(Form1.Canvas.Handle,
LOGPIXELSY), 72), 0, 0, 0, FW_DONTCARE, 0, 1, 0,
DEFAULT_CHARSET, OUT_TT_ONLY_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
FF_DECORATIVE, '');

{Selecciona la fuente en el contexto de dispositivo del formulario}
OldFont := SelectObject(Form1.Canvas.Handle, NewFont);

{Muestra una línea}
TextOut(Form1.Canvas.Handle, 8, Label3.Top + Label3.Height, 'Delphi Rocks!',
Length('Delphi Rocks!'));

{Selecciona la antigua fuente en el contexto de dispositivo y elimina la nueva}
SelectObject(Form1.Canvas.Handle, OldFont);
DeleteObject(NewFont);

{Crea una fuente itálica}
NewFont := CreateFont(-MulDiv(16, GetDeviceCaps(Form1.Canvas.Handle,
LOGPIXELSY), 72), 0, 0, 0, FW_DONTCARE, 1, 0, 0,
DEFAULT_CHARSET, OUT_TT_ONLY_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
FF_SCRIPT, '');

{Selecciona la fuente en el contexto de dispositivo del formulario}
OldFont := SelectObject(Form1.Canvas.Handle, NewFont);

{Muestra una línea de texto}
TextOut(Form1.Canvas.Handle, 8, Label4.Top + Label4.Height, 'Delphi Rocks!',
Length('Delphi Rocks!'));

{Selecciona la vieja fuente en el contexto de dispositivo y borra la nueva}

```

```
SelectObject(Form1.Canvas.Handle, OldFont);
DeleteObject(NewFont);
end;
```

Figura 8-3:  
Varias fuentes  
creadas con la  
función  
CreateFont



Tabla 8-3: Interpretación de los valores del parámetro nHeight de CreateFont

Valor	Descripción
nHeight > 0	El mapeador de fuentes convierte el valor de nHeight a unidades del dispositivo, haciendo corresponder el resultado con la altura de celda de las fuentes accesibles.
nHeight = 0	El mapeador de fuentes usa una altura de fuente por defecto para buscar una fuente adecuada.
nHeight < 0	El mapeador de fuentes convierte el valor de nHeight a unidades del dispositivo, haciendo corresponder el valor absoluto del resultado con la altura de los caracteres de las fuentes accesibles.

Tabla 8-4: Valores del parámetro fnWeight de CreateFont

Valor	Descripción
FW_DONTCARE	Utiliza el grosor por defecto de la negrita (0).
FW_THIN	Trazo de fuente extrafino (100).
FW_EXTRALIGHT	Trazo de fuente fino (200).
FW_LIGHT	Negrita con grosor ligeramente inferior a lo normal (300).
FW_NORMAL	Negrita normal (400).
FW_MEDIUM	Negrita con grosor ligeramente superior a lo normal (500).
FW_SEMIBOLD	Negrita ligera (600).
FW_BOLD	Negrita (700).
FW_EXTRABOLD	Negrita extra (800).
FW_HEAVY	Negrita extragruesa (900).

Tabla 8-5: Valores del parámetro `fdwCharSet` de `CreateFont`

Valor	Descripción
ANSI_CHARSET	El conjunto de caracteres ANSI.
DEFAULT_CHARSET	El conjunto de caracteres por defecto.
SYMBOL_CHARSET	El conjunto de caracteres Symbol.
SHIFTJIS_CHARSET	El conjunto de caracteres Shift-JIS.
GB2312_CHARSET	El conjunto de caracteres GB2312.
HANGEUL_CHARSET	El conjunto de caracteres coreano.
CHINESEBIG5_CHARSET	El conjunto de caracteres chino.
OEM_CHARSET	El conjunto de caracteres original del fabricante del equipo.
JOHAB_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Johab.
HEBREW_CHARSET	Sólo Windows 95/98: El conjunto de caracteres hebreo.
ARABIC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres árabe.
GREEK_CHARSET	Sólo Windows 95/98: El conjunto de caracteres griego.
TURKISH_CHARSET	Sólo Windows 95/98: El conjunto de caracteres turco.
VIETNAMESE_CHARSET	Sólo Windows 95/98: El conjunto de caracteres vietnamita.
THAI_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Thai.
EASTEUROPE_CHARSET	Sólo Windows 95/98: El conjunto de caracteres de Europa del Este.
RUSSIAN_CHARSET	Sólo Windows 95/98: El conjunto de caracteres cirílico.
MAC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Macintosh.
BALTIC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres báltico.

Tabla 8-6: Valores del parámetro `fdwOutputPrecision` de `CreateFont`

Valor	Descripción
OUT_DEFAULT_PRECIS	El comportamiento por defecto del mapeador de fuentes.
OUT_DEVICE_PRECIS	Elige una fuente de dispositivo cuando existe más de una fuente con el nombre especificado.
OUT_OUTLINE_PRECIS	Sólo Windows NT: Elige una fuente True Type o de otros tipos basados en vectores.
OUT_RASTER_PRECIS	Elige una fuente de barrido cuando existe más de una fuente con el nombre especificado.
OUT_STROKE_PRECIS	Sólo Windows NT: No usado por el mapeador de fuentes. Sin embargo, este indicador es devuelto cuando son enumeradas fuentes True Type u otras fuentes vectoriales.  Sólo Windows 95/98: Elige una fuente de las fuentes basadas en vectores.

Valor	Descripción
OUT_TT_ONLY_PRECIS	Elige una fuente sólo de las fuentes True Type. Si no existen ninguna fuente True Type, el mapeador de fuentes retorna al comportamiento por defecto.
OUT_TT_PRECIS	Elige una fuente True Type cuando existe más de una fuente con el nombre especificado.

Tabla 8-7: Valores del parámetro `fdwClipPrecision` de `CreateFont`

Valor	Descripción
CLIP_DEFAULT_PRECIS	El comportamiento predeterminado del recorte.
CLIP_STROKE_PRECIS	Este indicador es usado sólo durante la enumeración de fuentes.
CLIP_EMBEDDED	Este indicador tiene que ser incluido cuando se usa una fuente incrustada <i>de-sólo-lectura</i> .
CLIP_LH_ANGLES	Especifica que la rotación de la fuente es dependiente del sistema de coordenadas. Si este indicador no es especificado, las fuentes de dispositivo siempre rotarán en sentido contrario a las manecillas del reloj.

Tabla 8-8: Valores del parámetro `fdwQuality` de `CreateFont`

Valor	Descripción
DEFAULT_QUALITY	Utiliza la calidad predeterminada de la fuente.
DRAFT_QUALITY	El escalamiento de las fuentes de barrido está habilitado y las fuentes negritas, itálicas, subrayadas y tachadas serán construidas cuando se necesiten. La correspondencia exacta de los atributos es más importante que la calidad de la fuente.
PROOF_QUALITY	El escalamiento de las fuentes de barrido está deshabilitado y se elige la fuente con un tamaño físico más cercano al especificado. Las fuentes negritas, itálicas, subrayadas y tachadas serán construidas cuando se necesiten. La calidad de la fuente es más importante que la correspondencia exacta de los atributos.

Tabla 8-9: Valores de las opciones de pitch para el parámetro `fdwPitchAndFamily` de `CreateFont`

Valor	Descripción
DEFAULT_PITCH	Utiliza el pitch predeterminado de la fuente.
FIXED_PITCH	El ancho de todos los patrones de caracteres es el mismo.
VARIABLE_PITCH	El ancho de los patrones de los caracteres varía en dependencia de la imagen individual del patrón.

**Tabla 8-10: Valores de las opciones de familias de fuentes para el parámetro `fdwPitchAndFamily` de `CreateFont`**

Valor	Descripción
FF_DECORATIVE	Indica una fuente decorativa, como Old English.
FF_DONTCARE	El estilo general de la fuente es desconocido o no es importante.
FF_MODERN	Indica una fuente monoespaciada con trazos de ancho consistentes, con o sin serifs, como Courier New.
FF_ROMAN	Indica una fuente proporcional con trazos de ancho variable y con serifs, como Times New Roman.
FF_SCRIPT	Indica una fuente que imita el manuscrito, como Brush Script.
FF_SWISS	Indica una fuente proporcional con trazos de ancho variable y sin serifs, como Arial.

### **CreateFontIndirect**

### **Windows.Pas**

#### **Sintaxis**

```
CreateFontIndirect(
    const p1: TLogFont      {puntero a registro de fuente lógica}
): HFONT;                  {devuelve un manejador de la nueva fuente}
```

#### **Descripción**

Esta función crea una fuente lógica, haciéndole corresponder los atributos de fuente especificados en el registro *TLogFont* al que apunta el parámetro *p1*. Esta fuente puede ser seleccionada en cualquier contexto de dispositivo que soporte funciones de salida de texto. Cuando la fuente deje de ser necesaria, deberá ser eliminada usando la función *DeleteObject*.

#### **Parámetros**

*p1*: Puntero a un registro *TLogFont* que describe los atributos de la fuente deseada. El registro *TLogFont* se define como:

*TLogFont* = **packed record**

<i>lfHeight</i> : Longint;	{altura de la fuente en unidades lógicas}
<i>lfWidth</i> : Longint;	{ancho promedio del carácter}
<i>lfEscapement</i> : Longint;	{ángulo del vector de inclinación}
<i>lfOrientation</i> : Longint;	{ángulo de la línea base del carácter}
<i>lfWeight</i> : Longint;	{grosor de la negrita}
<i>lfItalic</i> : Byte;	{indicador de itálica}
<i>lfUnderline</i> : Byte;	{indicador de subrayado}
<i>lfStrikeOut</i> : Byte;	{indicador de tachado}
<i>lfCharSet</i> : Byte;	{conjunto de caracteres}

lfOutPrecision: Byte;	{indicador de precisión de salida}
lfClipPrecision: Byte;	{indicador de precisión de recorte}
lfQuality: Byte;	{indicador de calidad de salida}
lfPitchAndFamily: Byte;	{indicador de pitch y familia}
lfFaceName: <b>array</b> [0..LF_FACESIZE - 1] <b>of</b> AnsiChar;	{nombre del tipo de letra}

**end;**

*lfHeight*: Especifica la altura del carácter o de las celdas de caracteres de la fuente. La altura del carácter es la medida de la altura de la celda del carácter menos el valor de la guía interna. Este valor se expresa en unidades lógicas y dependerá del modo de mapeado actual. El mapeador de fuentes de Windows interpreta el valor del campo *lfHeight* según se describe en la Tabla 8-11 y recuperará la fuente accesible más grande para el tamaño especificado. Para el modo de mapeado *MM\_TEXT*, utilice la siguiente fórmula para expresar la altura de una fuente para un tamaño de punto específico:

```
lfHeight := -MulDiv(PointSize, GetDeviceCaps(hDeviceContext, LOGPIXELSY), 72);
```

*lfWidth*: Especifica el ancho promedio de los caracteres de la fuente. Este valor se expresa en unidades lógicas y dependerá del modo de mapeado actual. Si a este campo se le asigna cero, el mapeador de fuentes de Windows elegirá una fuente apropiada, basándose en los valores absolutos de la diferencia entre la relación de aspecto del dispositivo actual y la relación de aspecto digitalizada de las fuentes apropiadas.

*lfEscapement*: Especifica el ángulo entre la línea base de una línea de texto y el eje X, en décimas de grado. Bajo Windows NT, si al modo gráfico se le asigna *GM\_ADVANCED*, el ángulo de una línea de texto y el ángulo de cada carácter dentro de esa línea de texto pueden ser asignados independientemente. Si al modo gráfico se le asigna *GM\_COMPATIBLE*, el parámetro *lfEscapement* especifica el ángulo para ambos, la línea de texto y los caracteres dentro de ella, y a los parámetros *lfEscapement* y *lfOrientation* se les debe asignar el mismo valor. Bajo Windows 95/98, el parámetro *lfEscapement* siempre especifica el ángulo para ambos, la línea de texto y los caracteres dentro de ella, y a los parámetros *lfEscapement* y *lfOrientation* se les debe asignar el mismo valor.

*lfOrientation*: Especifica el ángulo entre la línea base de cada carácter individual y el eje X, en décimas de grado. Bajo Windows 95/98, el campo *lfEscapement* siempre especifica el ángulo para ambos, la línea de texto y los caracteres dentro de esa línea, y a los campos *lfEscapement* y *lfOrientation* se les debe asignar el mismo valor.

*lfWeight*: Especifica el grosor de la negrita de la fuente. El valor de este campo puede estar en el rango 0 a 1000, o se le puede asignar un valor de la Tabla 8-12. Un valor cero indica el valor del grosor predeterminado de la negrita para la fuente especificada.

*lfItalic*: Especifica el atributo de itálica para la fuente. Si a este campo se le asigna TRUE, la fuente será mostrada en *itálica*.

*lfUnderline*: Especifica el atributo de subrayado para la fuente. Si a este campo se le asigna TRUE, la fuente será subrayada.

*lfStrikeOut*: Especifica el atributo de tachado para la fuente. Si a este campo se le asigna TRUE, la fuente será ~~tachada~~.

*lfCharSet*: Especifica el conjunto de caracteres que utilizará el mapeador de fuentes de Windows para elegir una fuente apropiada, y puede tomar un valor de la Tabla 8-13. El nombre del tipo de letra de la fuente, especificado en el campo *lfFaceName*, tiene que corresponder a una fuente que defina caracteres del conjunto especificado. Si a este campo se le asigna *DEFAULT\_CHARSET*, el tamaño de la fuente y el nombre del tipo de letra serán usados para encontrar una fuente apropiada. Sin embargo, si el nombre del tipo de letra no se encuentra, cualquier otra fuente de un conjunto de caracteres que se corresponda con los valores especificados podrá ser utilizado, lo que puede provocar resultados imprevisibles.

*lfOutPrecision*: Especifica cómo la fuente resultante tiene que corresponderse con los valores de altura, ancho, orientación del carácter, escape, *pitch* y tipo de fuente. A este campo se le puede asignar un valor de la Tabla 8-14. Observe que las opciones *OUT\_DEVICE\_PRECIS*, *OUT\_RASTER\_PRECIS* y *OUT\_TT\_PRECIS* controlan el comportamiento del mapeador de fuentes de Windows cuando existe más de una fuente con el nombre especificado en el campo *lfFaceName*.

*lfClipPrecision*: Especifica cómo serán dibujados los caracteres que estén parcialmente fuera del área de recorte. A este campo se le puede asignar uno a más valores de la Tabla 8-15.

*lfQuality*: Especifica cómo el mapeador de fuentes de Windows hace corresponder los atributos de la fuente especificada con la fuente actual. A este campo se le puede asignar un valor de la Tabla 8-16.

*lfPitchAndFamily*: Las opciones de *pitch* y familia de fuentes. Este campo puede contener una combinación de un valor de la tabla de opciones de *pitch* (Tabla 8-17) y un valor de la tabla de opciones de familia de fuentes (Tabla 8-18). Los valores de estas tablas se combinan mediante el operador booleano **or**. El *pitch* describe cómo varía el ancho del patrón de los caracteres individuales, y la familia describe la apariencia de la fuente. Si el nombre del tipo de letra no es accesible, la función devuelve la fuente más cercana de la familia especificada.

*lfFaceName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del tipo de letra de la fuente. El nombre del tipo de letra de la fuente no puede exceder los 32 caracteres de longitud, incluyendo el terminador nulo. Utilice la función *EnumFontFamilies* para recuperar una lista de todos los nombres de tipos de letras de fuentes instaladas. Si este campo es **nil**, el mapeador de fuentes de Windows elegirá la primera fuente de la familia especificada que se corresponda con los atributos especificados.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de la fuente lógica recién creada; en caso contrario, devuelve cero.

**Véase además**

*CreateFont, DeleteObject, EnumFontFamilies, EnumFontFamiliesEx, SelectObject*

**Ejemplo****Listado 8-3: Creando una fuente indirectamente**

```

procedure TForm1.FormPaint(Sender: TObject);
var
    FontInfo: TLogFont;      // registro de fuente lógica
    NewFont, OldFont: HFont; // fuentes antigua y nueva
begin
    {Establece el modo de fondo transparente}
    SetBkMode(Form1.Canvas.Handle, TRANSPARENT);

    {Inicializa la información de la fuente lógica, estableciendo el grosor e
    inclinación especificados por las barras de desplazamiento}
    with FontInfo do
    begin
        lfHeight := 24;
        lfWidth := 0;
        lfEscapement := TrackBar1.Position * 10;
        lfOrientation := TrackBar1.Position * 10;
        lfWeight := TrackBar2.Position;
        lfItalic := 0;
        lfUnderline := 0;
        lfStrikeOut := 0;
        lfFaceName := 'Arial';
    end;

    {Crea una nueva fuente}
    NewFont := CreateFontIndirect(FontInfo);

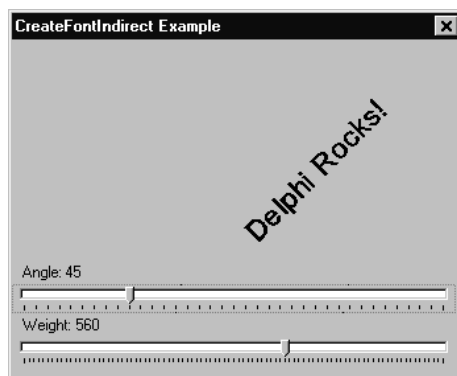
    {Selecciona la nueva fuente en el contexto de dispositivo del formulario}
    OldFont := SelectObject(Form1.Canvas.Handle, NewFont);

    {Muestra una cadena de texto rotado}
    TextOut(Form1.Canvas.Handle, Form1.Width div 2, 140, 'Delphi Rocks!',
        Length('Delphi Rocks!'));

    {Selecciona la fuente original en el contexto de dispositivo, y elimina la nueva}
    SelectObject(Form1.Canvas.Handle, OldFont);
    DeleteObject(NewFont);
end;

```

Figura 8-4:  
Una fuente  
rotada



**Tabla 8-II: Interpretación de los valores del campo `pl.lfHeight` de `CreateFontIndirect`**

Valor	Descripción
<code>nHeight &gt; 0</code>	El mapeador de fuentes convierte el valor de <code>nHeight</code> a unidades del dispositivo, haciendo corresponder el resultado con la altura de celda de las fuentes accesibles.
<code>nHeight = 0</code>	El mapeador de fuentes usa una altura de fuente por defecto para buscar una fuente adecuada.
<code>nHeight &lt; 0</code>	El mapeador de fuentes convierte el valor de <code>nHeight</code> a unidades del dispositivo, haciendo corresponder el valor absoluto del resultado con la altura de los caracteres de las fuentes accesibles.

**Tabla 8-I2: Valores del campo `pl.lfWeight` de `CreateFontIndirect`**

Valor	Descripción
<code>FW_DONTCARE</code>	Utiliza el grosor por defecto de la negrita (0).
<code>FW_THIN</code>	Trazo de fuente extrafino (100).
<code>FW_EXTRALIGHT</code>	Trazo de fuente fino (200).
<code>FW_LIGHT</code>	Negrita con grosor ligeramente inferior a lo normal (300).
<code>FW_NORMAL</code>	Negrita normal (400).
<code>FW_MEDIUM</code>	Negrita con grosor ligeramente superior a lo normal (500).
<code>FW_SEMIBOLD</code>	Negrita ligera (600).
<code>FW_BOLD</code>	Negrita (700).
<code>FW_EXTRABOLD</code>	Negrita extra (800).
<code>FW_HEAVY</code>	Negrita extragruesa (900).

Tabla 8-13: Valores del campo pl.IfCharSet de CreateFontIndirect

Valor	Descripción
ANSI_CHARSET	El conjunto de caracteres ANSI.
DEFAULT_CHARSET	El conjunto de caracteres por defecto.
SYMBOL_CHARSET	El conjunto de caracteres Symbol.
SHIFTJIS_CHARSET	El conjunto de caracteres Shift-JIS.
GB2312_CHARSET	El conjunto de caracteres GB2312.
HANGEUL_CHARSET	El conjunto de caracteres coreano.
CHINESEBIG5_CHARSET	El conjunto de caracteres chino.
OEM_CHARSET	El conjunto de caracteres original del fabricante del equipo.
JOHAB_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Johab.
HEBREW_CHARSET	Sólo Windows 95/98: El conjunto de caracteres hebreo.
ARABIC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres árabe.
GREEK_CHARSET	Sólo Windows 95/98: El conjunto de caracteres griego.
TURKISH_CHARSET	Sólo Windows 95/98: El conjunto de caracteres turco.
VIETNAMESE_CHARSET	Sólo Windows 95/98: El conjunto de caracteres vietnamita.
THAI_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Thai.
EASTEUROPE_CHARSET	Sólo Windows 95/98: El conjunto de caracteres de Europa del Este.
RUSSIAN_CHARSET	Sólo Windows 95/98: El conjunto de caracteres cirílico.
MAC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Macintosh.
BALTIC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres báltico.

Tabla 8-14: Valores del campo pl.IfOutputPrecision de CreateFontIndirect

Valor	Descripción
OUT_DEFAULT_PRECIS	El comportamiento por defecto del mapeador de fuentes.
OUT_DEVICE_PRECIS	Elige una fuente de dispositivo cuando existe más de una fuente con el nombre especificado.
OUT_OUTLINE_PRECIS	Sólo Windows NT: Elige una fuente True Type o de otros tipos basados en vectores.
OUT_RASTER_PRECIS	Elige una fuente de barrido cuando existe más de una fuente con el nombre especificado.

Valor	Descripción
OUT_STROKE_PRECIS	Sólo Windows NT: No usado por el mapeador de fuentes. Sin embargo, este indicador es devuelto cuando son enumeradas fuentes True Type u otras fuentes vectoriales. Sólo Windows 95/98: Elige una fuente de las fuentes basadas en vectores.
OUT_TT_ONLY_PRECIS	Elige una fuente sólo de las fuentes True Type. Si no existen ninguna fuente True Type, el mapeador de fuentes retorna al comportamiento por defecto.
OUT_TT_PRECIS	Elige una fuente True Type cuando existe más de una fuente con el nombre especificado.

Tabla 8-15: Valores del campo `pl.IfClipPrecision` de `CreateFontIndirect`

Valor	Descripción
CLIP_DEFAULT_PRECIS	El comportamiento predeterminado del recorte.
CLIP_STROKE_PRECIS	Este indicador es usado sólo durante la enumeración de fuentes.
CLIP_EMBEDDED	Este indicador tiene que ser incluido cuando se usa una fuente incrustada <i>de-sólo-lectura</i> .
CLIP_LH_ANGLES	Especifica que la rotación de la fuente es dependiente del sistema de coordenadas. Si este indicador no es especificado, las fuentes de dispositivo siempre rotarán en sentido contrario a las manecillas del reloj.

Tabla 8-16: Valores del campo `pl.IfQuality` de `CreateFontIndirect`

Valor	Descripción
DEFAULT_QUALITY	Utiliza la calidad predeterminada de la fuente.
DRAFT_QUALITY	El escalamiento de las fuentes de barrido está habilitado y las fuentes negritas, itálicas, subrayadas y tachadas serán construidas cuando se necesiten. La correspondencia exacta de los atributos es más importante que la calidad de la fuente.
PROOF_QUALITY	El escalamiento de las fuentes de barrido está deshabilitado y se elige la fuente con un tamaño físico más cercano al especificado. Las fuentes negritas, itálicas, subrayadas y tachadas serán construidas cuando se necesiten. La calidad de la fuente es más importante que la correspondencia exacta de los atributos.

**Tabla 8-17: Valores de las opciones de pitch para el campo `pl.IfPitchAndFamily` de `CreateFontIndirect`**

Valor	Descripción
DEFAULT_PITCH	Utiliza el pitch predeterminado de la fuente.
FIXED_PITCH	El ancho de todos los patrones de caracteres es el mismo.
VARIABLE_PITCH	El ancho de los patrones de los caracteres varía en dependencia de la imagen individual del patrón.

**Tabla 8-18: Valores de las opciones de familias de fuentes para el campo `pl.IfPitchAndFamily` de `CreateFontIndirect`**

Valor	Descripción
FF_DECORATIVE	Indica una fuente decorativa, como Old English.
FF_DONTCARE	El estilo general de la fuente es desconocido o no es importante.
FF_MODERN	Indica una fuente monoespaciada con trazos de ancho consistentes, con o sin serifs, como Courier New.
FF_ROMAN	Indica una fuente proporcional con trazos de ancho variable y con serifs, como Times New Roman.
FF_SCRIPT	Indica una fuente que imita los caracteres manuscritos, como Brush Script.
FF_SWISS	Indica una fuente proporcional con trazos de ancho variable y sin serifs, como Arial.

**CreateScalableFontResource      Windows.Pas****Sintaxis**

```
CreateScalableFontResource(
    p1: DWORD;           {indicador de sólo lectura}
    p2: PChar;           {nombre de fichero de recursos de la fuente}
    p3: PChar;           {nombre de fichero de la fuente escalable}
    p4: PChar;           {ruta de fichero de la fuente escalable}
): BOOL;               {devuelve TRUE o FALSE}
```

**Descripción**

Esta función es usada para crear un fichero de recursos de fuente que puede ser utilizado luego por la función *AddFontResource* para añadir fuentes True Type a la tabla de fuentes interna de Windows, lo que hará accesible a todas las aplicaciones la fuente True Type. Cuando una aplicación termina de utilizar la fuente True Type, deberá eliminarla del sistema llamando a la función *RemoveFontResource*.

### Parámetros

*pl*: Indica si la fuente es de sólo lectura. Si a este parámetro se le asigna cero, la fuente tiene permiso de lectura y escritura. El valor 1 indica que se trata de una fuente *de-sólo-lectura* y la fuente será ocultada en otras aplicaciones, y además no aparecerá en las listas de fuentes devueltas por las funciones `EnumFontFamilies` o `EnumFontFamiliesEx`.

*p2*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre de fichero y la extensión (normalmente .FOT) del fichero de recursos de fuente que será creado por esta función.

*p3*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del fichero de fuente True Type que será utilizado para crear el fichero de recursos de fuente escalable. Si esta cadena contiene únicamente el nombre de fichero y la extensión, el parámetro *p4* deberá apuntar a una cadena que contenga la ruta del fichero especificado.

*p4*: Puntero a una cadena de caracteres terminada en nulo con la ruta del fichero de fuentes escalables. Si el nombre de fichero especificado en el parámetro *p3* contiene un nombre de fichero con la ruta incluida, a este parámetro debe asignársele **nil**.

*Valor que devuelve*

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

*Véase además*

*AddFontResource, EnumFontFamilies, EnumFontFamiliesEx, RemoveFontResource*

### Ejemplo

### Listado 8-4: Instalando una nueva fuente True Type

```
{=====
    La fuente Ventilate usada en este ejemplo fue generosamente donada por su dueño
Copyright © 1997, Brian J. Bonislowsky - Astigmatic One Eye. Usada con permiso.
    Astigmatic One Eye es un gran proveedor de shareware y freeware de fuentes de
todo tipo. Visite su sitio http://www.comptechdev.com/cavop/aoe/
=====}
```

[illegible]

```

nil);

{Añade la fuente a la tabla interna de fuentes de Windows, haciéndola
 accesible a cualquier aplicación}
AddFontResource(PChar(CurDir + '\ventilat.fot'));

{Informa a todas las aplicaciones del cambio en la tabla de fuentes}
SendMessage(HWND_BROADCAST, WM_FONTCHANGE, 0, 0);
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  CurDir: array[0..MAX_PATH] of Char; // almacena el directorio actual
begin
  {Recupera el directorio actual}
  GetCurrentDirectory(MAX_PATH, @CurDir[0]);

  {Borra la fuente de la tabla interna de fuentes de Windows}
  RemoveFontResource(PChar(CurDir + '\ventilat.fot'));

  {Informa a todas las aplicaciones del cambio en la tabla de fuentes}
  SendMessage(HWND_BROADCAST, WM_FONTCHANGE, 0, 0);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
  NewFont, OldFont: HFont; // almacenan la fuente anterior y la nueva
begin
  {Asigna el modo del fondo transparente}
  SetBkMode(Form1.Canvas.Handle, TRANSPARENT);

  {Crea una fuente desde el recurso de fuentes recientemente instalado}
  NewFont := CreateFont(-MulDiv(48, GetDeviceCaps(Form1.Canvas.Handle,
    LOGPIXELSY), 72), 0, 0, 0, FW_DONTCARE, 0, 0, 0,
    DEFAULT_CHARSET, OUT_TT_ONLY_PRECIS,
    CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or
    FF_DONTCARE, 'Ventilate');

  {Selecciona la fuente en el contexto de dispositivo del formulario}
  OldFont := SelectObject(Form1.Canvas.Handle, NewFont);

  {Muestra una línea de texto}
  TextOut(Form1.Canvas.Handle, 8, 8, 'Delphi Rocks!', Length('Delphi Rocks!'));

  {Selecciona la fuente anterior en el contexto de dispositivo y elimina la nueva}
  SelectObject(Form1.Canvas.Handle, OldFont);
  DeleteObject(NewFont);
end;

```

Figura 8-5:  
Usando la  
nueva fuente



**DrawText****Windows.Pas****Sintaxis**

```

DrawText(
  hDC: HDC;           {manejador de contexto de dispositivo}
  lpString: PChar;    {cadena a dibujar}
  nCount: Integer;    {longitud de la cadena a dibujar}
  var lpRect: TRect;  {rectángulo de formato}
  uFormat: UINT       {opciones de formato de texto}
): Integer;           {devuelve la altura del texto dibujado}

```

**Descripción**

Esta función dibuja la cadena de texto indicada por *lpString* sobre el contexto de dispositivo especificado por el parámetro *hDC*. El texto es dibujado dentro del rectángulo especificado y es formateado de acuerdo a las opciones de formato especificadas en el parámetro *uFormat*. Para dibujar el texto se utilizan la fuente, color del texto, color del fondo y modo del fondo seleccionados en el contexto de dispositivo. A menos que se indique lo contrario mediante un indicador de formato específico, se asume que el texto tiene múltiples líneas y será recortado en los límites del rectángulo especificado.

Observe que las cadenas que contienen el carácter de prefijo mnemotécnico ('&') provocarán que se subraye el carácter siguiente a cada '&', y dos prefijos mnemotécnicos seguidos serán interpretados como un carácter '&' literal.

**Parámetros**

*hDC*: Manejador del contexto de dispositivo sobre el cual el texto será dibujado.

*lpString*: Puntero a una cadena de caracteres terminada en nulo que contiene el texto que será dibujado.

*nCount*: Especifica la longitud de la cadena a la que apunta el parámetro *lpString*, en caracteres. Si a este parámetro se le asigna -1, se asume que la cadena a la que apunta el parámetro *lpString* es una cadena de caracteres terminada en nulo y la función calculará automáticamente el longitud de la cadena.

*lpRect*: Especifica las coordenadas del rectángulo, en unidades lógicas, dentro del cual será dibujado y formateado el texto.

*uFormat*: Conjunto de indicadores que especifican cómo el texto será mostrado y formateado dentro del rectángulo especificado. Este parámetro puede contener uno o más valores de la Tabla 8-19.

**Valor que devuelve**

Si la función tiene éxito, devuelve la altura del texto en unidades lógicas; en caso contrario, devuelve cero.

Véase además

*DrawTextEx, GrayString, TabbedTextOut, TextOut*

Ejemplo

#### Listado 8-5: Dibujando texto formateado

```

procedure TForm1.FormPaint(Sender: TObject);
var
    BoundingRect: TRect;           // rectángulo de formato del texto
    CurDirectory: array[0..MAX_PATH] of Char; // directorio
begin
    {Crea el rectángulo límite para el texto}
    BoundingRect := Rect(Label1.Left, Label1.Top + Label1.Height + 3,
                        Form1.Width - (Label1.Left*2), Label1.Top + Label1.Height + 83);

    {Dibuja el rectángulo sobre el formulario}
    Form1.Canvas.Rectangle(BoundingRect.Left, BoundingRect.Top,
                        BoundingRect.Right, BoundingRect.Bottom);

    {Pone el modo del fondo del formulario en transparente}
    SetBkMode(Form1.Canvas.Handle, TRANSPARENT);

    {Dibuja texto en la parte inferior izquierda del rectángulo}
    DrawText(Form1.Canvas.Handle, 'Delphi Rocks!', -1, BoundingRect,
                DT_BOTTOM or DT_SINGLELINE);

    {Dibuja texto en el centro del rectángulo}
    DrawText(Form1.Canvas.Handle, 'Delphi Rocks!', -1, BoundingRect,
                DT_CENTER or DT_VCENTER or DT_SINGLELINE);

    {Dibuja texto en la parte superior derecha del rectángulo}
    DrawText(Form1.Canvas.Handle, 'Delphi Rocks!', -1, BoundingRect,
                DT_TOP or DT_RIGHT);

    {Crea un nuevo rectángulo límite de formato del texto}
    BoundingRect := Rect(Label2.Left, Label2.Top + Label2.Height + 3,
                        Label2.Width + Label2.Left, Label2.Top + Label2.Height + 73);

    {Dibuja el rectángulo}
    Form1.Canvas.Rectangle(BoundingRect.Left, BoundingRect.Top,
                        BoundingRect.Right, BoundingRect.Bottom);

    {Dibuja el texto con cambio de líneas dentro del rectángulo}
    DrawText(Form1.Canvas.Handle, 'Delphi is the most awesome Windows '+
                'development environment on the market.', -1, BoundingRect,
                DT_WORDBREAK);

    {Crea un nuevo rectángulo límite de formato del texto}
    BoundingRect := Rect(Label3.Left, Label3.Top + Label3.Height + 3,
                        Label3.Width + Label3.Left, Label3.Top + Label3.Height + 25);

    {Recupera el directorio actual}
    GetCurrentDirectory(MAX_PATH, CurDirectory);

```

```

    {Muestra el directorio dentro de rectángulo, reduciéndolo si es necesario}
    DrawText(Form1.Canvas.Handle, CurDirectory, -1, BoundingRect,
            DT_PATH_ELLIPSIS);
end;

```

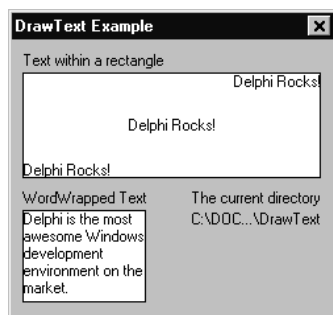


Figura 8-6:  
Salida de texto  
formateado

Tabla 8-19: Valores del parámetro uFormat de DrawText

Valor	Descripción
DT_BOTTOM	El texto a dibujar será justificado al borde inferior del rectángulo. Este indicador tiene que ser combinado con el indicador DT_SINGLELINE.
DT_CALCRECT	Determina automáticamente el ancho y la altura del rectángulo. Para textos de múltiples líneas, el borde inferior del rectángulo es extendido de forma que incluya la última línea del texto. Para textos de una sola línea, el borde derecho del rectángulo es extendido de forma que incluya el último carácter del texto. La función devuelve la altura del texto, pero el texto no es dibujado.
DT_CENTER	Centra el texto horizontalmente dentro del rectángulo.
DT_EDITCONTROL	Duplica el comportamiento de mostrar texto de un control de edición. Específicamente, la función no dibujará la última línea de texto si ésta es visible sólo parcialmente.
DT_END_ELLIPSIS	Si la cadena es demasiado larga para ser mostrada completamente dentro del rectángulo especificado, este indicador hace que la función reemplace los caracteres al final de la cadena con puntos suspensivos (...), de manera que la cadena resultante quepa en el rectángulo.
DT_EXPANDTABS	Los caracteres de tabulación (Tab) son expandidos cuando el texto es dibujado. Por defecto, un carácter de tabulación se expande en 8 caracteres.
DT_EXTERNALLEADING	La altura de la fuente devuelta incluirá el valor de la guía externa para la fuente seleccionada.
DT_LEFT	El texto es justificado al borde izquierdo del rectángulo.

Valor	Descripción
DT_MODIFYSTRING	Modifica la cadena especificada para que se corresponda con el texto mostrado. Este indicador es útil solamente cuando se combina con DT_END_ELLIPSES o DT_PATH_ELLIPSIS.
DT_NOCLIP	Hace que el texto sea dibujado sin recortarlo a los límites del rectángulo. Esto tiene un efecto colateral de mejora en el rendimiento.
DT_NOPREFIX	Desactiva el tratamiento de los caracteres de prefijo mnemotécnico. Específicamente, cualquier carácter de prefijo mnemotécnico ('&') en la cadena será interpretado literalmente y no hará que el siguiente carácter sea subrayado.
DT_PATH_ELLIPSIS	Si la cadena es demasiado larga para ser mostrada completamente dentro del rectángulo especificado, este indicador hace que la función reemplace los caracteres en la mitad de la cadena con puntos suspensivos (...), de manera que la cadena resultante quepa en el rectángulo. Si la cadena contiene contrabarras ('\'), como en el caso de una ruta, la función intentará preservar tanto texto como sea posible a partir de la última contrabarra en la cadena.
DT_RIGHT	El texto es justificado al borde derecho del rectángulo.
DT_RTLREADING	Dibuja el texto de derecha a izquierda. Este indicador puede usarse solamente cuando la fuente seleccionada en el contexto de dispositivo especificado es una fuente hebrea o árabe; en caso contrario, es ignorado.
DT_SINGLELINE	El texto especificado es interpretado como una línea simple, y los caracteres de retorno de carro y cambio de línea son ignorados.
DT_TABSTOP	Indica la cantidad de caracteres que resultan de la expansión de las tabulaciones. Al byte más significativo de la palabra menos significativa del parámetro uFormat (bits 8-15) se le debe asignar la cantidad de caracteres a los que las tabulaciones son expandidas.
DT_TOP	El texto es justificado al borde superior del rectángulo. Este indicador tiene que ser combinado con el indicador DT_SINGLELINE.
DT_VCENTER	Centra el texto verticalmente dentro de la ventana.
DT_WORDBREAK	Implementa un algoritmo de cambio de línea tal que cualquier palabra que se extienda más allá del borde del rectángulo provocará que un cambio de línea sea insertado en el texto, con la palabra de ruptura dibujada en la siguiente línea.

**DrawTextEx**      **Windows.Pas****Sintaxis**

```

DrawTextEx(
    DC: HDC;                                {manejador de contexto de dispositivo}
    lpchText: PChar;                        {cadena a dibujar}
    cchText: Integer;                       {longitud de la cadena a dibujar}
    var p4: TRect;                          {rectángulo de formato}
    dwDTFormat: UINT;                      {opciones de formato del texto}
    DTParams: PDrawTextParams             {opciones adicionales de formato}
): Integer;                               {devuelve la altura del texto dibujado}

```

**Descripción**

Esta función dibuja la cadena de texto indicada por el parámetro *lpchText* sobre el contexto de dispositivo especificado por el parámetro *DC*. El texto es dibujado dentro del rectángulo especificado y es formateado de acuerdo a las opciones de formato especificadas en el parámetro *dwDTFormat* y a las opciones adicionales especificadas por el parámetro *DTParams*. Para dibujar el texto se utilizan la fuente, color del texto, color del fondo y modo del fondo seleccionados en el contexto de dispositivo. A menos que se indique lo contrario mediante un indicador de formato específico, se asume que el texto tiene múltiples líneas y será recortado en los límites del rectángulo especificado.

Observe que las cadenas que contienen el carácter de prefijo mnemotécnico ('&') provocarán que se subraye el carácter siguiente a cada '&', y dos prefijos mnemotécnicos seguidos serán interpretados como un carácter '&' literal.

**Parámetros**

*DC*: Manejador del contexto de dispositivo sobre el cual el texto será dibujado.

*lpchText*: Puntero a una cadena de caracteres terminada en nulo que contiene el texto que será dibujado.

*cchText*: Especifica la longitud de la cadena a la que apunta el parámetro *lpchText*, en caracteres. Si a este parámetro se le asigna -1, se asume que la cadena a la que apunta el parámetro *lpchText* es una cadena de caracteres terminada en nulo y la función calcula automáticamente la longitud de esa cadena.

*p4*: Especifica las coordenadas del rectángulo, en unidades lógicas, dentro del cual el texto será dibujado y formateado.

*dwDTFormat*: Conjunto de indicadores que determinan cómo el texto será mostrado y formateado dentro del rectángulo especificado. Este parámetro puede contener uno o más valores de la Tabla 8-20.

*DTParams*: Puntero al registro *TDrawTextParams* que contiene opciones adicionales de formato del texto. Si a este parámetro se le asigna **nil**, la función *DrawTextEx* se comportará exactamente igual que *DrawText*. El registro *TDrawTextParams* se define:

TDrawTextParams = **packed record**

cbSize: UINT;	{tamaño del registro TDrawTextParams}
iTabLength: Integer;	{tamaño de las tabulaciones}
iLeftMargin: Integer;	{margen izquierdo}
iRightMargin: Integer;	{margen derecho}
uiLengthDrawn: UINT;	{recibe la cantidad de caracteres dibujados}

**end;**

*cbSize*: Especifica el tamaño del registro *TDrawTextParams*. A este campo debe asignársele *SizeOf(TDrawTextParams)*.

*iTabLength*: Especifica el ancho de cada tabulación, en unidades iguales al ancho promedio de los caracteres.

*iLeftMargin*: Especifica el margen izquierdo dentro del rectángulo de formato, en unidades lógicas.

*iRightMargin*: Especifica el margen derecho dentro del rectángulo de formato, en unidades lógicas.

*uiLengthDrawn*: Recupera la cantidad de caracteres dibujados por la función *DrawTextEx*, incluyendo los espacios en blanco.

#### Valor que devuelve

Si la función tiene éxito, devuelve la altura del texto en unidades lógicas; en caso contrario, devuelve cero.

#### Véase además

*DrawText*, *GrayString*, *TabbedTextOut*, *TextOut*

#### Ejemplo

##### Listado 8-6: Dibujando texto con márgenes<sup>1</sup>

{Cadena larga que será dibujada}

**const**

```
TheString = 'The companion CD-ROM that accompanies this book is a multimedia' +
' experience containing all of the source code from the book, a complete' +
' Delphi Syntax compliant help file, shareware, freeware, and an assortment' +
' of third party development and evaluation tools. Using the CD-Browser you' +
' can navigate through the CD and choose which applications and chapter code' +
' to install with a single mouse click. Using the CD browser is simple; on' +
' a Windows 95 or Windows NT system, simply insert the CD and the browser' +
' will begin automatically.';
```

1 La versión en castellano no incluye CD-ROM. Vea en el apéndice F como conseguir el código fuente de los ejemplos del libro y el contenido del CD.

```

var
    Form1: TForm1;
    ResizingMargins: Boolean;    // indica si los márgenes están siendo cambiados

implementation

{$R *.DFM}

procedure TForm1.PaintBox1Paint(Sender: TObject);
var
    BoundingRect: TRect;        // el rectángulo límite de formato del texto
    DrawingParams: TDrawTextParams; // opciones adicionales de formato de texto
begin
    with PaintBox1.Canvas do
        begin
            {Borra la última imagen}
            Brush.Color := clWhite;
            FillRect(ClipRect);

            {El rectángulo de formato de texto es del tamaño de la caja de dibujo}
            BoundingRect := ClipRect;

            with DrawingParams do
                begin
                    {Selecciona el tamaño de la estructura de parámetros de formato adicionales}
                    cbSize := SizeOf(TDrawTextParams);

                    {Inicializa la longitud de las tabulaciones y los márgenes}
                    iTabLength := 0;
                    iLeftMargin := (Panel1.Left - PaintBox1.Left);
                    iRightMargin := 200 - Panel2.Width;
                end;

                {Dibuja el texto con márgenes}
                DrawTextEx(PaintBox1.Canvas.Handle, TheString, Length(TheString),
                    BoundingRect, DT_WORDBREAK, @DrawingParams);
            end;
        end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    {No redimensionamos los márgenes inicialmente}
    ResizingMargins := FALSE;
end;

procedure TForm1.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    {El usuario está arrastrando un panel y redimensionando los márgenes}
    ResizingMargins := TRUE;
end;

procedure TForm1.Panel1MouseUp(Sender: TObject; Button: TMouseButton;

```

```

    Shift: TShiftState; X, Y: Integer);
begin
    {Los márgenes han sido redimensionados, se actualiza la pantalla}
    ResizingMargins := FALSE;
    PaintBox1.Refresh;
end;

procedure TForm1.Panel1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    {Redimensiona el panel si el usuario ha comenzado a redimensionar los márgenes}
    if ResizingMargins then
    begin
        Panel1.Left := Panel1.Left + X;
        Panel1.Width := Panel2.Left - Panel1.Left;
    end;

    {Limita el panel a un tamaño máximo}
    if Panel1.Left < PaintBox1.Left then
    begin
        Panel1.Left := PaintBox1.Left;
        Panel1.Width := 200;
    end;
end;

procedure TForm1.Panel2MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    {Redimensiona el panel si el usuario ha comenzado a redimensionar los márgenes}
    if ResizingMargins then
        Panel2.Width := X;

    {Confina el panel a un tamaño máximo}
    if Panel2.Width > 200 then
        Panel2.Width := 200;
end;

```

Figura 8-7:  
Texto  
formateado  
con márgenes

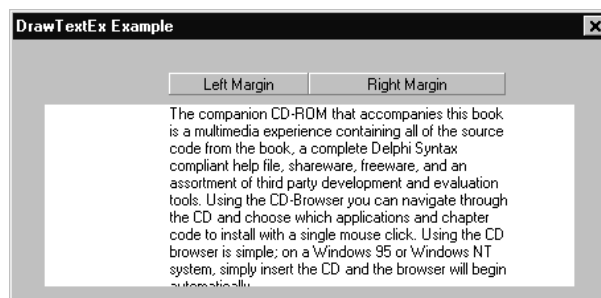


Tabla 8-20: Valores del parámetro dwDTFormat de DrawTextEx

Valor	Descripción
DT_BOTTOM	El texto a dibujar será justificado al borde inferior del rectángulo. Este indicador tiene que ser combinado con el indicador DT_SINGLELINE.
DT_CALCRECT	Determina automáticamente el ancho y la altura del rectángulo. Para textos de múltiples líneas, el borde inferior del rectángulo es extendido de forma que incluya la última línea del texto. Para textos de una sola línea, el borde derecho del rectángulo es extendido de forma que incluya el último carácter del texto. La función devuelve la altura del texto, pero el texto no es dibujado.
DT_CENTER	Centra el texto horizontalmente dentro del rectángulo.
DT_EDITCONTROL	Duplica el comportamiento de mostrar texto de un control de edición. Específicamente, la función no dibujará la última línea de texto si ésta es visible sólo parcialmente.
DT_END_ELLIPSIS	Si la cadena es demasiado larga para ser mostrada completamente dentro del rectángulo especificado, este indicador hace que la función reemplace los caracteres al final de la cadena con puntos suspensivos (...), de manera que la cadena resultante quepa en el rectángulo.
DT_EXPANDTABS	Los caracteres de tabulación (Tab) son expandidos cuando el texto es dibujado. Por defecto, un carácter de tabulación se expande en 8 caracteres.
DT_EXTERNALLEADING	La altura de la fuente devuelta incluirá el valor de la guía externa para la fuente seleccionada.
DT_LEFT	El texto es justificado al borde izquierdo del rectángulo.
DT_MODIFYSTRING	Modifica la cadena especificada para que se corresponda con el texto mostrado. Este indicador es útil solamente cuando se combina con DT_END_ELLIPSES o DT_PATH_ELLIPSIS.
DT_NOCLIP	Hace que el texto sea dibujado sin recortarlo a los límites del rectángulo. Esto tiene un efecto colateral de mejora en el rendimiento.
DT_NOPREFIX	Desactiva el tratamiento de los caracteres de prefijo mnemotécnico. Específicamente, cualquier carácter de prefijo mnemotécnico ('&') en la cadena será interpretado literalmente, y no subrayará el siguiente carácter.
DT_PATH_ELLIPSIS	Si la cadena es demasiado larga para ser mostrada completamente dentro del rectángulo especificado, este indicador hace que la función reemplace los caracteres en la mitad de la cadena con puntos suspensivos (...), de manera que la cadena resultante quepa en el rectángulo. Si la cadena contiene contrabarras ('\'), como en el caso de una ruta, la función intentará preservar tanto texto como sea posible a partir de la última contrabarra en la cadena.
DT_RIGHT	El texto es justificado al borde derecho del rectángulo.

Valor	Descripción
DT_RTLREADING	Dibuja el texto de derecha a izquierda . Este indicador puede usarse solamente cuando la fuente seleccionada en el contexto de dispositivo especificado es una fuente hebrea o arábica; en caso contrario, es ignorado.
DT_SINGLELINE	El texto especificado es interpretado como una línea simple, y los caracteres de retorno de carro y cambio de línea son ignorados.
DT_TABSTOP	Indica la cantidad de caracteres que resultan de la expansión de las tabulaciones. Al byte más significativo de la palabra menos significativa del parámetro dwDTFormat (bits 8-15) se le debe asignar la cantidad de caracteres a los que las tabulaciones son expandidas.
DT_TOP	El texto es justificado al borde superior del rectángulo. Este indicador tiene que ser combinado con el indicador DT_SINGLELINE.
DT_VCENTER	Centra el texto verticalmente dentro de la ventana.
DT_WORDBREAK	Implementa un algoritmo de cambio de línea tal que cualquier palabra que se extienda más allá del borde del rectángulo provocará que un cambio de línea sea insertado en el texto, con la palabra de ruptura dibujada en la siguiente línea.

**EnumFontFamilies****Windows.Pas****Sintaxis**

```
EnumFontFamilies(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: PChar;               {nombre del tipo de letra}
    p3: TFNFontEnumProc;    {puntero a una función de respuesta}
    p4: LPARAM               {valor de 32 bits definido por la aplicación}
): BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función suministra la información asociada a cada fuente accesible en el contexto de dispositivo *DC* con el nombre del tipo de letra especificado por el parámetro *p2* a una función de respuesta definida por la aplicación. En la enumeración se incluyen las fuentes True Type, de barrido y vectoriales, pero se excluye cualquier fuente True Type *de-sólo-lectura*. La enumeración continuará hasta que todas las fuentes hayan sido enumeradas o hasta que la función de respuesta devuelva cero.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyas fuentes serán enumeradas. La función enumera todas las fuentes accesibles en el contexto de dispositivo especificado.

*p2*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del tipo de letra cuyas fuentes asociadas serán enumeradas. Si a este parámetro se le asigna **nil**, la función enumera sólo una fuente, aleatoriamente seleccionada, de cada nombre de tipo de letra disponible.

*p3*: La dirección de la función de respuesta definida por la aplicación.

*p4*: Contiene un valor de 32 bits definido por la aplicación, que es pasado a la función de enumeración.

#### Valor que devuelve

Si el último valor devuelto por la función de respuesta es distinto de cero, la función devuelve TRUE. Si el último valor devuelto por la función de respuesta es cero, la función devuelve FALSE. Esta función no indica ningún error en caso de fallo.

#### Sintaxis de la Función de Respuesta

```
EnumFontFamProc(
    LogFont: PEnumLogFont;      {puntero a los atributos de fuente lógica}
    TextMetrics: PNewTextMetric; {puntero a los atributos de fuente física}
    FontType: Integer;          {opciones de tipo de fuente}
    lParam: LPARAM              {valor de 32 bits definido por la aplicación}
): Integer;                    {devuelve un valor distinto de cero para continuar
                                la enumeración}
```

#### Descripción

Esta función recibe punteros a registros de tipos *TEnumLogFont* y *TNewTextMetric* para cada fuente enumerada, y puede ejecutar cualquier tarea deseada.

#### Parámetros

*LogFont*: Puntero a un registro *TEnumLogFont* que contiene los atributos de fuente lógica para la fuente actualmente enumerada. El registro *TEnumLogFont* se define como:

```
TEnumLogFont = packed record
    elfLogFont: TLogFont;          {información de la fuente lógica}
    elfFullName: array[0..LF_FULLFACESIZE - 1] of AnsiChar;
                                     {nombre completo de la fuente}
    elfStyle: array[0..LF_FACESIZE - 1] of AnsiChar; {estilo de la fuente}
end;
```

*elfLogFont*: Especifica un registro *TLogFont* que describe los atributos lógicos de la fuente. El registro *TLogFont* se define como:

```
TLogFont = packed record
    lfHeight: Longint;              {altura de la fuente}
    lfWidth: Longint;               {ancho de caracteres}
    lfEscapement: Longint;          {ángulo de escape}
```

```

lfOrientation: Longint;           {ángulo de la línea base}
lfWeight: Longint;               {grosor de la negrita}
lfItalic: Byte;                  {indicador de itálica}
lfUnderline: Byte;               {indicador de subrayado}
lfStrikeOut: Byte;               {indicador de tachado}
lfCharSet: Byte;                 {conjunto de caracteres}
lfOutPrecision: Byte;            {indicador de precisión}
lfClipPrecision: Byte;           {precisión de recorte}
lfQuality: Byte;                 {indicador de calidad}
lfPitchAndFamily: Byte;          {pitch y familia de fuentes}
lfFaceName: array[0..LF_FACESIZE - 1] of AnsiChar;
                                   {nombre del tipo de letra de la fuente}

```

**end;**

Consulte la función *CreateFontIndirect* para ver una descripción de este registro.

*elfFullName*: Cadena de caracteres terminada en nulo que contiene el nombre completo (que es único) de la fuente enumerada.

*elfStyle*: Cadena de caracteres terminada en nulo que contiene el estilo de la fuente.

*TextMetrics*: Puntero a un registro *TNewTextMetric* que contiene los atributos físicos de la fuente. Observe que si la fuente actualmente enumerada no es una fuente True Type, este parámetro apuntará a un registro *TTextMetric*. Todas las medidas devueltas en este registro están en unidades lógicas y dependen del modo de mapeado actual del contexto de dispositivo especificado. El registro *TNewTextMetric* se define como:

**TNewTextMetric = record**

```

tmHeight: Longint;               {altura del carácter}
tmAscent: Longint;               {ascenso del carácter}
tmDescent: Longint;              {descenso del carácter}
tmInternalLeading: Longint;        {guía interna}
tmExternalLeading: Longint;        {guía externa}
tmAveCharWidth: Longint;          {ancho promedio del carácter}
tmMaxCharWidth: Longint;          {ancho máximo del carácter}
tmWeight: Longint;               {valor del peso de la negrita}
tmOverhang: Longint;             {ancho que sobresale}
tmDigitizedAspectX: Longint;      {aspecto horizontal}
tmDigitizedAspectY: Longint;      {aspecto vertical }
tmFirstChar: AnsiChar;            {primer carácter}
tmLastChar: AnsiChar;             {último carácter}
tmDefaultChar: AnsiChar;          {carácter por defecto}
tmBreakChar: AnsiChar;            {carácter de ruptura de palabra}
tmItalic: Byte;                   {indicador de itálica}
tmUnderlined: Byte;               {indicador de subrayado}
tmStruckOut: Byte;                {indicador de tachado}
tmPitchAndFamily: Byte;           {opciones de pitch y familia}

```

tmCharSet: Byte;	{conjunto de caracteres}
ntmFlags: DWORD;	{atributos de la máscara de bits}
ntmSizeEM: UINT;	{tamaño de la pica cuadrada, en unidades teóricas}
ntmCellHeight: UINT;	{altura de la celda, en unidades teóricas}
ntmAvgWidth: UINT;	{ancho promedio del carácter, en unidades teóricas}

**end;**

Excepto por los últimos cuatro campos, este registro es idéntico a *TTextMetric*. Consulte la función *GetTextMetrics* para ver una descripción del registro *TTextMetric*. Los nuevos campos incluidos en *TNewTextMetric* son:

*ntmFlags*: Una máscara de bits que especifica varios atributos de la fuente. Cada bit de la máscara identifica un atributo de fuente diferente, como se describe en la Tabla 8-21. Si se selecciona un bit, ese atributo estará presente en la fuente actualmente enumerada.

*ntmSizeEM*: Especifica el tamaño de la pica cuadrada de la fuente, en unidades teóricas (*notional units*). La unidad teórica es la unidad para la cual la fuente fue creada originalmente.

*ntmCellHeight*: Especifica la altura de la celda de un carácter para la fuente, en unidades teóricas.

*ntmAvgWidth*: Especifica el ancho del carácter promedio para la fuente, en unidades teóricas.

*FontType*: Especifica una serie de opciones que indican el tipo de la fuente que está siendo enumerada. Este parámetro puede contener uno o más valores de la Tabla 8-22. Observe que si no están presentes el indicador *RASTER\_FONTTYPE* ni el indicador *TRUETYPE\_FONTTYPE*, la fuente enumerada es vectorial.

*lParam*: Especifica el valor de 32 bits definido por la aplicación pasado a la función *EnumFontFamilies* en el parámetro *p4*.

#### Valor que devuelve

La función de respuesta debe devolver un valor distinto de cero para continuar la enumeración o cero para terminarla.

#### Véase además

*CreateFontIndirect*, *EnumFontFamiliesEx*, *GetTextMetrics*

#### Ejemplo

##### Listado 8-7: Enumeración de fuentes accesibles

```
{El prototipo de la función de respuesta}
function FontEnumProc(LogFont: PEnumLogFont; TextMetrics: PNewTextMetric;
                      FontType: Integer; lParam: LPARAM): Integer; stdcall;

var
```

```
Form1: TForm1;
```

### implementation

```
{ $R *.DFM }
```

```
procedure TForm1.FormActivate(Sender: TObject);
var
    RasterStatus: TRasterizerStatus;    // almacena capacidades de barrido
begin
    {Selecciona el tamaño del registro de estado de barrido}
    RasterStatus.nSize := SizeOf(TRasterizerStatus);

    {Recupera el estado de barrido}
    GetRasterizerCaps(RasterStatus, SizeOf(TRasterizerStatus));

    {Indica si la fuente True Type está habilitada y accesible}
    if (RasterStatus.wFlags and TT_ENABLED) = TT_ENABLED then
        CheckBox1.Checked := TRUE;
    if (RasterStatus.wFlags and TT_AVAILABLE) = TT_AVAILABLE then
        CheckBox2.Checked := TRUE;

    {Enumera todas las fuentes instaladas}
    EnumFontFamilies(Form1.Canvas.Handle, nil, @FontEnumProc, 0);
end;

function FontEnumProc(LogFont: PEnumLogFont; TextMetrics: PNewTextMetric;
    FontType: Integer; lParam: LPARAM): Integer; stdcall;
begin
    {Añade el nombre de la fuente y su tipo de fuente al cuadro de lista}
    Form1.ListBox1.Items.AddObject(TEnumLogFont(LogFont^).elfLogFont.lfFaceName,
        TObject(FontType));

    {Continúa la enumeración}
    Result := 1;
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer;
    Rect: TRect; State: TOwnerDrawState);
begin
    {Indica si la fuente es True Type o de otro tipo}
    if Integer(ListBox1.Items.Objects[Index]) = TRUETYPE_FONTTYPE then
        ListBox1.Canvas.Draw(Rect.Left, Rect.Top, Image2.Picture.Bitmap)
    else
        ListBox1.Canvas.Draw(Rect.Left, Rect.Top, Image1.Picture.Bitmap);

    {Dibuja el nombre de la fuente}
    Rect.Left := Rect.Left + 18;
    Rect.Top := Rect.Top + 2;
    TextOut(ListBox1.Canvas.Handle, Rect.Left, Rect.Top,
        PChar(ListBox1.Items[Index]), Length(ListBox1.Items[Index]));
end;
```

Figura 8-8:  
Los nombres  
de fuente  
accesibles

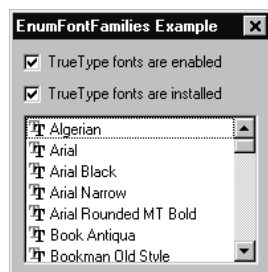


Tabla 8-21: Valores de los bits del campo `TextMetrics.ntmFlags` de la función de respuesta `EnumFontFamProc`

Posición del bit	Descripción
0	Indica una fuente itálica.
1	Indica una fuente subrayada.
2	Indica una fuente en imagen inversa.
3	Indica una fuente contorneada.
4	Indica una fuente tachada.
5	Indica una fuente negrita.

Tabla 8-22: Valores del parámetro `FontType` de la función de respuesta `EnumFontFamProc`

Valor	Descripción
<code>DEVICE_FONTTYPE</code>	Indica una fuente residente en el dispositivo, o que el dispositivo soporta descarga de fuentes True Type.
<code>RASTER_FONTTYPE</code>	Indica una fuente de barrido o de mapa de bits.
<code>TRUETYPE_FONTTYPE</code>	Indica una fuente True Type.

## **EnumFontFamiliesEx**      **Windows.Pas**

### **Sintaxis**

```
EnumFontFamiliesEx(
    DC: HDC;                {manejador de contexto de dispositivo}
    var p2: TLogFont;        {registro TLogFont}
    p3: TFNFontEnumProc;     {puntero a la función de respuesta}
    p4: LPARAM;              {valor de 32 bits definidos por la aplicación}
    p5: DWORD                {este parámetro está reservado}
): BOOL;                    {devuelve TRUE o FALSE}
```

### **Descripción**

Esta función suministra la información asociada a cada fuente accesible en el contexto de dispositivo *DC* cuyos atributos coincidan con los especificados en el parámetro *p2*, de tipo *TLogFont*, a una función de respuesta definida por la aplicación. En la

enumeración se incluyen las fuentes True Type, de barrido y vectoriales, pero se excluye cualquier fuente True Type *de-sólo-lectura*. La enumeración continuará hasta que todas las fuentes hayan sido enumeradas o hasta que la función de respuesta devuelva cero.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuyas fuentes serán enumeradas. La función enumera todas las fuentes accesibles en el contexto de dispositivo especificado.

*p2*: Puntero a un registro *TLogFont* que contiene la información que determina qué fuentes serán enumeradas. El registro *TLogFont* se define como:

```
TLogFont = packed record
    lfHeight: Longint;           {altura de la fuente en
                                unidades lógicas}
    lfWidth: Longint;           {ancho del caracter en
                                unidades lógicas}
    lfEscapement: Longint;      {ángulo de escape}
    lfOrientation: Longint;     {ángulo de la línea base}
    lfWeight: Longint;         {grosor de la negrita}
    lfItalic: Byte;            {indicador de itálica}
    lfUnderline: Byte;         {indicador de subrayado}
    lfStrikeOut: Byte;         {indicador de tachado}
    lfCharSet: Byte;           {conjunto de caracteres}
    lfOutPrecision: Byte;      {indicador de precisión}
    lfClipPrecision: Byte;     {precisión de recorte}
    lfQuality: Byte;           {indicador de calidad}
    lfPitchAndFamily: Byte;    {pitch y familia de fuentes}
    lfFaceName: array[0..LF_FACESIZE - 1] of AnsiChar;
                                {nombre del tipo de letra de
                                la fuente}
end;
```

Consulte la función *CreateFontIndirect* para ver una descripción de este registro. Sólo los campos *lfCharSet*, *lfFaceName* y *lfPitchAndFamily* determinan el comportamiento de la función *EnumFontFamiliesEx*.

*lfCharSet*: Si a este parámetro se le asigna *DEFAULT\_CHARSET*, la función enumera todas las fuentes en todos los conjuntos de caracteres. Si a este campo se le asigna un conjunto de caracteres específico, sólo las fuentes que definen caracteres para ese conjunto serán enumeradas.

*lfFaceName*: Si a este campo se le asigna una cadena vacía, una fuente, seleccionada aleatoriamente, de cada tipo de letra es enumerada. Si a este campo se le asigna un nombre de tipo de letra válido, sólo las fuentes con ese tipo de letra serán enumeradas.

*lfPitchAndFamily*: Este campo es usado solamente con fuentes hebreas o arábigas, y hay que asignarle cero para cualquier otro tipo de fuente. Para

fuentes hebreas o árabigas, a este campo se le puede asignar *MONO\_FONT*, para enumerar únicamente las fuentes que contienen todos los caracteres de la página de códigos.

*p3*: La dirección de la función de respuesta definida por la aplicación.

*p4*: Contiene un valor de 32 bits definido por la aplicación que es pasado a la función de enumeración.

*p5*: Este parámetro está reservado para uso futuro y debe asignársele cero.

#### Valor que devuelve

Si el último valor devuelto por la función de respuesta es un valor distinto de cero, la función devuelve TRUE. Si el último valor devuelto por función de respuesta es cero, la función devuelve FALSE. Esta función no indica errores en caso de fallos.

#### Sintaxis de la Función de Respuesta

```
EnumFontFamExProc(
    LogFont: PEnumLogFontEx;    {puntero a los atributos de fuente lógica}
    TextMetrics: PNewTextMetric; {puntero a los atributos de fuente física}
    FontType: Integer;           {opciones de tipo de fuente}
    lParam: LPARAM               {valor de 32 bits definido por la aplicación}
): Integer;                     {devuelve un valor distinto de cero para continuar
                                la enumeración}
```

#### Descripción

Esta función recibe punteros a registros de tipo *TEnumLogFontEx* y *TNewTextMetricEx* para cada fuente enumerada, y puede ejecutar cualquier tarea deseada.

#### Parámetros

*LogFont*: Puntero a un registro *TEnumLogFontEx* que contiene los atributos de fuente lógica para la fuente actualmente enumerada. El registro *TEnumLogFontEx* se define como:

```
TEnumLogFontEx = packed record
    elfLogFont: TLogFont;           {información de la fuente lógica}
    elfFullName: array[0..LF_FULLFACESIZE - 1] of Char;
                                     {nombre completo de la fuente}
    elfStyle: array[0..LF_FACESIZE - 1] of Char;    {estilo de la fuente}
    elfScript: array[0..LF_FACESIZE - 1] of Char;   {guión de la fuente}
end;
```

*elfLogFont*: Especifica un registro *TLogFont* que describe los atributos lógicos de la fuente. Consulte la función *CreateFontIndirect* para ver una descripción de este registro.

*elfFullName*: Cadena de caracteres terminada en nulo que contiene el nombre completo (que es único) de la fuente enumerada.

*elfStyle*: Cadena de caracteres terminada en nulo con el estilo de la fuente.

*elfScript*: Una cadena de caracteres terminada en nulo con la descripción de la fuente.

*TextMetrics*: Puntero a un registro *TNewTextMetricEx* que contiene los atributos físicos de la fuente. Observe que si la fuente actualmente enumerada no es True Type, este parámetro apuntará a un registro *TTextMetric*. También tenga en cuenta que bajo Windows 95/98 el registro *TNewTextMetricEx* no está implementado y este parámetro apuntará, en su lugar, a un registro de tipo *TNewTextMetric*. El registro *TNewTextMetricEx* se define como:

*TNewTextMetricEx* = **packed record**

    ntmTm: *TNewTextMetric*;                   {registro *TNewTextMetric*}

    ntmFontSig: *TFontSignature*;           {registro *TFontSignature*}

**end;**

*ntmTm*: Un registro *TNewTextMetric* que contiene los atributos de la fuente física para la fuente actualmente enumerada. El registro *TNewTextMetric* se define como:

*TNewTextMetric* = **record**

tmHeight: Longint;	{altura del carácter}
tmAscent: Longint;	{ascenso del carácter}
tmDescent: Longint;	{descenso del carácter}
tmInternalLeading: Longint;	{guía interna}
tmExternalLeading: Longint;	{guía externa}
tmAveCharWidth: Longint;	{ancho promedio del carácter}
tmMaxCharWidth: Longint;	{ancho máximo del carácter}
tmWeight: Longint;	{valor del peso de la negrita}
tmOverhang: Longint;	{ancho que sobresale}
tmDigitizedAspectX: Longint;	{aspecto horizontal}
tmDigitizedAspectY: Longint;	{aspecto vertical}
tmFirstChar: AnsiChar;	{primer carácter}
tmLastChar: AnsiChar;	{último carácter}
tmDefaultChar: AnsiChar;	{carácter por defecto}
tmBreakChar: AnsiChar;	{carácter de ruptura de palabra}
tmItalic: Byte;	{indicador de itálica}
tmUnderlined: Byte;	{indicador de subrayado}
tmStruckOut: Byte;	{indicador de tachado}
tmPitchAndFamily: Byte;	{opciones de pitch y familia}
tmCharSet: Byte;	{conjunto de caracteres}
ntmFlags: DWORD;	{atributos de la máscara de bits}
ntmSizeEM: UINT;	{tamaño de la pica cuadrada, en unidades teóricas}
ntmCellHeight: UINT;	{altura de la celda, en unidades teóricas}
ntmAvgWidth: UINT;	{ancho promedio del carácter}

**end;**

Consulte la función *EnumFontFamilies* para ver una descripción de este registro.

*ntmFontSig*: Un registro *TFontSignature* que identifica las páginas de código y los subrangos Unicode para los cuales la fuente actualmente enumerada ofrece patrones de imágenes. El registro *TFontSignature* se define como:

*TFontSignature* = **packed record**

fsUsb: **array**[0..3] **of** DWORD; {máscara del subrango de Unicode}

fsCsb: **array**[0..1] **of** DWORD; {máscara de la página de códigos}

**end;**

*fsUsb*: Una máscara de 128 bits que identifica uno de 126 posibles subrangos de Unicode, donde cada bit, excepto los dos más significativos, identifica un subrango simple. El bit más significativo siempre está activo, y el segundo más significativo actualmente no se utiliza y siempre es 0.

*fsCsb*: Una máscara de 64 bits que identifica un conjunto de caracteres específico o una página de códigos, donde cada bit identifica una única página de códigos. La doble palabra menos significativa especifica páginas de códigos de Windows y la doble palabra más significativa páginas de códigos no Windows. Las páginas de códigos para cada bit individual se listan en la Tabla 8-23.

*FontType*: Especifica una serie de opciones que indican el tipo de fuente que está siendo enumerada. Este parámetro puede contener uno o más valores de la Tabla 8-24. Observe que si no están presentes el indicador *RASTER\_FONTTYPE* ni el indicador *TRUETYPE\_FONTTYPE*, la fuente enumerada es vectorial.

*lParam*: Especifica el valor de 32 bits definido por la aplicación pasado a la función *EnumFontFamiliesEx* en el parámetro *p4*.

#### Valor que devuelve

La función de respuesta debe devolver un valor distinto de cero para continuar la enumeración y cero para terminarla.

#### Véase además

*CreateFontIndirect*, *EnumFontFamilies*, *GetTextMetrics*

#### Ejemplo

##### Listado 8-8: Enumerando sólo las fuentes de símbolos

{El prototipo de la función de respuesta}

**function** FontEnumExProc(LogFont: PEnumLogFontEx; TextMetrics: PNewTextMetric;  
FontType: Integer; lParam: LPARAM): Integer; **stdcall**;

**var**

Form1: TForm1;

**implementation**

```

{$R *.DFM}

procedure TForm1.FormActivate(Sender: TObject);
var
    FontInfo: TLogFont; // almacena la información de enumeración de fuente
begin
    {Inicializa registro FontInfo para enumerar todas las fuentes pertenecientes
     al conjunto de caracteres de símbolos}
    FontInfo.lfCharSet      := SYMBOL_CHARSET;
    FontInfo.lfFaceName     := '';
    FontInfo.lfPitchAndFamily := 0;

    {Enumera las fuentes}
    EnumFontFamiliesEx(Form1.Canvas.Handle, FontInfo, @FontEnumExProc, 0, 0);
end;

function FontEnumExProc(LogFont: PEnumLogFontEx; TextMetrics: PNewTextMetric;
                        FontType: Integer; lParam: LPARAM): Integer; stdcall;
begin
    {Añade el nombre de tipo de letra y su tipo al cuadro de lista}
    Form1.ListBox1.Items.AddObject(TEnumLogFontEx(LogFont^).elfLogFont.lfFaceName,
                                   TObject(FontType));

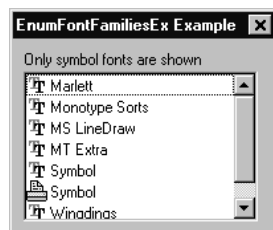
    {Continúa la enumeración}
    Result := 1;
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer;
                                Rect: TRect; State: TOwnerDrawState);
begin
    {Indica si la fuente es True Type o de otro tipo}
    if Integer(ListBox1.Items.Objects[Index]) = TRUETYPE_FONTTYPE then
        ListBox1.Canvas.Draw(Rect.Left, Rect.Top, Image2.Picture.Bitmap)
    else
        ListBox1.Canvas.Draw(Rect.Left, Rect.Top, Image1.Picture.Bitmap);

    {Dibuja el nombre de la fuente}
    Rect.Left := Rect.Left + 18;
    Rect.Top  := Rect.Top + 2;
    TextOut(ListBox1.Canvas.Handle, Rect.Left, Rect.Top,
            PChar(ListBox1.Items[Index]), Length(ListBox1.Items[Index]));
end;

```

Figura 8-9:  
Todas las  
fuentes de  
símbolos  
accesibles



**Tabla 8-23: Valores del campo TextMetrics.ntmFontSig.fsCsb de la función de respuesta EnumFontFamExProc**

Bit	Pág. Códigos	Descripción
0	1252	Latin 1 (Europa Occidental).
1	1250	Latin 2 (Europa del Este).
2	1251	Cirílico.
3	1253	Griego.
4	1254	Turco.
5	1255	Hebreo.
6	1256	Arábigo.
7	1257	Báltico.
8-16		Reservado para ANSI.
17	874	Tailandés.
18	932	JIS/Japón.
19	936	Caracteres chinos simplificados.
20	949	Coreano Unificado (Hangeul).
21	950	Caracteres chinos tradicionales.
22-29		Reservado para uso alternativo de ANSI y OEM.
30-31		Reservado por el sistema.
32-47		Reserved para OEM.
48	869	Griego IBM.
49	866	Ruso MS-DOS.
50	865	Nórdico MS-DOS.
51	864	Arábigo.
52	863	Francés Canadiense MS-DOS.
53	862	Hebreo.
54	861	Islandés MS-DOS.
55	860	Portugués MS-DOS.
56	857	Turco IBM.
57	855	Cirílico IBM.
58	852	Latin 2.
59	776	Báltico.
60	737	Griego.
61	708	Arábigo (ASMO 708).
62	850	WE/Latin 1.
63	437	Estados Unidos.

**Tabla 8-24: Valores del parámetro `FontType` de la función de respuesta `EnumFontFamExProc`**

Valor	Descripción
<code>DEVICE_FONTTYPE</code>	Indica una fuente residente en el dispositivo, o que el dispositivo soporta la descarga de fuentes True Type.
<code>RASTER_FONTTYPE</code>	Indica una fuente de barrido o mapa de bits.
<code>TRUETYPE_FONTTYPE</code>	Indica una fuente True Type.

### ***GetCharABCWidths***

### ***Windows.Pas***

#### ***Sintaxis***

```
GetCharABCWidths(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: UINT;                {primer carácter en el rango}
    p3: UINT;                {último carácter en el rango}
    const ABCStructs        {apunta a un array de registros TABC}
): BOOL;                    {devuelve TRUE o FALSE}
```

#### ***Descripción***

Esta función recupera varios valores de ancho y espaciado de caracteres para la fuente True Type actualmente seleccionada en el contexto de dispositivo identificado por el parámetro *DC*. Estos valores son recuperados para un rango de caracteres consecutivos dentro de la fuente. Para cada carácter en el rango, el registro *TABC* correspondiente en el *array* de registros *TABC* al que apunta el parámetro *ABCStructs*, recibe tres valores de ancho. El valor de espaciado “A” es la distancia que se añade a la posición actual antes de ubicar el próximo patrón del carácter cuando se está escribiendo una línea de texto. El valor de espaciado “B” es el ancho actual del patrón del carácter. El valor de espaciado “C” es la distancia que se añade a la derecha del patrón para ofrecer espacios para la separación de los caracteres. Valores de espaciado “A” y “C” negativos, indican una fuente con un desplazamiento hacia abajo o hacia arriba.

Tenga en cuenta que esta función es aplicable únicamente a fuentes True Type. Para recuperar el ancho de fuentes que no sean True Type utilice la función *GetCharWidth*.

#### ***Parámetros***

*DC*: Manejador del contexto de dispositivo cuyos anchos de caracteres para la fuente actualmente seleccionada serán recuperados.

*p2*: Especifica el valor del primer carácter en el rango de caracteres.

*p3*: Especifica el valor del último carácter en el rango de caracteres.

*ABCStructs*: Puntero a un *array* de registros de tipo *TABC* que recibirá los anchos de espaciado ABC de cada carácter en el rango especificado. Debe haber al menos tantos

registros *TABC* en el array como caracteres en el rango definido por los parámetros *p2* y *p3*. El registro *TABC* se define como:

**TABC = packed record**

```
    abcA: Integer;           {desplazamiento inicial}
    abcB: UInt;             {ancho del patrón}
    abcC: Integer;          {espacio en blanco}
```

**end;**

*abcA*: Especifica la distancia que se añade a la posición actual antes de ubicar el patrón del próximo carácter cuando se está escribiendo una línea de texto, en unidades lógicas.

*abcB*: Especifica el ancho del patrón del carácter, en unidades lógicas.

*abcC*: Especifica la distancia que se añade a la derecha del patrón para ofrecer espacio de separación de caracteres, en unidades lógicas.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetCharWidth*, *GetOutlineTextMetrics*, *GetTextMetrics*

#### Ejemplo

##### Listado 8-9: Recuperando los anchos ABC para todas las letras mayúsculas

```
procedure TForm1.FormActivate(Sender: TObject);
var
    CharWidths: array[0..25] of TABC; // almacena anchos ABC de caracteres
    Count: Integer; // variable de control de bucle
begin
    {Inicializa la rejilla de cadenas}
    StringGrid1.Cells[0,0] := 'Character';
    StringGrid1.Cells[1,0] := ''A'' Width';
    StringGrid1.Cells[2,0] := ''B'' Width';
    StringGrid1.Cells[3,0] := ''C'' Width';

    {Recupera los anchos ABC de todas las mayúsculas}
    GetCharABCWidths(Form1.Canvas.Handle, Ord('A'), Ord('Z'), CharWidths);

    {Muestra los anchos ABC de todas las mayúsculas}
    for Count := 0 to 26 do
    begin
        StringGrid1.Cells[0, Count+1] := Char(Ord('A') + Count);
        StringGrid1.Cells[1, Count+1] := IntToStr(CharWidths[Count].abcA);
        StringGrid1.Cells[2, Count+1] := IntToStr(CharWidths[Count].abcB);
        StringGrid1.Cells[3, Count+1] := IntToStr(CharWidths[Count].abcC);
```

```
end;
end;
```

Character	'A' Width	'B' Width	'C' Width
A	0	7	1
B	0	6	1
C	0	6	1
D	0	6	1

Figura 8-10:  
El ancho ABC  
de las letras  
mayúsculas

## GetCharWidth Windows.Pas

### Sintaxis

```
GetCharWidth(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: UINT;                {primer carácter del rango}
    p3: UINT;                {último carácter del rango}
    const Widths             {puntero a un array de enteros}
): BOOL;                   {devuelve TRUE o FALSE}
```

### Descripción

Esta función recupera el ancho de cada carácter en el rango de caracteres indicado para la fuente actualmente seleccionada en el contexto de dispositivo identificado por el parámetro *DC*. Para cada carácter en el rango, el ancho del carácter se deposita en el entero correspondiente en el *array* de enteros al que apunta el parámetro *Widths*. Esta función es útil tanto para fuentes True Type como no True Type. Sin embargo, las fuentes True Type deben utilizar la función *GetCharABCWidths* para recuperar valores más exactos.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuyos anchos de caracteres para la fuente actualmente seleccionada serán recuperados.

*p2*: Especifica el valor del primer carácter en el rango de caracteres.

*p3*: Especifica el valor del último carácter en el rango de caracteres.

*Widths*: Puntero a un *array* de enteros que recibe el ancho de cada carácter en el rango especificado. Debe haber tantos enteros en el array como caracteres en el rango definido por los parámetros *p2* y *p3*.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*GetCharABCWidths, GetTextExtentExPoint, GetTextExtentPoint32*

Ejemplo

#### Listado 8-10: Recuperando anchos de caracteres para las mayúsculas

```

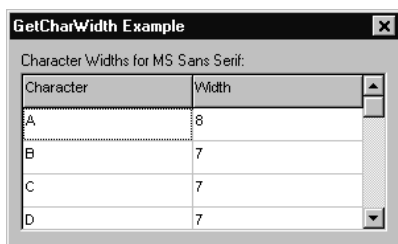
procedure TForm1.FormActivate(Sender: TObject);
var
    CharWidths: array[0..25] of Integer;    // almacena los anchos de los caracteres
    Count: Integer;                          // variable de control de bucle
begin
    {Inicializa la rejilla de cadenas}
    StringGrid1.Cells[0,0] := 'Character';
    StringGrid1.Cells[1,0] := 'Width';

    {Recupera los anchos de las mayúsculas}
    GetCharWidth(Form1.Canvas.Handle, Ord('A'), Ord('Z'), CharWidths);

    {Muestra los anchos de los caracteres}
    for Count := 0 to 26 do
    begin
        StringGrid1.Cells[0, Count+1] := Char(Ord('A') + Count);
        StringGrid1.Cells[1, Count+1] := IntToStr(CharWidths[Count]);
    end;
end;

```

Figura 8-11:  
Los anchos de  
las mayúsculas



Character	Width
A	8
B	7
C	7
D	7

#### GetFontData

#### Windows.Pas

#### Sintaxis

```

GetFontData(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: DWORD;              {tabla de medidas de las fuentes}
    p3: DWORD;              {desplazamiento en la tabla de medidas de la fuente}
    p4: Pointer;            {puntero a un buffer que recibe la información}
    p5: DWORD;              {cantidad de datos a recuperar}
): DWORD;                  {devuelve la cantidad de bytes recuperados}

```

**Descripción**

Esta función recupera información de la tabla de medidas de las fuentes especificada por el parámetro *p2*, para la fuente True Type seleccionada en el contexto de dispositivo identificada por el parámetro *DC*. *GetFontData* puede ser utilizada para recuperar un fichero completo de fuentes True Type, con el objetivo de incrustar una fuente en un documento.

**Parámetros**

*DC*: Manejador del contexto de dispositivo en el que está seleccionada la fuente cuya información será recuperada.

*p2*: Especifica la tabla de medidas de la fuente desde la cual se recuperarán datos. Las tablas de medidas de las fuentes True Type se describen en las especificaciones de ficheros de fuentes True Type, publicadas por Microsoft. Si a este parámetro se le asigna cero, la función recupera información comenzando desde el inicio del fichero de fuentes.

*p3*: Especifica el desplazamiento desde el inicio de la tabla de medidas especificada, a partir de dónde la función comenzará a recuperar información. Si a este parámetro se le asigna cero, la función recupera información comenzando desde el inicio de la tabla de medidas especificada.

*p4*: Puntero a un *buffer* que recibirá la información recuperada. Si a este parámetro se le asigna **nil**, la función devuelve el tamaño del *buffer* necesario para almacenar la información solicitada.

*p5*: Especifica la cantidad de información a recuperar, en bytes. Si a este parámetro se le asigna cero, la función devuelve el tamaño de la tabla de medidas especificada por el parámetro *p2*.

**Valor que devuelve**

Si la función tiene éxito, devuelve la cantidad de bytes recuperados de los datos de la fuente; en caso contrario, devuelve *GDI\_ERROR*.

**Véase además**

*AddFontResource*, *CreateScalableFontResource*, *GetTextMetrics*,  
*RemoveFontResource*

**Ejemplo**

Vea el Listado 8-1, que muestra cómo incrustar una fuente.

**GetGlyphOutline****Windows.Pas****Sintaxis**

GetGlyphOutline(

DC: HDC ;	{manejador de contexto de dispositivo}
p2: UINT;	{carácter}
p3: UINT;	{opciones de formato de datos}
<b>const</b> p4: TGlyphMetrics;	{puntero a un registro TGlyphMetrics}
p5: DWORD;	{tamaño del <i>buffer</i> de datos}
p6: Pointer;	{puntero al <i>buffer</i> de datos}
<b>const</b> p7: TMat2	{matriz de rotación}
); DWORD;	{devuelve un código de error}

### Descripción

Esta función recupera la información de delineado del carácter especificado para la fuente True Type seleccionada en el contexto de dispositivo identificado por el parámetro *DC*. La información de delineado recuperada se obtiene en la forma de un mapa de bits monocromático o de una serie de líneas y curvas que describen la forma nativa del patrón. Esta información se almacena en el *buffer* a la que apunta el parámetro *p6*.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuya fuente True Type actualmente seleccionada es utilizada cuando se recupera la información de delineado.

*p2*: Identifica el código del carácter cuyo delineado será recuperado.

*p3*: Especifica el formato de la información de delineado. Este parámetro contiene un valor de la Tabla 8-25.

*p4*: Puntero a un registro *TGlyphMetrics* que recibe la información concerniente a los atributos físicos del patrón del carácter. El registro *TGlyphMetrics* se define de la siguiente manera:

TGlyphMetrics = **packed record**

gmBlackBoxX: UINT;	{ancho del rectángulo más pequeño}
gmBlackBoxY: UINT;	{altura del rectángulo más pequeño}
gmptGlyphOrigin: TPoint;	{origen del rectángulo más pequeño}
gmCellIncX: SHORT;	{desplazamiento horizontal del próximo carácter}
gmCellIncY: SHORT;	{desplazamiento vertical del próximo carácter}

**end;**

*gmBlackBoxX*: Indica el ancho del rectángulo más pequeño en el que cabría completamente la imagen del patrón, en unidades de dispositivo.

*gmBlackBoxY*: Indica la altura del rectángulo más pequeño en el que cabría completamente la imagen del patrón, en unidades de dispositivo.

*gmptGlyphOrigin*: Indica las coordenadas horizontal y vertical, dentro de la celda del carácter, del origen del rectángulo más pequeño en el que cabría completamente la imagen del patrón, en unidades del dispositivo.

*gmCellIncX*: Indica el desplazamiento horizontal desde el principio de la celda del carácter actual, hasta el principio de la celda del próximo carácter, en unidades del dispositivo.

*gmCellIncY*: Indica el desplazamiento vertical desde el principio de la celda del carácter actual, hasta el principio de la celda del próximo carácter, en unidades del dispositivo.

*p5*: Especifica el tamaño del *buffer* de datos al que apunta el parámetro *p6*. Si este parámetro es cero, la función devuelve el tamaño necesario para el *buffer*.

*p6*: Puntero al *buffer* que recibe la información de delineado del patrón. Si a este parámetro se le asigna **nil**, la función devuelve el tamaño necesario para el *buffer*.

*p7*: Puntero a un registro *TMat2* que define una matriz de transformación 3 x 3 que se utiliza para rotar la fuente en cualquier ángulo. El registro *TMat2* se define como:

*TMat2* = **packed record**

<i>eM11</i> : TFixed;	{ángulo en formato de coma fija}
<i>eM12</i> : TFixed;	{ángulo en formato de coma fija}
<i>eM21</i> : TFixed;	{ángulo en formato de coma fija}
<i>eM22</i> : TFixed;	{ángulo en formato de coma fija}

**end;**

*eM11*: Identifica el ángulo de rotación de la fuente, en la forma del registro *TFixed*, para el valor M11 de la matriz de transformación 3X3.

*eM12*: Identifica el ángulo de rotación de la fuente, en la forma del registro *TFixed*, para el valor M12 de la matriz de transformación 3X3.

*eM21*: Identifica el ángulo de rotación de la fuente, en la forma del registro *TFixed*, para el valor M21 de la matriz de transformación 3X3.

*eM22*: Identifica el ángulo de rotación de la fuente, en la forma del registro *TFixed*, para el valor M22 de la matriz de transformación 3X3.

El registro *TFixed* define un número real en un formato de coma fija. El registro *TFixed* se define como:

*TFixed* = **packed record**

<i>fract</i> : Word;	{parte fraccionaria}
<i>value</i> : SHORT;	{parte entera}

**end;**

*fract*: Identifica la parte fraccionaria del número real.

*value*: Identifica la parte entera del número real.

#### Valor que devuelve

Si la función tiene éxito, devuelve un valor distinto de cero y el *buffer* al que apunta el parámetro *p6* contendrá la información de delineado del patrón. Si la función falla, devuelve *GDI\_ERROR*.

Véase además

*GetOutlineTextMetrics*

Ejemplo

#### Listado 8-II: Recuperando mapas de bits de patrones de caracteres

```

var
  Form1: TForm1;
  SelectedChar: Byte; // almacena el carácter seleccionado
  Angle: Integer;     // almacena el ángulo de rotación

implementation

{$R *.DFM}

function MakeFixed(Value: Double): TFixed;
var
  TheValue: longint; // variable de almacenamiento intermedio
begin
  {Convierte el número indicado en un registro TFixed}
  TheValue := Trunc(Value*65536);
  Result := TFixed(Longint(TheValue));
end;

procedure DrawGlyph;
var
  BitmapSize: Longint;           // tamaño necesario del mapa de bits
  BitmapBits: Pointer;          // un puntero al mapa de bits
  BitmapInfo: Windows.TBitmap;  // información del mapa de bits de Windows
  GlyphBitmap: HBITMAP;         // manejador del mapa de bits final
  GlyphMetrics: TGlyphMetrics;  // información sobre medidas del patrón
  Matrix: TMat2;                // matriz de rotación
begin
  {Inicializa la matriz de rotación. Observe que los valores de todos los ángulos
  deben convertirse a radianes}
  Matrix.eM11 := MakeFixed(Cos(Angle * (PI/180)));
  Matrix.eM12 := MakeFixed(Sin(Angle * (PI/180)));
  Matrix.eM21 := MakeFixed(-Sin(Angle * (PI/180)));
  Matrix.eM22 := MakeFixed(Cos(Angle * (PI/180)));

  {Recupera el tamaño requerido por el mapa de bits}
  BitmapSize := GetGlyphOutline(Form1.Canvas.Handle, SelectedChar, GGO_BITMAP,
                                GlyphMetrics, 0, nil, Matrix);

  {Reserva memoria suficiente para almacenar el mapa de bits}
  GetMem(BitmapBits, BitmapSize);

  {Recupera el mapa de bits del patrón}
  GetGlyphOutline(Form1.Canvas.Handle, SelectedChar, GGO_BITMAP, GlyphMetrics,
                  BitmapSize, BitmapBits, Matrix);

  {Inicializa el registro BitmapInfo para crear un mapa de bits de Windows}
  with BitmapInfo do

```

```

begin
    bmType := 0;
    bmWidth := (GlyphMetrics.gmBlackBoxX + 31) and not 31;
    bmHeight := GlyphMetrics.gmBlackBoxY;
    bmWidthBytes := bmWidth shr 3;
    bmPlanes := 1;
    bmBitsPixel := 1;
    bmBits := BitmapBits;
end;

{Crea el mapa de bits de Windows}
GlyphBitmap := CreateBitmapIndirect(BitmapInfo);

{Asigna el mapa de bits final a la imagen para mostrar}
Form1.Image1.Picture.Bitmap.Handle := GlyphBitmap;
Form1.Image1.Picture.Bitmap.Width := GlyphMetrics.gmBlackBoxX;

{Libera la memoria reservada para el mapa de bits}
FreeMem(BitmapBits, BitmapSize);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Crea el mapa de bits de la imagen e inicializa las variables}
    Image1.Picture.Bitmap := TBitmap.Create;
    SelectedChar := Ord('A');
    Angle := 0;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
    {Dibuja el mapa de bits en la activación}
    DrawGlyph;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    {Selecciona el carácter y dibuja su mapa de bits}
    SelectedChar := Ord(PChar(TSpeedButton(Sender).Caption)[0]);
    DrawGlyph;
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    {Cambia el ángulo de rotación y actualiza la pantalla}
    Angle := ScrollBar1.Position;
    Label2.Caption := IntToStr(Angle);
    DrawGlyph;
end;

```

Figure 8-12:  
El patrón  
rotado

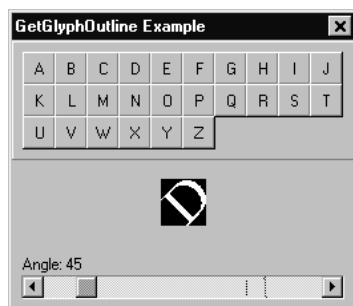


Tabla 8-25: Valores del parámetro p3 de GetGlyphOutline

Valor	Descripción
GGO_BITMAP	Recupera el delineado del patrón en forma de mapa de bits monocromático por filas y alineado a frontera de doble palabra.
GGO_NATIVE	Recupera el delineado del patrón en su formato nativo (una serie de líneas y curvas), medidas en unidades de diseño de la fuente. El parámetro p7 es ignorado.
GGO_METRICS	Recupera solamente la información de TGlyphMetrics para el parámetro p4.

### GetKerningPairs

### Windows.Pas

#### Sintaxis

```
GetKerningPairs(
    DC: HDC;                {manejador de contexto de dispositivo}
    Count: DWORD;           {cantidad de registros TKerningPair en el array}
    var KerningPairs        {puntero a un array de registros TKerningPair}
): DWORD;                  {devuelve la cantidad de registros recuperados}
```

#### Descripción

Esta función recupera las *parejas de kerning* (*kerning pairs*) de caracteres para la fuente actualmente seleccionada en el contexto de dispositivo identificado por el parámetro *DC*.

#### Parámetros

*DC*: Manejador del contexto de dispositivo de cuya fuente actualmente seleccionada serán recuperadas las parejas de *kerning* de caracteres.

*Count*: Indica la cantidad de registros *TKerningPair* en el *array* al que apunta el parámetro *KerningPairs*. Si la fuente seleccionada contiene más parejas de *kerning* que los que este parámetro indica, la función falla.

*KerningPairs*: Puntero a un *array* de registros *TKerningPair* que recibe las parejas de *kerning* de caracteres de la fuente seleccionada. Este *array* debe contener al menos tantos registros *TKerningPair* como los indicados por el parámetro *Count*. Si a este parámetro se le asigna **nil**, la función devuelve el número total de parejas de *kerning* en la fuente. El registro *TKerningPair* se define como:

*TKerningPair* = **packed record**

wFirst: Word;	{primer carácter de la pareja}
wSecond: Word;	{segundo carácter de la pareja}
iKernAmount: Integer;	{valor del <i>kerning</i> }

**end;**

*wFirst*: Especifica el valor del primer carácter de la pareja de *kerning*.

*wSecond*: Especifica el valor del segundo carácter de la pareja de *kerning*.

*iKernAmount*: Especifica el ajuste del espacio entre caracteres, en unidades lógicas, si los dos caracteres aparecen uno junto a otro en el mismo nombre de tipo de letra y tamaño. Generalmente este valor es negativo, lo que provoca que los caracteres sean situados más cerca uno del otro.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de parejas de *kerning* recuperadas; en caso contrario, devuelve cero.

#### Véase además

*GetTextCharacterExtra*, *SetTextCharacterExtra*

#### Ejemplo

##### Listado 8-12: Recuperando las parejas de kerning para la fuente actualmente seleccionada

{iOJO! Delphi importa incorrectamente esta función; debemos reimportarla manualmente para obtener toda la funcionalidad que nos ofrece esta función}

```
function GetKerningPairs(DC: HDC; Count: DWORD;
    KerningPairs: Pointer): DWORD; stdcall;
```

**var**

Form1: TForm1;

#### implementation

{ \$R \*.DFM }

{ Importación de la función }

```
function GetKerningPairs; external gdi32 name 'GetKerningPairs';
```

```
procedure TForm1.FormActivate(Sender: TObject);
```

**type**

TKerningPairs = **array**[0..0] **of** TKerningPair; // parejas de kerning

**var**

FaceName: **array**[0..255] **of** Char; // nombre de fuente de la letra seleccionada

```

KerningPairs: ^TKerningPairs;    // puntero al array de parejas
NumPairs: DWORD;                // almacena el número parejas
Count: Integer;                 // variable de control de bucle
begin
  {Recupera el nombre de la fuente actualmente seleccionada y la muestra}
  GetTextFace(Form1.Canvas.Handle, 255, @FaceName[0]);
  Label2.Caption := FaceName;

  {Recupera el número total de parejas de kerning en la fuente seleccionada}
  NumPairs := GetKerningPairs(Form1.Canvas.Handle, 0, nil);

  {Reserva memoria suficiente para almacenar todas las parejas}
  GetMem(KerningPairs, SizeOf(TKerningPair)*NumPairs);

  {Recupera las parejas de kerning para la fuente}
  GetKerningPairs(Form1.Canvas.Handle, NumPairs, KerningPairs);

  {Muestra todas las parejas y sus valores de kerning}
  Memo1.Lines.Clear;
  Memo1.Lines.Add('Pair' + #9 + 'Kern Amount');
  for Count := 0 to NumPairs-1 do
    Memo1.Lines.Add(Char(KerningPairs^[Count].wFirst) +
                     Char(KerningPairs^[Count].wSecond) + #9 +
                     IntToStr(KerningPairs^[Count].iKernAmount));

  {Libera la memoria del array de parejas}
  FreeMem(KerningPairs, SizeOf(TKerningPair) * NumPairs);
end;

```

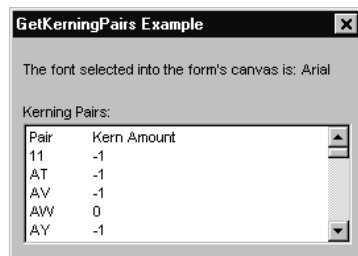


Figura 8-13:  
Las parejas de  
kerning

## GetOutlineTextMetrics

## Windows.Pas

### Sintaxis

```

GetOutlineTextMetrics(
  DC: HDC;                {manejador de contexto de dispositivo}
  p2: UINT;               {tamaño del buffer TOutlineTextMetric}
  OTMetricStructs: Pointer {puntero al buffer TOutlineTextMetric}
): UINT;                 {devuelve un código de error}

```

### Descripción

Esta función recupera información sobre las medidas, tales como la altura, ascenso, descenso y otras medidas físicas sólo para la fuente True Type actualmente seleccionada en el contexto de dispositivo identificado por el parámetro *DC*. Esta función ofrece información específica a las fuentes True Type, en adición a la información recuperada por la función *GetTextMetrics*.

### Parámetros

*DC*: Manejador del contexto de dispositivo de cuya fuente True Type actualmente seleccionada son recuperadas las medidas de texto.

*p2*: Especifica el tamaño del *buffer* al que apunta el parámetro *OTMetricStructs*, en bytes.

*OTMetricStructs*: Puntero a un *buffer* que recibe un registro *TOutlineTextMetric* que describe las medidas de texto de la fuente True Type. Si a este parámetro se le asigna **nil**, la función devuelve el tamaño necesario para el *buffer*. Debido a que al final de este registro se almacenan cadenas de caracteres, su tamaño es variable. El desarrollador debe llamar inicialmente a la función para determinar el tamaño apropiado y entonces reservar el *buffer* dinámicamente. Observe que las dimensiones devueltas en los campos de este registro se expresan en unidades lógicas y dependen del modo de mapeado del contexto de dispositivo especificado. El registro *TOutlineTextMetric* se define como:

*TOutlineTextMetric* = **record**

<i>otmSize</i> : UINT;	{tamaño del registro}
<i>otmTextMetrics</i> : TTextMetric;	{información adicional sobre la fuente}
<i>otmFiller</i> : Byte;	{relleno de alineación}
<i>otmPanoseNumber</i> : TPanose;	{especifica información PANOSE}
<i>otmfsSelection</i> : UINT;	{atributos inherentes de la fuente}
<i>otmfsType</i> : UINT;	{opciones de incrustación y licencia}
<i>otmsCharSlopeRise</i> : Integer;	{numerador de la inclinación del cursor de itálica}
<i>otmsCharSlopeRun</i> : Integer;	{denominador de la inclinación del cursor de itálica}
<i>otmItalicAngle</i> : Integer;	{ángulo de itálica}
<i>otmEMSquare</i> : UINT;	{dimensiones de la pica cuadrada}
<i>otmAscent</i> : Integer;	{ascenso tipográfico}
<i>otmDescent</i> : Integer;	{descenso tipográfico}
<i>otmLineGap</i> : UINT;	{espaciado de línea tipográfico}
<i>otmsCapEmHeight</i> : UINT;	{no se utiliza}
<i>otmsXHeight</i> : UINT;	{no se utiliza}
<i>otmrcFontBox</i> : TRect;	{rectángulo límite}
<i>otmMacAscent</i> : Integer;	{ascenso Macintosh}
<i>otmMacDescent</i> : Integer;	{descenso Macintosh}
<i>otmMacLineGap</i> : UINT;	{espaciado de línea Macintosh}

otmusMinimumPPEM: UINT;	{tamaño más pequeño recomendado}
otmptSubscriptSize: TPoint;	{tamaño de subíndice recomendado}
otmptSubscriptOffset: TPoint;	{desplazamiento de subíndice recomendado}
otmptSuperscriptSize: TPoint;	{tamaño de superíndice recomendado}
OtmptSuperscriptOffset: Tpoint;	{desplazamiento de superíndice aconsejado}
otmsStrikeoutSize: UINT;	{ancho de la línea de tachado}
otmsStrikeoutPosition: Integer;	{desplazamiento de tachado}
otmsUnderscoreSize: Integer;	{ancho de la línea de subrayado}
otmsUnderscorePosition: Integer;	{posición del subrayado}
otmpFamilyName: PAnsiChar;	{desplazamiento del nombre de la familia de fuentes}
otmpFaceName: PAnsiChar;	{desplazamiento del nombre}
otmpStyleName: PAnsiChar;	{desplazamiento del nombre del estilo}
otmpFullName: PAnsiChar;	{desplazamiento del nombre completo}

**end;**

*otmSize*: Especifica el tamaño del registro *TOutlineTextMetric* reservado, en bytes.

*otmTextMetrics*: Especifica un registro *TTextMetric* que contiene información física adicional para la fuente. El registro *TTextMetric* se define como:

**TTextMetric = record**

tmHeight: Longint;	{altura de un carácter}
tmAscent: Longint;	{ascenso de un carácter}
tmDescent: Longint;	{descenso de un carácter}
tmInternalLeading: Longint;	{guía interna}
tmExternalLeading: Longint;	{guía externa}
tmAveCharWidth: Longint;	{ancho promedio de un carácter}
tmMaxCharWidth: Longint;	{ancho máximo de un carácter}
tmWeight: Longint;	{grosor de la negrita}
tmOverhang: Longint;	{ancho que sobresale}
tmDigitizedAspectX: Longint;	{aspecto horizontal}
tmDigitizedAspectY: Longint;	{aspecto vertical}
tmFirstChar: AnsiChar;	{primer carácter}
tmLastChar: AnsiChar;	{último carácter}
tmDefaultChar: AnsiChar;	{carácter por defecto}
tmBreakChar: AnsiChar;	{carácter de ruptura de palabras}
tmItalic: Byte;	{indicador de itálica}
tmUnderlined: Byte;	{indicador de subrayado}
tmStruckOut: Byte;	{indicador de tachado}
tmPitchAndFamily: Byte;	{indicador de <i>pitch</i> y familia de fuentes}
tmCharSet: Byte;	{conjunto de caracteres}

**end;**

Consulte la función *GetTextMetrics* para ver una descripción de este registro.

*otmFiller*: Especifica un valor usado únicamente para alinear el registro.

*otmPanoseNumber*: Un registro *TPanose* que contiene la información de clasificación de fuente PANOSE para la fuente True Type. Esto es usado para asociar la fuente con otras fuentes que tienen una apariencia similar pero nombres distintos. El registro *TPanose* se define como:

*TPanose* = **packed record**

<i>bFamilyType</i> : Byte;	{tipo de familia}
<i>bSerifStyle</i> : Byte;	{estilo del <i>serif</i> }
<i>bWeight</i> : Byte;	{grosor}
<i>bProportion</i> : Byte;	{proporcionalidad}
<i>bContrast</i> : Byte;	{contraste}
<i>bStrokeVariation</i> : Byte;	{variación del trazo}
<i>bArmStyle</i> : Byte;	{estilo del brazo}
<i>bLetterform</i> : Byte;	{forma de la letra}
<i>bMidline</i> : Byte;	{posición de la línea media}
<i>bXHeight</i> : Byte;	{altura X}

end;

*bFamilyType*: Especifica el tipo de familia y puede contener un valor de la Tabla 8-26.

*bSerifStyle*: Especifica el estilo *serif* y puede contener un valor de la Tabla 8-27.

*bWeight*: Especifica el grosor de la fuente (intensidad de la negrita) y puede contener un valor de la Tabla 8-28.

*bProportion*: Especifica la proporcionalidad de la fuente y puede contener un valor de la Tabla 8-29.

*bContrast*: Especifica el contraste de la fuente y puede contener un valor de la Tabla 8-30.

*bStrokeVariation*: Especifica la variación del trazo dentro de la fuente y puede contener un valor de la Tabla 8-31.

*bArmStyle*: Especifica el estilo de brazo (*arm style*) de los patrones de la fuente y puede contener un valor de la Tabla 8-32.

*bLetterform*: Especifica la forma de letra de los patrones y puede contener un valor de la Tabla 8-33.

*bMidline*: Especifica la línea media de la fuente y puede contener un valor de la Tabla 8-34.

*bXHeight*: Especifica la altura X (*xheight*) de la fuente y puede contener un valor de la Tabla 8-35.

*otmfsSelection*: Especifica una máscara de bits que indica ciertos atributos que están implícitos en el patrón de la fuente, tales como negrita o itálica. Los bits de este campo indican los atributos, como se indica en la Tabla 8-36.

*otmfsType*: Especifica una máscara de bits que indica los atributos de licencia de la fuente. Si el bit 1 está seleccionado, la fuente no puede ser incrustada en un documento; si no está seleccionado, se puede incrustar. Si el bit 2 está seleccionado, la fuente puede ser incrustada solamente como una fuente *de-sólo-lectura*.

*otmsCharSlopeRise*: Conjuntamente con el campo *otmsCharSlopeRun*, este valor especifica el numerador de la relación usada para crear un cursor en itálica que tenga la misma inclinación que la fuente itálica, como se indica en el campo *otmItalicAngle*.

*otmsCharSlopeRun*: Conjuntamente con el campo *otmsCharSlopeRise*, este valor especifica el denominador de la relación usada para crear un cursor en itálica que tenga la misma inclinación que la fuente itálica, como se indica en el campo *otmItalicAngle*.

*otmItalicAngle*: Especifica el ángulo de la itálica para la fuente, en décimas de grado, rotando en dirección contraria a las manecillas del reloj, desde la vertical. La mayoría de las fuentes tienen un valor negativo que indica una fuente inclinada a la derecha. Este campo tiene valor cero para fuentes no itálica.

*otmEMSsquare*: Especifica las dimensiones horizontal y vertical, en unidades lógicas, de la pica cuadrada de la fuente.

*otmAscent*: El valor tipográfico que especifica la dimensión máxima en la cual los caracteres en esta fuente se elevan por encima de la línea base.

*otmDescent*: El valor tipográfico que especifica la dimensión máxima en la cual los caracteres en esta fuente descienden por debajo de la línea base.

*otmLineGap*: Especifica el espaciado tipográfico de la línea.

*otmsCapEmHeight*: Este campo ya no se utiliza.

*otmsXHeight*: Este campo ya no se utiliza.

*otmrcFontBox*: Especifica el rectángulo límite de la fuente.

*otmMacAscent*: La dimensión máxima en la cual los caracteres en esta fuente se elevan por encima de la línea base en un ordenador Macintosh.

*otmMacDescent*: La dimensión máxima en la cual los caracteres en esta fuente descienden por debajo de la línea base en un ordenador Macintosh.

*otmMacLineGap*: El espaciado de línea usado por esta fuente en un ordenador Macintosh.

*otmusMinimumPPEM*: Especifica el tamaño más pequeño recomendado de la fuente, en píxeles por pica cuadrada.

*otmptSubscriptSize*: Un registro *TPoint* que especifica el ancho y la altura recomendados para el estilo subíndice.

*otmptSubscriptOffset*: Un registro *TPoint* que especifica el desplazamiento horizontal y vertical recomendados para el estilo subíndice, desde el origen del carácter hasta el origen del subíndice.

*otmptSuperscriptSize*: Un registro *TPoint* que especifica el ancho y la altura recomendados para el estilo del superíndice.

*otmptSuperscriptOffset*: Un registro *TPoint* que especifica el desplazamiento horizontal y vertical recomendado para el estilo del superíndice, desde la línea base del carácter hasta la línea base del superíndice.

*otmsStrikeoutSize*: Especifica el ancho de la línea de tachado.

*otmsStrikeoutPosition*: Especifica el desplazamiento de la línea de tachado desde la línea base.

*otmsUnderscoreSize*: Especifica el ancho de la línea de subrayado.

*otmsUnderscorePosition*: Especifica el ancho de la línea de subrayado desde la línea base.

*otmpFamilyName*: Especifica el desplazamiento desde el principio del registro *TOutlineTextMetric* hasta el principio de la cadena que contiene el nombre de la familia de la fuente.

*otmpFaceName*: Especifica el desplazamiento desde el principio del registro *TOutlineTextMetric* hasta el principio de la cadena que contiene el nombre de la fuente.

*otmpStyleName*: Especifica el desplazamiento desde el principio del registro *TOutlineTextMetric* hasta el principio de la cadena que contiene el nombre del estilo de la fuente.

*otmpFullName*: Especifica el desplazamiento desde el principio del registro *TOutlineTextMetric* hasta el principio de la cadena que contiene el nombre completo y único de la fuente.

#### Valor que devuelve

Si la función tiene éxito, devuelve un valor distinto de cero; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetGlyphOutline*, *GetTextMetrics*

#### Ejemplo

##### Listado 8-13: Recuperando medidas del texto de una fuente True Type

{Nota: el formulario debe tener una fuente True Type seleccionada antes para que este ejemplo funcione correctamente}

```
procedure TForm1.FormActivate(Sender: TObject);
var
  FontInfo: POutlineTextMetric;      // puntero a información de medidas de texto
  FaceName: array[0..255] of Char;  // nombre de la fuente
  TheSize: LongInt;                 // tamaño requerido del buffer
begin
  {Recupera y muestra el nombre de la fuente seleccionada}
```

```

GetTextFace(Form1.Canvas.Handle, 256, FaceName);
Label2.Caption := FaceName;

{Recupera el tamaño requerido del buffer}
TheSize := GetOutlineTextMetrics(Form1.Canvas.Handle, 0, nil);

{Reserva el buffer}
GetMem(FontInfo, TheSize);

{Selecciona el campo del tamaño}
FontInfo^.otmSize := TheSize;

{Recupera los atributos de la fuente True Type}
GetOutlineTextMetrics(Form1.Canvas.Handle, TheSize, FontInfo);

{Borra el cuadro de lista y comienza a mostrar los atributos de la fuente física}
ListBox1.Items.Clear;
with FontInfo^.otmTextMetrics, ListBox1.Items do
begin
    {Muestra varias medidas de la fuente}
    Label15.Caption := IntToStr(tmHeight);
    Label14.Caption := IntToStr(tmAscent);
    Label13.Caption := IntToStr(tmDescent);
    Label12.Caption := IntToStr(tmInternalLeading);
    Label11.Caption := IntToStr(tmExternalLeading);

    {Muestra el ancho promedio y máximo de un carácter}
    Add('Average Char Width: ' + IntToStr(tmAveCharWidth));
    Add('Max Char Width: ' + IntToStr(tmMaxCharWidth));

    {Muestra los atributos de la negrita}
    case tmWeight of
        FW_DONTCARE: Add('Weight: Don't care');
        FW_THIN:     Add('Weight: Thin');
        FW_EXTRALIGHT: Add('Weight: Extra light');
        FW_LIGHT:    Add('Weight: Light');
        FW_NORMAL:   Add('Weight: Normal');
        FW_MEDIUM:   Add('Weight: Medium');
        FW_SEMIBOLD: Add('Weight: Semibold');
        FW_BOLD:     Add('Weight: Bold');
        FW_EXTRABOLD: Add('Weight: Extra bold');
        FW_HEAVY:    Add('Weight: Heavy');
    end;

    {Muestra el ancho de lo que sobresale}
    Add('Overhang: ' + IntToStr(tmOverhang));

    {Muestra los aspectos horizontal y vertical}
    Add('Digitized Aspect X: ' + IntToStr(tmDigitizedAspectX));
    Add('Digitized Aspect Y: ' + IntToStr(tmDigitizedAspectY));

    {Muestra los caracteres importantes de la fuente}
    Add('First Character: ' + Char(tmFirstChar));
    Add('Last Char: ' + Char(tmLastChar));
    Add('Default Char: ' + Char(tmDefaultChar));

```

```

Add('Break Char: ' + Char(tmBreakChar));

{Indica atributos de itálica, subrayado o tachado}
CheckBox1.Checked := (tmItalic>0);
CheckBox2.Checked := (tmUnderlined>0);
CheckBox3.Checked := (tmStruckOut>0);

{Muestra el pitch de la fuente}
Add('Pitch: ');
if ((tmPitchAndFamily and $0F) and TMPF_FIXED_PITCH) = TMPF_FIXED_PITCH then
  Add('    Fixed pitch');
if ((tmPitchAndFamily and $0F) and TMPF_VECTOR) = TMPF_VECTOR then
  Add('    Vector');
if ((tmPitchAndFamily and $0F) and TMPF_TRUETYPE) = TMPF_TRUETYPE then
  Add('    True Type');
if ((tmPitchAndFamily and $0F) and TMPF_DEVICE) = TMPF_DEVICE then
  Add('    Device');
if (tmPitchAndFamily and $0F) = 0 then
  Add('    Monospaced bitmap font');

{Muestra la familia de la fuente}
case (tmPitchAndFamily and $F0) of
  FF_DECORATIVE: Add('Family: Decorative');
  FF_DONTCARE:   Add('Family: Don't care');
  FF_MODERN:     Add('Family: Modern');
  FF_ROMAN:      Add('Family: Roman');
  FF_SCRIPT:     Add('Family: Script');
  FF_SWISS:      Add('Family: Swiss');
end;

{Muestra el conjunto de caracteres}
case tmCharSet of
  ANSI_CHARSET:      Add('Character set: ANSI');
  DEFAULT_CHARSET:   Add('Character set: Default');
  SYMBOL_CHARSET:    Add('Character set: Symbol');
  SHIFTJIS_CHARSET:  Add('Character set: ShiftJis');
  GB2312_CHARSET:    Add('Character set: GB2312');
  HANGEUL_CHARSET:   Add('Character set: Hangeul');
  CHINESEBIG5_CHARSET: Add('Character set: Chinese Big5');
  OEM_CHARSET:       Add('Character set: OEM');
else
  Add('Windows 95 only character set');
end;
end;

{Muestra información True Type específica}
with FontInfo^, ListBox1.Items do
begin
  Add('');
  Add('');
  Add('True Type specific information');
  Add('-----');
  Add('');
  Add('Panose Information: ');

```

```

{Muestra el tipo de familia Panose}
case otmPanoseNumber.bFamilyType of
  PAN_ANY:           Add('    Family Type: Any');
  PAN_NO_FIT:        Add('    Family Type: No fit');
  PAN_FAMILY_TEXT_DISPLAY: Add('    Family Type: Text and display');
  PAN_FAMILY_SCRIPT: Add('    Family Type: Script');
  PAN_FAMILY_DECORATIVE: Add('    Family Type: Decorative');
  PAN_FAMILY_PICTORIAL: Add('    Family Type: Pictorial');
end;

{Muestra el estilo serif Panose}
case otmPanoseNumber.bSerifStyle of
  PAN_ANY:           Add('    Serif Style: Any');
  PAN_NO_FIT:        Add('    Serif Style: No fit');
  PAN_SERIF_COVE:     Add('    Serif Style: Cove');
  PAN_SERIF_OBTUSE_COVE: Add('    Serif Style: Obtuse cove');
  PAN_SERIF_SQUARE_COVE: Add('    Serif Style: Square cove');
  PAN_SERIF_OBTUSE_SQUARE_COVE: Add('    Serif Style: Obtuse square cove');
  PAN_SERIF_SQUARE:   Add('    Serif Style: Square');
  PAN_SERIF_THIN:     Add('    Serif Style: Thin');
  PAN_SERIF_BONE:     Add('    Serif Style: Bone');
  PAN_SERIF_EXAGGERATED: Add('    Serif Style: Exaggerated');
  PAN_SERIF_TRIANGLE: Add('    Serif Style: Triangle');
  PAN_SERIF_NORMAL_SANS: Add('    Serif Style: Normal sans serif');
  PAN_SERIF_OBTUSE_SANS: Add('    Serif Style: Obtuse sans serif');
  PAN_SERIF_PERP_SANS: Add('    Serif Style: Perp sans serif');
  PAN_SERIF_FLARED:   Add('    Serif Style: Flared');
  PAN_SERIF_ROUNDED:  Add('    Serif Style: Rounded');
end;

{Muestra el grosor Panose}
case otmPanoseNumber.bWeight of
  PAN_ANY:           Add('    Weight: Any');
  PAN_NO_FIT:        Add('    Weight: No fit');
  PAN_WEIGHT_VERY_LIGHT: Add('    Weight: Very light');
  PAN_WEIGHT_LIGHT:   Add('    Weight: Light');
  PAN_WEIGHT_THIN:    Add('    Weight: Thin');
  PAN_WEIGHT_BOOK:    Add('    Weight: Book');
  PAN_WEIGHT_MEDIUM:  Add('    Weight: Medium');
  PAN_WEIGHT_DEMI:    Add('    Weight: Demi');
  PAN_WEIGHT_BOLD:    Add('    Weight: Bold');
  PAN_WEIGHT_HEAVY:   Add('    Weight: Heavy');
  PAN_WEIGHT_BLACK:   Add('    Weight: Black');
  PAN_WEIGHT_NORD:    Add('    Weight: Nord');
end;

{Muestra la proporción Panose}
case otmPanoseNumber.bProportion of
  PAN_ANY:           Add('    Proportion: Any');
  PAN_NO_FIT:        Add('    Proportion: No fit');
  PAN_PROP_OLD_STYLE: Add('    Proportion: Old style');
  PAN_PROP_MODERN:    Add('    Proportion: Modern');
  PAN_PROP_EVEN_WIDTH: Add('    Proportion: Even width');
  PAN_PROP_EXPANDED:  Add('    Proportion: Expanded');

```

```

PAN_PROP_CONDENSED:      Add('    Proportion: Condensed');
PAN_PROP_VERY_EXPANDED:  Add('    Proportion: Very expanded');
PAN_PROP_VERY_CONDENSED: Add('    Proportion: Very condensed');
PAN_PROP_MONOSPACED:     Add('    Proportion: Monospaced');
end;

{Muestra el contraste Panose}
case otmPanoseNumber.bContrast of
  PAN_ANY:      Add('    Contrast: Any');
  PAN_NO_FIT:   Add('    Contrast: No fit');
  PAN_CONTRAST_NONE: Add('    Contrast: None');
  PAN_CONTRAST_VERY_LOW: Add('    Contrast: Very low');
  PAN_CONTRAST_LOW: Add('    Contrast: Low');
  PAN_CONTRAST_MEDIUM_LOW: Add('    Contrast: Medium low');
  PAN_CONTRAST_MEDIUM: Add('    Contrast: Medium');
  PAN_CONTRAST_MEDIUM_HIGH: Add('    Contrast: Medium high');
  PAN_CONTRAST_HIGH: Add('    Contrast: High');
  PAN_CONTRAST_VERY_HIGH: Add('    Contrast: Very high');
end;

{Muestra la variación de trazo Panose}
case otmPanoseNumber.bStrokeVariation of
  PAN_ANY:      Add('    Stroke variation: Any');
  PAN_NO_FIT:   Add('    Stroke variation: No fit');
  PAN_STROKE_GRADUAL_DIAG: Add('    Stroke variation: Gradual diagonal');
  PAN_STROKE_GRADUAL_TRAN: Add('    Stroke variation: Gradual transition');
  PAN_STROKE_GRADUAL_VERT: Add('    Stroke variation: Gradual vertical');
  PAN_STROKE_GRADUAL_HORZ: Add('    Stroke variation: Gradual horizontal');
  PAN_STROKE_RAPID_VERT: Add('    Stroke variation: Rapid vertical');
  PAN_STROKE_RAPID_HORZ: Add('    Stroke variation: Rapid horizontal');
  PAN_STROKE_INSTANT_VERT: Add('    Stroke variation: Instant vertical');
end;

{Muestra el estilo de brazo Panose}
case otmPanoseNumber.bArmStyle of
  PAN_ANY:      Add('    Arm style: Any');
  PAN_NO_FIT:   Add('    Arm style: No fit');
  PAN_STRAIGHT_ARMS_HORZ: Add('    Arm style: Straight horizontal');
  PAN_STRAIGHT_ARMS_WEDGE: Add('    Arm style: Straight wedge');
  PAN_STRAIGHT_ARMS_VERT: Add('    Arm style: Straight vertical');
  PAN_STRAIGHT_ARMS_SINGLE_SERIF: Add('    Arm style: Straight single-serif');
  PAN_STRAIGHT_ARMS_DOUBLE_SERIF: Add('    Arm style: Straight double-serif');
  PAN_BENT_ARMS_HORZ: Add('    Arm style: Nonstraight '+'
                                'horizontal');
  PAN_BENT_ARMS_WEDGE: Add('    Arm style: Nonstraight wedge');
  PAN_BENT_ARMS_VERT: Add('    Arm style: Nonstraight vertical');
  PAN_BENT_ARMS_SINGLE_SERIF: Add('    Arm style: Nonstraight '+'
                                'single-serif');
  PAN_BENT_ARMS_DOUBLE_SERIF: Add('    Arm style: Nonstraight '+'
                                'double-serif');
end;

{Muestra la forma de letra Panose}
case otmPanoseNumber.bLetterform of
  PAN_ANY:      Add('    Letter form: Any');

```

```

PAN_NO_FIT: Add(' Letter form: No fit');
PAN_LETT_NORMAL_CONTACT: Add(' Letter form: Normal contact');
PAN_LETT_NORMAL_WEIGHTED: Add(' Letter form: Normal weighted');
PAN_LETT_NORMAL_BOXED: Add(' Letter form: Normal boxed');
PAN_LETT_NORMAL_FLATTENED: Add(' Letter form: Normal flattened');
PAN_LETT_NORMAL_ROUNDED: Add(' Letter form: Normal rounded');
PAN_LETT_NORMAL_OFF_CENTER: Add(' Letter form: Normal off center');
PAN_LETT_NORMAL_SQUARE: Add(' Letter form: Normal square');
PAN_LETT_OBLIQUE_CONTACT: Add(' Letter form: Oblique contact');
PAN_LETT_OBLIQUE_WEIGHTED: Add(' Letter form: Oblique weighted');
PAN_LETT_OBLIQUE_BOXED: Add(' Letter form: Oblique boxed');
PAN_LETT_OBLIQUE_FLATTENED: Add(' Letter form: Oblique flattened');
PAN_LETT_OBLIQUE_ROUNDED: Add(' Letter form: Oblique rounded');
PAN_LETT_OBLIQUE_OFF_CENTER: Add(' Letter form: Oblique off center');
PAN_LETT_OBLIQUE_SQUARE: Add(' Letter form: Oblique square');
end;

{Muestra la línea media Panose}
case otmPanoseNumber.bMidline of
  PAN_ANY: Add(' Midline: Any');
  PAN_NO_FIT: Add(' Midline: No fit');
  PAN_MIDLINE_STANDARD_TRIMMED: Add(' Midline: Standard trimmed');
  PAN_MIDLINE_STANDARD_POINTED: Add(' Midline: Standard pointed');
  PAN_MIDLINE_STANDARD_SERIFED: Add(' Midline: Standard serifed');
  PAN_MIDLINE_HIGH_TRIMMED: Add(' Midline: High trimmed');
  PAN_MIDLINE_HIGH_POINTED: Add(' Midline: High pointed');
  PAN_MIDLINE_HIGH_SERIFED: Add(' Midline: High serifed');
  PAN_MIDLINE_CONSTANT_TRIMMED: Add(' Midline: Constant trimmed');
  PAN_MIDLINE_CONSTANT_POINTED: Add(' Midline: Constant pointed');
  PAN_MIDLINE_CONSTANT_SERIFED: Add(' Midline: Constant serifed');
  PAN_MIDLINE_LOW_TRIMMED: Add(' Midline: Low trimmed');
  PAN_MIDLINE_LOW_POINTED: Add(' Midline: Low pointed');
  PAN_MIDLINE_LOW_SERIFED: Add(' Midline: Low serifed');
end;

{Muestra la altura X Panose}
case otmPanoseNumber.bXHeight of
  PAN_ANY: Add(' XHeight: Any');
  PAN_NO_FIT: Add(' XHeight: No fit');
  PAN_XHEIGHT_CONSTANT_SMALL: Add(' XHeight: Constant small');
  PAN_XHEIGHT_CONSTANT_STD: Add(' XHeight: Constant standard');
  PAN_XHEIGHT_CONSTANT_LARGE: Add(' XHeight: Constant large');
  PAN_XHEIGHT_DUCKING_SMALL: Add(' XHeight: Ducking small');
  PAN_XHEIGHT_DUCKING_STD: Add(' XHeight: Ducking standard');
  PAN_XHEIGHT_DUCKING_LARGE: Add(' XHeight: Ducking large');
end;

{Muestra los atributos inherentes de la fuente}
Add('Selection: ');
if (otmfsSelection and $01) > 0 then
  Add(' Italic');
if (otmfsSelection and $02) > 0 then
  Add(' Underscore');
if (otmfsSelection and $04) > 0 then
  Add(' Negative');

```

```

if (otmfsSelection and $08) > 0 then
    Add('    Outline');
if (otmfsSelection and $10) > 0 then
    Add('    Strikeout');
if (otmfsSelection and $20) > 0 then
    Add('    Bold');

{Muestra información de incrustación de la fuente}
Add('Type:');
if (otmfsType and $02) > 0 then
    Add('    Embedding Forbidden');
if (otmfsType and $02) < 1 then
    Add('    Embedding Allowed');
if (otmfsType and $04) > 0 then
    Add('    Embedding Read-Only');

{Muestra atributos de itálica}
Add('Slope Rise: ' + IntToStr(otmsCharSlopeRise));
Add('Slope Run: ' + IntToStr(otmsCharSlopeRun));
Add('Italic Angle: ' + IntToStr(otmItalicAngle));

{Muestra atributos físicos importantes}
Add('EM Square: ' + IntToStr(otmEMSquare));
Add('Typographic Ascent: ' + IntToStr(otmAscent));
Add('Typographic Descent: ' + IntToStr(otmDescent));
Add('Typographic Line Gap: ' + IntToStr(otmLineGap));

{Muestra las coordenadas de la caja límite}
Add('Font Bounding Box: ');
Add('    Left: ' + IntToStr(otmrcFontBox.Left));
Add('    Top: ' + IntToStr(otmrcFontBox.Top));
Add('    Right: ' + IntToStr(otmrcFontBox.Right));
Add('    Bottom: ' + IntToStr(otmrcFontBox.Bottom));

{Muestra los atributos Macintosh}
Add('Mac Ascent: ' + IntToStr(otmMacAscent));
Add('MacDescent: ' + IntToStr(otmMacDescent));
Add('Mac Line Gap: ' + IntToStr(otmMacLineGap));

{Muestra el tamaño mínimo}
Add('Minimum Size: ' + IntToStr(otmusMinimumPPEM));

{Muestra sugerencias de subíndice}
Add('Subscript Size: ');
Add('    Horizontal: ' + IntToStr(otmptSubscriptSize.X));
Add('    Vertical: ' + IntToStr(otmptSubscriptSize.Y));
Add('Subscript Offset: ');
Add('    Horizontal: ' + IntToStr(otmptSubscriptOffset.X));
Add('    Vertical: ' + IntToStr(otmptSubscriptOffset.Y));

{Muestra sugerencias de superíndice}
Add('Superscript Size: ');
Add('    Horizontal: ' + IntToStr(otmptSuperscriptSize.X));
Add('    Vertical: ' + IntToStr(otmptSuperscriptSize.Y));

```

```

Add('Superscript Offset: ');
Add('    Horizontal: ' + IntToStr(otmptSuperscriptOffset.X));
Add('    Vertical: ' + IntToStr(otmptSuperscriptOffset.Y));

{Muestra el tamaño y posiciones de tachado}
Add('Strikeout Size: ' + IntToStr(otmsStrikeoutSize));
Add('Strikeout Position: ' + IntToStr(otmsStrikeoutPosition));
Add('Underscore Size: ' + IntToStr(otmsUnderscoreSize));
Add('Underscore Position: ' + IntToStr(otmsUnderscorePosition));

{Muestra cadenas de familia de fuente, tipo de letra y nombre}
Add('Family Name: ' + PChar(Longint(FontInfo) + FontInfo^.otmpFamilyName));
Add('Face Name: ' + PChar(Longint(FontInfo) + FontInfo^.otmpFaceName));
Add('Style Name: ' + PChar(Longint(FontInfo) + FontInfo^.otmpStyleName));
end;

{Muestra el nombre completo de la fuente}
Label17.Caption := PChar(Longint(FontInfo) + FontInfo^.otmpFullName);

{Libera el buffer reservado}
FreeMem(FontInfo, TheSize);
end;

```

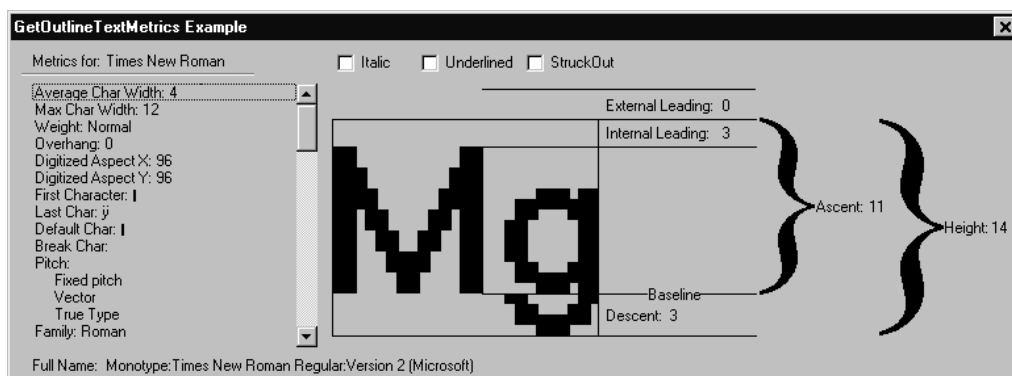


Figura 8-14: La información de la fuente True Type

**Tabla 8-26: Valores del campo OTMetricStructs.otmPanoseNumber.bFamilyType de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier familia.
PAN_NO_FIT	No hay coincidencia.
PAN_FAMILY_TEXT_DISPLAY	Familia de texto y presentación.
PAN_FAMILY_SCRIPT	Familia Script.
PAN_FAMILY_DECORATIVE	Familia Decorative.
PAN_FAMILY_PICTORIAL	Familia Pictorial.

**Tabla 8-27: Valores del campo OTMetricStructs.otmPanoseNumber.bSerifStyle de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier estilo serif.
PAN_NO_FIT	No hay coincidencia.
PAN_SERIF_COVE	Serifs internos.
PAN_SERIF_OBTUSE_COVE	Serifs internos obtusos.
PAN_SERIF_SQUARE_COVE	Serifs internos cuadrados.
PAN_SERIF_OBTUSE_SQUARE_COVE	Serifs obtusos y cuadrados.
PAN_SERIF_SQUARE	Serifs cuadrados.
PAN_SERIF_THIN	Serifs delgados.
PAN_SERIF_BONE	Serifs “de hueso”.
PAN_SERIF_EXAGGERATED	Serifs exagerados.
PAN_SERIF_TRIANGLE	Serifs triangulares.
PAN_SERIF_NORMAL_SANS	Sans serif normal.
PAN_SERIF_OBTUSE_SANS	Sans serif obtuso.
PAN_SERIF_PERP_SANS	Sans serif perpendicular.
PAN_SERIF_FLARED	Serifs iluminados.
PAN_SERIF_ROUNDED	Serifs redondeados.

**Tabla 8-28: Valores del campo OTMetricStructs.otmPanoseNumber.bWeight de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier negrita.
PAN_NO_FIT	No hay coincidencia.
PAN_WEIGHT_VERY_LIGHT	Negrita muy ligera.
PAN_WEIGHT_LIGHT	Negrita ligera.
PAN_WEIGHT_THIN	Negrita delgada.
PAN_WEIGHT_BOOK	Negrita Book.
PAN_WEIGHT_MEDIUM	Negrita Medium.
PAN_WEIGHT_DEMI	Demibold.
PAN_WEIGHT_BOLD	Negrita.
PAN_WEIGHT_HEAVY	Negrita Heavy.
PAN_WEIGHT_BLACK	Negrita Black.
PAN_WEIGHT_NORD	Negrita Nord.

**Tabla 8-29: Valores del campo OTMetricStructs.otmPanoseNumber.bProportion de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier proporción.
PAN_NO_FIT	No hay coincidencia.
PAN_PROP_OLD_STYLE	Proporción Old style.
PAN_PROP_MODERN	Proporción Modern.
PAN_PROP_EVEN_WIDTH	Proporción Even width.
PAN_PROP_EXPANDED	Proporción Expanded.
PAN_PROP_CONDENSED	Proporción Condensed.
PAN_PROP_VERY_EXPANDED	Proporción Very expanded.
PAN_PROP_VERY_CONDENSED	Proporción Very condensed.
PAN_PROP_MONOSPACED	Proporción Monospaced.

**Tabla 8-30: Valores del campo OTMetricStructs.otmPanoseNumber.bContrast de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier contraste.
PAN_NO_FIT	No hay coincidencia.
PAN_CONTRAST_NONE	Sin contraste.
PAN_CONTRAST_VERY_LOW	Contraste muy bajo.
PAN_CONTRAST_LOW	Contraste bajo.
PAN_CONTRAST_MEDIUM_LOW	Contraste medio bajo.
PAN_CONTRAST_MEDIUM	Contraste medio.
PAN_CONTRAST_MEDIUM_HIGH	Contraste medio alto.
PAN_CONTRAST_HIGH	Contraste alto.
PAN_CONTRAST_VERY_HIGH	Contraste muy alto.

**Tabla 8-31: Valores del campo OTMetricStructs.otmPanoseNumber.bStrokeVariation de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier variación de trazo.
PAN_NO_FIT	No hay coincidencia.
PAN_STROKE_GRADUAL_DIAG	Variación diagonal gradual del trazo.
PAN_STROKE_GRADUAL_TRAN	Variación transiccional gradual del trazo.
PAN_STROKE_GRADUAL_VERT	Variación vertical gradual del trazo.
PAN_STROKE_GRADUAL_HORZ	Variación horizontal gradual del trazo.
PAN_STROKE_RAPID_VERT	Variación vertical rápida del trazo.
PAN_STROKE_RAPID_HORZ	Variación horizontal rápida del trazo.
PAN_STROKE_INSTANT_VERT	Variación vertical instantánea del trazo.

**Tabla 8-32: Valores del campo OTMetricStructs.otmPanoseNumber.bArmStyle de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier estilo de brazo.
PAN_NO_FIT	No hay coincidencia.
PAN_STRAIGHT_ARMS_HORZ	Brazos rectos en horizontal.
PAN_STRAIGHT_ARMS_WEDGE	Brazos rectos en cuña.
PAN_STRAIGHT_ARMS_VERT	Brazos rectos en vertical.
PAN_STRAIGHT_ARMS_SINGLE_SERIF	Brazos rectos, con serif sencillo.
PAN_STRAIGHT_ARMS_DOUBLE_SERIF	Brazos rectos, con serif doble.
PAN_BENT_ARMS_HORZ	Brazos doblados en horizontal.
PAN_BENT_ARMS_WEDGE	Brazos doblados en cuña.
PAN_BENT_ARMS_VERT	Brazos doblados en vertical.
PAN_BENT_ARMS_SINGLE_SERIF	Brazos doblados, serif sencillo.
PAN_BENT_ARMS_DOUBLE_SERIF	Brazos doblados, serif doble.

**Tabla 8-33: Valores del campo OTMetricStructs.otmPanoseNumber.bLetterform de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier forma de letra.
PAN_NO_FIT	No hay coincidencia.
PAN_LETT_NORMAL_CONTACT	Letra normal contacto.
PAN_LETT_NORMAL_WEIGHTED	Letra normal ponderada.
PAN_LETT_NORMAL_BOXED	Letra normal rectangular.
PAN_LETT_NORMAL_FLATTENED	Letra normal aplanada.
PAN_LETT_NORMAL_ROUNDED	Letra normal redondeada.
PAN_LETT_NORMAL_OFF_CENTER	Letra normal descentrada.
PAN_LETT_NORMAL_SQUARE	Letra normal cuadrada.
PAN_LETT_OBLIQUE_CONTACT	Letra oblicua de contacto.
PAN_LETT_OBLIQUE_WEIGHTED	Letra oblicua ponderada.
PAN_LETT_OBLIQUE_BOXED	Letra oblicua rectangular.
PAN_LETT_OBLIQUE_FLATTENED	Letra oblicua aplanada.
PAN_LETT_OBLIQUE_ROUNDED	Letra oblicua redondeada.
PAN_LETT_OBLIQUE_OFF_CENTER	Letra oblicua descentrada.
PAN_LETT_OBLIQUE_SQUARE	Letra oblicua cuadrada.

**Tabla 8-34: Valores del campo OTMetricStructs.otmPanoseNumber.bMidline de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier línea media.
PAN_NO_FIT	No hay coincidencia.
PAN_MIDLINE_STANDARD_TRIMMED	Línea media estándar recortada.
PAN_MIDLINE_STANDARD_POINTED	Línea media estándar punteada.
PAN_MIDLINE_STANDARD_SERIFED	Línea media estándar con serif.
PAN_MIDLINE_HIGH_TRIMMED	Línea media alta recortada.
PAN_MIDLINE_HIGH_POINTED	Línea media alta punteada.
PAN_MIDLINE_HIGH_SERIFED	Línea media alta con serif.
PAN_MIDLINE_CONSTANT_TRIMMED	Línea media constante recortada.
PAN_MIDLINE_CONSTANT_POINTED	Línea media constante punteada.
PAN_MIDLINE_CONSTANT_SERIFED	Línea media constante con serif.
PAN_MIDLINE_LOW_TRIMMED	Línea media baja recortada.
PAN_MIDLINE_LOW_POINTED	Línea media baja punteada.
PAN_MIDLINE_LOW_SERIFED	Línea media baja con serif.

**Tabla 8-35: Valores del campo OTMetricStructs.otmPanoseNumber.bXHeight de GetOutlineTextMetrics**

Valor	Descripción
PAN_ANY	Cualquier altura X.
PAN_NO_FIT	No hay coincidencia.
PAN_XHEIGHT_CONSTANT_SMALL	Altura X constante pequeña.
PAN_XHEIGHT_CONSTANT_STD	Altura X constante estándar.
PAN_XHEIGHT_CONSTANT_LARGE	Altura X constante grande.
PAN_XHEIGHT_DUCKING_SMALL	Cursiva con altura X pequeña.
PAN_XHEIGHT_DUCKING_STD	Cursiva con altura X estándar.
PAN_XHEIGHT_DUCKING_LARGE	Cursiva con altura X grande.

**Tabla 8-36: Valores del campo OTMetricStructs.otmfsSelection de GetOutlineTextMetrics**

Bit	Descripción
0	Indica una fuente itálica.
1	Indica una fuente con subrayado.
2	Indica una fuente negativa.
3	Indica una fuente delineada.
4	Indica una fuente tachada.
5	Indica una fuente negrita.

**GetRasterizerCaps**      **Windows.Pas****Sintaxis**

```
GetRasterizerCaps(
    var p1: TRasterizerStatus; {puntero a registro TRasterizerStatus}
    p2: UINT                    {tamaño del registro TRasterizerStatus}
); BOOL;                       {devuelve TRUE o FALSE}
```

**Descripción**

Esta función devuelve información en el registro *TRasterizerStatus* al que apunta el parámetro *p1*, que indica si las fuentes True Type están instaladas o habilitadas en el sistema.

**Parámetros**

*p1*: Puntero a un registro *TRasterizerStatus* que recibe información concerniente a la accesibilidad de las fuentes True Type en el sistema. El registro *TRasterizerStatus* se define como:

*TRasterizerStatus* = **packed record**

```
    nSize: SHORT;           {tamaño del registro TRasterizerStatus}
    wFlags: SHORT;          {indicadores de accesibilidad de True Type}
    nLanguageID: SHORT;     {identificador de idioma}
```

**end;**

*nSize*: Especifica el tamaño del registro *TRasterizerStatus*, en bytes. A este campo se le debe asignar *SizeOf(TRasterizerStatus)*.

*wFlags*: Una serie de indicadores que especifican la disponibilidad de fuentes True Type. Este campo puede contener uno o más valores de la Tabla 8-37.

*nLanguageID*: Especifica el identificador de idioma, según se indica en el fichero del sistema **Setup.inf**.

*p2*: Especifica el número de bytes a copiar en el registro *TRasterizerStatus*. La cantidad de bytes copiados será el valor de este parámetro o el tamaño del registro *TRasterizerStatus*, el que sea menor.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetOutlineTextMetrics*, *GetTextMetrics*

**Ejemplo**

Vea el Listado 8-7 bajo *EnumFontFamilies*.

Tabla 8-37: Valores del campo `pl.wFlags` de `GetRasterizerCaps`

Valor	Descripción
TT_AVAILABLE	Al menos una fuente True Type está instalada y es accesible.
TT_ENABLED	Las fuentes True Type son soportadas por el sistema.

**GetTabbedTextExtent**      **Windows.Pas****Sintaxis**

```
GetTabbedTextExtent(
    hDC: HDC;                {manejador de contexto de dispositivo}
    lpString: PChar;          {cadena cuyas dimensiones serán determinadas}
    nCount: Integer;          {cantidad de caracteres en la cadena}
    nTabPositions: Integer;    {cantidad de tabulaciones}
    lpnTabStopPositions: Pointer {puntero a un array de posiciones de tabulación}
); DWORD;                    {devuelve el ancho y la altura de la cadena}
```

**Descripción**

Esta función devuelve el ancho y la altura que necesitará para su visualización una cadena que contiene caracteres de tabulación. La fuente actualmente seleccionada en el contexto de dispositivo especificado es utilizada para determinar las dimensiones de la cadena, y cualquier carácter de tabulación en la cadena es expandido hasta la parada de tabulación que se indica en el *array* al que apunta el parámetro *lpnTabStopPositions*. La región de recorte actual del contexto de dispositivo especificado no afecta a las dimensiones calculadas. En los casos en que una cadena que contenga parejas de *kerning* sea mostrada sobre un dispositivo que soporte el *kerning*, las dimensiones devueltas por esta función pueden no corresponderse con la suma de las dimensiones individuales de los caracteres en la cadena.

**Parámetros**

*hDC*: Manejador del contexto de dispositivo cuya fuente actualmente seleccionada es usada para determinar la longitud de la cadena.

*lpString*: Puntero a una cadena de caracteres terminada en nulo que contiene el texto con caracteres de tabulación.

*nCount*: Especifica la longitud de la cadena a la que apunta el parámetro *lpString*, en caracteres.

*nTabPositions*: Especifica la cantidad de entradas en el *array* de paradas de tabulación al que apunta el parámetro *lpnTabStopPositions*.

*lpnTabStopPositions*: Puntero a un *array* de enteros. Cada entrada en el *array* indica una posición (*parada*) de tabulación, en unidades del dispositivo. Las paradas de tabulación tienen que ser colocadas en un orden creciente, con la más pequeña como primera entrada en el *array*. Si a este parámetro se le asigna **nil** y el parámetro

*nTabPositions* tiene valor cero, los caracteres de tabulación son expandidos a ocho veces el ancho promedio de los caracteres de la fuente seleccionada.

#### Valor que devuelve

Si la función tiene éxito, devuelve el ancho y la altura de la cadena, donde la altura es la palabra más significativa del valor devuelto y el ancho la palabra menos significativa. Si la función falla, devuelve cero.

#### Véase además

*GetTextExtentPoint32*, *TabbedTextOut*

#### Ejemplo

Vea el Listado 8-17 bajo *TabbedTextOut*.

### **GetTextAlign**      **Windows.Pas**

#### Sintaxis

```
GetTextAlign(
    DC: HDC                      {manejador de contexto de dispositivo}
): UINT;                        {devuelve las opciones de alineación del texto}
```

#### Descripción

Esta función recupera un conjunto de indicadores de las características de alineación de texto actuales para el contexto de dispositivo especificado. La alineación se ejecuta en base a un rectángulo límite que rodea los caracteres de la cadena a mostrar. Las dimensiones del rectángulo límite para una cadena pueden ser recuperadas llamando a *GetTextExtentPoint32*. La alineación de texto se basa en el punto de comienzo de la cadena, según se define en las funciones de salida de texto tales como *TextOut*.

#### Parámetros

*DC*: Manejador del contexto de dispositivo cuya alineación de texto será recuperada.

#### Valor que devuelve

Si la función tiene éxito, devuelve una o más opciones de alineación de texto de la Tabla 8-38; en caso contrario, devuelve *GDI\_ERROR*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*. A diferencia de la mayoría de las funciones, las posibles opciones de alineación de texto no representan bits individuales y no pueden ser combinadas con el valor devuelto para determinar si un indicador determinado está presente. En su lugar, los indicadores deben detectarse inspeccionando los siguientes grupos de opciones relacionadas:

TA\_LEFT, TA\_RIGHT y TA\_CENTER  
 TA\_BOTTOM, TA\_TOP y TA\_BASELINE  
 TA\_NOUPDATECP y TA\_UPDATECP

Para fuentes de línea base vertical, las opciones relacionadas son:

TA\_LEFT, TA\_RIGHT y VTA\_BASELINE  
 TA\_BOTTOM, TA\_TOP y VTA\_CENTER  
 TA\_NOUPDATECP y TA\_UPDATECP

Para determinar si un indicador particular está presente en el valor devuelto por la función, el grupo de opciones relacionadas tiene que ser combinado mediante el operador booleano **or**, y el resultado entonces combinado con el valor devuelto usando el operador booleano **and**. Por ejemplo, para determinar si el texto está alineado a la derecha, asuma que *TextAlignment* contiene el valor de retorno de una llamada a la función *GetTextAlign* y use la fórmula:

```
if (TextAlignment and (TA_LEFT or TA_CENTER or TA_RIGHT)) = TA_RIGHT then
  Label2.Caption := 'Right';
```

Véase además

*DrawText*, *DrawTextEx*, *GetTextExtentPoint32*, *SetTextAlign*, *TextOut*

**Ejemplo**

Vea el Listado 8-16 bajo *SetTextAlign*.

**Tabla 8-38: Valores que devuelve GetTextAlign**

Valor	Descripción
TA_BASELINE	El punto de comienzo está sobre la línea base del texto.
TA_BOTTOM	El punto de comienzo está sobre el borde inferior del rectángulo límite del texto.
TA_TOP	El punto de comienzo está sobre el borde superior del rectángulo límite del texto.
TA_CENTER	El punto de comienzo es el centro horizontal del rectángulo límite del texto.
TA_LEFT	El punto de comienzo está a la izquierda del rectángulo límite del texto.
TA_RIGHT	El punto de comienzo está a la derecha del rectángulo límite del texto.
TA_RTLREADING	Sólo Windows 95/98: Indica que el orden de lectura del texto es de derecha a izquierda. Este valor sólo tiene sentido cuando la fuente seleccionada es hebrea o árabe.
TA_NOUPDATECP	La posición actual no se actualiza después de dibujar texto.
TA_UPDATECP	La posición actual se actualiza después de dibujar texto.
VTA_BASELINE	Sólo para fuentes de línea base vertical: El punto de comienzo está sobre la línea base del texto.
VTA_CENTER	Solo para fuentes de línea base vertical: El punto de comienzo es el centro vertical del rectángulo límite del texto.

**GetTextCharacterExtra****Windows.Pas****Sintaxis**

```
GetTextCharacterExtra(
    DC: HDC                {manejador de contexto de dispositivo}
): Integer;                {devuelve el espaciado entre caracteres}
```

**Descripción**

Esta función recupera la cantidad de espacio extra, en unidades lógicas, que se añade entre caracteres cuando se dibuja una línea de texto sobre el contexto de dispositivo especificado.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo valor de espaciado extra entre caracteres será recuperado.

**Valor que devuelve**

Si la función tiene éxito, devuelve la cantidad de espacio extra que se añade entre caracteres; en caso contrario, devuelve \$80000000.

**Véase además**

*DrawText*, *DrawTextEx*, *SetTextCharacterExtra*, *TextOut*

**Ejemplo**

Vea el Listado 8-16 bajo *SetTextAlign*.

**GetTextColor****Windows.Pas****Sintaxis**

```
GetTextColor(
    DC: HDC                {manejador de contexto de dispositivo}
): COLORREF;              {devuelve un especificador de colores de 32 bits}
```

**Descripción**

Esta función recupera el color actual usado cuando se dibuja texto sobre el contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo color de texto será recuperado.

**Valor que devuelve**

Si la función tiene éxito, devuelve el especificador de color de 32 bits que identifica el color usado cuando se dibuja texto. Si la función falla, devuelve *CLR\_INVALID*.

Véase además

*SetTextColor*, *TextOut*

### Ejemplo

Vea el Listado 8-16 bajo *SetTextAlign*.

## GetTextExtentExPoint Windows.Pas

### Sintaxis

```
GetTextExtentExPoint(
    DC: HDC;                {manejador de contexto de dispositivo}
    p2: PChar;               {cadena cuya extensión de caracteres se desea obtener}
    p3: Integer;             {cantidad de caracteres en la cadena}
    p4: Integer;             {ancho máximo de la cadena}
    p5: PInteger;            {entero que recibe la cantidad máxima de caracteres}
    p6: Pointer;             {puntero a un array de enteros que recibe las extensiones}
    var p7: TSize;           {registro TSize que recibe las dimensiones de la cadena}
    ): BOOL;                 {devuelve TRUE o FALSE}
```

### Descripción

Esta función recupera la cantidad máxima de caracteres de la cadena a la que apunta el parámetro *p2*, que cabrán dentro del ancho máximo permitido, especificado por el parámetro *p4*. Adicionalmente, la función rellena un *array* de enteros cuyos elementos se corresponden con cada carácter de la cadena con el desplazamiento desde el principio de la cadena al principio del carácter cuando sea dibujado sobre el contexto de dispositivo *DC*. La fuente actualmente seleccionada en el contexto de dispositivo especificado es utilizada para determinar el máximo permitido de caracteres y desplazamientos.

### Parámetros

*DC*: Manejador del contexto de dispositivo cuya fuente seleccionada es utilizada para determinar la dimensión del texto.

*p2*: Puntero a una cadena de caracteres terminada en nulo cuyas extensiones de texto serán recuperadas.

*p3*: Especifica el tamaño de la cadena a la que apunta el parámetro *p2*, en bytes.

*p4*: Especifica el ancho máximo permitido de la cadena de salida sobre el contexto de dispositivo, en unidades lógicas.

*p5*: Puntero a un entero que recibirá la cantidad máxima de caracteres que cabrán en el espacio lógico del contexto de dispositivo identificado mediante el parámetro *p4*. Si a este parámetro se le asigna **nil**, el parámetro *p4* es ignorado.

*p6*: Puntero a un *array* de enteros que recibe las posiciones en las que estará situado cada carácter individual de la cadena a la que apunta el parámetro *p2*. Cada entrada en el *array* está asociada con el carácter en la posición correspondiente de la cadena y contiene el desplazamiento desde el principio de la cadena al origen del carácter cuando es dibujado en la pantalla. Este desplazamiento caerá siempre dentro del ancho máximo especificado en el parámetro *p4*. Si bien debe haber tantas entradas en el array como caracteres en la cadena *p2*, la función sólo rellenará la cantidad de entradas del array indicadas a través del parámetro *p5*. A este parámetro se le puede asignar **nil**, si las dimensiones individuales de los caracteres no son necesarias.

*p7*: Puntero a un registro *TSize* que recibe el ancho y la altura de la cadena especificada, en unidades lógicas.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetTextExtentPoint32*

#### Ejemplo

##### Listado 8-14: Justificando texto por programa

{iOJO! Delphi importa esta función incorrectamente. La reimportamos manualmente para obtener su funcionalidad completa}

**function** GetTextExtentExPoint(DC: HDC; p2: PChar;  
p3, p4: Integer; p5: PInteger; p6: Pointer; **var** p7: TSize): BOOL; **stdcall**;

**var**

Form1: TForm1;

**implementation**

{ \$R \*.DFM }

{ Reimportar la función }

**function** GetTextExtentExPoint; **external** gdi32 **name** 'GetTextExtentExPointA';

**procedure** TForm1.PaintBox1Paint(Sender: TObject);

**var**

TheString: PChar;	// almacena la cadena de salida
StrPointer: PChar;	// puntero a la cadena de salida
DisplayString: PChar;	// almacena la cadena actualmente mostrada
MaxChars: Integer;	// recibe el máximo de caracteres visualizables
StringSize: TSize;	// recibe las dimensiones de la cadena
LineNum: Integer;	// un contador de número de líneas
ExtraSpace: Integer;	// almacena el espacio extra a añadir
NumBreaks: Integer;	// almacena el número de espacios en una cadena
Count: Integer;	// variable de control de bucle

```

begin
    {Borra la imagen del área de dibujo del cuadro de dibujo}
    with PaintBox1.Canvas do
    begin
        Brush.Color := clWhite;
        FillRect(ClipRect);
    end;

    {Inicializa la cadena original}
    TheString:='Delphi is the most awesome Windows development environment ever!';

    {Inicializa el número de líneas y el puntero de la cadena}
    LineNum := 0;
    StrPointer := TheString;

    {Recupera suficiente memoria para la cadena mostrada}
    GetMem(DisplayString, Length(TheString));

    {Recorre la cadena hasta que es mostrada completamente}
    while Length(StrPointer) > 0 do
    begin
        {Recupera la cantidad máxima de caracteres que caben en una línea del paintbox}
        GetTextExtentExPoint(PaintBox1.Canvas.Handle, TheString,
                               Length(TheString), PaintBox1.Width, @MaxChars,
                               nil, StringSize);

        {Si el resto de la cadena supera lo que puede ser mostrado en una línea y el
        último carácter a mostrar no es un espacio, continua decrementando el máximo
        de caracteres visualizables hasta que se encuentra un espacio}
        while (Length(StrPointer) > MaxChars) and (StrPointer[MaxChars] <> ' ') do
            Inc(MaxChars, -1);

        {Copia solamente la cantidad calculada de caracteres en la cadena visualizable.
        Esta nueva cadena debe caber dentro del paintbox sin dividir ninguna palabra}
        StrLCopy(DisplayString, StrPointer, MaxChars);

        {Si el resto de la cadena supera lo que se puede mostrar, mueve el puntero de
        la cadena más allá del final de la cadena visualizable; en caso contrario,
        hace apuntar el puntero de la cadena a una cadena vacía}
        if Length(StrPointer) > MaxChars then
            StrPointer := @StrPointer[MaxChars+1]
        else
            StrPointer := #0;

        {Recupera el ancho y altura de la cadena}
        GetTextExtentPoint32(PaintBox1.Canvas.Handle, DisplayString,
                               Length(DisplayString), StringSize);

        {Para justificar el texto de manera que llene la línea completa, calcula la
        cantidad de espacio entre el tamaño de la cadena y el ancho del paintbox}
        ExtraSpace := PaintBox1.Width - StringSize.cx;

        {Cuenta el número de caracteres de ruptura en la cadena mostrada. Observe que
        se asume que el carácter de ruptura es un espacio (' ')}

```

```

NumBreaks := 0;
for Count := 0 to Length(DisplayString) - 1 do
  if DisplayString[Count] = ' ' then
    Inc(NumBreaks);

{Si hay al menos un espacio, asigna la justificación de texto. Esto añadirá la
cantidad calculada de espacios extra de forma regular entre todos los
espacios en la línea, ejecutando así una justificación completa cuando la
cadena sea dibujada en el contexto de dispositivo}
if NumBreaks > 0 then
  SetTextJustification(PaintBox1.Canvas.Handle, ExtraSpace, NumBreaks);

{Dibuja la cadena completamente justificada en el contexto del paintbox}
TextOut(PaintBox1.Canvas.Handle, 0, LineNum * Stringsized.cy, DisplayString,
  Length(DisplayString));

{Reinicia la justificación de texto a su valor original para el paso próximo}
SetTextJustification(PaintBox1.Canvas.Handle, 0, 0);

{Incrementa el número de línea del texto actual}
Inc(LineNum);
end;

{Libera la memoria}
FreeMem(DisplayString, Length(TheString));
end;

```

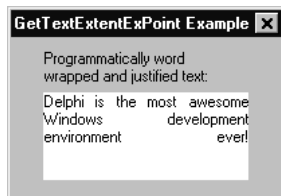


Figura 8-15:  
El texto  
justificado

## GetTextExtentPoint32 Windows.Pas

### Sintaxis

```

GetTextExtentPoint32(
  DC: HDC;           {manejador de contexto de dispositivo}
  Str: PChar;        {puntero a una cadena}
  Count: Integer;    {cantidad de caracteres en la cadena}
  var Size: TSize;   {apunta al registro TSize que recibe las dimensiones}
); BOOL;             {devuelve TRUE o FALSE}

```

### Descripción

Esta función recupera el ancho y la altura de la cadena a la que apunta el parámetro *Str*, en unidades lógicas. El ancho y la altura se calculan en base a los atributos de la cadena

actualmente seleccionada en el contexto de dispositivo identificado por el parámetro *DC*. La región de recorte del contexto de dispositivo especificado no afecta a las dimensiones calculadas. En los casos en que una cadena que contiene parejas de *kerning* se muestre por un dispositivo que soporte el *kerning* de caracteres, las dimensiones devueltas por esta función pueden no corresponderse con la suma de las dimensiones de los caracteres individuales en la cadena.

#### Parámetros

*DC*: Manejador del contexto de dispositivo cuya fuente actualmente seleccionada es usada para determinar el ancho y altura de la cadena.

*Str*: Puntero a una cadena cuyo ancho y altura serán recuperados. No tiene que ser una cadena de caracteres terminada en nulo, ya que el parámetro *Count* especifica la longitud.

*Count*: Especifica la cantidad de caracteres que hay en la cadena a la que apunta el parámetro *Str*.

*Size*: Puntero a un registro *TSize* que recibe el ancho y la altura de la cadena especificada, basado en los atributos de la fuente seleccionada en el contexto de dispositivo especificado.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetTabbedTextExtent*, *GetTextExtentExPoint*, *SetTextCharacterExtra*

#### Ejemplo

Vea el Listado 8-14 bajo *GetTextExtentExPoint*.

### GetTextFace

### Windows.Pas

#### Sintaxis

```
GetTextFace(
    DC: HDC;                {manejador de contexto de dispositivo}
    Count: Integer;          {longitud del buffer}
    Buffer: PChar             {buffer que recibe el nombre del tipo de letra}
): Integer;                 {devuelve el número de caracteres copiados}
```

#### Descripción

Esta función recupera el nombre del tipo de letra de la fuente actualmente seleccionada en el contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo nombre de tipo de letra actualmente seleccionada será recuperada.

*Count*: Especifica el tamaño del *buffer* al que apunta el parámetro *Buffer*, en caracteres. Si el nombre del tipo de letra recuperado es más largo que el valor especificado por este parámetro, la cadena es truncada.

*Buffer*: Puntero a un *buffer* de cadena de caracteres terminada en nulo que recibe el nombre del tipo de letra de la fuente actualmente seleccionada. Si a este parámetro se le asigna **nil**, la función devuelve el tamaño necesario para el *buffer*, en caracteres e incluyendo el terminador nulo.

**Valor que devuelve**

Si la función tiene éxito, devuelve el número de caracteres copiados al *buffer* al que apunta el parámetro *Buffer*. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*EnumFontFamilies*, *EnumFontFamiliesEx*, *GetTextAlign*, *GetTextColor*, *GetTextMetrics*

**Ejemplo**

Vea el Listado 8-15 bajo *GetTextMetrics*.

**GetTextMetrics****Windows.Pas****Sintaxis**

```
GetTextMetrics(
    DC: HDC;                {manejador de contexto de dispositivo}
    var TM: TTextMetric      {puntero a un registro TTextMetric}
); BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recupera información sobre las medidas, tales como altura, ascenso, descenso y otras medidas físicas, para la fuente seleccionada en el contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyas medidas de la fuente actualmente seleccionada serán recuperadas.

*TM*: Puntero a un registro *TTextMetric* que recibe las medidas físicas y otros atributos de la fuente seleccionada en el contexto de dispositivo especificado. Observe que todas

las medidas están en unidades lógicas y son dependientes del modo de mapeado del contexto de dispositivo. El registro *TTextMetric* se define como:

TNewTextMetric = **record**

tmHeight: Longint;	{altura del carácter}
tmAscent: Longint;	{ascenso del carácter}
tmDescent: Longint;	{descenso del carácter}
tmInternalLeading: Longint;	{guía interna}
tmExternalLeading: Longint;	{guía externa}
tmAveCharWidth: Longint;	{ancho promedio del carácter}
tmMaxCharWidth: Longint;	{ancho máximo del carácter}
tmWeight: Longint;	{valor del peso de la negrita}
tmOverhang: Longint;	{ancho que sobresale}
tmDigitizedAspectX: Longint;	{aspecto horizontal}
tmDigitizedAspectY: Longint;	{aspecto vertical }
tmFirstChar: AnsiChar;	{primer carácter}
tmLastChar: AnsiChar;	{último carácter}
tmDefaultChar: AnsiChar;	{carácter por defecto}
tmBreakChar: AnsiChar;	{carácter de ruptura de palabra}
tmItalic: Byte;	{indicador de itálica}
tmUnderlined: Byte;	{indicador de subrayado}
tmStruckOut: Byte;	{indicador de tachado}
tmPitchAndFamily: Byte;	{opciones de pitch y familia}
tmCharSet: Byte;	{conjunto de caracteres}

end;

*tmHeight*: Especifica la altura de los caracteres de la fuente. La altura de los caracteres se calcula mediante la suma *tmAscent* + *tmDescent*.

*tmAscent*: Especifica el ascenso de los caracteres en la fuente. El ascenso es medido desde la línea base a la cima del carácter, incluyendo la guía interna.

*tmDescent*: Especifica el descenso de los caracteres en la fuente. El descenso es medido desde la línea base al final del carácter e incluye la parte inferior de caracteres como “g” o “y”.

*tmInternalLeading*: Especifica la cantidad de espacio dentro del ascenso reservado para elementos como los acentos y las marcas diacríticas. El diseñador de la fuente puede poner este valor a cero.

*tmExternalLeading*: Especifica la cantidad de espacio extra sobre la cima de la fuente. Este espacio está destinado a añadir un área extra entre las líneas de texto y no contiene marcas. El diseñador de la fuente puede poner este valor a cero.

*tmAveCharWidth*: Especifica el ancho promedio de los caracteres en la fuente, excluyendo cualquier ancho adicional requerido para caracteres en itálica o negrita.

*tmMaxCharWidth*: Especifica el ancho del carácter más ancho en la fuente.

*tmWeight*: Especifica el grosor de la negrita de la fuente. El valor de este campo puede estar en el rango de 0-1000, ó puede ser un valor de la Tabla 8-39.

*tmOverhang*: Especifica el ancho extra por cadena que es añadido cuando se sintetizan fuentes en itálica o negrita. Para fuentes en negrita, este valor indica el desplazamiento de la línea superior (*overstrike*). Para fuentes en itálica, este valor indica la distancia de *shearing*. Utilice el valor devuelto por una llamada a la función *GetTextExtentPoint32* sobre un carácter simple, menos el valor de este campo, para determinar el ancho actual del carácter.

*tmDigitizedAspectX*: Especifica el aspecto horizontal del dispositivo para el cual la fuente fue originalmente diseñada.

*tmDigitizedAspectY*: Especifica el aspecto vertical del dispositivo para el cual la fuente fue originalmente diseñada.

*tmFirstChar*: Especifica el valor del primer carácter definido.

*tmLastChar*: Especifica el valor del último carácter definido.

*tmDefaultChar*: Especifica el valor del carácter por defecto. Este carácter es utilizado cuando la salida de texto contiene un carácter no definido en la fuente.

*tmBreakChar*: Especifica el valor del carácter usado para la ruptura de palabras y la justificación de texto.

*tmItalic*: Especifica el atributo de itálica para la fuente. Si a este miembro se le asigna TRUE, la fuente es *itálica*.

*tmUnderlined*: Especifica el atributo de subrayado para la fuente. Si a este miembro se le asigna TRUE, la fuente es subrayada.

*tmStruckOut*: Especifica el atributo de tachado para la fuente. Si a este miembro se le asigna TRUE, la fuente es ~~tachada~~.

*tmPitchAndFamily*: Especifica la familia y el *pitch* de la fuente. Los cuatro bits menos significativos especifican el *pitch* de la fuente y pueden contener uno o más valores de la Tabla 8-40. Los cuatro bits más significativos indican la familia de la fuente. Combinando este campo con un valor \$F0 usando el operador booleano **and** se recuperará un valor que se corresponde con alguno de los valores de la Tabla 8-41.

*tmCharSet*: Especifica el conjunto de caracteres de la fuente. Este campo puede contener un valor de la Tabla 8-42.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*EnumFontFamilies*, *EnumFontFamiliesEx*, *GetTextAlign*, *GetTextExtentExPoint*, *GetTextExtentPoint32*, *GetTextFace*, *SetTextJustification*

*Ejemplo***Listado 8-15: Recuperando información sobre las medidas de la fuente**

```

procedure TForm1.FormActivate(Sender: TObject);
var
    FontInfo: TTextMetric;           // información sobre las medidas de la fuente
    FaceName: array[0..255] of Char; // nombre de la fuente
begin
    {Recupera el nombre de la fuente actualmente seleccionada y la muestra}
    GetTextFace(Form1.Canvas.Handle, 256, FaceName);
    Label2.Caption := FaceName;

    {Recupera los atributos físicos de la fuente seleccionada}
    GetTextMetrics(Form1.Canvas.Handle, FontInfo);

    {Limpia el cuadro de lista y comienza a mostrar los atributos de la fuente}
    ListBox1.Items.Clear;
    with FontInfo, ListBox1.Items do
    begin
        {Muestra las diferentes medidas de la fuente}
        Label15.Caption := IntToStr(tmHeight);
        Label14.Caption := IntToStr(tmAscent);
        Label13.Caption := IntToStr(tmDescent);
        Label12.Caption := IntToStr(tmInternalLeading);
        Label11.Caption := IntToStr(tmExternalLeading);

        {Muestra el ancho promedio y máximo de los caracteres}
        Add('Average Char Width: ' + IntToStr(tmAveCharWidth));
        Add('Max Char Width: ' + IntToStr(tmMaxCharWidth));

        {Muestra los atributos de la negrita}
        case tmWeight of
            FW_DONTCARE: Add('Weight: Don't care');
            FW_THIN:     Add('Weight: Thin');
            FW_EXTRALIGHT: Add('Weight: Extra light');
            FW_LIGHT:     Add('Weight: Light');
            FW_NORMAL:    Add('Weight: Normal');
            FW_MEDIUM:    Add('Weight: Medium');
            FW_SEMIBOLD:  Add('Weight: Semibold');
            FW_BOLD:      Add('Weight: Bold');
            FW_EXTRABOLD: Add('Weight: Extra bold');
            FW_HEAVY:     Add('Weight: Heavy');
        end;

        {Muestra el ancho que sobresale}
        Add('Overhang: ' + IntToStr(tmOverhang));

        {Muestra los aspectos horizontal y vertical. Nota: Hay un error en la función
        GetTextMetrics que provoca que estos dos valores sean intercambiados. El
        valor AspectX es devuelto en el campo AspectY y viceversa}
        Add('Digitized Aspect X: ' + IntToStr(tmDigitizedAspectY));
        Add('Digitized Aspect Y: ' + IntToStr(tmDigitizedAspectX));
    end;

```

```

{Muestra los caracteres importantes de la fuente}
Add('First Character: ' + Char(tmFirstChar));
Add('Last Char: ' + Char(tmLastChar));
Add('Default Char: ' + Char(tmDefaultChar));
Add('Break Char: ' + Char(tmBreakChar));

{Indica los atributos de itálica, subrayado o tachado}
CheckBox1.Checked := (tmItalic > 0);
CheckBox2.Checked := (tmUnderlined > 0);
CheckBox3.Checked := (tmStruckOut > 0);

{Muestra el pitch de la fuente}
Add('Pitch: ');
if ((tmPitchAndFamily and $OF) and TMPF_FIXED_PITCH) = TMPF_FIXED_PITCH then
  Add('    Fixed pitch');
if ((tmPitchAndFamily and $OF) and TMPF_VECTOR) = TMPF_VECTOR then
  Add('    Vector');
if ((tmPitchAndFamily and $OF) and TMPF_TRUETYPE) = TMPF_TRUETYPE then
  Add('    True Type');
if ((tmPitchAndFamily and $OF) and TMPF_DEVICE) = TMPF_DEVICE then
  Add('    Device');
if (tmPitchAndFamily and $OF) = 0 then
  Add('    Monospaced bitmap font');

{Muestra la familia de la fuente}
case (tmPitchAndFamily and $F0) of
  FF_DECORATIVE: Add('Family: Decorative');
  FF_DONTCARE:   Add('Family: Don't care');
  FF_MODERN:     Add('Family: Modern');
  FF_ROMAN:      Add('Family: Roman');
  FF_SCRIPT:     Add('Family: Script');
  FF_SWISS:      Add('Family: Swiss');
end;

{Muestra el conjunto de caracteres}
case tmCharSet of
  ANSI_CHARSET:      Add('Character set: ANSI');
  DEFAULT_CHARSET:   Add('Character set: Default');
  SYMBOL_CHARSET:    Add('Character set: Symbol');
  SHIFTJIS_CHARSET:  Add('Character set: ShiftJis');
  GB2312_CHARSET:    Add('Character set: GB2312');
  HANGEUL_CHARSET:   Add('Character set: Hangeul');
  CHINESEBIG5_CHARSET: Add('Character set: Chinese Big5');
  OEM_CHARSET:       Add('Character set: OEM');
else
  Add('Windows 95 only character set');
end;
end;
end;
end;

```

Figura 8-16:  
La información  
de las medidas  
de la fuente  
actual

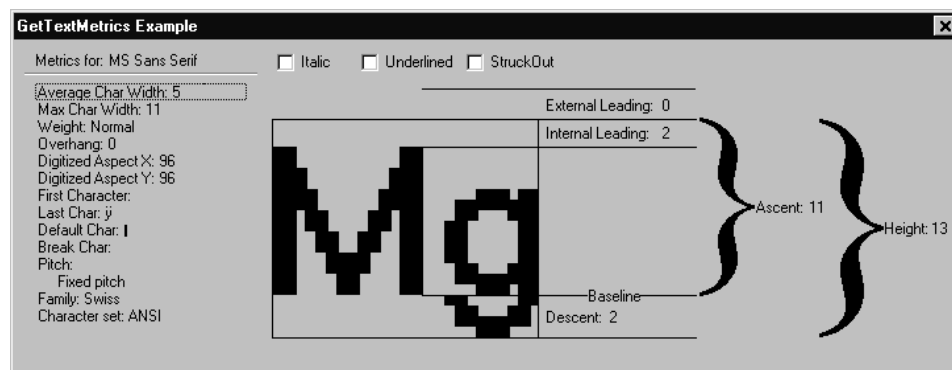


Tabla 8-39: Valores del campo **TM.tmWeight** de **GetTextMetrics**

Valor	Descripción
FW_THIN	Trazo de fuente extrafino (100).
FW_EXTRALIGHT	Trazo de fuente fino (200).
FW_LIGHT	Negrita con grosor ligeramente inferior a lo normal (300).
FW_NORMAL	Negrita normal (400).
FW_MEDIUM	Negrita con grosor ligeramente superior a lo normal (500).
FW_SEMIBOLD	Negrita ligera (600).
FW_BOLD	Negrita (700).
FW_EXTRABOLD	Negrita extra (800).
FW_HEAVY	Negrita extragruesa (900).

Tabla 8-40: Valores de pitch de fuentes de **GetTextMetrics** **TM.tmPitchAndFamily**

Valor	Descripción
TMPF_FIXED_PITCH	Si este indicador está presente, la fuente es de pitch variable. Si no está presente, la fuente es de pitch fijo, o monoespaciada.
TMPF_VECTOR	Indica una fuente vectorial.
TMPF_TRUETYPE	Indica una fuente True Type.
TMPF_DEVICE	Indica una fuente de dispositivo.

Tabla 8-41: Valores de familia de fuentes de **GetTextMetrics** **TM.tmPitchAndFamily**

Valor	Descripción
FF_DECORATIVE	Indica una fuente decorativa, como Old English.
FF_DONTCARE	El estilo general de la fuente es desconocido o no es importante.
FF_MODERN	Indica una fuente monoespaciada con trazos de ancho consistentes, con o sin serifs, como Courier New.

Valor	Descripción
FF_ROMAN	Indica una fuente proporcional con trazos de ancho variable y con serifs, como Times New Roman.
FF_SCRIPT	Indica una fuente que imita el manuscrito, como Brush Script.
FF_SWISS	Indica una fuente proporcional con trazos de ancho variable y sin serifs, como Arial.

Tabla 8-42: Valores del campo **TM.tmCharSet** de **GetTextMetrics**

Valor	Descripción
ANSI_CHARSET	El conjunto de caracteres ANSI.
DEFAULT_CHARSET	El conjunto de caracteres por defecto.
SYMBOL_CHARSET	El conjunto de caracteres Symbol.
SHIFTJIS_CHARSET	El conjunto de caracteres Shift-JIS.
GB2312_CHARSET	El conjunto de caracteres GB2312.
HANGEUL_CHARSET	El conjunto de caracteres coreano.
CHINESEBIG5_CHARSET	El conjunto de caracteres chino.
OEM_CHARSET	El conjunto de caracteres original del fabricante del equipo.
JOHAB_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Johab.
HEBREW_CHARSET	Sólo Windows 95/98: El conjunto de caracteres hebreo.
ARABIC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres árabe.
GREEK_CHARSET	Sólo Windows 95/98: El conjunto de caracteres griego.
TURKISH_CHARSET	Sólo Windows 95/98: El conjunto de caracteres turco.
VIETNAMESE_CHARSET	Sólo Windows 95/98: El conjunto de caracteres vietnamita.
THAI_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Thai.
EASTEUROPE_CHARSET	Sólo Windows 95/98: El conjunto de caracteres de Europa del Este.
RUSSIAN_CHARSET	Sólo Windows 95/98: El conjunto de caracteres cirílico.
MAC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres Macintosh.
BALTIC_CHARSET	Sólo Windows 95/98: El conjunto de caracteres báltico.

**RemoveFontResource****Windows.Pas***Sintaxis*

```
RemoveFontResource(
    p1: PChar           {nombre de fichero de recursos de fuentes}
); BOOL;               {devuelve TRUE o FALSE}
```

**Descripción**

Esta función elimina los recursos de fuentes contenidos en el fichero de recursos de fuentes especificado de la tabla interna de fuentes del sistema. Si la fuente es eliminada con éxito, la aplicación que la eliminó debe informar a las demás aplicaciones del cambio. Esto se logra enviando un mensaje *WM\_FONTCHANGE* mediante la función *SendMessage*, especificando *HWND\_BROADCAST* como valor del parámetro *hWnd*. El recurso de fuente no será eliminado hasta que no sea deseleccionado de cualquier contexto de dispositivo.

**Parámetros**

*p1*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del fichero de recursos de fuentes cuyos recursos de fuentes serán eliminados de la tabla interna de fuentes del sistema.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*AddFontResource*, *CreateScalableFontResource*, *GetFontData*, *WM\_FONTCHANGE*

**Ejemplo**

Vea el Listado 8-4 bajo *CreateScalableFontResource*.

**SetTextAlign      Windows.Pas****Sintaxis**

```
SetTextAlign(
    DC: HDC;                {manejador de contexto de dispositivo}
    Flags: UINT              {opciones de alineación de texto}
): UINT;                   {devuelve las opciones de alineación anteriores}
```

**Descripción**

Esta función selecciona los atributos de alineación usados cuando se dibuja texto sobre el contexto de dispositivo especificado. La alineación se ejecuta en base a un rectángulo límite que rodea los caracteres de la cadena a mostrar. Las dimensiones del rectángulo límite para una cadena pueden ser recuperadas llamando a *GetTextExtentPoint32*. La alineación de texto se basa en el punto de comienzo de la cadena, según se define en las funciones de salida de texto tales como *TextOut*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuya alineación de texto será establecida.

*Flags*: Una combinación de valores que indican el nuevo modo de alineación de texto para el contexto de dispositivo especificado. A este parámetro se le pueden asignar uno o más valores de la Tabla 8-43, combinándolos mediante el operador booleano **or**. Sin embargo, sólo puede elegirse UN indicador de aquellos que modifican la alineación horizontal o vertical, y sólo UN indicador de aquellos que modifican la posición actual.

#### Valor que devuelve

Si esta función tiene éxito, devuelve las opciones de alineación de texto anteriores; en caso contrario, devuelve *GDI\_ERROR*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*DrawText*, *DrawTextEx*, *GetTextAlign*, *TabbedTextOut*, *TextOut*

#### Ejemplo

##### Listado 8-16: Manipulando texto

```
var
  Form1: TForm1;
  HorzAlignmentValue: UINT;           // alineación horizontal
  VertAlignmentValue: UINT;           // alineación vertical
  IntercharacterSpacing: Integer;     // espaciado entre caracteres

implementation

{$R *.DFM}

procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  TextAlignment: UINT; // almacena la alineación de texto
begin
  {Asigna la alineación de texto}
  SetTextAlign(PaintBox1.Canvas.Handle,
    HorzAlignmentValue or VertAlignmentValue);

  {Selecciona el espaciado entre caracteres}
  SetTextCharacterExtra(PaintBox1.Canvas.Handle, SpinEdit1.Value);

  {Recupera y muestra el espaciado actual entre caracteres}
  Label17.Caption := IntToStr(GetTextCharacterExtra(PaintBox1.Canvas.Handle));

  {Selecciona el color del texto}
  SetTextColor(PaintBox1.Canvas.Handle, ColorGrid1.ForegroundColor);

  {Recupera y muestra el color actual del texto}
  Label19.Caption := IntToHex(GetTextColor(PaintBox1.Canvas.Handle), 8);

  {Dibuja algún texto (afectado por alineación, espaciado y color) en el
  contexto de dispositivo}
  TextOut(PaintBox1.Canvas.Handle, PaintBox1.Width div 2,
    PaintBox1.Height div 2, 'ABCabc', Length('ABCabc'));
```

```

{Recupera la alineación actual del texto}
TextAlignment := GetTextAlign(PaintBox1.Canvas.Handle);

{Muestra la alineación horizontal}
if (TextAlignment and (TA_LEFT or TA_CENTER or TA_RIGHT)) = TA_LEFT then
  Label2.Caption := 'Left';
if (TextAlignment and (TA_LEFT or TA_CENTER or TA_RIGHT)) = TA_CENTER then
  Label2.Caption := 'Center';
if (TextAlignment and (TA_LEFT or TA_CENTER or TA_RIGHT)) = TA_RIGHT then
  Label2.Caption := 'Right';

{Muestra la alineación vertical}
if (TextAlignment and (TA_TOP or TA_BASELINE or TA_BOTTOM)) = TA_TOP then
  Label4.Caption := 'Top';
if (TextAlignment and (TA_TOP or TA_BASELINE or TA_BOTTOM)) = TA_BASELINE then
  Label4.Caption := 'Baseline';
if (TextAlignment and (TA_TOP or TA_BASELINE or TA_BOTTOM)) = TA_BOTTOM then
  Label4.Caption := 'Bottom';
end;

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  {Indica la alineación horizontal seleccionada}
  HorzAlignmentValue := 0;
  case TRadioButton(Sender).Tag of
    1: HorzAlignmentValue := TA_LEFT;
    2: HorzAlignmentValue := TA_CENTER;
    3: HorzAlignmentValue := TA_RIGHT;
  end;

  {Refresca la pantalla}
  PaintBox1.Refresh;
end;

procedure TForm1.RadioButton4Click(Sender: TObject);
begin
  {Indica la alineación vertical seleccionada}
  VertAlignmentValue := 0;
  case TRadioButton(Sender).Tag of
    1: VertAlignmentValue := TA_TOP;
    2: VertAlignmentValue := TA_BASELINE;
    3: VertAlignmentValue := TA_BOTTOM;
  end;

  {Refresca la pantalla}
  PaintBox1.Refresh;
end;

```

Figura 8-17:  
Propiedades  
de alineación,  
color y  
espaciado  
extra del texto

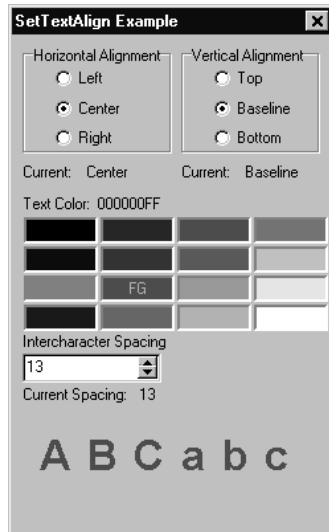


Tabla 8-43: Valores de opciones de SetTextAlign

Valor	Descripción
TA_BASELINE	El punto de comienzo está sobre la línea base del texto.
TA_BOTTOM	El punto de comienzo está sobre el borde inferior del rectángulo límite del texto.
TA_TOP	El punto de comienzo está sobre el borde superior del rectángulo límite del texto.
TA_CENTER	El punto de comienzo es el centro horizontal del rectángulo límite del texto.
TA_LEFT	El punto de comienzo está a la izquierda del rectángulo límite del texto.
TA_RIGHT	El punto de comienzo está a la derecha del rectángulo límite del texto.
TA_RTLREADING	Sólo Windows 95/98: Indica que el orden de lectura del texto es de derecha a izquierda. Este valor sólo tiene sentido cuando la fuente seleccionada es hebrea o árabe.
TA_NOUPDATECP	La posición actual no se actualiza después de dibujar texto.
TA_UPDATECP	La posición actual se actualiza después de dibujar texto.
VTA_BASELINE	Sólo para fuentes de línea base vertical: El punto de comienzo está sobre la línea base del texto.
VTA_CENTER	Solo para fuentes de línea base vertical: El punto de comienzo es el centro vertical del rectángulo límite del texto.

**SetTextCharacterExtra****Windows.Pas****Sintaxis**

```

SetTextCharacterExtra(
    DC: HDC;                {manejador de contexto de dispositivo}
    CharExtra: Integer      {cantidad de espacio extra entre caracteres}
): Integer;                {devuelve el espaciado entre caracteres anterior extra}

```

**Descripción**

Esta función selecciona la cantidad de espacio extra, en unidades lógicas, que se añade entre caracteres cuando se dibuja una línea de texto sobre el contexto de dispositivo especificado.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo espaciado extra entre caracteres será establecido.

*CharExtra*: Especifica la cantidad de espacio a añadir entre caracteres, en unidades lógicas. Si al modo de mapeado del contexto de dispositivo no se le asigna *MM\_TEXT*, este valor será convertido al modo de mapeado actual y redondeado a la cantidad de píxeles más cercana.

**Valor que devuelve**

Si la función tiene éxito, devuelve el valor anterior del espaciado extra; en caso contrario, devuelve \$80000000.

**Véase además**

*DrawText*, *DrawTextEx*, *GetTextCharacterExtra*, *TextOut*

**Ejemplo**

Vea el Listado 8-16 bajo *SetTextAlign*.

**SetTextColor****Windows.Pas****Sintaxis**

```

SetColor(
    DC: HDC;                {manejador de contexto de dispositivo}
    Color: COLORREF         {nuevo especificador de color de texto de 32 bits}
): COLORREF;               {devuelve el especificador de color de texto anterior}

```

**Descripción**

Esta función selecciona el color a utilizar cuando se dibuje texto sobre el contexto de dispositivo identificado por el parámetro *DC*.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo color de texto será establecido.

*Color*: Especificador de color de 32 bits que define el nuevo color para dibujar el texto. El color que realmente será usado es el más cercano al especificado en la paleta activa del contexto de dispositivo especificado.

**Valor que devuelve**

Si la función tiene éxito, devuelve el especificador del color de texto anterior; en caso contrario, devuelve *CLR\_INVALID*. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*DrawText*, *DrawTextEx*, *GetTextColor*, *RealizePalette*, *RGB*, *SetBkColor*, *SetBkMode*, *TabbedTextOut*, *TextOut*

**Ejemplo**

Vea el Listado 8-16 bajo *SetTextAlign*.

**SetTextJustification****Windows.Pas****Sintaxis**

```
SetTextJustification(
    DC: HDC;                {manejador de contexto de dispositivo}
    BreakExtra: Integer;     {espacio extra total}
    BreakCount: Integer;     {cantidad de caracteres de ruptura}
): Integer;                 {devuelve cero o un valor distinto de cero}
```

**Descripción**

Esta función especifica la cantidad de espacio extra, en unidades lógicas, que debe ser añadido a cada carácter de ruptura cuando se dibujan cadenas de caracteres en el contexto de dispositivo especificado. La mayoría de las fuentes utilizan como carácter de ruptura el espacio (' '), pero algunas fuentes no latinas pueden definir un carácter diferente. Utilice la función *GetTextMetrics* para recuperar cualquier carácter de ruptura específico definido para la fuente. La función *GetTextExtentPoint32* puede ser usada para recuperar el ancho del texto de salida, de manera que el espaciado extra apropiado pueda ser determinado. La función *TextOut* distribuye el espaciado extra proporcionalmente entre los caracteres de ruptura de una línea de texto de salida.

**Parámetros**

*DC*: Manejador del contexto de dispositivo cuyo espaciado extra para justificación de de texto será seleccionado.

*BreakExtra*: Especifica el total de espacio extra que será añadido a las líneas de salida de texto, en unidades lógicas. Si el modo de mapeado del contexto de dispositivo actual no es *MM\_TEXT*, este valor será convertido al modo de mapeado actual y redondeado a la cantidad de píxeles más cercana.

*BreakCount*: Especifica la cantidad total de caracteres de ruptura en la cadena que será justificada.

#### Valor que devuelve

Si la función tiene éxito, devuelve un valor distinto de cero; en caso contrario, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*DrawText*, *DrawTextEx*, *GetTextExtentExPoint*, *GetTextExtentPoint32*, *GetTextMetrics*, *TextOut*

#### Ejemplo

Vea el Listado 8-14 bajo *GetTextExtentExPoint*.

### TabbedTextOut

### Windows.Pas

#### Sintaxis

```
TabbedTextOut(
    hDC: HDC;                {manejador de contexto de dispositivo}
    X: Integer;               {origen horizontal del texto}
    Y: Integer;               {origen vertical del texto}
    lpString: PChar;          {cadena a dibujar sobre el contexto de dispositivo}
    nCount: Integer;          {número de caracteres en la cadena}
    nTabPositions: Integer;    {número de entradas en el array de tabulaciones}
    lpnTabStopPositions: Pointer {puntero a un array de posiciones de tabulación}
    nTabOrigin: Integer        {origen horizontal de las tabulaciones}
): Longint;                  {devuelve las dimensiones de la cadena}
```

#### Descripción

Esta función muestra la cadena de caracteres apuntada por *lpString* sobre el contexto de dispositivo identificado por el parámetro *hDC*, expandiendo cualquier carácter de tabulación en la cadena a la parada de tabulación indicada en el *array* de enteros al que apunta el parámetro *lpnTabStopPositions*. Las tabulaciones son expandidas a los valores en este *array* cuando son encontradas, con el primer carácter de tabulación en la cadena expandido a la posición indicada por la primera entrada del array, el segundo carácter de tabulación en la cadena expandido a la posición indicada por la segunda entrada en el *array*, y así sucesivamente. La fuente actualmente seleccionada en el contexto de dispositivo es utilizada para dibujar el texto. La función no actualizará la

posición actual, a menos que la función *SetTextAlign* haya sido llamada antes con la opción *TA\_UPDATECP* especificada.

### Parámetros

*hDC*: Manejador del contexto de dispositivo sobre el cual el texto es dibujado.

*X*: La coordenada horizontal de la línea de texto de salida, en unidades lógicas. Si la posición actual del contexto de dispositivo va a ser actualizada, este parámetro será ignorado en las llamadas subsiguientes.

*Y*: La coordenada vertical de la línea de texto de salida, en unidades lógicas. Si la posición actual del contexto de dispositivo va a ser actualizada, este parámetro será ignorado en las llamadas subsiguientes.

*lpString*: Puntero a la cadena de caracteres (que puede contener caracteres de tabulación) que será dibujada sobre el contexto de dispositivo especificado. No tiene necesariamente que ser una cadena de caracteres terminada en nulo, ya que el parámetro *nCount* indica la longitud.

*nCount*: Especifica la cantidad de caracteres en la cadena a la que apunta el parámetro *lpString*.

*nTabPositions*: Especifica la cantidad de entradas en el *array* de posiciones de tabulación, al que apunta el parámetro *lpnTabStopPositions*.

*lpnTabStopPositions*: Puntero a un *array* de enteros. Cada entrada del *array* indica una posición de tabulación, en unidades de dispositivo. Las posiciones de tabulación tienen que ser colocadas en orden creciente, con la más pequeña como primera entrada del *array*. Si a este parámetro se le asigna **nil** y el parámetro *nTabPositions* es puesto a cero, los caracteres de tabulación son expandidos a ocho veces el ancho promedio del carácter de la fuente seleccionada. Bajo Windows 95/98, una posición de tabulación negativa indica una tabulación alineada a la derecha.

*nTabOrigin*: Especifica la coordenada horizontal desde la cual comenzar a expandir las tabulaciones, en unidades lógicas.

### Valor que devuelve

Si la función tiene éxito, devuelve las dimensiones de la cadena de salida, en unidades lógicas, con la altura de la cadena en la palabra más significativa del valor devuelto y el ancho en la palabra menos significativa. Si la función falla, devuelve cero.

### Véase además

*DrawText*, *DrawTextEx*, *GetTabbedTextExtent*, *GrayString*, *SetTextAlign*, *TextOut*

### Ejemplo

#### Listado 8-17: Mostrando texto en forma de tabla

```
{iOJO! Delphi importa incorrectamente estas funciones. Tenemos que importarl as
manualmente para obtener toda su funcionalidad}
```

```
function TabbedTextOut(hDC: HDC; X, Y: Integer; lpString: PChar;
    nCount, nTabPositions: Integer;
    lpnTabStopPositions: Pointer; nTabOrigin: Integer): Longint; stdcall;
```

```
function GetTabbedTextExtent(hDC: HDC; lpString: PChar;
    nCount, nTabPositions: Integer;
    lpnTabStopPositions: Pointer): DWORD; stdcall;
```

```
var
    Form1: TForm1;
```

#### implementation

```
{$R *.DFM}
```

```
{Reimportación de funciones}
```

```
function GetTabbedTextExtent; external user32 name 'GetTabbedTextExtentA';
```

```
function TabbedTextOut; external user32 name 'TabbedTextOutA';
```

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
```

```
const
```

```
    {Define algunos arrays de cadenas estáticos}
```

```
    NameCol: array[0..8] of string = ('Name', 'John', 'David', 'Larry', 'Phil',
    'Kenneth', 'Rod', 'Ovais', 'Mike');
```

```
    IDCol: array[0..8] of string = ('ID Number', '100', '101', '102', '103',
    '104', '105', '106', '107');
```

```
    ScoreCol: array[0..8] of string = ('Score', '9,000,000', '8,345,678',
    '7,325,876', '8,324,689', '5,234,761',
    '5,243,864', '8,358,534', '6,538,324');
```

```
var
```

```
    TabStops: array[0..2] of Integer; // almacena las posiciones de tabulación
```

```
    FontSize: TSize; // almacena el tamaño de la fuente
```

```
    Count: Integer; // variable de control de bucle
```

```
begin
```

```
    {Define nuestras posiciones de tabulación}
```

```
    TabStops[0] := 10;
```

```
    TabStops[1] := PaintBox1.Width div 2;
```

```
    TabStops[2] := -PaintBox1.Width; // una tabulación alineada a la derecha
```

```
    {Recupera la altura de una cadena}
```

```
    GetTextExtentPoint32(PaintBox1.Canvas.Handle, 'ABC', Length('ABC'), FontSize);
```

```
with PaintBox1.Canvas do
```

```
begin
```

```
    {Borra la última imagen}
```

```
    Brush.Color := clWhite;
```

```
    FillRect(ClipRect);
```

```
    {Muestra el array de cadenas, usando las posiciones de tabulación para
    formatear las cadenas como una tabla}
```

```
for Count := 0 to 8 do
```

```
    TabbedTextOut(Handle, 0, FontSize.cy * Count,
```

```
        PChar(NameCol[Count] + #9 + IDCol[Count] + #9 + ScoreCol[Count]),
```

```
        Length(NameCol[Count] + #9 + IDCol[Count] + #9 + ScoreCol[Count]),
```

```

3, @TabStops, 0);

end;

{Recupera la longitud de una cadena que contiene las tabulaciones, en píxeles.
Este valor tiene que ser igual al ancho del paintbox.}
Label13.Caption := IntToStr(LoWord(GetTabbedTextExtent(PaintBox1.Canvas.Handle,
PChar(NameCol[0] + #9 + IDCol[0] + #9 + ScoreCol[0]),
Length(NameCol[0] + #9 + IDCol[0] + #9 + ScoreCol[0]),
3, @TabStops)));

end;

```

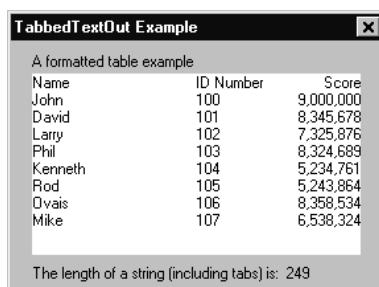


Figura 8-18:  
Salida de texto  
con  
tabulaciones

## TextOut

## Windows.Pas

### Sintaxis

```

TextOut(
  DC: HDC;           {manejador de contexto de dispositivo}
  X: Integer;         {origen horizontal del texto}
  Y: Integer;         {origen vertical del texto}
  Str: PChar;         {cadena que será dibujada en el contexto de dispositivo}
  Count: Integer;     {cantidad de caracteres en la cadena}
); BOOL;             {devuelve TRUE o FALSE}

```

### Descripción

Esta función muestra la cadena de texto especificada sobre el contexto de dispositivo identificado por el parámetro *DC*. La fuente actualmente seleccionada en el contexto de dispositivo es utilizada para dibujar el texto. La función no actualizará la posición actual, a menos que la función *SetTextAlign* haya sido llamada antes con la opción *TA\_UPDATECP* especificada.

### Parámetros

*DC*: Manejador del contexto de dispositivo sobre el cual el texto es dibujado.

*X*: La coordenada horizontal de la línea de texto de salida, en unidades lógicas. Si la posición actual del contexto de dispositivo va a ser actualizada, este parámetro será ignorado en las llamadas subsiguientes.

*Y*: La coordenada vertical de la línea de texto de salida, en unidades lógicas. Si la posición actual del contexto de dispositivo va a ser actualizada, este parámetro será ignorado en las llamadas subsiguientes.

*Str*: Puntero a la cadena que será dibujada sobre el contexto de dispositivo especificado. No tiene necesariamente que ser una cadena de caracteres terminada en nulo, ya que el parámetro *Count* especifica su longitud.

*Count*: Especifica la cantidad de caracteres que hay en la cadena a la que apunta el parámetro *Str*.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*DrawText*, *DrawTextEx*, *GetTextAlign*, *SetTextAlign*, *TabbedTextOut*

#### Ejemplo

Vea el Listado 8-2 bajo *CreateFont*, y otros ejemplos a lo largo de este capítulo.

## Capítulo 9

# Funciones de recursos

Los recursos ofrecen un mecanismo mediante el cual los desarrolladores pueden almacenar datos extra necesarios para la aplicación dentro de la propia aplicación. Esto elimina muchos ficheros externos que de otro modo serían necesarios, permitiendo una distribución más sencilla de los productos, a la vez que protege elementos como los gráficos de ser estropeados por los usuarios. La encapsulación que hace Delphi de ciertos aspectos de la programación de Windows hace innecesario el uso de ciertos tipos de recursos. Por ejemplo, el desarrollador no necesita crear recursos de menú o de cuadros de diálogo, ya que esta funcionalidad ha sido fuertemente simplificada por Delphi. Sin embargo, los desarrolladores de Delphi pueden aprovechar ciertos tipos de recursos para obtener beneficios como los mencionados anteriormente. Este capítulo cubre las funciones de recursos del API de Windows que son utilizadas más comúnmente en las aplicaciones de Delphi.

Tenga en cuenta que algunos ejemplos de este capítulo tendrán listados adicionales para los ficheros de descripción de recursos.

## Creando recursos

Para usar recursos, el desarrollador tiene primero que crear un *fichero de descripción de recursos (resource script file)*. Estos ficheros son algo más que simples ficheros de texto con extensión .RC. Un fichero de descripción de recursos puede ser creado manualmente usando el Bloc de Notas. Sin embargo, Borland ofrece un producto llamado Resource Workshop<sup>1</sup> que permite simplificar enormemente la creación de ficheros de recursos. Este producto incluye editores para todos los tipos estándar de recursos, tales como mapas de bits, y permite crear ficheros de recursos compilados tanto para aplicaciones de 16 bits como para aplicaciones de 32 bits. La descripción de esta herramienta de creación de recursos va más allá del alcance de este libro.

1 El Resource Workshop está incluido en el CD de Delphi.

Una vez que un fichero de descripción de recursos ha sido creado, debe ser compilado a un formato binario que es utilizado por el enlazador de Delphi para incrustar los recursos en un fichero ejecutable. El *Resource Workshop* puede compilar ficheros de recursos desde su entorno integrado. Para aquellos que no posean este producto, Delphi incluye un compilador de recursos de línea de comandos DOS, llamado BRCC32.EXE, ubicado en directorio BIN. Para compilar un fichero de descripción de recursos, pase el nombre del fichero como argumento a BRCC32 en la línea de comandos (por ejemplo, 'BRCC32 <ficheroDescripciónRecursos>.RC'). Esto creará un fichero con el mismo nombre del fichero fuente, pero con extensión .RES. Este fichero contiene los recursos binarios compilados que serán enlazados con la aplicación.



**Nota:** Es muy importante que el nombre del fichero de recursos compilados sea diferente del nombre del proyecto al cual el recurso será enlazado. Delphi crea automáticamente un fichero de recursos para el proyecto con el nombre <nombreProyecto>.RES. Este fichero es mantenido por Delphi y puede ser recreado cuando se añade o elimina algo del proyecto, como controles visuales. Por esta razón, ningún recurso debe ser añadido al fichero de recursos mantenido por Delphi. Modificar este fichero en cualquier sentido puede tener drásticos efectos colaterales, tales como mensajes de error de recursos duplicados cuando el proyecto es compilado, u otros incluso peores.

## Nombres y tipos de recursos

Cuando se define un nombre o tipo de recurso en un fichero de descripción de recursos, es una práctica común utilizar una cadena de caracteres para identificar al recurso o tipo de recurso de una manera lógica (por ejemplo, *Splash BITMAP "Splash.bmp"*). Alternativamente, el recurso (o tipo de recurso, en el caso de recursos definidos por el usuario) puede ser identificado mediante un valor entero. Este último enfoque tiene la ventaja de producir una pequeña ganancia de rendimiento al buscar y cargar recursos. Al utilizar cualquier función que requiera un nombre de recurso o de tipo de recurso, el desarrollador puede utilizar el identificador entero como parámetro, convirtiéndolo en un valor adecuado mediante la función *MakeIntResource*. El identificador de recurso puede ser especificado también en forma de una cadena literal en la que el primer carácter es la almohadilla ('#'), seguida del identificador entero del recurso (por ejemplo, '#256').

## Enlazando los recursos

Una vez que los recursos han sido compilados, pueden enlazarse con la aplicación de Delphi especificando la directiva de compilación *\$R* en algún lugar del código. La directiva puede colocarse en cualquier lugar del código del proyecto. Su sintaxis es

'{\$R <ficheroRecursosCompilados>.RES}'. Cuando el proyecto es construido, los recursos compilados son incrustados y enlazados en el ejecutable resultante.

Los recursos no tienen por qué ser enlazados únicamente a ficheros ejecutables. También pueden ser colocados en DLLs. Esta es una práctica muy común, especialmente útil cuando se trata de desarrollar aplicaciones para el mercado internacional. Por ejemplo, la librería *MoreIcons.DLL*, que forma parte de Windows, contiene solamente recursos y ningún código ejecutable. Utilizar los recursos de esta manera permite actualizar las aplicaciones simplemente compilando una nueva versión de la DLL de recursos, la cual tiene un tamaño mucho más pequeño que un ejecutable y debe, por lo tanto, ser más fácil de distribuir.

## Usando recursos

Para hacer uso de un recurso, una aplicación tiene primero que recuperar un manejador mediante las funciones *FindResource* o *FindResourceEx*. Entonces se debe cargar el recurso en memoria pasándole este manejador a la función *LoadResource*, que a su vez devuelve un manejador del bloque de memoria global que contiene el recurso. Por último, la aplicación debe recuperar un puntero al recurso, pasándole el manejador de memoria global obtenido a la función *LockResource*. Observe que bajo el API Win32 ya no es necesario desbloquear un recurso “bloqueado” o liberar un recurso cargado mediante la función *LoadResource*.

La manipulación de recursos mediante las funciones *FindResource*, *LoadResource* y *LockResource* es necesaria solamente si la aplicación necesita acceso directo a los datos del recurso binario. Para todos los tipos de recursos estándar se ofrecen funciones que cargan un recurso y devuelven un manejador del objeto de recurso correspondiente en su formato nativo. La siguiente tabla lista las funciones más comúnmente utilizadas para cargar recursos estándar.

**Tabla 9-I: Funciones de carga de recursos estándar**

Function	Descripción
LoadBitmap	Carga un recurso de mapa de bits, devolviendo un manejador de mapa de bits.
LoadCursor	Carga un recurso de cursor, devolviendo un manejador de cursor.
LoadIcon	Carga un recurso de icono, devolviendo un manejador de icono.
LoadImage	Carga un recurso de icono, cursor, o mapa de bits, devolviendo un manejador de icono, cursor, o mapa de bits.
LoadString	Carga un recurso de cadena desde una tabla de cadenas, almacenándola en un <i>buffer</i> de cadena de caracteres terminada en nulo.

## Recursos definidos por el usuario

En adición a los tipos de recursos estándar, los desarrolladores pueden crear sus propios tipos de recursos. La forma más simple de utilizar un tipo de recurso definido por el usuario es con una declaración de recurso *simple* (de una sola línea). El formato de estas declaraciones es muy similar al de otras declaraciones de recursos de una sola línea, que consisten en el nombre del recurso, seguido del tipo de recurso, seguido del nombre del fichero externo que contiene los datos del recurso. El compilador de recursos creará un fichero de recursos como si se tratase de un recurso de tipo estándar. Sin embargo, los recursos definidos por el usuario pueden ser accedidos únicamente utilizando las funciones *FindResource*, *LoadResource* y *LockResource*. Por ejemplo, para incluir un fichero de audio digitalizado (WAV) en un recurso, el fichero de descripción de recursos tiene que contener la línea *DRocks WAV "Delphi.wav"*, donde *DRocks* es el nombre del recurso, *WAV* es el tipo de recurso definido por el usuario y *"Delphi.wav"* es el fichero externo de audio. Para cargar el recurso, simplemente se debe utilizar *DRocks* como nombre del recurso y *WAV* como el tipo de recurso en las llamadas a las funciones *FindResource*, *LoadResource* y *LockResource*. El siguiente ejemplo muestra la implementación de esta técnica.

### Listado 9-1: Fichero de recursos definidos por el usuario de una sola línea

```
DRocks WAV "Delphi.wav"
```

### Listado 9-2: Cargando un recurso definido por el usuario

#### implementation

```
uses MMSystem;
```

```
{ $R *.DFM }
```

```
{ $R WAVES.RES }
```

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
```

```
    TheResource: HRSRC;    // manejador del recurso
```

```
    TheWave: Pointer;      // puntero a los datos del recurso
```

```
begin
```

```
    {Encuentra y carga el recurso definido por el usuario (un fichero WAV)}
```

```
    TheResource := LoadResource(hInstance, FindResource(hInstance,  
                                                         'DRocks', 'WAV'));
```

```
    {Recupera un puntero a los datos del fichero de audio}
```

```
    TheWave := LockResource(TheResource);
```

```
    {Ejecuta el fichero de audio}
```

```
    sndPlaySound(TheWave, SND_MEMORY or SND_SYNC);
```

```
end;
```

Los recursos definidos por el usuario de múltiples líneas (*compuestos*) son también muy fáciles de crear y usar. En este caso, se debe utilizar el tipo de recurso estándar

*RCDATA* define un recurso multilinea de usuario, y los valores que se coloquen entre los bloques *BEGIN* y *END* de tales definiciones de recursos pueden contener los datos que se deseen. Repetimos que no existe otra forma de cargar estos recursos que la secuencia *FindResource*, *LoadResource* y *LockResource*. Al igual que en el caso de los recursos definidos por el usuario de una sola línea, la aplicación debe tener un conocimiento profundo del formato de los datos del recurso. El siguiente ejemplo muestra cómo usar recursos definidos por el usuario de múltiples líneas.

### Listado 9-3: Fichero de recursos definidos por el usuario de múltiples líneas

```
CustomInfo RCDATA
BEGIN
    0x0000001B,
    "John Ayres\0",
    "Lead Author\0",
END
```

### Listado 9-4: Usando recursos definidos por el usuario de múltiples líneas

#### implementation

```
{ $R *.DFM }
{ $R CustomInfo.RES }

procedure TForm1.Button1Click(Sender: TObject);
var
    TheResource: HRSRC;    // manejador del recurso
    TheInfo: Pointer;      // puntero al recurso
begin
    {Encuentra y carga el recurso que deberá ser un recurso multilinea, definido por
    el usuario}
    TheResource := LoadResource(hInstance, FindResource(hInstance, 'CustomInfo',
                                                         RT_RCDATA));

    {Recupera un puntero al recurso}
    TheInfo := LockResource(TheResource);

    {Extrae el primer elemento del recurso definido por el usuario (una edad)}
    Label1.Caption := IntToStr(LongInt(TheInfo^));

    {Extrae el segundo elemento del recurso definido por el usuario (un nombre en
    forma de cadena de caracteres terminada en nulo)}
    TheInfo := Pointer(LongInt(TheInfo) + SizeOf(LongInt));
    Label2.Caption := PChar(TheInfo);

    {Extrae el tercer elemento del recurso definido por el usuario (un título en
    forma de una cadena de caracteres terminada en nulo). Observe que añadimos uno a
    la longitud de la cadena anterior para posicionar el puntero más allá del
    terminador nulo de la primera cadena.}
    TheInfo := Pointer(LongInt(TheInfo) + StrLen(TheInfo) + 1);
    Label3.Caption := PChar(TheInfo);
end;
```

Figura 9-1:  
El recurso  
definido por el  
usuario



## Funciones de recursos

En este capítulo se describen las siguientes funciones de recursos:

**Tabla 9-2: Funciones de recursos**

Función	Descripción
EnumResourceLanguages	Enumera todos los idiomas en los cuales un recurso específico está disponible.
EnumResourceNames	Enumera todos los recursos accesibles de un tipo de recursos específico.
EnumResourceTypes	Enumera todos los tipos de recursos accesibles en un módulo específico.
FindResource	Encuentra un recurso específico.
FindResourceEx	Encuentra un recurso específico para un idioma específico.
LoadResource	Carga un recurso.
LoadString	Carga una cadena desde una tabla de cadenas.
LockResource	Obtiene un puntero a los datos de un recurso.
MakeIntResource	Convierte un identificador de recurso entero en un valor adecuado para ser usado en las funciones de recursos.
SizeofResource	Recupera el tamaño de un recurso.

### **EnumResourceLanguages**

### **Windows.Pas**

#### *Sintaxis*

```
EnumResourceLanguages(
    hModule: HMODULE;           {manejador de instancia del módulo}
    lpType: Pchar;              {tipo del recurso}
    lpName: Pchar;              {nombre del recurso}
    lpEnumFunc: ENUMRESLANGPROC; {puntero a función de respuesta}
    lParam: LongInt;            {valor definido por la aplicación}
); BOOL;                       {devuelve TRUE o FALSE}
```

**Descripción**

Esta función enumera todos los recursos que se corresponden con el tipo y nombre especificado, para todos los idiomas en los cuales el recurso está disponible, pasando el idioma de cada recurso hallado a la función de respuesta a la que apunta el parámetro *lpEnumFunc*. La enumeración continuará hasta que la función de respuesta devuelva el valor FALSE o todos los recursos hayan sido enumerados.

**Parámetros**

*hModule*: Especifica el manejador de instancia del módulo que contiene los recursos que serán enumerados. Si a este parámetro se le asigna cero, la función enumera los recursos asociados con el proceso que hace la llamada.

*lpType*: Puntero a una cadena de caracteres terminada en nulo que contiene el tipo de recurso a enumerar. Este parámetro puede apuntar a una cadena que contiene un tipo de recurso definido por el usuario, o puede contener un valor de la Tabla 9-3. Para especificar un tipo de recurso mediante su identificador entero, la palabra más significativa del puntero deberá ser cero, y la menos significativa deberá contener el valor del identificador del tipo de recurso. Ese valor puede construirse usando la función *MakeIntResource*. Alternativamente, un tipo de recurso puede expresarse en forma de una cadena de caracteres en la que el primer carácter es la almohadilla ('#'), seguida por el identificador entero del tipo de recurso (por ejemplo, "#256").

*lpName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso a enumerar. Para especificar un recurso mediante su identificador entero, la palabra más significativa del puntero deberá ser cero, y la menos significativa deberá contener el valor del identificador del recurso. Ese valor puede construirse usando la función *MakeIntResource*. Alternativamente, un recurso puede expresarse en forma de una cadena de caracteres en la que el primer carácter es la almohadilla ('#'), seguida por el identificador entero del recurso (por ejemplo, "#256").

*lpEnumFunc*: Puntero a la función de respuesta, que será llamada una vez por cada recurso que se corresponda con el nombre y tipo especificados.

*lParam*: Un valor de 32 bits definido por la aplicación que es pasado a la función de respuesta.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Sintaxis de la función de respuesta**

```
EnumResLangProc(
    hModule: HMODULE;           {manejador de instancia del módulo}
    lpszType: PChar;            {tipo del recurso}
```

<code>lp.szName: PChar;</code>	{nombre del recurso}
<code>wIDLanguage: LANGID;</code>	{identificador de idioma del recurso}
<code>lParam: LongInt</code>	{valor definido por la aplicación}
<code>):BOOL;</code>	{devuelve TRUE o FALSE}

### Descripción

Esta función de respuesta recibe un identificador de idioma por cada idioma en el que un nombre y tipo de recurso específicos están disponibles. Esta función de respuesta puede ejecutar cualquier tarea que se estime necesaria.

### Parámetros

*hModule*: Manejador de instancia del módulo que contiene los recursos que serán enumerados, pasado a través del parámetro *hModule* de la función *EnumResourceLanguages*.

*lp.szType*: Puntero a una cadena de caracteres terminada en nulo que contiene el tipo de recurso a enumerar, pasado a través del parámetro *lpType* de la función *EnumResourceLanguages*. Si el tipo de recurso es especificado en forma de un identificador entero, la palabra más significativa de este puntero tendrá valor cero, y la palabra menos significativa contendrá el identificador entero del tipo de recurso.

*lp.szName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso a enumerar, pasado a través del parámetro *lpName* de la función *EnumResourceLanguages*. Si el tipo de recurso es especificado en forma de un identificador entero, la palabra más significativa de este puntero tendrá valor cero y la palabra menos significativa contendrá el identificador entero del recurso.

*wIDLanguage*: El identificador de idioma de un idioma para el que está disponible el recurso indicado.

*lParam*: El valor de 32 bits definido por la aplicación pasado a través del parámetro *lParam* de la función *EnumResourceLanguages*. Este valor se suministra para uso específico de la aplicación dentro de la función de respuesta.

### Valor que devuelve

Para indicar que continúe la enumeración, la función de respuesta debe devolver TRUE; en caso contrario, debe devolver FALSE.

### Véase además

*EnumResourceNames*, *EnumResourceTypes*

### Ejemplo

#### Listado 9-5: Fichero de recursos multilingüe

```
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
Hello BITMAP "hienglish.bmp"
```

```
LANGUAGE LANG_SPANISH, SUBLANG_SPANISH_MEXICAN
Hello BITMAP "hispansh.bmp"
```

#### Listado 9-6: Enumerando todos los idiomas en que un recurso específico está disponible

```
var
    Form1: TForm1;
    NumResources: Integer;    // almacena la cantidad de recursos enumerados

{prototipo de la función de respuesta de enumeración}
function ResLangCallBack(hModule: HMODULE; lpszType: PChar; lpszName: PChar;
                        wIDLanguage: LANGID; lParam: Integer): BOOL; stdcall;

implementation

{$R *.DFM}
{$R reslangs.RES}

function ResLangCallBack(hModule: HMODULE; lpszType: PChar; lpszName: PChar;
                        wIDLanguage: LANGID; lParam: Integer): BOOL;
var
    LanName: array[0..255] of Char;    // almacena un nombre de idioma
    ResHandle: THandle;                // almacena un manejador de recurso
    TheBitmap: TBitmap;               // un objeto de mapa de bits
    ResPtr: Pointer;                  // puntero a los datos de un recurso
    WinBitmap: HBitmap;               // manejador de un mapa de bits de Windows
begin
    {Recupera el nombre del idioma en el que el recurso está disponible}
    VerLanguageName(wIDLanguage, LanName, 255);

    {Recupera un manejador de los datos del recurso}
    ResHandle := LoadResource(hInstance, FindResourceEx(hInstance, lpszType,
                                                         lpszName, wIDLanguage));

    {Recupera un puntero a los datos del recurso}
    ResPtr := LockResource(ResHandle);

    {Si el recurso fue localizado...}
    if ResHandle <> 0 then
    begin
        {Sabemos que se trata de imágenes de 256 colores, creamos un DIB a partir
        de la información del recurso de mapa de bits}
        WinBitmap := CreateDIBitmap(Form1.Canvas.Handle,
                                    Windows.TBitmapInfo(ResPtr^).bmiHeader, CBM_INIT,
                                    Pointer(Integer(ResPtr) + SizeOf(Windows.TBitmapInfo) +
                                    (SizeOf(TRGBQuad)*255)),
                                    Windows.TBitmapInfo(ResPtr^), DIB_RGB_COLORS);

        {Crea un objeto de mapa de bits de Delphi}
        TheBitmap := TBitmap.Create;

        {Asigna el DIB al objeto de mapa de bits de Delphi}
        TheBitmap.Handle := WinBitmap;
```

```

    {Muestra el idioma en que el recurso está disponible y
    el recurso de mapa de bits como tal}
    SetBkMode(Form1.Canvas.Handle, TRANSPARENT);
    Form1.Canvas.TextOut(16, 128 + (64*NumResources) + (3*NumResources), LanName);
    Form1.Canvas.Draw(168, 128 + (64*NumResources) + (3*NumResources), TheBitmap);

    {Libera el mapa de bits cuando ya no es necesario}
    TheBitmap.Free;

    {Incrementa el número de recursos cargados}
    Inc(NumResources);
end;

{Indica que la enumeración debe continuar}
ResLangCallBack := TRUE;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Inicializa el contador de recursos cargados}
    NumResources := 0;

    {Enumera todos los idiomas en los que el recurso de mapa de bits llamado
    'Hello' está disponible}
    EnumResourceLanguages(hInstance, RT_BITMAP, 'Hello', @ResLangCallBack, 0);
end;

```

**Figura 9-2:**  
Todos los  
idiomas en los  
que un  
recurso de  
mapa de bits  
específico está  
disponible



**Tabla 9-3: Valores del parámetro lpType de EnumResourceLanguages y lpszType de EnumResLangProc.**

Valor	Descripción
RT_ACCELERATOR	Un recurso de tabla de aceleradores.
RT_ANICURSOR	Un recurso de cursor animado.
RT_ANIICON	Un recurso de icono animado.
RT_BITMAP	Un recurso de mapa de bits.

RT_CURSOR	Un recurso de cursor dependiente del hardware.
RT_DIALOG	Un recurso de cuadro de diálogo.
RT_FONT	Un recurso de fuente.
RT_FONTPATH	Un recurso de directorio de fuentes.
RT_GROUP_CURSOR	Un recurso de cursor independiente del hardware.
RT_GROUP_ICON	Un recurso de icono independiente del hardware.
RT_ICON	Un recurso de icono dependiente del hardware.
RT_MENU	Un recurso de menú.
RT_MESSAGEBOX	Un recurso de tabla de mensajes.
RT_RCDATA	Un recurso definido por la aplicación.
RT_STRING	Un recurso de tabla de cadenas.
RT_VERSION	Un recurso de información de versión.

**EnumResourceNames****Windows.Pas****Sintaxis**

```
EnumResourceNames(
    hModule: HMODULE;           {manejador de instancia del módulo}
    lpType: PChar;              {tipo de recurso}
    lpEnumFunc: ENUMRESNAMEPROC; {puntero a función de respuesta}
    lParam: Longint;            {valor definido por la aplicación}
): BOOL;                       {devuelve TRUE o FALSE}
```

**Descripción**

Esta función enumera todos los recursos del tipo especificado por el parámetro *lpType*, pasando el nombre de cada recurso encontrado a la función de respuesta a la que apunta el parámetro *lpEnumFunc*. Los recursos continuarán siendo enumerados hasta que la función de respuesta devuelva FALSE o todos los recursos hayan sido enumerados.

**Parámetros**

*hModule*: Manejador de instancia del módulo que contiene los recursos a ser enumerados. Si a este parámetro se le asigna cero, la función enumera los recursos asociados al proceso que hace la llamada.

*lpType*: Puntero a una cadena de caracteres terminada en nulo que contiene el tipo de recurso a enumerar. Este parámetro puede apuntar a una cadena que contiene un tipo de recurso definido por el usuario, o puede contener un valor de la Tabla 9-4. Para especificar un tipo de recurso mediante su identificador entero, la palabra más significativa del puntero deberá ser cero, y la menos significativa deberá contener el valor del identificador del tipo de recurso. Ese valor puede construirse usando la función *MakeIntResource*. Alternativamente, un tipo de recurso puede expresarse en forma de una cadena de caracteres en la que el primer carácter es la almohadilla ('#'), seguida por el identificador entero del tipo de recurso (por ejemplo, "#256").

*lpEnumFunc*: Puntero a una función de respuesta que será llamada una vez por cada recurso que se corresponda con el tipo especificado.

*lParam*: Un valor de 32 bits definido por la aplicación que es pasado a la función de respuesta.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Sintaxis de la función de respuesta

```
EnumResNameProc(
    hModule: HMODULE;           {manejador de instancia del módulo}
    lpszType: PChar;            {tipo de recurso}
    lpszName:PChar;             {nombre del recurso}
    lParam: LongInt             {valor definido por la aplicación}
):BOOL;                        {devuelve TRUE o FALSE}
```

#### Descripción

Esta función recupera el nombre de cada recurso del tipo especificado por el parámetro *lpszType* perteneciente al módulo identificado por el parámetro *hModule*. Esta función de respuesta puede ejecutar cualquier tarea que se estime necesaria.

#### Parámetros

*hModule*: Manejador de instancia del módulo que contiene los recursos que serán enumerados, pasado a través del parámetro *hModule* de la función *EnumResourceNames*.

*lpszType*: Puntero a una cadena de caracteres terminada en nulo que contiene el tipo de recurso a enumerar, pasado a través del parámetro *lpType* de la función *EnumResourceNames*. Si el tipo de recurso es especificado en forma de un identificador entero, la palabra más significativa de este puntero tendrá valor cero, y la palabra menos significativa contendrá el identificador entero del tipo de recurso.

*lpszName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso a enumerar, del tipo especificado a través del parámetro *lpszType*. Si el tipo de recurso es especificado en forma de un identificador entero, la palabra más significativa de este puntero tendrá valor cero, y la palabra menos significativa contendrá el identificador entero del recurso.

*lParam*: El valor de 32 bits definido por la aplicación pasado a través del parámetro *lParam* de la función *EnumResourceNames*. Este valor se suministra para uso específico de la aplicación dentro de la función de respuesta.

**Valor que devuelve**

Para indicar que continúe la enumeración, la función de respuesta debe devolver TRUE; en caso contrario, debe devolver FALSE.

**Véase además**

*EnumResourceLanguages*, *EnumResourceTypes*

**Ejemplo**

Consulte el Listado 9-7 bajo *EnumResourceTypes*.

**Tabla 9-4: Valores del parámetro lpType de EnumResourceNames y lpszType de EnumResNameProc.**

Valor	Descripción
RT_ACCELERATOR	Un recurso de tabla de aceleradores.
RT_ANICURSOR	Un recurso de cursor animado.
RT_ANIICON	Un recurso de icono animado.
RT_BITMAP	Un recurso de mapa de bits.
RT_CURSOR	Un recurso de cursor dependiente del hardware.
RT_DIALOG	Un recurso de cuadro de diálogo.
RT_FONT	Un recurso de fuente.
RT_FONTDIR	Un recurso de directorio de fuentes.
RT_GROUP_CURSOR	Un recurso de cursor independiente del hardware.
RT_GROUP_ICON	Un recurso de icono independiente del hardware.
RT_ICON	Un recurso de icono dependiente del hardware.
RT_MENU	Un recurso de menú.
RT_MESSAGE TABLE	Un recurso de tabla de mensajes.
RT_RCDATA	Un recurso definido por la aplicación.
RT_STRING	Un recurso de tabla de cadenas.
RT_VERSION	Un recurso de información de versión.

**EnumResourceTypes****Windows.Pas****Sintaxis**

```
EnumResourceTypes(
    hModule: HMODULE;           {manejador de instancia del módulo}
    lpEnumFunc: ENUMRESTYPEPROC; {puntero a función de respuesta}
    lParam: LongInt             {valor definido por la aplicación}
): BOOL;                       {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recorre todos los recursos pertenecientes al módulo identificado por el parámetro *hModule*. Para cada tipo de recurso diferente que se encuentre en el módulo, se pasará una cadena de caracteres correspondiente al tipo de recurso a la función de respuesta a la que apunta el parámetro *lpEnumFunc*. Los recursos continuarán siendo enumerados hasta que la función de respuesta devuelva FALSE o todos los recursos hayan sido enumerados.

**Parámetros**

*hModule*: Manejador de instancia del módulo que contiene los recursos que serán enumerados. Si a este parámetro se le asigna cero, la función enumera los recursos asociados al proceso que hace la llamada.

*lpEnumFunc*: Puntero a la función de respuesta que será llamada una vez por cada tipo de recurso encontrado.

*lParam*: Un valor de 32 bits definido por la aplicación, que es pasado a la función de respuesta.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Sintaxis de la función de respuesta**

```
EnumResTypeProc(
    hModule: HMODULE;      {manejador de instancia del módulo}
    lpszType: PChar;        {tipo de recurso}
    lParam: LongInt         {valor definido por la aplicación}
):BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recibe una cadena de caracteres que contiene el nombre del tipo de recurso para cada tipo de recurso diferente hallado en el módulo identificado por el parámetro *hModule*. Esta función de respuesta puede ejecutar cualquier tarea que se estime necesaria.

**Parámetros**

*hModule*: Manejador de instancia del módulo que contiene los recursos que serán enumerados, pasado al parámetro *hModule* de la función *EnumResourceTypes*.

*lpszType*: Puntero a una cadena de caracteres terminada en nulo, que contiene un tipo de recurso encontrado en el módulo especificado. Esta cadena puede contener un nombre de tipo de recurso definido por el usuario o un valor de la Tabla 9-5. Si el tipo de recurso es especificado en forma de un identificador entero, la palabra más

significativa de este puntero tendrá valor cero, y la palabra menos significativa contendrá el identificador entero del tipo de recurso.

*lParam*: El valor de 32 bits definido por la aplicación pasado a través del parámetro *lParam* de la función *EnumResourceTypes*. Este valor está previsto para su uso específico dentro de la función de respuesta.

#### Valor que devuelve

Para indicar que continúe la enumeración, la función de respuesta debe devolver TRUE; en caso contrario, debe devolver FALSE.

#### Véase además

*EnumResourceLanguages*, *EnumResourceNames*

#### Ejemplo

##### Listado 9-7: Enumerando los tipos y nombres de todos los recursos incluidos

```
{prototipo de la función de respuesta de EnumResourceTypes}
function EnumResTypeProc(hModule: HMODULE; lpszType: PChar;
                        lParam: LongInt): BOOL; stdcall;

{prototipo de la función de respuesta de EnumResourceNames}
function EnumResNameProc(hModule: HMODULE; lpszType: PChar; lpszName: PChar;
                        lParam: LongInt): BOOL; stdcall;

implementation

{$R *.DFM}

function EnumResTypeProc(hModule: HMODULE; lpszType: PChar;
                        lParam: LongInt): BOOL;
var
    TypeName: string;           // almacena la descripción del tipo
    ParentNode: TTreeNode;       // almacena un nodo de árbol
begin
    {Indica que queremos que la enumeración continúe}
    Result := TRUE;

    {Si el tipo de recurso es estándar, el puntero contendrá un identificador de
    recurso que se corresponde con uno de los tipos de recursos estándar. Debemos
    convertirlo a LongInt para que la sentencia case funcione correctamente. Sin
    embargo, para cualquier tipo de recurso no estándar (como los tipos de recurso
    definidos por el usuario), asumimos que el parámetro lpszType está apuntando a
    una cadena}
    case LongInt(lpszType) of
        LongInt(RT_CURSOR)       : TypeName := 'Cursor';
        LongInt(RT_BITMAP)       : TypeName := 'Bitmap';
        LongInt(RT_ICON)         : TypeName := 'Icon';
        LongInt(RT_MENU)         : TypeName := 'Menu';
        LongInt(RT_DIALOG)       : TypeName := 'Dialog';
        LongInt(RT_STRING)       : TypeName := 'String';
```

```

LongInt(RT_FONTDIR)      : Typename := 'Font Directory';
LongInt(RT_FONT)         : Typename := 'Font';
LongInt(RT_ACCELERATOR)  : Typename := 'Accelerator';
LongInt(RT_RCDATA)       : Typename := 'RC Data';
LongInt(RT_MESSAGETABLE) : Typename := 'Message Table';
LongInt(RT_GROUP_CURSOR) : Typename := 'Cursor Group';
LongInt(RT_GROUP_ICON)   : Typename := 'Icon Group';
LongInt(RT_VERSION)      : Typename := 'Version Table';
LongInt(RT_DLGINCLUDE)   : Typename := 'Dialog Include';
LongInt(RT_PLUGPLAY)     : Typename := 'Plug and Play';
LongInt(RT_VXD)          : Typename := 'VXD';
LongInt(RT_ANICURSORS)   : Typename := 'Animated Cursor';
LongInt(RT_ANIICON)      : Typename := 'Animated Icon';
else
  TypeName := lpszType;
end;

{Añade un nodo al árbol, especificando el tipo de recurso enumerado como texto}
ParentNode := Form1.TreeView1.Items.Add(nil, TypeName);

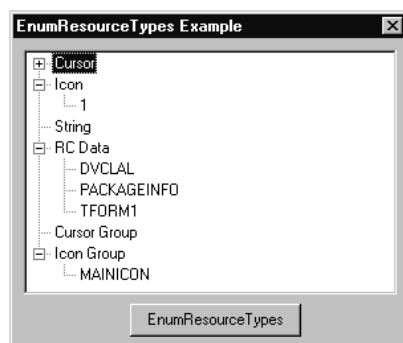
{Ahora se enumeran todos los recursos de este tipo contenidos en el ejecutable}
EnumResourceNames(hInstance, lpszType, @EnumResNameProc, LongInt(ParentNode));
end;

function EnumResNameProc(hModule: HMODULE; lpszType: PChar; lpszName: PChar;
  lParam: LongInt): BOOL;
begin
  {Si la palabra más significativa del puntero al nombre del recurso es cero, el
  puntero actualmente contiene un identificador de recurso. En ese caso se añade
  el identificador del recurso al árbol}
  if Hi(LongInt(lpszName)) = 0 then
    TTreeNode(lParam).Owner.AddChild(TTreeNode(lParam), IntToStr(
      LongInt(lpszName)))
  else
    {En caso contrario, el puntero contiene un nombre de recurso, que es añadido
    al árbol}
    TTreeNode(lParam).Owner.AddChild(TTreeNode(lParam), lpszName);

    {Indica que queremos continuar la enumeración}
    Result := TRUE;
  end;
end;

```

*Figura 9-3:  
Recursos  
estándar  
incluidos en  
un proyecto  
Delphi*



```

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Enumera todos los tipos de recursos hallados en el ejecutable}
    EnumResourceTypes(hInstance, @EnumResTypeProc, 0);
end;

```

**Tabla 9-5: Valores del parámetro `lpszType` de `EnumResTypeProc`**

Valor	Descripción
RT_ACCELERATOR	Un recurso de tabla de aceleradores.
RT_ANICURSOR	Un recurso de cursor animado.
RT_ANIICON	Un recurso de icono animado.
RT_BITMAP	Un recurso de mapa de bits.
RT_CURSOR	Un recurso de cursor dependiente del hardware.
RT_DIALOG	Un recurso de cuadro de diálogo.
RT_FONT	Un recurso de fuente.
RT_FONTDIR	Un recurso de directorio de fuentes.
RT_GROUP_CURSOR	Un recurso de cursor independiente del hardware.
RT_GROUP_ICON	Un recurso de icono independiente del hardware.
RT_ICON	Un recurso de icono dependiente del hardware.
RT_MENU	Un recurso de menú.
RT_MESSAGE TABLE	Un recurso de tabla de mensajes.
RT_RCDATA	Un recurso definido por la aplicación.
RT_STRING	Un recurso de tabla de cadenas.
RT_VERSION	Un recurso de información de versión.

## FindResource

## Windows.Pas

### Sintaxis

```

FindResource(
    hModule: HMODULE;    {manejador de instancia del módulo}
    lpName: PChar;        {nombre del recurso}
    lpType: PChar;        {tipo del recurso}
): HRSRC;                {devuelve un manejador del bloque de información del
                           recurso}

```

### Descripción

Esta función localiza dentro del módulo especificado por el parámetro *hModule* el recurso con el nombre y el tipo indicados en los parámetros *lpName* y *lpType*. El manejador devuelto por esta función es usado por la función *LoadResource*.

**Parámetros**

*hModule*: Manejador de instancia del módulo que contiene el recurso a buscar. Si a este parámetro se le asigna cero, la función busca los recursos asociados al proceso que hace la llamada.

*lpName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso a buscar. Para especificar un identificador entero de recurso, la palabra más significativa de este puntero deberá ser cero, y la menos significativa deberá contener el identificador entero del nombre de recurso. Esto puede lograrse utilizando la función *MakeIntResource*. Alternativamente, un identificador entero puede expresarse en forma de una cadena de caracteres en la que el primer carácter es la almohadilla ('#'), seguida por el identificador entero del tipo de recurso (por ejemplo, "#256").

*lpType*: Puntero a una cadena de caracteres terminada en nulo que contiene el tipo del recurso a encontrar. Este parámetro puede apuntar a una cadena que contiene un tipo de recurso definido por el usuario, o puede contener un valor de la Tabla 9-6. Para especificar un tipo de recurso mediante su identificador entero, la palabra más significativa del puntero deberá ser cero, y la menos significativa deberá contener el valor del identificador del tipo de recurso. Ese valor puede construirse usando la función *MakeIntResource*. Alternativamente, un identificador entero puede expresarse en forma de una cadena de caracteres en la que el primer carácter es la almohadilla ('#'), seguida por el identificador entero del tipo de recurso (por ejemplo, "#256").

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de un bloque de información del recurso localizado. Este manejador puede ser pasado a la función *LoadResource* para cargar el recurso en memoria. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*FindResourceEx*, *FormatMessage*, *LoadBitmap*, *LoadCursor*, *LoadIcon*, *LoadResource*, *LoadString*, *LockResource*, *SizeofResource*

**Ejemplo**

Consulte el Listado 9-2 y el Listado 9-4 bajo los ejemplos de recursos definidos por el usuario en la introducción del capítulo.

**Tabla 9-6: Valores del parámetro *lpType* de *FindResource***

Valor	Descripción
RT_ACCELERATOR	Un recurso de tabla de aceleradores.
RT_ANICURSOR	Un recurso de cursor animado.
RT_ANIICON	Un recurso de icono animado.
RT_BITMAP	Un recurso de mapa de bits.

RT_CURSOR	Un recurso de cursor dependiente del hardware.
RT_DIALOG	Un recurso de cuadro de diálogo.
RT_FONT	Un recurso de fuente.
RT_FONTDIR	Un recurso de directorio de fuentes.
RT_GROUP_CURSOR	Un recurso de cursor independiente del hardware.
RT_GROUP_ICON	Un recurso de icono independiente del hardware.
RT_ICON	Un recurso de icono dependiente del hardware.
RT_MENU	Un recurso de menú.
RT_MESSAGE	Un recurso de tabla de mensajes.
RT_RCDATA	Un recurso definido por la aplicación.
RT_STRING	Un recurso de tabla de cadenas.
RT_VERSION	Un recurso de información de versión.

**FindResourceEx****Windows.Pas****Sintaxis**

```
FindResourceEx(
    hModule: HMODULE;      {manejador de instancia del módulo}
    lpType: PChar;         {tipo del recurso}
    lpName: PChar;         {nombre del recurso}
    wLanguage: Word        {idioma del recurso}
): HRSRC;                 {devuelve un manejador del bloque de información del
                           recurso}
```

**Descripción**

Esta función localiza dentro del módulo especificado por el parámetro *hModule* el recurso con el nombre, tipo e idioma indicados. El manejador devuelto por esta función es usado por la función *LoadResource*.

**Parámetros**

*hModule*: Manejador de instancia del módulo que contiene el recurso a buscar. Si a este parámetro se le asigna cero, la función busca los recursos asociados al proceso que hace la llamada.

*lpType*: Puntero a una cadena de caracteres terminada en nulo que contiene el tipo del recurso a encontrar. Este parámetro puede apuntar a una cadena que contiene un tipo de recurso definido por el usuario, o puede contener un valor de la Tabla 9-7. Para especificar un tipo de recurso mediante su identificador entero, la palabra más significativa del puntero deberá ser cero, y la menos significativa deberá contener el valor del identificador del tipo de recurso. Ese valor puede construirse usando la función *MakeIntResource*. Alternativamente, un identificador entero puede expresarse

en forma de una cadena de caracteres en la que el primer carácter es la almohadilla ('#'), seguida por el identificador entero del tipo de recurso (por ejemplo, "#256").

*lpName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre del recurso a buscar. Para especificar un identificador entero de recurso, la palabra más significativa de este puntero deberá ser cero, y la menos significativa deberá contener el identificador entero del nombre de recurso. Esto puede lograrse utilizando la función *MakeIntResource*. Alternativamente, un identificador entero puede expresarse en forma de una cadena de caracteres en la que el primer carácter es la almohadilla ('#'), seguida por el identificador entero del tipo de recurso (por ejemplo, "#256").

*wLanguage*: El identificador de idioma del recurso a encontrar. Si a este parámetro se le asigna *MakeLangId(LANG\_NEUTRAL, SUBLANG\_NEUTRAL)*, la función utiliza el idioma actual del hilo que hace la llamada.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador de un bloque de información del recurso localizado. Este manejador puede ser pasado a la función *LoadResource* para cargar el recurso en memoria. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*FindResource*, *FormatMessage*, *LoadBitmap*, *LoadCursor*, *LoadIcon*, *LoadResource*, *LoadString*, *LockResource*, *SizeofResource*

#### Ejemplo

Consulte el Listado 9-6, correspondiente a la función *EnumResourceLanguages*.

**Tabla 9-7: Valores del parámetro *lpType* de *FindResourceEx***

Valor	Descripción
RT_ACCELERATOR	Un recurso de tabla de aceleradores.
RT_ANICURSOR	Un recurso de cursor animado.
RT_ANIICON	Un recurso de icono animado.
RT_BITMAP	Un recurso de mapa de bits.
RT_CURSOR	Un recurso de cursor dependiente del hardware.
RT_DIALOG	Un recurso de cuadro de diálogo.
RT_FONT	Un recurso de fuente.
RT_FONTDIR	Un recurso de directorio de fuentes.
RT_GROUP_CURSOR	Un recurso de cursor independiente del hardware.
RT_GROUP_ICON	Un recurso de icono independiente del hardware.
RT_ICON	Un recurso de icono dependiente del hardware.
RT_MENU	Un recurso de menú.
RT_MESSAGETABLE	Un recurso de tabla de mensajes.

RT_RCDATA	Un recurso definido por la aplicación.
RT_STRING	Un recurso de tabla de cadenas.
RT_VERSION	Un recurso de información de versión.

**LoadResource****Windows.Pas****Sintaxis**

```
LoadResource(
    hModule: HINST;           {manejador de instancia del módulo}
    hResInfo: HRSRC           {manejador del recurso}
): HGLOBAL;                  {devuelve un manejador de bloque de memoria global}
```

**Descripción**

Esta función carga el recurso identificado por el parámetro *hResInfo* en memoria. Un puntero a los datos del recurso puede ser recuperado pasando el valor devuelto por esta función a la función *LockResource*. Bajo Windows 95/98 y NT, cualquier recurso cargado con la función *LoadResource* será liberado automáticamente por el sistema.

**Parámetros**

*hModule*: Manejador de instancia del módulo que contiene el recurso que será cargado. Si a este parámetro se le asigna cero, la función carga el recurso desde el proceso que hace la llamada.

*hResInfo*: Manejador del bloque de información del recurso. Este manejador es recuperado por las funciones *FindResource* y *FindResourceEx*.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del bloque de memoria global que identifica los datos del recurso. Este manejador puede ser utilizado en una llamada posterior a la función *LockResource* para recuperar un puntero a los datos del recurso. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*FindResource*, *FindResourceEx*, *LoadLibrary*, *LockResource*, *SizeofResource*

**Ejemplo**

Consulte el Listado 9-11 bajo *MakeIntResource*.

**LoadString****Windows.Pas****Sintaxis**

```
LoadString(
```

<code>hInstance: HINST;</code>	<code>{ manejador de instancia del módulo }</code>
<code>uID: UINT;</code>	<code>{ identificador de la cadena a cargar }</code>
<code>lpBuffer: PChar;</code>	<code>{ puntero al <i>buffer</i> que recibe la cadena }</code>
<code>nBufferMax: Integer</code>	<code>{ tamaño del <i>buffer</i> en caracteres }</code>
<code>): Integer;</code>	<code>{ devuelve la cantidad de caracteres copiados al <i>buffer</i> }</code>

**Descripción**

Esta función recupera la cadena asociada con el identificador entero especificado por los recursos asociados al módulo identificado por el parámetro *hInstance*. La función añade un terminador nulo a la cadena cuando ésta es colocada en el *buffer*.

**Parámetros**

*hInstance*: Manejador de instancia del módulo que contiene la cadena que será cargada. Si a este parámetro se le asigna cero, la función carga la cadena desde los recursos del proceso que hace la llamada.

*uID*: El identificador entero de la cadena a cargar.

*lpBuffer*: Puntero al *buffer* de cadena de caracteres terminada en nulo, reservado por la aplicación, que recibirá el recurso de cadena.

*nBufferMax*: Especifica el tamaño del *buffer* al que apunta el parámetro *lpBuffer*, en caracteres. Si el recurso de cadena es más largo que este valor, es truncado.

**Valor que devuelve**

Si la función tiene éxito, devuelve la cantidad de caracteres copiados al *buffer*, sin incluir el terminador nulo. Si la función falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*FindResource*, *FindResourceEx*, *LoadBitmap*, *LoadCursor*, *LoadIcon*, *LoadResource*, *LockResource*.

**Ejemplo****Listado 9-8: Fichero de recursos que contiene una tabla de cadenas**

```

STRINGTABLE
{
    1, "An error has occurred."
    2, "A really, really bad error has occurred."
    3, "A massive, critical error has occurred."
}

```

**Listado 9-9: Utilizando una tabla de cadenas para almacenar mensajes de error****implementation**

```

{$R *.DFM}
{$R Strings.RES}

```

```

procedure TForm1.Button1Click(Sender: TObject);
var
    Buffer: array[0..255] of char; // la cadena de la tabla de cadenas
begin
    {Simula una condición de error. Hay 3 cadenas en la tabla, numeradas del 1 al 3.
    El código de error se corresponde con el número de la cadena}
    SetLastError(Random(3) + 1);

    {Inicializa el buffer para almacenar la cadena}
    FillMemory(@Buffer, 256, 0);

    {Recupera el código de error asignado anteriormente, y carga la cadena
    correspondiente de la tabla de cadenas}
    LoadString(hInstance, GetLastError, Buffer, 255);

    {Muestra la cadena (un mensaje de error simulado)}
    Panel1.Caption := Buffer
end;

```

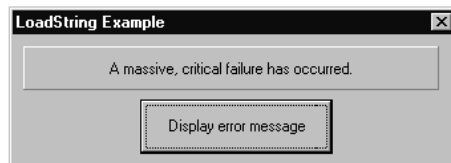


Figura 9-4:  
La cadena  
cargada

## LockResource Windows.Pas

### Sintaxis

```

LockResource(
    hResData: HGLOBAL    {manejador del recurso a bloquear}
): Pointer;              {devuelve un puntero a los datos del recurso}

```

### Descripción

Esta función bloquea un recurso en memoria, devolviendo un puntero al primer byte de los datos del recurso. Esta función es llamada pasándole como parámetro el valor devuelto por la función *LoadResource*. No es necesario desbloquear los recursos bloqueados mediante esta función.

### Parámetros

*hResData*: Manejador del bloque de memoria global que contiene los datos del recurso. Este manejador es devuelto por la función *LoadResource*.

**Valor que devuelve**

Si la función tiene éxito, devuelve un puntero al primer byte de los datos del recurso. Si la función falla, devuelve **nil**. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*FindResource*, *FindResourceEx*, *LoadResource*

**Ejemplo**

Consulte el Listado 9-11, correspondiente a la función *MakeIntResource*.

**MakeIntResource****Windows.Pas****Sintaxis**

```
MakeIntResource(
    wInteger: Integer           {identificador de recurso}
); PChar;                     {devuelve el identificador del recurso}
```

**Descripción**

Esta función convierte un identificador entero de recurso en un valor que puede ser utilizado en los parámetros de nombres y tipos de recursos de las funciones del API.

**Parámetros**

*wInteger*: El identificador entero del recurso.

**Valor que devuelve**

Esta función devuelve un valor de puntero correspondiente a un identificador entero de recurso. La palabra más significativa del puntero es cero, y la menos significativa contendrá el identificador entero del recurso. Este valor puede ser utilizado posteriormente en lugar de un tipo o nombre de recurso en cualquier función de recursos del API. Esta función no indica ninguna condición de error.

**Véase además**

*EnumResourceLanguages*, *EnumResourceNames*, *FindResource*, *FindResourceEx*

**Ejemplo****Listado 9-10: El fichero de recursos de ejemplo de MakeIntResource**

```
#define CoolPic 256

CoolPic BITMAP "image6.bmp"
```

**Listado 9-II: Cargando un mapa de bits de 256 colores desde un recurso****implementation**

```
{ $R *.DFM }
{ $R intresource.RES }
```

{Esta función carga un recurso de mapa de bits de 256 colores mientras mantiene su paleta original. Esta funcionalidad está ahora presente en los métodos LoadFromResourceID y LoadFromResourceName de la clase TBitmap. Este ejemplo debe ejecutarse bajo una configuración de vídeo de 256 colores para apreciar completamente sus efectos}

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
```

```
  ResHandle: HGLOBAL;           // manejador del recurso cargado
  ResPointer: Pointer;          // puntero a los datos del recurso
  FileHeader: TBitmapFileHeader; // almacena información sobre el encabezamiento
                                // del fichero de un mapa de bits
  ResSize: LongInt;             // almacena el tamaño del recurso
  InfoStream: TMemoryStream;    // usado para cargar el recurso en un mapa de
                                // bits
```

```
begin
```

```
  {Inicializa el encabezamiento del tipo del fichero con los caracteres 'BM'.
   Este es el único campo del encabezamiento del mapa de bits que requiere
   inicialización}
  FileHeader.bfType := $4D42;
```

```
  {Encuentra y carga el recurso del mapa de bits, especificando el nombre como una
   cadena identificada por el identificador entero del recurso}
```

```
  ResHandle := LoadResource(hInstance, FindResource(hInstance, '#256',
                                                    RT_BITMAP));
```

```
  {Recupera el tamaño del recurso, esta vez, encuentra el recurso especificando el
   nombre como un identificador entero usando MakeIntResource, este otro método es
   perfectamente correcto}
```

```
  ResSize := SizeofResource(hInstance, FindResource(hInstance,
                                                    MakeIntResource(256), RT_BITMAP));
```

```
  {Recupera un puntero a los datos del recurso}
  ResPointer := LockResource(ResHandle);
```

```
  {Crea un flujo en memoria e inicializa su tamaño para almacenar el recurso
   entero y la información del encabezamiento del fichero del mapa de bits}
```

```
  InfoStream := TMemoryStream.Create;
  InfoStream.SetSize(ResSize + SizeOf(TBitmapFileHeader));
```

```
  {Escribe el encabezamiento del fichero del mapa de bits al flujo en memoria}
  InfoStream.Write(FileHeader, SizeOf(TBitmapFileHeader));
```

```
  {Escribe los datos del recurso al flujo en memoria}
  InfoStream.Write(ResPointer^, ResSize);
```

```
  {Reposiciona el puntero del flujo al principio}
  InfoStream.Seek(0, 0);
```

```

{Crea la imagen del mapa de bits}
Image1.Picture.Bitmap := TBitmap.Create;

{Carga la información del recurso en el nuevo mapa de bits, mostrándolo}
Image1.Picture.Bitmap.LoadFromStream(InfoStream);

{No necesitamos ya el flujo en memoria, lo liberamos}
InfoStream.Free;
end;

```



Figura 9-5:  
El recurso de  
mapa de bits  
cargado

## SizeofResource

## Windows.Pas

### Sintaxis

```

SizeofResource(
    hModule: HINST;           {manejador de instancia del módulo}
    hResInfo: HRSRC           {manejador del recurso}
): DWORD;                   {devuelve el tamaño del recurso}

```

### Descripción

Esta función recupera el tamaño del recurso especificado por el parámetro *hResInfo*, en bytes. Esta función utiliza como parámetro el manejador devuelto por las funciones *FindResource* o *FindResourceEx*. El valor devuelto por esta función puede ser mayor que el tamaño real del recurso debido a la alineación de la memoria.

### Parámetros

*hModule*: Manejador de instancia del módulo que contiene el recurso. Si a este parámetro se le asigna cero, la función utiliza los recursos del proceso que hace la llamada.

*hResInfo*: Manejador del recurso cuyo tamaño se desea recuperar. Este manejador es devuelto por las funciones *FindResource* y *FindResourceEx*.

**Valor que devuelve**

Si la función tiene éxito, devuelve el tamaño del recurso, en bytes. Si falla, devuelve cero. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*FindResource, FindResourceEx, LoadResource, LockResource*

**Ejemplo**

Consulte el Listado 9-11 bajo *MakeIntResource*.



## Capítulo 10

# Funciones de movimiento de ventanas

El API Win32 incluye un grupo de funciones que permiten al desarrollador controlar desde su aplicación el tamaño y la posición de las ventanas. Si bien una ventana puede ser fácilmente movida o redimensionada utilizando las propiedades *Left*, *Top*, *Width*, o *Height* de la clase *TForm*, estas funciones proporcionan al desarrollador un control que se extiende mucho más allá de lo que Delphi ofrece.

## Orden Z

Muchas de las funciones de movimiento de ventanas tienen como objetivo modificar el *orden Z* de una ventana. El concepto de orden Z se refiere al orden en que unas ventanas solapan a otras. Está basado en el eje Z, que puede imaginarse como una línea que penetra la pantalla perpendicularmente. Las ventanas son colocadas de acuerdo a su posición en este eje Z. Aquellas ventanas que están más cerca de la cima del orden Z solapan y aparecen sobre las otras ventanas, mientras que las solapadas se dice que están más cerca del fondo del orden Z. La Figura 10-1 ilustra este concepto.

10

Capítulo

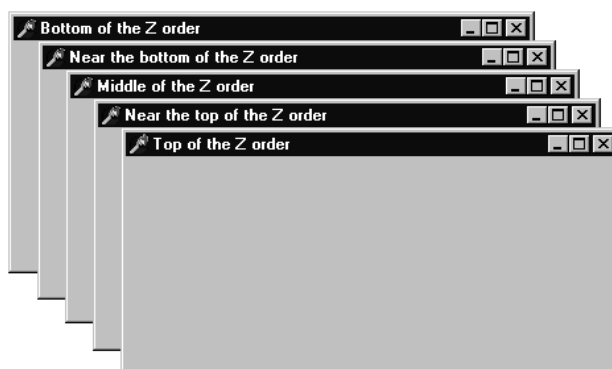


Figura 10-1:  
El orden Z de  
las ventanas

Windows mantiene el orden Z de todas las ventanas en una lista única. El orden Z de una ventana está determinado por el orden en que apareció en la pantalla, en relación con las demás ventanas. Cuando una ventana es creada, es colocada sobre las ventanas creadas previamente, de manera que la primera ventana creada está en el fondo del orden Z y la última en la cima. Sin embargo, la posición de las ventanas en la lista del orden Z depende de su tipo. Todas las ventanas pueden ser clasificadas como sigue:

- Ventanas *siempre visibles* (*topmost*): Una ventana siempre visible solapa a todas las demás que no tengan ese atributo. Esto se cumple incluso aunque la ventana no sea la ventana activa o de primer plano. Este tipo de ventana contiene el atributo de estilo extendido *WS\_EX\_TOPMOST* y aparece en el orden Z antes que todas las ventanas que no lo tengan.
- Ventanas de nivel superior (*top level*): Una ventana de nivel superior es cualquier ventana normal que no sea una ventana hija. Este tipo de ventana no contiene el atributo de estilo extendido *WS\_EX\_TOPMOST*, y siempre estará solapada, apareciendo siempre debajo de cualquier ventana visible.
- Ventanas hijas: Una ventana hija contiene el atributo de estilo *WS\_CHILD*. Las ventanas hijas tienen su propio orden Z dentro de una ventana madre, pero a nivel general reflejan la misma posición de orden Z que su ventana madre.

Cuando una ventana es activada, es colocada en la cima del orden Z de todas las ventanas del mismo tipo, trayendo consigo a todas sus ventanas hijas. Si una ventana es propietaria de otras ventanas, éstas últimas son colocadas encima de la ventana activada en el orden Z, de manera que siempre son mostradas sobre sus propietarias.

## Efectos especiales

Una utilización imaginativa de las funciones de movimiento de ventanas permite dar a las aplicaciones un toque profesional. Por ejemplo, si una aplicación ocupa un espacio relativamente pequeño de la pantalla, pero tiene barras de herramientas flotantes u otras ventanas emergentes que estén abiertas constantemente, el desarrollador puede usar las funciones de movimiento de ventanas para hacer que la ventana de la barra de herramientas se mueva junto con la ventana principal cuando el usuario arrastre ésta a una nueva posición. Este es un efecto muy elegante que requiere muy pocas líneas de código. El siguiente ejemplo muestra el uso de esta técnica.

### Listado 10-1: Moviendo la barra de herramientas junto con su ventana propietaria

```
unit WinMoveU;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
Unit2;
```

```

type
  TForm1 = class(TForm)
    procedure FormShow(Sender: TObject);
  private
    {Declaraciones privadas}
    {Debemos sobrescribir el mensaje WM_MOVE}
    procedure WMMove(var Msg: TWMMove); message WM_MOVE;
  public
    {Declaraciones públicas}
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

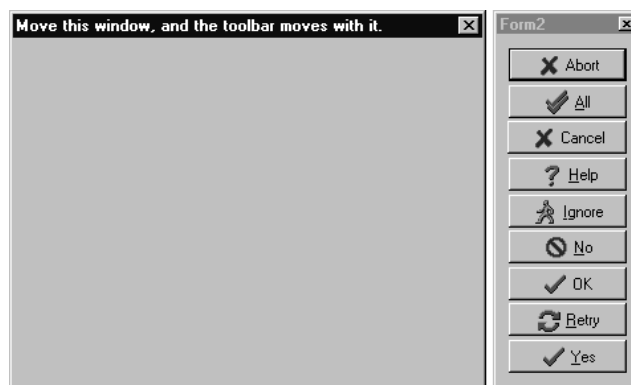
procedure TForm1.FormShow(Sender: TObject);
begin
  {Muestra la ventana de la barra de herramientas}
  Form2.Show;
end;

{Esto se ejecuta cada vez que la ventana principal se mueve}
procedure TForm1.WMMove(var Msg: TWMMove);
begin
  {Si la ventana de la barra de herramientas existe...}
  if Form2 <> nil then
    {...mueve la barra de herramientas junto con la ventana principal.}
    MoveWindow(Form2.Handle, Form1.Left + Form1.Width + 5, Form1.Top, Form2.Width,
      Form2.Height, TRUE);
end;

end.

```

Figura 10-2:  
La ventana  
principal y su  
barra de  
herramientas



## Funciones de movimiento de ventanas

En este capítulo se describen las siguientes funciones de movimiento de ventanas:

**Tabla 10-1: Funciones de movimiento de ventanas**

Función	Descripción
AdjustWindowRect	Calcula el tamaño de una ventana en base al tamaño deseado del área cliente.
AdjustWindowRectEx	Calcula el tamaño de una ventana con un estilo extendido en base al tamaño deseado del área cliente.
BeginDeferWindowPos	Comienza un proceso de movimiento de varias ventanas simultáneamente.
BringWindowToTop	Lleva la ventana especificada a la cima del orden Z.
CascadeWindows	Ordena las ventanas especificadas en cascada.
CloseWindow	Minimiza la ventana especificada.
DeferWindowPos	Define un nuevo tamaño y posición para la ventana especificada.
EndDeferWindowPos	Finaliza el proceso de mover varias ventanas simultáneamente.
GetWindowPlacement	Recupera el estado de presentación y la posición de la ventana especificada.
MoveWindow	Mueve una ventana.
OpenIcon	Restaura una ventana desde un estado minimizado.
SetWindowPlacement	Asigna el estado de presentación y la posición de la ventana especificada.
SetWindowPos	Cambia el tamaño, posición y orden Z de la ventana especificada.
ShowOwnedPopups	Cambia la visibilidad de todas las ventanas emergentes propiedad de la ventana especificada.
ShowWindow	Muestra una ventana.
ShowWindowAsync	Muestra una ventana e inmediatamente retorna a la función que hizo la llamada.
TileWindows	Ordena las ventanas especificadas en cascada.

### **AdjustWindowRect**

#### *Sintaxis*

```
AdjustWindowRect(
    var lpRect: TRect;
    dwStyle: DWORD;
    bMenu: BOOL
); BOOL;
```

### **Windows.Pas**

```
{puntero a registro de rectángulo cliente}
{opciones de estilo de ventana}
{opción de menú}
{devuelve TRUE o FALSE}
```

### Descripción

Calcula el tamaño del rectángulo de la ventana a partir del tamaño del rectángulo cliente especificado en *lpRect*. El rectángulo de la ventana incluirá el tamaño del borde, de la barra de título y la barra de menú. Este rectángulo puede ser usado con las funciones *CreateWindow* o *CreateWindowEx* para crear una ventana con el tamaño exacto que se desea para el área cliente. El resultado devuelto por esta función se expresa en términos de coordenadas de pantalla, e incluye las esquinas superior izquierda e inferior derecha. Sin embargo, la función *CreateWindow* necesita esos parámetros en términos de una coordenada superior izquierda y el ancho y la altura de la ventana. Por eso, el desarrollador debe restar la coordenada superior de la inferior para obtener el ancho apropiado, y restar la coordenada superior de la inferior para obtener la altura apropiada.

### Parámetros

*lpRect*: La dirección del registro de tipo *TRect* que contiene las coordenadas superior izquierda e inferior derecha del área cliente, relativas a la pantalla. Si la función tiene éxito, esta información será modificada para contener las coordenadas superior izquierda e inferior derecha del rectángulo de la ventana que contiene el área cliente especificada, también relativa a la pantalla.

*dwStyle*: Un valor de 32 bits que indica los estilos de ventana utilizados por la ventana específica.

*bMenu*: Si la ventana tiene un menú, este parámetro debe valer TRUE; en caso contrario, debe valer FALSE.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*AdjustWindowRectEx*, *CreateWindow*, *CreateWindowEx*

### Ejemplo

#### Listado 10-2: Creando una ventana con un área cliente de 300 x 300 píxeles

```
procedure TForm1.CreateParams(var Params: TCreateParams);
var
  TheRect: TRect; // almacena las coordenadas del rectángulo
begin
  {Rellena los parámetros estándar}
  inherited CreateParams(Params);

  {Nuestra ventana comenzará en las coordenadas 100,100 y su rectángulo cliente
  tendrá 300 píxeles de altura y 300 píxeles de ancho}
  TheRect.Left := 100;
```

```

TheRect.Top := 100;
TheRect.Right := 400;
TheRect.Bottom := 400;

{Ajusta las coordenadas del rectángulo a una ventana con un área cliente
de 300 por 300 píxeles}
AdjustWindowRect(TheRect, Params.Style, FALSE);

{El resultado de AdjustWindowRect está en términos de coordenadas exactas, pero
la función CreateWindowEx lo necesita en términos de coordenadas superior
izquierda, y medidas de ancho y altura}
Params.X := TheRect.Left;
Params.Y := TheRect.Top;
Params.Width := TheRect.Right - TheRect.Left;    // ancho de la ventana
Params.Height := TheRect.Bottom - TheRect.Top;    // altura de la ventana
end;

```

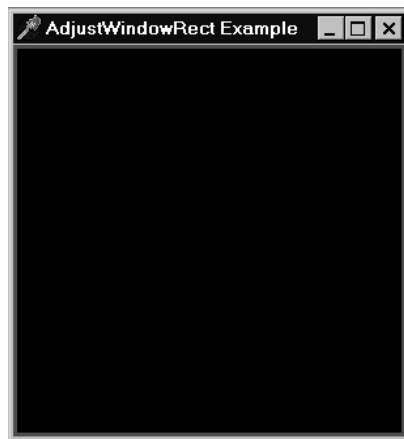


Figura 10-3:  
El resultado de  
la función  
AdjustWindow  
Rect sobre la  
ventana

## AdjustWindowRectEx Windows.Pas

### Sintaxis

```

AdjustWindowRectEx(
  var lpRect: TRect;           {puntero al registro de rectángulo cliente}
  dwStyle: DWORD;             {opciones de estilo de ventana}
  bMenu: BOOL                 {opción de menú}
  dwExStyle: DWORD            {opciones de estilo extendido}
): BOOL;                     {devuelve TRUE o FALSE}

```

### Descripción

Calcula el tamaño del rectángulo de la ventana a partir del tamaño del rectángulo cliente especificado en *lpRect*. El rectángulo de la ventana incluirá el tamaño del borde, de la barra de título y la barra de menú. Este rectángulo puede ser usado con las

funciones *CreateWindow* o *CreateWindowEx* para crear una ventana con el tamaño exacto que se desea para el área cliente. El resultado devuelto por esta función se expresa en términos de coordenadas de pantalla, e incluye las esquinas superior izquierda e inferior derecha. Sin embargo, la función *CreateWindow* necesita esos parámetros en términos de una coordenada superior izquierda y el ancho y la altura de la ventana. Por eso, el desarrollador debe restar la coordenada izquierda de la derecha para obtener el ancho apropiado, y restar la coordenada superior de la inferior para obtener la altura apropiada. Esta funcionalidad es equivalente a *AdjustWindowRect*.

### Parámetros

*lpRect*: La dirección del registro de tipo *TRect* que contiene las coordenadas superior izquierda e inferior derecha del área cliente, relativas a la pantalla. Si la función tiene éxito, esta información será modificada para contener las coordenadas superior izquierda e inferior derecha del rectángulo de la ventana que contiene el área cliente especificada, también relativa a la pantalla.

*dwStyle*: Un valor de 32 bits que indica los estilos de ventana utilizados por la ventana específica.

*bMenu*: Si la ventana tiene un menú, este parámetro debe valer TRUE; en caso contrario, debe valer FALSE.

*dwExStyle*: Un valor de 32 bits que indica los estilos de ventana extendidos utilizados por la ventana especificada.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*AdjustWindowRect*, *CreateWindow*, *CreateWindowEx*

### Ejemplo

#### Listado 10-3: Asignando a una ventana con estilos extendidos un área cliente de 300 x 300 píxeles

```
procedure TForm1.CreateParams(var Params: TCreateParams);
var
    TheRect: TRect; // almacena las coordenadas del rectángulo
begin
    {Rellena los parámetros estándar}
    inherited CreateParams(Params);

    {Nuestra ventana comenzará en las coordenadas 100,100 y el rectángulo cliente
    tendrá 300 píxeles de altura y 300 píxeles de ancho}
    TheRect.Left := 100;
    TheRect.Top := 100;
```

```

TheRect.Right := 400;
TheRect.Bottom := 400;

{Ajusta las coordenadas del rectángulo para lograr una ventana con un área
 cliente de 300 por 300 píxeles}
AdjustWindowRectEx(TheRect, Params.Style, FALSE, Params.ExStyle);

{El resultado de AdjustWindowRect está en términos de coordenadas exactas, pero
 la función CreateWindowEx lo necesita en términos de coordenadas superior
 izquierda y medidas del ancho y altura}
Params.X := TheRect.Left;
Params.Y := TheRect.Top;
Params.Width := TheRect.Right - TheRect.Left;    // ancho de la ventana
Params.Height := TheRect.Bottom - TheRect.Top;    // altura de la ventana
end;

```

**BeginDeferWindowPos****Windows.Pas****Sintaxis**

```

BeginDeferWindowPos(
  nNumWindows: Integer    {la cantidad de ventanas que serán movidas}
): HDWP;                  {devuelve un manejador de un registro de posición}

```

**Descripción**

Esta es la primera de una serie de funciones usadas para reposicionar y redimensionar múltiples ventanas simultáneamente, con un mínimo de refresco de pantalla. Esta función reserva memoria para un registro interno que almacenará la posición y el tamaño de las ventanas que serán modificadas. La función *DeferWindowPos* llenará este registro con información relativa al nuevo tamaño y posición para cada ventana. Finalmente, la función *EndDeferWindowPos* utiliza esa información para mover y redimensionar las ventanas simultáneamente. La pantalla no es actualizada hasta que la función *EndDeferWindowPos* no es llamada.

**Parámetros**

*nNumWindows*: Especifica la cantidad de ventanas sobre las que se almacenará información en un registro de posición de múltiples ventanas. La función *DeferWindowPos* puede incrementar el tamaño de este registro si es necesario, pero si no hay suficiente memoria para aumentar ese tamaño, toda la secuencia fallará.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador de un registro de posición de múltiples ventanas; en caso contrario, devuelve cero.

**Véase además**

*DeferWindowPos*, *EndDeferWindowPos*, *GetWindowPlacement*, *MoveWindow*, *SetWindowPlacement*, *SetWindowPos*, *ShowWindow*, *WM\_MOVE*, *WM\_SIZE*, *WM\_WINDOWPOSCHANGED*, *WM\_WINDOWPOSCHANGING*

*Ejemplo***Listado 10-4: Reposicionando múltiples ventanas**

```

procedure TForm1.FormShow(Sender: TObject);
begin
    {Muestra las otras dos ventanas}
    Form2.Show;
    Form3.Show;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    WindowPosInfo: HDWP; // almacena el registro interno de posición de ventanas
begin
    {Reserva memoria para mover tres ventanas}
    WindowPosInfo := BeginDeferWindowPos(3);

    {Prepara la primera ventana}
    WindowPosInfo:=DeferWindowPos(WindowPosInfo, Form1.Handle, HWND_NOTOPMOST,
                                   50, 50, 400, 100, SWP_SHOWWINDOW);

    {Prepara la segunda ventana}
    WindowPosInfo:=DeferWindowPos(WindowPosInfo, Form2.Handle, HWND_NOTOPMOST,
                                   50, 150, 400, 100, SWP_SHOWWINDOW);

    {Prepara la tercera ventana}
    WindowPosInfo:=DeferWindowPos(WindowPosInfo, Form3.Handle, HWND_NOTOPMOST,
                                   50, 250, 400, 100, SWP_SHOWWINDOW);

    {Completa la secuencia y reposiciona las ventanas}
    EndDeferWindowPos(WindowPosInfo);
end;

```

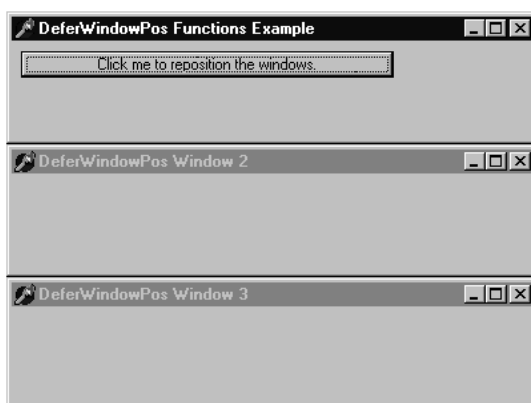


Figura 10-4:  
Las ventanas  
reposicionadas

**BringWindowToTop****Windows.Pas****Sintaxis**

```

BringWindowToTop(
    hWnd: HWND           {manejador de una ventana}
): BOOL;                {devuelve TRUE o FALSE}

```

**Descripción**

Esta función colocará la ventana especificada en la cima de su orden Z relativo, situándola encima de las otras ventanas con el mismo orden Z (por ejemplo, una ventana hija estará encima de otras ventanas hijas, una ventana de nivel superior estará encima de otras ventanas de nivel superior y una ventana siempre visible estará encima de otras ventanas siempre visibles). Si la ventana es de nivel superior o siempre visible, será activada, pero no será restaurada desde un estado minimizado. Esta función no puede utilizarse para convertir una ventana en una ventana siempre visible. Una aplicación debe llamar a *SetForegroundWindow* para convertirse en la aplicación de primer plano.

**Parámetros**

*hWnd*: Manejador de la ventana que será colocada en la cima del orden Z.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*EnableWindow*, *IsWindowVisible*, *SetActiveWindow*, *SetFocus*, *SetForegroundWindow*, *SetWindowPos*, *WM\_ENABLE*, *WM\_SETFOCUS*

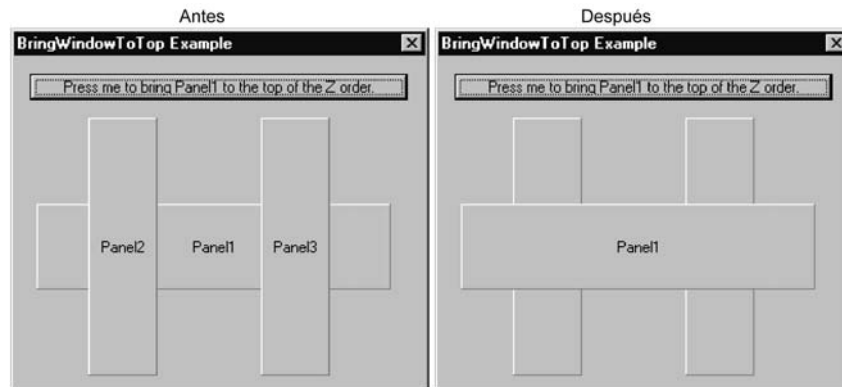
**Ejemplo****Listado 10-5: Modificando el orden Z de ventanas hijas**

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Coloca a Panel1 en la cima del orden Z de las ventanas hijas}
    BringWindowToTop(Panel1.Handle);
end;

```

Figura 10-5:  
El orden Z  
modificado



### CascadeWindows

### Windows.Pas

#### Sintaxis

```
CascadeWindows(
    hwndParent: HWND;    {manejador de la ventana madre}
    wHow: UINT;           {opciones de control}
    lpRect: PRect;        {área rectangular para ordenar las ventanas}
    cKids: UINT;          {la cantidad de ventanas a ordenar}
    lpKids: Pointer       {la dirección de un array de manejadores de ventana}
); WORD;                 {devuelve la cantidad de ventanas ordenadas}
```

#### Descripción

Esta función ordena en cascada las ventanas asociadas con los manejadores colocados en el array *lpKids* o las ventanas hijas de la ventana especificada.

#### Parámetros

*hwndParent*: Manejador de la ventana madre. Si este parámetro es cero, el escritorio es usado como ventana madre. Si la función va a ser utilizada para ordenar en cascada ventanas hijas MDI, a este parámetro se le debe asignar la propiedad *ClientHandle* del formulario particular.

*wHow*: Se le puede asignar *MDITILE\_SKIPDISABLED* para no colocar en cascada las ventanas hijas desactivadas. Si a este parámetro se le asigna cero, todas las ventanas hijas serán ordenadas en cascada.

*lpRect*: Puntero a un registro *TRect* que describe un área rectangular en coordenadas de pantalla, dentro del cual las ventanas serán ordenadas. Si este parámetro es **nil**, se utilizará el área cliente de la ventana madre.

*cKids*: Indica la cantidad de elementos en el array *lpKids*. Si *lpKids* es **nil**, este parámetro es ignorado.

*lpKids*: Puntero a un *array* de manejadores de ventanas que identifican las ventanas que serán ordenadas. Si a este parámetro se le asigna **nil**, se ordenarán todas las ventanas hijas de la ventana madre.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de ventanas que fueron ordenadas; en caso contrario, devuelve cero.

#### Véase además

*BeginDeferWindowPos*, *DeferWindowPos*, *EndDeferWindowPos*, *MoveWindow*, *SetWindowPlacement*, *TileWindows*, *WM\_MDICASCADE*, *WM\_MDITILE*

#### Ejemplo

##### Listado 10-6: Ordenando en cascada ventanas hijas MDI

```
procedure TForm1.Cascade1Click(Sender: TObject);
begin
    {Esto ordenará todas las ventanas hijas MDI que no estén deshabilitadas}
    CascadeWindows(Form1.ClientHandle, MDITILE_SKIPDISABLED, nil, 0, nil);
end;
```

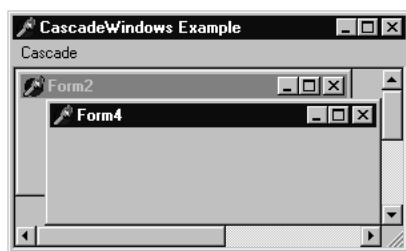


Figura 10-6:  
Ventanas hijas  
MDI en  
cascada

#### CloseWindow      Windows.Pas

##### Sintaxis

```
CloseWindow(
    hWnd: HWND                      {manejador de ventana}
): BOOL;                            {devuelve TRUE o FALSE}
```

##### Descripción

*CloseWindow* minimiza la ventana especificada, pero no la destruye.

##### Parámetros

*hWnd*: Manejador de la ventana que será minimizada.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*ArrangeIconicWindows*, *DestroyWindow*, *IsIconic*, *IsWindowVisible*, *IsZoomed*, *OpenIcon*, *ShowWindow*, *WM\_SIZE*

**Ejemplo****Listado 10-7: Ejemplo de CloseWindow usando OpenIcon e IsIconic**

```
{Esto minimiza y restaura el formulario continuamente}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    {Si nuestro formulario está minimizado...}
    if IsIconic(Form1.Handle) then
        {...lo restaura...}
        OpenIcon(Form1.Handle)
    else
        {...en caso contrario, lo minimiza}
        CloseWindow(Form1.Handle);
end;
```

**DeferWindowPos****Windows.Pas****Sintaxis**

DeferWindowPos(	
hWinPosInfo: HDWP;	{manejador de registro de posición}
hWnd: HWND;	{manejador de ventana a posicionar}
hWndInsertAfter: HWND;	{manejador de la ventana precedente}
X: Integer;	{coordenada horizontal}
Y: Integer;	{coordenada vertical}
CX: Integer;	{ancho, en píxeles}
CY: Integer;	{altura, en píxeles}
uFlags: UINT	{opciones de tamaño y posición}
): HDWP;	{devuelve un manejador de un registro de posición}

**Descripción**

Esta función actualiza el registro de posición de múltiples ventanas especificado, asignando la nueva posición y el nuevo tamaño a la ventana indicada. Utilice la función *BeginDeferWindowPos* para reservar memoria suficiente para este registro. La función *DeferWindowPos* puede incrementar el tamaño de este registro si es necesario, pero si no hay suficiente memoria para ello, la secuencia entera fallará. Cuando la función *EndDeferWindowPos* es llamada, Windows utiliza este registro para mover y

redimensionar las múltiples ventanas simultáneamente. La pantalla no es refrescada hasta que la función *EndDeferWindowPos* no termine.

Observe que las ventanas que son propiedad de una ventana siempre visible serán también convertidas en siempre visibles, de manera que aparezcan sobre su propietaria, pero las ventanas propietarias de la ventana especificada no serán cambiadas. Por eso, una ventana que no sea siempre visible puede ser propietaria de una ventana siempre visible, pero una ventana siempre visible no puede ser propietaria de una ventana que no lo sea. Si se quita el atributo de siempre visible a una ventana, las ventanas de su propiedad también dejarán de ser siempre visibles.

### Parámetros

*hWinPosInfo*: Manejador del registro de posición de múltiples ventanas devuelto por *BeginDeferWindowPos* o la llamada más reciente a *DeferWindowPos*.

*hWnd*: Manejador de la ventana que será movida o redimensionada.

*hWndInsertAfter*: Identifica a la ventana que precederá a la ventana reposicionada en el orden Z. Puede ser un manejador de ventana o un valor de la Tabla 10-2. Este parámetro es ignorado si la opción *SWP\_NOZORDER* se asigna al parámetro *Flags*. Si a este parámetro se le asigna cero, la ventana será ubicada en la cima del orden Z. Si el orden Z de la posición de una ventana es situado encima de todas las demás ventanas siempre visibles, la ventana se convertirá en una ventana siempre visible. Esto tiene el mismo efecto que si se especifica la opción *HWND\_TOPMOST* para este parámetro.

*X*: La coordenada horizontal de la esquina superior izquierda de la ventana. Si esta ventana es una ventana hija, las coordenadas son relativas al área cliente de su ventana madre.

*Y*: La coordenada vertical de la esquina superior izquierda de la ventana. Si esta ventana es una ventana hija, las coordenadas son relativas al área cliente de su ventana madre.

*CX*: El nuevo ancho de la ventana, en píxeles.

*CY*: La nueva altura de la ventana, en píxeles.

*uFlags*: Especifica una combinación de valores de la Tabla 10-3 que afectarán el tamaño y la posición de la ventana. Dos o más valores pueden ser combinados usando el operador booleano **or**.

### Valor que devuelve

Si esta función tiene éxito, devuelve un manejador del registro de posición de múltiples ventanas. Este registro puede ser diferente del que la función recibió, y debe ser usado en las llamadas posteriores a *DeferWindowPos* y *EndDeferWindowPos*. En caso contrario, esta función devuelve 0. Si la función falla, la aplicación debe abandonar la operación de posicionamiento de ventana y no debe llamar a *EndDeferWindowPos*.

Véase además

*BeginDeferWindowPos*, *EndDeferWindowPos*, *GetWindowPlacement*, *MoveWindow*, *SetWindowPlacement*, *SetWindowPos*, *ShowWindow*, *WM\_MOVE*, *WM\_SIZE*, *WM\_WINDOWPOSCHANGED*, *WM\_WINDOWPOSCHANGING*.

### Ejemplo

Consulte el Listado 10-4 bajo *BeginDeferWindowPos*

**Tabla 10-2: Valores del parámetro *hWndInsertAfter* de *DeferWindowPos***

Valor	Descripción
HWND_BOTTOM	Coloca la ventana en el fondo del orden Z. Si la ventana era una ventana siempre visible, pierde ese estatus y es situada debajo de las demás.
HWND_NOTOPMOST	Coloca la ventana encima de todas las ventanas que no sean siempre visibles, pero detrás de las siempre visibles. Si la ventana no es siempre visible, esta opción no tiene efecto.
HWND_TOP	Coloca la ventana en la cima del orden Z.
HWND_TOPMOST	Coloca la ventana encima de todas las ventanas que no sean siempre visibles; mantendrá su condición de siempre visible aún cuando esté desactivada.

**Tabla 10-3: Valores del parámetro *Uflags* de *DeferWindowPos***

Valor	Descripción
SWP_DRAWFRAME	Dibuja el marco definido en la descripción de la clase de la ventana alrededor de la ventana.
SWP_FRAMECHANGED	Provoca que un mensaje <i>WM_NCCALCSIZE</i> sea enviado a la ventana, aún cuando el tamaño de la ventana no esté cambiando.
SWP_HIDEWINDOW	Oculto la ventana.
SWP_NOACTIVATE	No activa la ventana. Si esta opción no es especificada, la ventana es activada y movida a la cima del grupo de ventanas siempre visibles o de ventanas no siempre visibles, en dependencia del valor del parámetro <i>hWndInsertAfter</i> .
SWP_NOCOPYBITS	Descarta el contenido del área cliente. Si esta opción no es especificada, la parte válida del área cliente es guardada y restaurada después de que finalicen todos los movimientos y redimensionamientos de ventanas.
SWP_NOMOVE	Mantiene la posición actual, ignorando los parámetros X e Y.
SWP_NOOWNERZORDER	No cambia la posición de la ventana propietaria en el orden Z.

Valor	Descripción
SWP_NOREDRAW	Si esta opción es especificada, no se redibuja la ventana y la aplicación tiene que invalidar o redibujar explícitamente cualquier parte de la ventana que necesite ser redibujada, incluyendo el área no cliente y las barras de desplazamiento.
SWP_NOREPOSITION	Equivalente a la opción SWP_NOOWNERZORDER.
SWP_NOSENDCHANGING	La ventana no recibirá los mensajes WM_WINDOWPOSCHANGING.
SWP_NOSIZE	Mantiene el tamaño actual, ignorando los parámetros CX y CY.
SWP_NOZORDER	Mantiene el orden Z actual, haciendo que el parámetro hWndInsertAfter sea ignorado.
SWP_SHOWWINDOW	Muestra la ventana.

### **EndDeferWindowPos**      **Windows.Pas**

#### **Sintaxis**

```
EndDeferWindowPos(
    hWinPosInfo: HDWP      {manejador del registro de posición}
): BOOL;                    {devuelve TRUE o FALSE}
```

#### **Descripción**

Esta es la última función a llamar de una serie de funciones que se utilizan para mover y redimensionar simultáneamente múltiples ventanas con un mínimo de refresco de pantalla. La función *BeginDeferWindowPos* es llamada primero para reservar memoria para un registro de posición de múltiples ventanas, que almacenará la nueva posición y el tamaño de cada ventana a mover. A continuación, se puede llamar a la función *DeferWindowPos* para cada ventana a modificar. La función *EndDeferWindowPos* es llamada en último lugar. Esta función envía los mensajes *WM\_WINDOWPOSCHANGING* y *WM\_WINDOWPOSCHANGED* a cada ventana y actualiza la pantalla sólo cuando todas las ventanas han sido modificadas.

#### **Parámetros**

*hWinPosInfo*: Manejador del registro interno de posición de múltiples ventanas. Este manejador es devuelto por las funciones *BeginDeferWindowPos* y *DeferWindowPos*.

#### **Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*BeginDeferWindowPos*, *DeferWindowPos*, *GetWindowPlacement*, *MoveWindow*, *SetWindowPlacement*, *SetWindowPos*, *ShowWindow*, *WM\_MOVE*, *WM\_SIZE*, *WM\_WINDOWPOSCHANGED*, *WM\_WINDOWPOSCHANGING*,

**Ejemplo**

Consulte el Listado 10-4 bajo *BeginDeferWindowPos*.

## **GetWindowPlacement    Windows.Pas**

**Sintaxis**

```
GetWindowPlacement(
    hWnd: HWND;                      {manejador de ventana}
    WindowPlacement: PWindowPlacement {puntero a registro de posición de
                                     ventana}
): BOOL;                            {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recupera el estado de visualización y las posiciones normal, minimizada y maximizada de la ventana especificada.

**Parámetros**

*hWnd*: Manejador de la ventana cuya información de posición será recuperada.

*WindowPlacement*: Puntero a un registro de tipo *TWindowPlacement* que recibirá el estado actual y la información de posición de la ventana. Este registro se define de la siguiente manera:

```
TWindowPlacement = packed record
    length: UINT;          {tamaño del registro en bytes}
    flags: UINT;           {opciones de posicionamiento}
    showCmd: UINT;        {opciones de estado de visualización}
    ptMinPosition: TPoint; {coordenadas minimizadas}
    ptMaxPosition: TPoint; {coordenadas maximizadas}
    rcNormalPosition: TRect; {coordenada de la posición restaurada}
end;
```

Antes de llamar a esta función, al campo *Length* hay que asignarle *SizeOf(TWindowPlacement)*. Los campos de este registro se asignan con información de posición después de que esta función sea llamada. Consulte la función *SetWindowPlacement* para ver una descripción de este registro.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*SetWindowPlacement*, *SetWindowPos*, *ShowWindow*

### Ejemplo

Consulte el Listado 10-9 bajo *SetWindowPlacement*.

## MoveWindow Windows.Pas

### Sintaxis

```
MoveWindow(
    hWnd: HWND;           {manejador de la ventana a mover}
    X: Integer;            {la nueva coordenada horizontal}
    Y: Integer;            {la nueva coordenada vertical}
    nWidth: Integer;       {nuevo ancho de la ventana}
    nHeight: Integer;      {nueva altura de la ventana}
    bRepaint: BOOL         {la opción de redibujado}
): BOOL;                 {devuelve TRUE o FALSE}
```

### Descripción

Esta función cambia la posición y dimensiones de la ventana especificada. Si la ventana especificada es una ventana de nivel superior, las coordenadas son relativas a la pantalla. Si la ventana especificada es una ventana hija, las coordenadas son relativas al área cliente de su ventana madre.

Esta función envía los siguientes mensajes a la ventana especificada:

*WM\_WINDOWPOSCHANGING*, *WM\_WINDOWPOSCHANGED*, *WM\_MOVE*, *WM\_SIZE* y *WM\_NCCALCSIZE*.

### Parámetros

*hWnd*: Manejador de la ventana que será modificada.

*X*: La nueva coordenada horizontal del lado izquierdo de la ventana.

*Y*: La nueva coordenada vertical del lado superior de la ventana.

*nWidth*: Especifica el nuevo ancho de la ventana.

*nHeight*: Especifica la nueva altura de la ventana.

*bRepaint*: Determina cómo la ventana será redibujada. Si este parámetro es TRUE, la función *MoveWindow* llama a la función *UpdateWindow*. Esta envía un mensaje *WM\_PAINT* a la ventana, causando que ésta sea redibujada inmediatamente después de que la ventana sea movida. Si este parámetro es FALSE, no se realizará ninguna actualización de la pantalla, incluyendo el área no cliente completa y cualquier parte de la ventana madre descubierta por una ventana hija. La aplicación deberá invalidar o repintar explícitamente cualquier área que necesite ser actualizada como resultado de la llamada a la función *MoveWindow*. Un mensaje *WM\_PAINT* es colocado en la cola de

mensajes de la ventana especificada, pero su bucle de mensajes sólo despachará el mensaje *WM\_PAINT* después que todos los demás mensajes hayan sido despachados.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*BeginDeferWindowPos*, *DeferWindowPos*, *EndDeferWindowPos*,  
*SetWindowPlacement*, *SetWindowPos*

#### Ejemplo

##### Listado 10-8: Moviendo una ventana

```
procedure TForm1.Timer1Timer(Sender: TObject);
const
    XPos: Integer = 5;    // posición horizontal inicial
begin
    {Incrementa la posición horizontal}
    Inc(XPos);

    {Mueve el control de edición a la derecha}
    MoveWindow(Edit1.Handle, XPos, Edit1.Top, Edit1.Width, Edit1.Height, TRUE);
end;
```

#### OpenIcon

#### Windows.Pas

#### Sintaxis

```
OpenIcon(
    hWnd: HWND           {manejador de una ventana minimizada}
): BOOL;                {devuelve TRUE o FALSE}
```

#### Descripción

Esta función restaura y activa la ventana minimizada especificada. Un mensaje *WM\_QUERYOPEN* es enviado a la ventana cuando esta función es llamada.

#### Parámetros

*hWnd*: Manejador de la ventana que será restaurada y activada.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*ArrangeIconicWindows*, *CloseWindow*, *DestroyWindow*, *IsIconic*, *IsWindowVisible*,  
*IsZoomed*, *ShowWindow*, *WM\_SIZE*, *WM\_QUERYOPEN*

**Ejemplo**

Consulte el Listado 10-7 bajo *CloseWindow*.

**SetWindowPlacement      Windows.Pas****Sintaxis**

```
SetWindowPlacement(
  hWnd: HWND;                      {manejador de ventana}
  WindowPlacement: PWindowPlacement {puntero a un registro de posición de la
                                   ventana}
): BOOL;                          {devuelve TRUE o FALSE}
```

**Descripción**

Esta función selecciona el estado de presentación y las coordenadas normal, minimizada y maximizada de la ventana especificada.

**Parámetros**

*hWnd*: Manejador de la ventana cuya información de presentación será asignada.

*WindowPlacement*: Puntero a un registro de tipo *TWindowPlacement* que contiene el estado de presentación y la información sobre la ubicación de la ventana. Este registro se define de la siguiente manera:

```
TWindowPlacement = packed record
  length: UINT;          {tamaño del registro en bytes}
  flags: UINT;           {opciones de posicionamiento}
  showCmd: UINT;         {opciones de estado de presentación}
  ptMinPosition: TPoint;  {coordenadas minimizadas}
  ptMaxPosition: TPoint;  {coordenadas maximizadas}
  rcNormalPosition: TRect; {coordenadas de la posición restaurada}
end;
```

*length*: El tamaño del registro, en bytes. Antes de llamar a la función, a este campo hay que asignarle *SizeOf(TWindowPlacement)*.

*flags*: Especifica opciones que controlan la posición de una ventana minimizada y el método mediante el cual será restaurada. Este campo puede incluir los valores de la Tabla 10-4.

*showCmd*: Especifica el estado de presentación actual de la ventana y puede ser uno de los valores de la Tabla 10-5.

*ptMinPosition*: Las coordenadas de la esquina superior izquierda de la ventana cuando es minimizada, almacenadas en un registro *TPoint*.

*ptMaxPosition*: Las coordenadas de la esquina superior izquierda de la ventana cuando es maximizada, almacenadas en un registro *TPoint*.

*rcNormalPosition*: Las coordenadas de la ventana en una posición normal, restaurada, almacenadas en un registro *TPoint*.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*GetWindowPlacement, SetWindowPos, ShowWindow*

**Ejemplo****Listado 10-9: Información sobre la ubicación de una ventana**

```
{Obtiene la ubicación de la ventana}
procedure TForm1.Button1Click(Sender: TObject);
var
    PlacementInfo: TWindowPlacement;
begin
    {Primero tenemos que asignar el tamaño del registro al campo Length}
    PlacementInfo.Length := SizeOf(TWindowPlacement);

    {Obtiene la información sobre la ubicación de la ventana}
    GetWindowPlacement(Form1.Handle, @PlacementInfo);

    {vacía el cuadro de lista}
    ListBox1.Items.Clear;

    {Muestra toda la información del registro de ubicación de la ventana}
    ListBox1.Items.Add('Length: ' + IntToStr(PlacementInfo.Length));
    ListBox1.Items.Add('Flags: ' + IntToStr(PlacementInfo.Flags));
    ListBox1.Items.Add('Show Command: ' + IntToStr(PlacementInfo.showCmd));
    ListBox1.Items.Add('Min: ' + IntToStr(PlacementInfo.ptMinPosition.X) + ', ' +
        IntToStr(PlacementInfo.ptMinPosition.Y));
    ListBox1.Items.Add('Max: ' + IntToStr(PlacementInfo.ptMaxPosition.X) + ', ' +
        IntToStr(PlacementInfo.ptMaxPosition.Y));
    ListBox1.Items.Add('Normal position: ' +
        IntToStr(PlacementInfo.rcNormalPosition.Left) + ', ' +
        IntToStr(PlacementInfo.rcNormalPosition.Top) + ', ' +
        IntToStr(PlacementInfo.rcNormalPosition.Right) + ', ' +
        IntToStr(PlacementInfo.rcNormalPosition.Bottom));

end;

{Asigna la ubicación de la ventana}
procedure TForm1.Button2Click(Sender: TObject);
var
    PlacementInfo: TWindowPlacement;
begin
    {Primero tenemos que asignar el tamaño del registro al campo Length}
    PlacementInfo.Length := SizeOf(TWindowPlacement);

    {Llena el resto de los campos del registro de la ventana}
    PlacementInfo.flags := WPF_SETMINPOSITION;
    PlacementInfo.showCmd := SW_SHOW;
    PlacementInfo.ptMinPosition.X := 100;
```

```

PlacementInfo.ptMinPosition.Y := 100;
PlacementInfo.ptMaxPosition.X := 50;
PlacementInfo.ptMaxPosition.Y := 50;
PlacementInfo.rcNormalPosition.Left := 100;
PlacementInfo.rcNormalPosition.Top := 100;
PlacementInfo.rcNormalPosition.Right := 250;
PlacementInfo.rcNormalPosition.Bottom := 250;

{Asigna la información de ubicación de la ventana}
SetWindowPlacement(Form1.Handle, @PlacementInfo);
end;

```

Figura 10-7:  
Obteniendo la  
información  
sobre la  
ubicación de  
la ventana

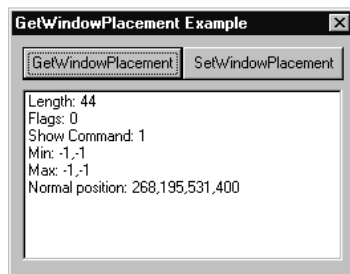


Tabla 10-4: Valores del campo `WindowPlacement.flags` de `SetWindowPlacement`

Valor	Descripción
WPF_RESTORETOMAXIMIZED	La ventana será maximizada la próxima vez que sea restaurada, independientemente de si fue o no maximizada antes de ser minimizada. Esto es válido solo para la próxima vez que la ventana sea restaurada y cuando se asigne el valor <code>SW_SHOWMINIMIZED</code> al campo <code>ShowCmd</code> . Esto no cambia el comportamiento por defecto de la restauración.
WPF_SETMINPOSITION	Las coordenadas de la ventana minimizada pueden ser especificadas. Esta opción tiene que ser incluida si se asignan coordenadas al campo <code>ptMinPosition</code> .

Tabla 10-5: Valores del campo `WindowPlacement.showCmd` de `SetWindowPlacement`

Valor	Descripción
SW_HIDE	La ventana es ocultada y otra ventana es activada.
SW_MINIMIZE	La ventana es minimizada y la siguiente ventana de nivel superior en la lista de ventanas del sistema es activada.
SW_RESTORE	La ventana es activada y mostrada en su tamaño y posición original.
SW_SHOW	La ventana es activada y mostrada en su tamaño y posición actual.

Valor	Descripción
SW_SHOWDEFAULT	La ventana es mostrada según indique el campo <code>wShowWindow</code> del registro <code>TStartupInfo</code> pasado a la función <code>CreateProcess</code> que lanzó la aplicación. Esto es usado para asignar el estado de presentación inicial de una ventana principal de una aplicación. Esta opción debe ser usada cuando se muestra la ventana por primera vez, si la aplicación puede ser ejecutada desde un acceso directo. Esta opción hace que la ventana sea mostrada usando los atributos especificados en las propiedades del acceso directo.
SW_SHOWMAXIMIZED	La ventana es activada y mostrada en estado maximizado.
SW_SHOWMINIMIZED	La ventana es activada y mostrada como un icono.
SW_SHOWMINNOACTIVE	La ventana es mostrada como un icono. La ventana activa permanece activa.
SW_SHOWNORMAL	La ventana es mostrada en su estado actual. La ventana activa permanece activa.
SW_SHOWNOACTIVE	La ventana es mostrada en su estado más reciente. La ventana activa permanece activa.
SW_SHOWNORMAL	Equivalente a <code>SW_RESTORE</code> .

## SetWindowPos Windows.Pas

### Sintaxis

```
SetWindowPos(
    hWnd: HWND;           {manejador de una ventana}
    hWndInsertAfter: HWND; {manejador de ventana u opción de ubicación}
    X: Integer;            {posición horizontal}
    Y: Integer;            {posición vertical}
    CX: Integer;           {ancho de la ventana}
    CY: Integer;           {altura de la ventana}
    uFlags: UINT           {opción de tamaño y posición}
): BOOL;                 {devuelve TRUE o FALSE}
```

### Descripción

Esta función cambia el tamaño, posición y orden Z de la ventana especificada. El orden Z de ventanas hijas, emergentes y de nivel superior es determinado por el orden en el que estas ventanas aparecen en pantalla. Una ventana siempre visible es la primera ventana en el orden Z.

Observe que las ventanas que son propiedad de una ventana siempre visible serán también convertidas en siempre visibles, de manera que aparezcan sobre su propietaria, pero las ventanas propietarias de la ventana especificada no serán cambiadas. Por eso,

una ventana que no sea siempre visible puede ser propietaria de una ventana siempre visible, pero una ventana siempre visible no puede ser propietaria de una ventana que no lo sea. Si se quita el atributo de siempre visible a una ventana, las ventanas de su propiedad también dejarán de ser siempre visibles.

### Parámetros

*hWnd*: Manejador de la ventana que va a ser movida o redimensionada.

*hWndInsertAfter*: Identifica a la ventana que precederá a la ventana reposicionada en el orden Z. Puede ser un manejador de ventana o un valor de la Tabla 10-6. Este parámetro es ignorado si la opción *SWP\_NOZORDER* se asigna al parámetro *Flags*. Si a este parámetro se le asigna cero, la ventana será ubicada en la cima del orden Z. Si el orden Z de la posición de una ventana es situado encima de todas las demás ventanas siempre visibles, la ventana se convertirá en una ventana siempre visible. Esto tiene el mismo efecto que si se especifica la opción *HWND\_TOPMOST* para este parámetro.

*X*: La coordenada horizontal de la esquina superior izquierda de la ventana. Si esta ventana es una ventana hija, las coordenadas son relativas al área cliente de su ventana madre.

*Y*: La coordenada vertical de la esquina superior izquierda de la ventana. Si esta ventana es una ventana hija, las coordenadas son relativas al área cliente de su ventana madre.

*CX*: El nuevo ancho de la ventana, en píxeles.

*CY*: La nueva altura de la ventana, en píxeles.

*uFlags*: Especifica una combinación de valores de la Tabla 10-7 que afectarán al tamaño y a la posición de la ventana. Dos o más valores pueden ser combinados usando el operador booleano **or**.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*BeginDeferWindowPos*, *DeferWindowPos*, *EndDeferWindowPos*, *MoveWindow*, *SetActiveWindow*, *SetForegroundWindow*, *SetWindowPlacement*, *ShowWindow*, *WM\_MOVE*, *WM\_SIZE*

### Ejemplo

#### Listado 10-10: Asignando la posición de la ventana

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    {Redimensiona el memo de modo que ocupe el área cliente bajo el botón}
    SetWindowPos(Memo1.Handle, 0, 0,
```

```

Button1.Top + Button1.Height + 5, Form1.ClientWidth,
Form1.ClientHeight - (Button1.Top + Button1.Height + 5),
SWP_SHOWWINDOW);
end;

```

Figura 10-8:  
El memo  
reposicionado

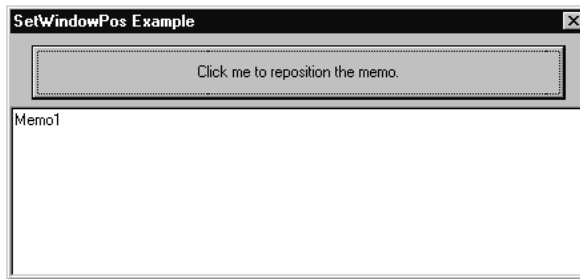


Tabla 10-6: Valores del parámetro `hWndInsertAfter` de `SetWindowPos`

Valor	Descripción
<code>HWND_BOTTOM</code>	Coloca la ventana en el fondo del orden Z. Si la ventana era una ventana siempre visible, pierde ese estatus y es situada debajo de las demás.
<code>HWND_NOTOPMOST</code>	Coloca la ventana encima de todas las ventanas que no sean siempre visibles, pero detrás de las siempre visibles. Si la ventana no es siempre visible, esta opción no tiene efecto.
<code>HWND_TOP</code>	Coloca la ventana en la cima del orden Z.
<code>HWND_TOPMOST</code>	Coloca la ventana encima de todas las ventanas que no sean siempre visibles; mantendrá su condición de siempre visible aún cuando esté desactivada.

Tabla 10-7: Valores del parámetro `Uflags` de `SetWindowPos`

Valor	Descripción
<code>SWP_DRAWFRAME</code>	Dibuja el marco definido en la descripción de la clase de la ventana alrededor de la ventana.
<code>SWP_FRAMECHANGED</code>	Provoca que un mensaje <code>WM_NCCALCSIZE</code> sea enviado a la ventana, aún cuando el tamaño de la ventana no esté cambiando.
<code>SWP_HIDEWINDOW</code>	Oculto la ventana.
<code>SWP_NOACTIVATE</code>	No activa la ventana. Si esta opción no es especificada, la ventana es activada y movida a la cima del grupo de ventanas siempre visibles o de ventanas no siempre visibles, dependiendo del valor del parámetro <code>hWndInsertAfter</code> .

Valor	Descripción
SWP_NOCOPYBITS	Descarta el contenido del área cliente. Si esta opción no es especificada, la parte válida del área cliente es guardada y restaurada después de que finalicen todos los movimientos y redimensionamientos de ventanas.
SWP_NOMOVE	Mantiene la posición actual, ignorando los parámetros X e Y.
SWP_NOOWNERZORDER	No cambia la posición de la ventana propietaria en el orden Z.
SWP_NOREDRAW	Si esta opción es especificada, no se redibuja la ventana y la aplicación tiene que invalidar o redibujar explícitamente cualquier parte de la ventana que necesite ser redibujada, incluyendo el área no cliente y las barras de desplazamiento.
SWP_NOREPOSITION	Equivalente a la opción SWP_NOOWNERZORDER.
SWP_NOSENDCHANGING	La ventana no recibirá los mensajes WM_WINDOWPOSCHANGING.
SWP_NOSIZE	Mantiene el tamaño actual, ignorando los parámetros CX y CY.
SWP_NOZORDER	Mantiene el orden Z actual, haciendo que el parámetro hWndInsertAfter sea ignorado.
SWP_SHOWWINDOW	Muestra la ventana.

**ShowOwnedPopups****Windows.Pas****Sintaxis**

```
ShowOwnedPopups(
    hWnd: HWND;           {manejador de ventana}
    fShow: BOOL            {opción de visibilidad de la ventana}
): BOOL;                 {devuelve TRUE o FALSE}
```

**Descripción**

Esta función mostrará u ocultará todas las ventanas emergentes que sean propiedad de la ventana especificada. Las ventanas emergentes sólo serán mostradas si han sido ocultadas previamente mediante una llamada a *ShowOwnedPopups* (una ventana ocultada con la función *ShowWindow* no será mostrada cuando se llame a *ShowOwnedPopups*).

**Parámetros**

*hWnd*: Manejador de la ventana propietaria de las ventanas emergentes que serán mostradas u ocultadas.

*fShow*: Determina si las ventanas emergentes son mostradas u ocultas. Un valor TRUE muestra todas las ventanas emergentes ocultas que posee la ventana especificada. Un valor FALSE ocultara todas las ventanas emergentes visibles.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener una información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*IsWindowVisible*, *SetWindowPos*, *ShowWindow*

#### Ejemplo

##### Listado 10-II: Cambiando el estado de presentación de ventanas emergentes

Este código pertenece a la unidad del formulario principal:

```
var
  Form1: TForm1;
  ShowIt: Boolean; // nuestra variable de estado

implementation

uses Unit2;

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  {Cambia la variable de estado de presentación}
  ShowIt := not ShowIt;

  {Muestra u oculta todas las ventanas emergentes que son propiedad de la
  ventana principal}
  ShowOwnedPopups(Form1.Handle, ShowIt);
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  {Muestra la segunda ventana cuando el programa arranca}
  Form2.Show;
end;

initialization
  {Inicializa la variable de cambio}
  ShowIt:=TRUE;
end.
```

Este código pertenece a la unidad Form2:

```

uses Unit2;

{Tenemos que sobrescribir CreateParams para asignar un propietario a la ventana}
procedure TForm2.CreateParams(var Params: TCreateParams);
begin
    {Rellena los parámetros de creación por defecto}
    inherited CreateParams(Params);

    {Asigna la ventana principal como propietaria de esta ventana}
    Params.WndParent := Form1.Handle;
end;

```

## ShowWindow Windows.Pas

### Sintaxis

```

ShowWindow(
    hWnd: HWND;           {manejador de ventana}
    nCmdShow: Integer     {estado de presentación de la ventana}
): BOOL;                 {devuelve TRUE o FALSE}

```

### Descripción

Esta función establece el estado de presentación de la ventana especificada. Cuando se muestra la ventana principal de una aplicación, el desarrollador debe especificar la opción *SW\_SHOWDEFAULT*. Esto mostrará la ventana según se indica en la información de arranque de la aplicación. Por ejemplo, si un acceso directo de Windows 95/98 tiene sus propiedades asignadas para que la aplicación se ejecute minimizada, la opción *SW\_SHOWDEFAULT* mostrará la ventana minimizada. Sin esta opción, las propiedades del acceso directo serán ignoradas.

### Parámetros

*hWnd*: Manejador de la ventana que será mostrada.

*nCmdShow*: Especifica cómo será mostrada la ventana. Puede ser un valor de la Tabla 10-8.

### Valor que devuelve

Si la función tiene éxito y la ventana era anteriormente visible, devuelve TRUE. Si la función falla o la ventana estaba anteriormente oculta, devuelve FALSE.

### Véase además

*CreateProcess*, *CreateWindow*, *SetWindowPlacement*, *SetWindowPos*, *ShowOwnedPopups*, *ShowWindowAsync*, *WM\_SHOWWINDOW*

Ejemplo

Listado 10-12: Mostrando una ventana basándose en la propiedades de un acceso directo

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  {Este ejemplo se ejecuta desde un acceso directo y esta línea mostrará la
   ventana en base a las propiedades del acceso directo}
  ShowWindow(Form1.Handle, SW_SHOWDEFAULT);
end;
```

Figura 10-9: Los atributos del acceso directo hacen que esta aplicación se inicie maximizada

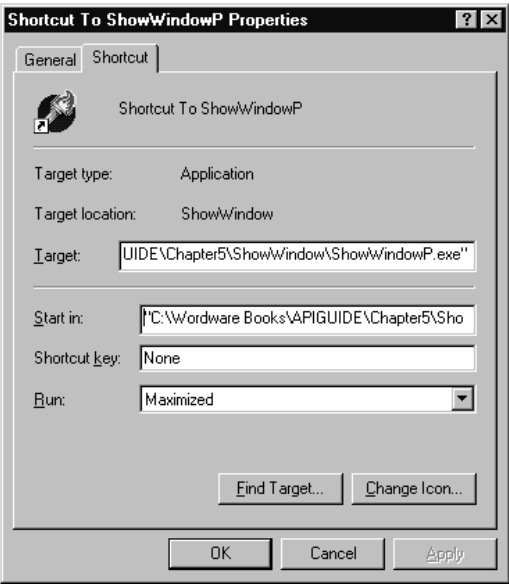


Tabla 10-8: Valores del parámetro nCmdShow de ShowWindow

Valor	Descripción
SW_HIDE	La ventana es ocultada y otra ventana es activada.
SW_MINIMIZE	La ventana es minimizada y la siguiente ventana de nivel superior en la lista de ventanas del sistema es activada.
SW_RESTORE	La ventana es activada y mostrada en su tamaño y posición original.
SW_SHOW	La ventana es activada y mostrada en su tamaño y posición actual.

Valor	Descripción
SW_SHOWDEFAULT	La ventana es mostrada según indique el campo <code>wShowWindow</code> del registro <code>TStartupInfo</code> pasado a la función <code>CreateProcess</code> que lanzó la aplicación. Esto es usado para asignar el estado de presentación inicial de una ventana principal de una aplicación. Esta opción debe ser usada cuando se muestra la ventana por primera vez, si la aplicación puede ser ejecutada desde un acceso directo. Esta opción hace que la ventana sea mostrada usando los atributos especificados en las propiedades del acceso directo.
SW_SHOWMAXIMIZED	La ventana es activada y mostrada en estado maximizado.
SW_SHOWMINIMIZED	La ventana es activada y mostrada como un icono.
SW_SHOWMINNOACTIVE	La ventana es mostrada como un icono. La ventana activa permanece activa.
SW_SHOWNORMAL	La ventana es mostrada en su estado actual. La ventana activa permanece activa.
SW_SHOWNOACTIVE	La ventana es mostrada en su estado más reciente. La ventana activa permanece activa.
SW_SHOWNORMAL	Equivalente a <code>SW_RESTORE</code> .

**ShowWindowAsync****Windows.Pas****Sintaxis**

```
ShowWindowAsync(
    hWnd: HWND;           {manejador de ventana}
    nCmdShow: Integer      {estado de presentación de la ventana}
): BOOL;                 {devuelve TRUE o FALSE}
```

**Descripción**

Esta función es similar a *ShowWindow*. Su propósito es establecer el estado de presentación de una ventana creada por un hilo de ejecución diferente. Esta función envía directamente un mensaje `WM_SHOWWINDOW` a la cola de mensajes de la ventana especificada. Esto permite a la aplicación que hace la llamada continuar la ejecución si la aplicación asociada con la ventana especificada se cuelga.

**Parámetros**

*hWnd*: Manejador de la ventana a mostrar.

*nCmdShow*: Especifica cómo se mostrará la ventana. Puede ser un valor de la Tabla 10-9.

#### Valor que devuelve

Si la función tiene éxito y la ventana era anteriormente visible, devuelve TRUE. Si la función falla o la ventana estaba anteriormente oculta, devuelve FALSE.

#### Véase además

*CreateProcess, CreateWindow, SetWindowPlacement, SetWindowPos, ShowOwnedPopups, ShowWindow, WM\_SHOWWINDOW*

#### Ejemplo

##### Listado 10-13: Mostrando una ventana asíncronamente

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TheWindow: HWND;
begin
  {Encuentra un manejador de ventana del Explorador de Windows. El Explorador
   debe estar ejecutándose}
  TheWindow := FindWindow('ExploreWClass', nil);

  {Lo muestra}
  ShowWindowAsync(TheWindow, SW_MAXIMIZE);
end;
```

**Tabla 10-9: Valores del parámetro nCmdShow de ShowWindowAsync**

Valor	Descripción
SW_HIDE	La ventana es ocultada y otra ventana es activada.
SW_MINIMIZE	La ventana es minimizada y la siguiente ventana de nivel superior en la lista de ventanas del sistema es activada.
SW_RESTORE	La ventana es activada y mostrada en su tamaño y posición original.
SW_SHOW	La ventana es activada y mostrada en su tamaño y posición actual.
SW_SHOWDEFAULT	La ventana es mostrada según indique el campo wShowWindow del registro TStartupInfo pasado a la función <i>CreateProcess</i> que lanzó la aplicación. Esto es usado para asignar el estado de presentación inicial de una ventana principal de una aplicación. Esta opción debe ser usada cuando se muestra la ventana por primera vez, si la aplicación puede ser ejecutada desde un acceso directo. Esta opción hace que la ventana sea mostrada usando los atributos especificados en las propiedades del acceso directo.
SW_SHOWMAXIMIZED	La ventana es activada y mostrada en estado maximizado.

Valor	Descripción
SW_SHOWMINIMIZED	La ventana es activada y mostrada como un icono.
SW_SHOWMINNOACTIVE	La ventana es mostrada como un icono. La ventana activa permanece activa.
SW_SHOWNA	La ventana es mostrada en su estado actual. La ventana activa permanece activa.
SW_SHOWNOACTIVE	La ventana es mostrada en su estado más reciente. La ventana activa permanece activa.
SW_SHOWNORMAL	Equivalente a SW_RESTORE.

**TileWindows****Windows.Pas****Sintaxis**

```

TileWindows(
    hwndParent: HWND;      {manejador de la ventana madre}
    wHow: UINT;             {opciones de control}
    lpRect: PRect;          {área rectangular para ordenar las ventanas}
    cKids: UINT;            {la cantidad de ventanas a ordenar}
    lpKids: Pointer         {la dirección de un array de manejadores de ventana}
): WORD;                  {devuelve la cantidad de ventanas ordenadas}

```

**Descripción**

Esta función ordena en mosaico las ventanas asociadas con los manejadores colocados en el *array* *lpKids* o las ventanas hijas de la ventana especificada. Las ventanas pueden ser puestas en mosaico horizontal o vertical y pueden ser restringidas a un área rectangular dentro de la ventana madre especificada.

**Parámetros**

*hwndParent*: Manejador de la ventana madre. Si este parámetro es cero, el escritorio es usado como ventana madre. Si la función va a ser utilizada para ordenar en cascada ventanas hijas MDI, a este parámetro se le debe asignar la propiedad *ClientHandle* del formulario particular.

*wHow*: *MDITILE\_HORIZONTAL* dispone las ventanas en mosaico horizontal y *MDITILE\_VERTICAL* en mosaico vertical. Estas opciones pueden combinarse con *MDITILE\_SKIPDISABLED* para no colocar en mosaico las ventanas hijas desactivadas. Consulte la Tabla 10-10.

*lpRect*: Puntero a un registro *TRect* que describe un área rectangular en coordenadas de pantalla, dentro del cual las ventanas serán ordenadas. Si este parámetro es **nil**, se utilizará el área cliente de la ventana madre.

*cKids*: Indica la cantidad de elementos en el *array* *lpKids*. Si *lpKids* es **nil**, este parámetro es ignorado.

*lpKids*: Puntero a un *array* de manejadores de ventanas que identifican las ventanas que serán ordenadas. Si a este parámetro se le asigna **nil**, se ordenarán todas las ventanas hijas de la ventana madre.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de ventanas que fueron ordenadas; en caso contrario, devuelve cero.

#### Véase además

*BeginDeferWindowPos*, *CascadeWindows*, *DeferWindowPos*, *EndDeferWindowPos*, *MoveWindow*, *SetWindowPlacement*, *WM\_MDICASCADE*, *WM\_MDITILE*

#### Ejemplo

##### Listado 10-14: Poniendo en mosaico vertical las ventanas hijas MDI

```
procedure TForm1.TileWindows1Click(Sender: TObject);
begin
    {Coloca en mosaico vertical todas las ventanas hijas MDI}
    TileWindows(Form1.ClientHandle, MDITILE_VERTICAL, nil, 0, nil);
end;
```

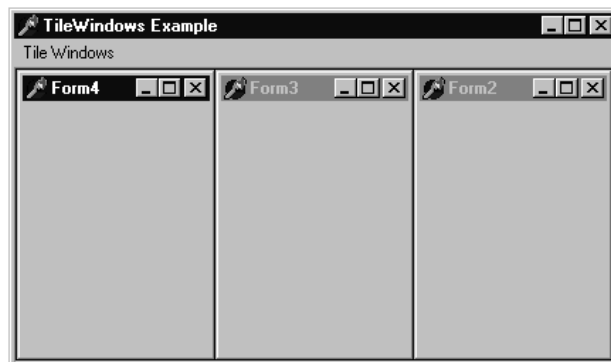


Figura 10-10:  
Ventanas en  
mosaico

Tabla 10-10: Valores del parámetro *wHow* de *TileWindow*

Valor	Descripción
MDITILE_HORIZONTAL	Las ventanas son colocadas en mosaico horizontal.
MDITILE_SKIPDISABLED	Las ventanas deshabilitadas no son colocadas en mosaico.
MDITILE_VERTICAL	Las ventanas son colocadas en mosaico vertical.



## Capítulo 11

## Funciones del shell

El *shell* de Windows ofrece al desarrollador un conjunto de funciones que tienen que ver con la organización y manipulación de ficheros. Estas funciones permiten al desarrollador consultar propiedades de un fichero tales como su tipo, sus iconos o su fichero ejecutable asociado. Las funciones para mover, copiar, renombrar, eliminar o ejecutar un fichero permiten dotar a las aplicaciones de una funcionalidad similar a la del Explorador de Windows. Las funciones de arrastrar y soltar posibilitan al usuario arrastrar ficheros desde el Explorador de Windows sobre una aplicación para abrirlos, imprimirlos o realizar cualquier acción deseada. Estas funciones constituyen el corazón de las barras de lanzamiento de aplicaciones y otras utilidades de organización y manipulación de ficheros. El nuevo *shell* de Windows 95/98 y Windows NT 4.0 también brinda nuevos elementos de interfaz de usuario que permiten lograr una apariencia más especializada.

### Aplicaciones basadas en ficheros

Las funciones de arrastrar y soltar del *shell* y las de manipulación de ficheros pueden mejorar enormemente la funcionalidad de aplicaciones basadas en ficheros. La posibilidad de arrastrar un fichero sobre una aplicación para abrirlo es esencial para cualquier aplicación actual basada en ficheros. Las funciones *DragAcceptFiles*, *DragQueryFile* y *DragFinish* son imprescindibles para implementar métodos amigables e intuitivos para que el usuario abra los ficheros.

El siguiente ejemplo de barra de lanzamiento muestra el uso de una variedad de funciones del *shell*. Cuando se arrastran ficheros desde el Explorador y se dejan caer en la ventana de la barra de lanzamiento, la aplicación mira en cada fichero que ha sido soltado y crea un botón para cada fichero ejecutable. Un icono es extraído del fichero ejecutable y utilizado como imagen del botón. Entonces, la función *ShellExecute* es llamada en el evento *OnClick* del botón para lanzar el fichero ejecutable.

**Listado II-1: Una aplicación de barra de lanzamiento con funcionalidad de arrastrar y soltar ficheros**

```

unit LaunchU;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics,
    Controls, Forms, Dialogs, Buttons, ShellAPI, ExtCtrls;

type
    TForm1 = class(TForm)
        procedure FormCreate(Sender: TObject);
        procedure SpeedButtonClick(Sender: TObject);
    private
        {Declaraciones privadas}
        {El manejador de mensaje necesario para procesar los ficheros soltados}
        procedure WMDropFiles(var Msg: TWMDropFiles); message WM_DROPFILES;
    public
        {Declaraciones públicas}
    end;

var
    Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Registra el formulario como un destino para soltar ficheros}
    DragAcceptFiles(Form1.Handle, TRUE);
end;

procedure TForm1.WMDropFiles(var Msg: TWMDropFiles);
var
    NumDroppedFiles: UINT;           // número de ficheros soltados
    TextBuffer: PChar;               // nombre de fichero
    BufferSize: UINT;                 // tamaño del buffer para el nombre de fichero
    Count: Integer;                  // variable de control de bucle
    LargeIcon, SmallIcon: HICON;     // almacena manejadores de iconos del fichero
begin
    {Recupera el número de ficheros que se han dejado caer sobre el formulario}
    NumDroppedFiles := DragQueryFile(Msg.Drop, $FFFFFFFF, nil, 0);

    {Para cada fichero dejado sobre el formulario...}
    for Count := 0 to NumDroppedFiles - 1 do
    begin
        {Obtiene la longitud del nombre del fichero y reserva una cadena de tamaño
         suficiente para almacenarla (añade un carácter para el terminador nulo)}
        BufferSize := DragQueryFile(Msg.Drop, Count, nil, 0);
        TextBuffer := StrAlloc(BufferSize + 1);
    
```

```

{El nombre del fichero}
DragQueryFile(Msg.Drop, Count, TextBuffer, BufferSize+1);

{Si el fichero es ejecutable...}
if (ExtractFileExt(UpperCase(string(TextBuffer))) = '.EXE') then
  {...crea un botón para él e inicializa las propiedades}
  with TSpeedButton.Create(Form1) do
    begin
      Parent := Form1;

      {En la propiedad Hint se almacena la ruta y nombre del fichero ejecutable
      dejado caer sobre el formulario. La parte corta almacena sólo el nombre
      de fichero, mientras que en la parte larga almacenamos la ruta completa
      y el nombre de fichero, que se utiliza para lanzar el ejecutable.}
      Hint := ExtractFileName(string(TextBuffer)) + '|' + TextBuffer;
      ShowHint := TRUE;

      {Asigna las coordenadas del botón}
      if Form1.ComponentCount = 1 then
        Left := 4
      else
        Left := TSpeedButton(Form1.Components[Form1.ComponentCount-2]).Left +
          TSpeedButton(Form1.Components[Form1.ComponentCount-2]).Width + 4;
      Top := 4;

      {Asigna el método OnClick de manera que el botón haga algo}
      OnClick := SpeedButtonClick;

      {Extrae el icono pequeño del ejecutable y lo muestra en el botón}
      with Glyph do
        begin
          ExtractIconEx(TextBuffer, 0, LargeIcon, SmallIcon, 1);

          {Tenemos que seleccionar el ancho y altura de la imagen de manera que
          sea lo suficientemente grande como para mostrar el icono pequeño}
          Width := GetSystemMetrics(SM_CXSMICON);
          Height := GetSystemMetrics(SM_CYSMICON);
          DrawIconEx(Canvas.Handle, 0, 0, SmallIcon,
            GetSystemMetrics(SM_CXSMICON),
            GetSystemMetrics(SM_CYSMICON), 0, 0, DI_NORMAL);

          DeleteObject(SmallIcon);
        end;
      end;

      {Libera el buffer de texto para el nombre de fichero}
      StrDispose(TextBuffer);
    end;

    {Libera la memoria reservada para el registro}
    DragFinish(Msg.Drop);
  end;
end;

```

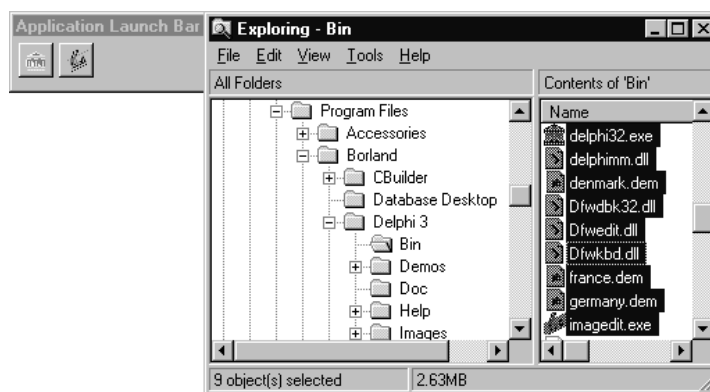
```

procedure TForm1.SpeedButtonClick(Sender: TObject);
begin
    {Cuando se pulsa el botón, la parte larga de su indicación contiene la ruta y
    el nombre de fichero del ejecutable, que ejecutamos.}
    ShellExecute(Form1.Handle, 'open', PChar(GetLongHint(TControl(Sender).Hint)),
        nil, nil, SW_SHOWNORMAL);
end;

end.

```

Figura 11-1:  
Ficheros  
soltados sobre  
la barra de  
lanzamiento



Una aplicación basada en ficheros puede querer darle al usuario la posibilidad de ir directamente a una ventana del Explorador de Windows mostrando el directorio de la aplicación. Con una sola línea de código, el desarrollador puede lograr esto, usando la función *ShellExecute*. Cuando *ShellExecute* “explora” una carpeta, abre una nueva ventana de Explorador de Windows con el directorio especificado seleccionado. El siguiente ejemplo muestra esta técnica.



**Nota:** Si Delphi no está instalado en su directorio por defecto, este ejemplo tendrá que ser modificado.

#### Listado 11-2: Explorando un carpeta

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Abre una nueva ventana del Explorador con la carpeta de Delphi seleccionada}
    ShellExecute(Form1.Handle, 'explore', 'C:\Program Files\Borland\Delphi5',
        nil, nil, SW_SHOWNORMAL);
end;

```

Las aplicaciones que generan ficheros temporales deben ofrecer la funcionalidad necesaria para borrar estos ficheros sin la intervención del usuario. Una aplicación basada en ficheros puede también necesitar ofrecer un mecanismo que permita al usuario copiar, mover, renombrar o borrar ficheros dentro de un ambiente controlado.

La función *SHFileOperation* permite satisfacer todas las necesidades típicas de manipulación de ficheros. El siguiente ejemplo muestra cómo enviar un fichero a la papelera de reciclaje.

### Listado II-3: Enviando un fichero a la papelera de reciclaje

```
procedure TForm1.FileListBox1DbClick(Sender: TObject);
var
    FileOperation: TSHFileOpStruct; // información sobre el fichero a borrar
begin
    {Inicializa la variable TSHFileOpStruct con la información necesaria. La opción
     FOF_ALLOWUNDO indica que el fichero borrado irá a la papelera de reciclaje}
    FileOperation.fFlags      := FOF_ALLOWUNDO or FOF_SIMPLEPROGRESS;
    FileOperation.Wnd        := Form1.Handle;
    FileOperation.wFunc       := FO_DELETE;
    FileOperation.pFrom       := PChar(FileListBox1.FileName + #0#0);
    FileOperation.pTo         := nil;
    FileOperation.hNameMappings := nil;
    FileOperation.lpszProgressTitle := 'Deleting Files';

    {Borra el fichero especificado}
    SHFileOperation(FileOperation);

    {Actualiza la información del cuadro de lista}
    FileListBox1.Update;
end;
```

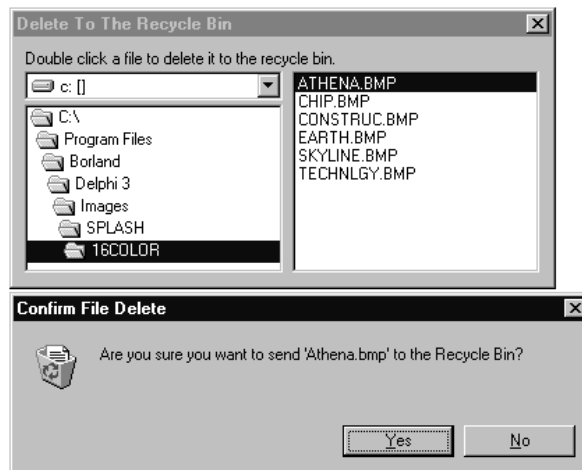


Figura II-2:  
Confirmación  
del borrado  
del fichero

## Listas de identificadores de elementos

Cada objeto en un espacio de nombres del *shell* (tales como ficheros, carpetas, servidores, grupos de trabajo, impresoras, etc.) es identificado de forma única mediante un objeto llamado *identificador de elemento* (*item identifier*). Un identificador de elemento es un registro binario de longitud variable, cuyo contenido y formato son

conocidos sólo por el creador del mismo. Los identificadores de elementos son devueltos por varias funciones de organización de ficheros.

La organización del espacio de nombres del *shell* es análoga a la organización de ficheros en una estructura de directorios. La raíz del espacio de nombres del *shell* es el escritorio, y todos los objetos por debajo de él pueden contener otros objetos. El identificador de elemento de cada objeto es único y tiene sentido solamente dentro del contexto de su padre. Dado que los objetos contenedores tienen un identificador de elemento que los identifica de modo único dentro del contenedor padre, cualquier objeto puede ser identificado de forma única mediante una lista de identificadores de elementos. Por lo tanto, una *lista de identificadores de elementos* identifica de forma única un objeto dentro del espacio de nombres del *shell* trazando una ruta que parte desde él y llega hasta el escritorio. Muchas de las funciones de organización y manipulación de ficheros utilizan listas de identificadores de elementos para especificar ficheros o carpetas.

Las listas de identificadores de elementos son comúnmente utilizadas con los objetos del Modelo de Objetos Componentes (*Component Object Model* - COM) del *shell*. Estos objetos ofrecen una interfaz más avanzada y compleja para el *shell* de Windows que las funciones tratadas en este capítulo. Se han escrito libros enteros acerca del modelo COM y de cómo trabajar con tales objetos (vea el libro “*Programación COM con Delphi*” editado por Danysoft). Por esa razón, una explicación del modelo COM está más allá del alcance de este capítulo.

### Las barras de aplicación

El *shell* de Windows 95/98 y NT introduce un nuevo elemento de interfaz de usuario conocido como *barra de aplicación* (*AppBar*). Una barra de aplicación es una ventana asociada con uno de los bordes de la pantalla. El espacio ocupado por la barra de aplicación es reservado para su propio uso, y el sistema evita que otras ventanas utilicen este área. Varias aplicaciones muy populares incluyen barras de aplicación, la mayoría de las cuales brindan al usuario una forma de organizar ficheros alternativa a la que ofrece el menú **Inicio**. La barra de tareas de Windows es un tipo especial de barra de aplicación.

La función *SHAppBarMessage* suministra la interfaz al *shell* de Windows para registrar una barra de aplicación y controlar su posición. La aplicación se comunica con el *shell* a través de esta función, enviándole mensajes de barras de aplicación. Una barra de aplicación es registrada usando la función *SHAppBarMessage* para enviarle al sistema un mensaje *ABM\_NEW*. Cuando una aplicación crea una *AppBar*, debe utilizar el mensaje *ABM\_QUERYPOS* para recuperar un área donde el sistema aprueba que la barra resida. El mensaje *ABM\_SETPOS* es usado entonces para informar al sistema que la barra está ocupando el área rectangular específico de la pantalla. Para mover físicamente la barra de aplicación dentro del área se utiliza la función *MoveWindow*. Una vez que la barra está situada en su posición, podrá recibir mensajes de notificación a través de mensajes definidos por la aplicación para informarle de eventos que pudieran afectar su apariencia. Estos eventos incluyen sucesos tales como un cambio en

el estado de la barra de tareas de Windows o el lanzamiento o cierre de una aplicación a pantalla completa.

Las barras de aplicación ofrecen al desarrollador Delphi una alternativa para usar una ventana de nivel superior como la interfaz de usuario primaria. El siguiente ejemplo muestra cómo desde Delphi se puede crear una barra de aplicación de Windows.

#### Listado II-4: Creando una barra de aplicación desde Delphi

```
unit AppBarMessageU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ShellAPI;

const
  {El identificador de mensaje de la barra, definido por la aplicación}
  WM_DELAPPBAR = WM_USER + 1;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    procedure FormActivate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
  private
    {Declaraciones privadas}
    procedure CreateParams(var Params: TCreateParams); override;
  public
    {Declaraciones públicas}
    {El manejador de mensajes de la barra}
    procedure WMDe1AppBar(var Msg: TMessage); message WM_DELAPPBAR;
  end;

var
  Form1: TForm1;
  {El registro TAppBarData debe ser global a la unit}
  AppBarInfo: TAppBarData;

implementation

{$R *.DFM}

{Sobreescribimos CreateParams para asegurarnos que se usen los estilos apropiados}
procedure TForm1.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);
  {La appbar tiene que ser una ventana de herramientas emergente para funcionar
  adecuadamente}
```

```

    Params.ExStyle := WS_EX_TOOLWINDOW;
    Params.Style := WS_POPUP or WS_CLIPCHILDREN;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    {Provee el registro TAppBarData con el manejador de la ventana de la barra}
    AppBarInfo.hWnd := Form1.Handle;

    {Registra la nueva barra}
    SHAppBarMessage(ABM_NEW, AppBarInfo);

    {Le pide al sistema una posición aprobada}
    SHAppBarMessage(ABM_QUERYPOS, AppBarInfo);

    {Ajusta la nueva posición a la altura de la ventana de la barra}
    AppBarInfo.rc.Bottom := AppBarInfo.rc.Top + 50;

    {Informa al sistema de la nueva posición de la barra}
    SHAppBarMessage(ABM_SETPOS, AppBarInfo);

    {Mueve físicamente la ventana de la barra a su posición}
    MoveWindow(AppBarInfo.hWnd, AppBarInfo.rc.Left, AppBarInfo.rc.Top,
        AppBarInfo.rc.Right - AppBarInfo.rc.Left,
        AppBarInfo.rc.Bottom - AppBarInfo.rc.Top, TRUE);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Rellena el registro TAppBarData con la información necesaria}
    AppBarInfo.cbSize := SizeOf(TAppBarData);
    AppBarInfo.hWnd := Form1.Handle;
    AppBarInfo.lParam := 0;

    {Desregistra la barra}
    SHAppBarMessage(ABM_REMOVE, AppBarInfo);
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
    {Activa la barra}
    SHAppBarMessage(ABM_ACTIVATE, AppBarInfo);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    Loop: Integer;
begin
    {Rellena la barra con un gradiente de amarillo a rojo}
    for Loop := 0 to (Width div 20) do
        begin
            Canvas.Brush.Color := RGB(255, 255 - ((255 div 20)*Loop), 0);
            Canvas.Brush.Style := bsSolid;
            Canvas.FillRect(Rect((Width div 20)*Loop, 0, ((Width div 20)*Loop) +
                (Width div 20), Height));
        end
    end

```

```

end;

{Dibuja una barra de título en la barra}
Canvas.Font.Name := 'Arial';
Canvas.Font.Size := 20;
Canvas.Font.Color := clBlue;
Canvas.Font.Style := [fsBold];
Canvas.Brush.Style := bsClear;
Canvas.TextOut(10, 10, 'Delphi App Bar');
end;

{Este manejador de mensajes es llamado cuando se produce un evento que puede
afectar a la barra}
procedure TForm1.WMDe1AppBar(var Msg: TMessage);
begin
  {El parámetro wParam del mensaje contiene el identificador de la notificación}
  case Msg.wParam of
    ABN_FULLSCREENAPP: ShowMessage('FullScreenApp notification message
                                received. ');
    ABN_POSCHANGED:    ShowMessage('PosChanged notification message received. ');
    ABN_STATECHANGE:   ShowMessage('StateChange notification message
                                received. ');
    ABN_WINDOWARRANGE: ShowMessage('WindowArrange notification message
                                received. ');
  end;
end;

initialization
  {Inicializa el registro TAppBarData con la información necesaria}
  AppBarInfo.uEdge := ABE_TOP;
  AppBarInfo.rc := Rect(0, 0, GetSystemMetrics(SM_CXSCREEN), 50);
  AppBarInfo.cbSize := SizeOf(TAppBarData);
  AppBarInfo.uCallbackMessage := WM_DELAPPBAR;
end.

```

Figura 11-3:  
La barra de  
aplicación de  
Delphi en  
acción



## Funciones del Shell

En este capítulo se describen las siguientes funciones del *shell*:

Tabla II-I: Funciones del shell

Función	Descripción
DragAcceptFiles	Registra una ventana como destino de ficheros arrastrados.
DragFinish	Completa el proceso de arrastrar y soltar.
DragQueryFile	Recupera información sobre un fichero soltado.
DragQueryPoint	Recupera las coordenadas del ratón en el momento en que un fichero es soltado.
FindExecutable	Recupera el nombre del fichero ejecutable asociado con un fichero especificado.
SHAddToRecentDocs	Añade o elimina un tipo de documento registrado en el elemento de menú 'Documentos' bajo el botón de Inicio.
SHAppBarMessage	Registra y controla una barra de aplicaciones.
SHBrowseForFolder	Crea un cuadro de diálogo que permite al usuario seleccionar una carpeta del shell.
ShellAbout	Muestra el cuadro de diálogo 'Acerca de....' del shell.
ShellExecute	Lanza un fichero ejecutable.
ShellExecuteEx	Lanza un fichero ejecutable. Esta función ofrece más opciones que ShellExecute.
Shell_NotifyIcon	Registra un icono de notificación en la bandeja de iconos.
SHFileOperation	Copia, mueve, renombra o borra un fichero.
SHFreeNameMappings	Libera un objeto de mapa de nombres.
SHGetFileInfo	Recupera información sobre el fichero especificado.
SHGetPathFromIDList	Recupera un nombre de ruta a partir de una lista de identificadores de elementos.
SHGetSpecialFolderLocation	Recupera la ubicación de carpetas únicas.

**DragAcceptFiles****ShellAPI.Pas****Sintaxis**

```

DragAcceptFiles(
    Wnd: HWND;           {manejador de ventana}
    Accept: BOOL          {indicador de aceptación}
);                       {esta función no devuelve nada}

```

**Descripción**

Este procedimiento registra una ventana para aceptar o rechazar ficheros soltados sobre ella. Si una aplicación registra una ventana para aceptar ficheros soltados, recibirá un mensaje *WM\_DROPFILES* cuando uno o más ficheros sean arrastrados y dejados caer sobre la ventana.

**Parámetros**

*Wnd*: Manejador de la ventana que aceptará o rechazará los ficheros soltados sobre ella.

*Accept*: Un valor booleano que determina si la ventana aceptará o rechazará los ficheros soltados sobre ella. Un valor TRUE registra la ventana para aceptar los ficheros soltados sobre ella; un valor FALSE rechazará los ficheros dejados caer.

Véase además

*DragFinish*, *DragQueryFile*, *DragQueryPoint*, *WM\_DROPFILES*

### Ejemplo

#### Listado II-5: Recuperando información sobre ficheros soltados

Observe que este ejemplo requiere que esta línea sea añadida a la sección pública de la definición de la clase del formulario:

```
procedure WMDropFiles(var DropFileMsg: TWMDropFiles); message WM_DROPFILES;
```

Cuando un fichero es dejado caer sobre el formulario, Windows envía un mensaje *WM\_DROPFILES* al formulario. Esta línea declara un procedimiento que manejará este mensaje:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    {Esto registra la ventana para aceptar ficheros}
    DragAcceptFiles(Handle, TRUE);
end;

procedure TForm1.WMDropFiles(var DropFileMsg: TWMDropFiles);
var
    FileCount: Integer;           // la cantidad de ficheros soltados
    TheFileName: array[0..500] of Char; // nombre de fichero
    DropPoint: TPoint;           // coordenadas del punto donde es soltado
    LoopCount: Integer;         // una variable de control de bucle
begin
    {Borra el cuadro de lista que muestra información sobre el fichero}
    ListBox1.Items.Clear;

    {Obtiene el número de ficheros que se soltaron y lo muestra}
    FileCount := DragQueryFile(DropFileMsg.Drop, $FFFFFFFF, nil, 0);
    ListBox1.Items.Add('Number of files dropped: ' + IntToStr(FileCount));
    ListBox1.Items.Add('');

    {Obtiene las coordenadas relativas a la ventana dónde los ficheros fueron soltados}
    DragQueryPoint(DropFileMsg.Drop, DropPoint);
    ListBox1.Items.Add('Mouse Drop Point: ' + IntToStr(DropPoint.X) + ', ' +
        IntToStr(DropPoint.Y));
    ListBox1.Items.Add('');
    ListBox1.Items.Add('-----');
    ListBox1.Items.Add('');

    {Recupera la ruta completa y el nombre de cada fichero que fue dejado caer}
    for LoopCount:=0 to FileCount-1 do
    begin
```

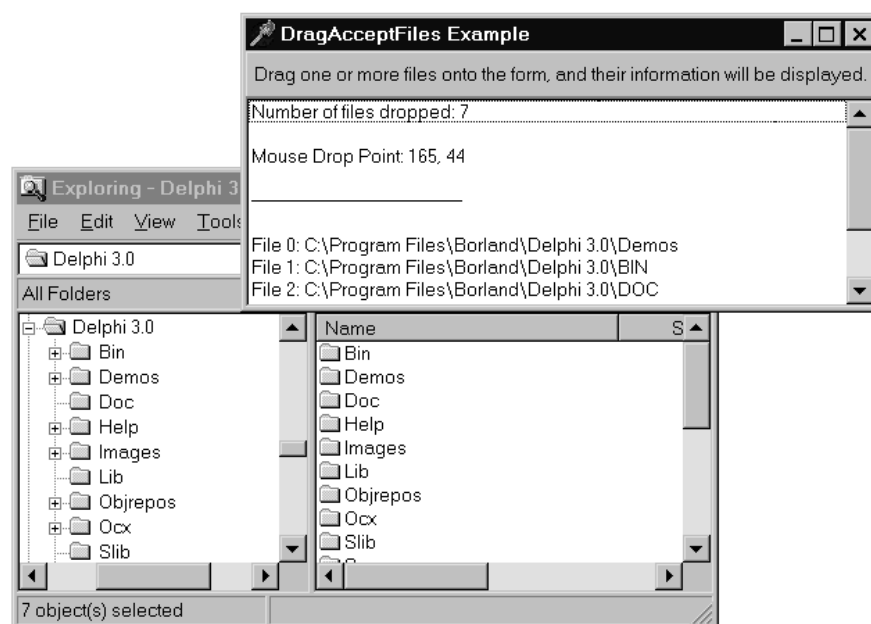
```

DragQueryFile(DropFileMsg.Drop, LoopCount, TheFileName, 500);
ListBox1.Items.Add('File ' + IntToStr(LoopCount) + ': ' +
                  string(TheFileName));
end;

{Libera la memoria reservada para el registro de datos de ficheros soltados}
DragFinish(DropFileMsg.Drop);
end;

```

Figura 11-4:  
Información  
sobre el  
fichero dejado  
caer



## DragFinish

## ShellAPI.Pas

### Sintaxis

```

DragFinish(
    Drop: HDROP           {manejador del registro de datos sobre ficheros soltados}
);                       {esta función no devuelve nada}

```

### Descripción

Este procedimiento libera memoria que Windows reservó para el registro de información sobre ficheros soltados.

### Parámetros

*Drop*: Manejador del registro de datos sobre ficheros soltados. Este manejador es pasado en el campo *wParam* del mensaje *WM\_DROPFILES*. Este parámetro es

también accesible desde Delphi, a través del campo *Drop* del registro de mensajes *TWMDropFiles* pasado a la rutina que maneja el mensaje *WM\_DROPFILES*.

Véase además

*DragAcceptFiles*, *WM\_DROPFILES*

### Ejemplo

Vea el Listado 11-5 bajo *DragAcceptFiles*.

## DragQueryFile ShellAPI.Pas

### Sintaxis

```
DragQueryFile(
    Drop: HDROP           {manejador del registro de datos sobre ficheros soltados}
    FileIndex: UINT;      {el índice de un nombre de fichero}
    FileName: PChar;      {puntero a un buffer para un nombre de fichero}
    cb: UINT              {el tamaño del buffer para el nombre de fichero}
): UINT;                {devuelve un entero sin signo}
```

### Descripción

Esta función recupera el nombre de un fichero soltado. El parámetro *FileIndex* indica la posición del fichero deseado en la lista de ficheros soltados que está almacenada en el registro indicado por el parámetro *Drop*. La ruta completa y el nombre del fichero soltado son copiados en el *buffer* al que apunta el parámetro *FileName*.

### Parámetros

*Drop*: Manejador del registro de datos sobre ficheros soltados. Este manejador es pasado en el campo *wParam* del mensaje *WM\_DROPFILES*. Este parámetro es también accesible desde Delphi, a través del campo *Drop* del registro de mensajes *TWMDropFiles* pasado a la rutina que maneja el mensaje *WM\_DROPFILES*.

*FileIndex*: Identifica el índice del fichero soltado deseado. Si este parámetro es \$FFFFFFFF, esta función devuelve el número total de ficheros soltados. El *array* de ficheros tiene base cero, y un valor entre cero y el número total de ficheros soltados menos uno, hará que la ruta completa y el nombre del fichero correspondiente sean copiados al *buffer* al que apunta el parámetro *FileName*.

*FileName*: Puntero a un *buffer* de cadena de caracteres que recibirá el nombre de fichero del fichero soltado. Si este parámetro es **nil**, esta función devuelve el tamaño necesario para el *buffer*, en caracteres.

*cb*: Especifica el tamaño del *buffer* al que apunta el parámetro *FileName*, en caracteres.

### Valor que devuelve

Si la función tiene éxito, el valor devuelto depende de los valores pasados en los parámetros. Si el valor del parámetro *FileIndex* es \$FFFFFFFF, la función devuelve el

número total de ficheros soltados. Si el parámetro *FileIndex* está entre cero y el número total de ficheros dejados caer menos uno y el valor del parámetro *FileName* es **nil**, la función devuelve el tamaño del *buffer* necesario para almacenar la ruta y el nombre del fichero correspondiente, en caracteres (sin incluir el terminador nulo). Si la función copia un nombre de fichero al *buffer* al que apunta el parámetro *FileName*, devuelve la cantidad de caracteres copiados, excluyendo el terminador nulo. Si la función falla, devuelve cero.

Véase además

*DragAcceptFiles*, *DragQueryPoint*, *WM\_DROPFILES*

### Ejemplo

Vea el Listado 11-5 bajo *DragAcceptFiles*.

## DragQueryPoint

## ShellAPI.Pas

### Sintaxis

```
DragQueryPoint(
  Drop: HDROP           {manejador del registro de datos sobre ficheros soltados}
  var Point: TPoint     {puntero a registro que recibirá las coordenadas}
): BOOL;               {devuelve TRUE o FALSE}
```

### Descripción

Esta función rellena un registro de tipo *TPoint* con las coordenadas del cursor del ratón en el momento en que los ficheros fueron soltados sobre la ventana.

### Parámetros

*Drop*: Manejador del registro de datos sobre ficheros soltados. Este manejador es pasado en el campo *wParam* del mensaje *WM\_DROPFILES*. Este parámetro es también accesible desde Delphi, a través del campo *Drop* del registro de mensajes *TWMDropFiles* pasado a la rutina que maneja el mensaje *WM\_DROPFILES*.

*Point*: Puntero a un registro *TPoint* que será rellenado con las coordenadas *X* e *Y* del cursor del ratón cuando los ficheros fueron soltados. Estas coordenadas son relativas a la ventana sobre la que se han soltado ficheros.

### Valor que devuelve

Si la función tiene éxito y el punto en que los ficheros son soltados está dentro del área cliente de la ventana, devuelve TRUE. Si la función falla, o el punto en que los ficheros son soltados no está dentro del área cliente, devuelve FALSE.

Véase además

*DragAcceptFiles*, *DragQueryFile*, *WM\_DROPFILES*

**Ejemplo**

Vea el Listado 11-5 bajo *DragAcceptFiles*.

**FindExecutable****ShellAPI.Pas****Sintaxis**

```
FindExecutable(
  FileName: PChar;           {puntero a nombre de fichero}
  Directory: PChar;          {puntero a directorio por defecto}
  Result: PChar              {puntero a un buffer que recibe un nombre de fichero}
): HINST;                   {devuelve un entero}
```

**Descripción**

Esta función recupera la ruta y el nombre del fichero ejecutable asociados con el nombre de fichero pasado en el parámetro *FileName*.

**Parámetros**

*FileName*: Puntero a una cadena de caracteres terminada en nulo que contiene un nombre de fichero. Este parámetro puede especificar un documento o un fichero ejecutable. Si este parámetro contiene un nombre de fichero ejecutable, el parámetro *Result* contendrá una copia exacta de este parámetro cuando la función retorne.

*Directory*: Puntero a una cadena de caracteres terminada en nulo que especifica una ruta a usar como directorio por defecto.

*Result*: Puntero a un *buffer* que recibe una cadena de caracteres terminada en nulo que identifica al fichero ejecutable asociado con el fichero o documento indicado por el parámetro *FileName*. Este fichero ejecutable es lanzado cada vez que una acción “abrir” es ejecutada sobre el fichero especificado por el parámetro *FileName*, ya sea haciendo clic con el botón derecho sobre el fichero y seleccionando **Abrir** en el Explorador de Windows o usando la función *ShellExecute*. El Registro de Windows almacena qué fichero ejecutable está asociado con tipos específicos de ficheros. La función *FindExecutable* busca inicialmente en el Registro bajo la clave

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes\<tipo de fichero>*.

Bajo esa clave se almacena el nombre de otra clave. *FindExecutable* toma este valor y busca en la clave

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes\<clave>\Shell\Open\Command*.

El valor situado en este lugar contiene la ruta y nombre de fichero completos del ejecutable asociado al tipo de documento. Por ejemplo, si el parámetro *FileName* especifica un fichero con la extensión .PAS, *FindExecutable* busca primero en

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes\pas*.

Bajo esta clave se almacena el valor “*DelphiUnit*”. *FindExecutable* toma este valor y busca en la entrada:

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes\DelphiUnit\Shell\Open\Command.*

Si Delphi ha sido instalado en el directorio por defecto, el valor hallado en este lugar será “C:\Archivos de programa\Borland\Delphi5\Bin\Delphi32.EXE /np”. Observe que si la ruta y el nombre del ejecutable almacenados en la entrada del registro contienen espacios, como en este ejemplo, y el valor no está encerrado entre comillas, la función *FindExecutable* reemplazará cualquier espacio con un carácter nulo.

#### Valor que devuelve

Si la función tiene éxito, devuelve un valor mayor que 32. Si la función falla, devuelve un valor de la Tabla 11-2.

#### Véase además

*ShellExecute, ShellExecuteEx*

#### Ejemplo

##### Listado 11-6: Encontrando un fichero ejecutable y abriendo documentos

```

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    {Selecciona una nueva máscara de edición basada en el contenido de Edit1}
    if ((Key = Chr(13)) AND (Edit1.Text <> '')) then
        begin
            FileListBox1.Mask := '*' + Edit1.Text;
            {Evita el "beep" del altavoz interno}
            Key := #0;
        end;
    end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Lanza el fichero ejecutable hallado por FindExecutable}
    if Label1.Caption <> '' then
        ShellExecute(Form1.Handle, 'open', PChar(Label1.Caption),
            nil, nil, SW_SHOWNORMAL);
    end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    {Abre el fichero seleccionado, ejecutando su aplicación asociada}
    ShellExecute(Form1.Handle, 'open', PChar(FileListBox1.FileName),
        nil, nil, SW_SHOWNORMAL);
    end;

procedure TForm1.FileListBox1Click(Sender: TObject);
var
    Buffer: array[0..500] of Char;    // un buffer para ruta y nombre de fichero
begin
    {Encuentra el ejecutable asociado con el fichero seleccionado}
    FindExecutable(PChar(FileListBox1.FileName), nil, @Buffer);

```

```
{Si el ejecutable no es hallado...}
if StrLen(Buffer) < 1 then
begin
  {...muestra un mensaje y deshabilita los botones de lanzamiento...}
  Label1.Caption := 'No Associated executable';
  Button1.Enabled := FALSE;
  Button2.Enabled := FALSE;
end
else
begin
  {...en caso contrario, muestra la ruta y nombre de fichero del ejecutable}
  Label1.Caption := string(Buffer);
  Button1.Enabled := TRUE;
  Button2.Enabled := TRUE;
end;
end;
```

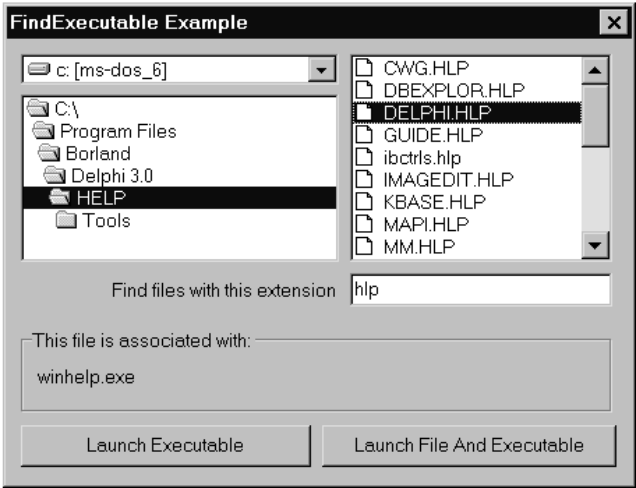


Figura 11-5:  
Se encontró  
un fichero  
ejecutable

Tabla II-2: Códigos de error devueltos por FindExecutable

Valor	Descripción
0	No hay suficiente memoria o recursos.
ERROR_GEN_FAILURE	El fichero especificado no tiene un fichero ejecutable asociado.
ERROR_FILE_NOT_FOUND	El fichero especificado por el parámetro FileName no pudo ser hallado.
ERROR_PATH_NOT_FOUND	La ruta del directorio por defecto especificada por el parámetro Directory no pudo ser hallada.
ERROR_BAD_FORMAT	El fichero ejecutable asociado no es válido o está corrupto.

**SHAddToRecentDocs**      **ShlObj.Pas****Sintaxis**

```
SHAddToRecentDocs(
  uFlags: UINT;           {valor que indica el contenido del parámetro pv}
  pv: Pointer             {puntero a un buffer o a una lista de identificadores}
);                        {esta función no devuelve nada}
```

**Descripción**

Esta función añade o quita ficheros de la lista de documentos recientes. A esta lista se accede a través del elemento de menú **Documentos** que aparece al pulsar el botón de **Inicio**. A esta lista sólo se pueden añadir documentos registrados (aquellos que tienen un fichero ejecutable asociado).

**Parámetros**

*uFlags*: Un valor que indica qué contiene el parámetro *pv*. Este parámetro puede contener un valor de la Tabla 11-3.

*pv*: Puede ser un puntero a una cadena de caracteres terminada en nulo que contiene la ruta y nombre de fichero de un documento, o un puntero a una lista de identificadores de elementos, que identifica el documento de forma única. Si este parámetro es **nil**, la lista de documentos recientes es vaciada.

**Véase además**

*SHGetFileInfo*

**Ejemplo****Listado 11-7: Añadiendo un documento a la lista de documentos recientes**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  {Borra todos los documentos de la lista de documentos recientes}
  SHAddToRecentDocs(SHARD_PATH, nil);
end;

procedure TForm1.FileListBox1DbClick(Sender: TObject);
var
  TheFileName: string;
begin
  {Recupera el nombre de fichero del documento seleccionado}
  TheFileName := FileListBox1.FileName;

  {Lo añade a la lista de documentos recientes. Observe que el fichero tiene que
  ser de un tipo registrado (debe tener asociado un ejecutable), para añadirlo
  a la lista}
  SHAddToRecentDocs(SHARD_PATH, PChar(TheFileName));
end;
```

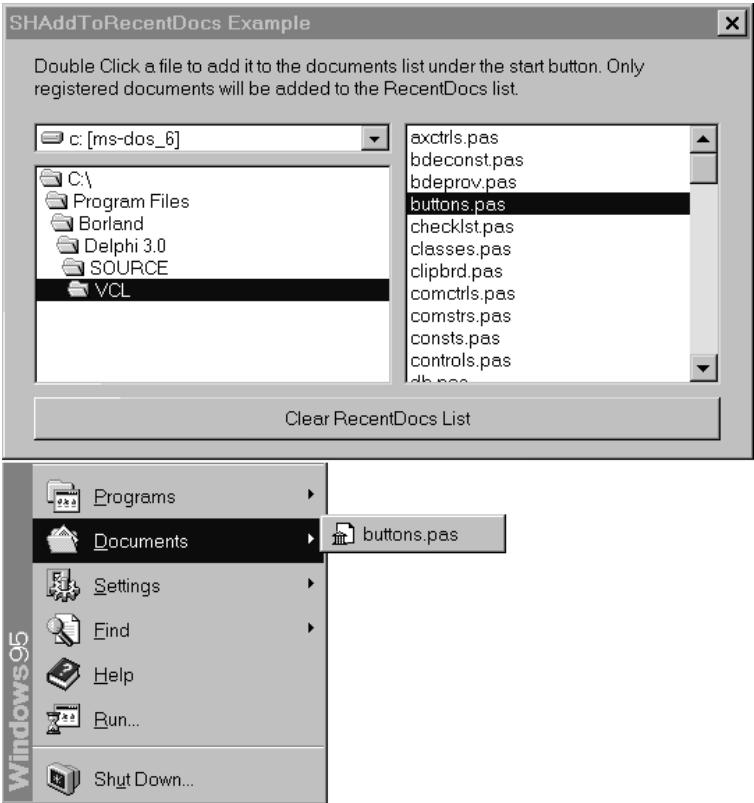


Figura 11-6:  
Un documento fue  
añadido a la  
lista

Tabla 11-3: Valores del parámetro uFlags de SHAddToRecentDocs

Valor	Descripción
SHARD_PATH	El parámetro pv contiene la dirección de una cadena de caracteres terminada en nulo con la ruta y el nombre de fichero de un documento.
SHARD_PIDL	El parámetro pv contiene la dirección de una lista de identificadores de elementos que identifica de manera única el documento.

**SHAppBarMessage      ShellAPI.Pas**

*Sintaxis*

```
SHAppBarMessage(  
  dwMessage: DWORD;           {mensaje de la barra de aplicación}  
  var pData: TAppBarData      {puntero a un registro TAppBarData}  
): UINT;                      {devuelve un valor dependiente del mensaje}
```

**Descripción**

Esta función crea una barra de aplicación. Una *barra de aplicación* es una ventana que está asociada a un borde particular de la pantalla y reserva ese espacio para su propio uso. Windows evita que otras ventanas utilicen este espacio, moviéndolas si es necesario. Observe que la ventana de una barra de aplicación tiene que usar los estilos *WS\_EX\_TOOLWINDOW* y *WS\_POPUP* para funcionar correctamente.

**Parámetros**

*dwMessage*: El identificador del mensaje de la barra de aplicación. Este parámetro puede tomar un valor de la Tabla 11-4.

*pData*: Puntero a un registro *TAppBarData*. Este registro ofrece información a la función *SHAppBarMessage* y recibe información como resultado de esa llamada. El registro *TAppBarData* se define de la siguiente manera:

**TAppBarData = record**

<i>cbSize</i> : DWORD;	{tamaño del registro TAppBarData}
<i>hWnd</i> : HWND;	{manejador de ventana}
<i>uCallbackMessage</i> : UINT;	{id. de mensaje definido por la aplicación}
<i>uEdge</i> : UINT;	{indicador de borde de pantalla}
<i>rc</i> : TRect;	{rectángulo en coordenadas de pantalla}
<i>lParam</i> : LPARAM;	{valor dependiente del mensaje}

**end;**

*cbSize*: El tamaño del registro *TAppBarData*, en bytes. A este campo se le debe asignar *SizeOf(TAppBarData)*.

*hWnd*: Manejador de la ventana de la barra de aplicación.

*uCallbackMessage*: Un identificador de mensaje definido por la aplicación. Este mensaje es enviado a la ventana identificada por el parámetro *hWnd* para notificarle eventos. El parámetro *wParam* de este mensaje contendrá un mensaje de notificación de la Tabla 11-5.

*uEdge*: Un valor que indica qué borde de la pantalla está asociado con la barra de aplicación. Este campo puede tomar un valor de la Tabla 11-6.

*rc*: Un registro *TRect* que almacena las coordenadas de la ventana de la barra de aplicación. Estas coordenadas se expresan en términos de la pantalla.

*lParam*: Un valor dependiente del mensaje. Consulte la Tabla 11-5 para conocer el uso de este campo en la tabla.

**Valor que devuelve**

Si la función tiene éxito, devuelve un valor específico al mensaje; en caso contrario, devuelve cero. Consulte la Tabla 11-5 para ver una descripción de los posibles valores devueltos.

**Véase además**

*CreateWindow*, *CreateWindowEx*, *MoveWindow*

Ejemplo

Listado II-8: Recuperando las coordenadas de la barra de tareas de Windows

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AppBarInfo: TAppBarData; // almacena la información de la barra
begin
  {Inicializa el registro de la barra con la información necesaria}
  AppBarInfo.cbSize := SizeOf(TAppBarData);
  AppBarInfo.hWnd := Form1.Handle;

  {Recupera las coordenadas de la barra de tareas de Windows...}
  SHAppBarMessage(ABM_GETTASKBARPOS, AppBarInfo);

  {...y las muestra}
  Button1.Caption := 'Left: ' + IntToStr(AppBarInfo.rc.Left) +
    ' Top: ' + IntToStr(AppBarInfo.rc.Top) +
    ' Right: ' + IntToStr(AppBarInfo.rc.Right) +
    ' Bottom: ' + IntToStr(AppBarInfo.rc.Bottom);
end;
```

Figura 11-7:  
Las  
coordenadas  
de la barra de  
tareas



Tabla II-4: Valores del parámetro dwMessage de SHAppBarMessage

Valor	Descripción
ABM_ACTIVATE	Notifica a Windows que una barra de aplicación ha sido activada. La barra de aplicación debe llamar a este mensaje cuando reciba un mensaje WM_ACTIVATE. Los campos cbSize y hWnd del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Este mensaje es ignorado si el parámetro hWnd identifica una barra de aplicación que se oculta automáticamente, para la que el sistema asigna automáticamente su orden Z. La función siempre devolverá un valor mayor que cero cuando utilice este mensaje.

Valor	Descripción
ABM_GETAUTOHIDEBAR	Recupera el manejador de ventana de la barra de aplicación que se oculta automáticamente asociada con el borde especificado de la pantalla. Los campos cbSize, hWnd, y uEdge del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Si la función tiene éxito, devuelve un manejador de la ventana de la barra de aplicación asociada con el borde especificado de la pantalla. Si la función falla o no hay una barra de aplicación asociada con el borde especificado de la pantalla, la función devuelve cero.
ABM_GETSTATE	Recupera los estados de ocultamiento automático y siempre-visible de la barra de tareas de Windows. Los campos cbSize y hWnd del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Si la función tiene éxito, devuelve ABS_ALWAYSONTOP, una constante que indica que la barra de tareas está en el estado siempre-visible o ABS_AUTOHIDE, una constante que indica que la barra de tareas está en estado de ocultamiento automático. La función puede devolver ambos valores si es necesario. Si la función falla o la barra de tareas de Windows no está en ninguno de los dos estados, devuelve cero.
ABM_GETTASKBARPOS	Recupera las coordenadas del rectángulo límite de la barra de tareas de Windows. Los campos cbSize y hWnd del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Si la función tiene éxito, devuelve un valor mayor que cero y el campo rc contendrá el rectángulo límite de la barra de tareas de Windows, en coordenadas de la pantalla. Si la función falla, devuelve cero.
ABM_NEW	Registra una nueva barra de aplicación en el sistema. La función especifica el identificador de mensaje definido por la aplicación que será utilizado para enviar los mensajes de notificación de la barra de aplicación. Este mensaje debe ser llamado antes que cualquier otro mensaje de la barra de aplicación. Para registrar una barra de aplicación que se oculte automáticamente, utilice el mensaje ABM_SETAUTOHIDEBAR. Los campos cbSize, hWnd y uCallbackMessage del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Si la función tiene éxito, devuelve un valor distinto de cero. Si la función falla o la barra de aplicación especificada ya está registrada, devuelve cero.

Valor	Descripción
ABM_QUERYPOS	Solicita un rectángulo límite y un borde de pantalla para la barra de aplicación. El sistema ajusta el rectángulo especificado de manera que la barra de aplicación no interfiera con la barra de tareas de Windows u otra barra de aplicación. La barra de aplicación debe enviar este mensaje antes del mensaje ABM_SETPOS. Los campos cbSize, hWnd, uEdge y rc del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Cuando la función retorna, el campo rc contiene las coordenadas ajustadas para la nueva posición de la barra de aplicación. Este mensaje hace que la función siempre devuelva un valor distinto de cero.
ABM_REMOVE	Desregistra una barra de aplicación del sistema. Los campos cbSize y hWnd del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. La función siempre devolverá un valor distinto de cero cuando utilice este mensaje. El mensaje de notificación ABN_POSCHANGED es enviado a todas las demás barras de aplicación una vez que este mensaje es procesado.
ABM_SETAUTOHIDEBAR	Registra o desregistra una barra de aplicación con ocultamiento automático. El sistema sólo permite una barra de aplicación con ocultamiento automático por borde de la pantalla. Al campo lParam del registro TAppBarData se le asigna un valor distinto de cero para registrar una barra de aplicación con ocultamiento automático, o cero para desregistrarla. Los campos cbSize, hWnd, uEdge y lParam del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Si la función tiene éxito, devuelve un valor distinto de cero. Si la función falla o ya está registrada una barra de aplicación con ocultamiento automático para el borde especificado, la función devuelve cero.

Valor	Descripción
ABM_SETPOS	Asigna un rectángulo límite y un borde de pantalla para la barra de aplicación. El sistema ajusta el rectángulo especificado de manera que la barra de aplicación no interfiera con la barra de tareas de Windows u otra barra de aplicación. Los campos cbSize, hWnd, uEdge y rc del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Cuando la función retorna, el campo rc contiene las coordenadas ajustadas para la nueva posición de la barra de aplicación. Este mensaje hace que la función siempre retorne un valor distinto de cero. El sistema envía un mensaje de notificación ABN_POSCHANGED a todas las demás barras de aplicación una vez que este mensaje es procesado.
ABM_WINDOWPOSCHANGED	Notifica al sistema que la posición de la barra de aplicación ha cambiado. La barra de aplicación debe enviar este mensaje cuando responda al mensaje WM_WINDOWPOSCHANGED. Los campos cbSize y hWnd del registro TAppBarData tienen que ser inicializados. Todos los demás campos son ignorados. Este mensaje hace que la función siempre devuelva un valor distinto de cero. Este mensaje es ignorado si el campo hWnd identifica una barra de aplicación con ocultamiento automático.

Tabla 11-5: Valores de wParam para los mensajes de notificación de SHAppBarMessage

Valor	Descripción
ABN_FULLSCREENAPP	Notifica a una barra de aplicación cuando una aplicación a pantalla completa se está abriendo o cerrando. Cuando se inicia la ejecución de una aplicación a pantalla completa, la barra de aplicación tiene que pasar al final del orden Z. Cuando una aplicación a pantalla completa se cierra, la barra de aplicación puede restaurar su posición original en el orden Z. Si el parámetro lParam es distinto de cero, ello indica que la aplicación a pantalla completa está arrancando; si es cero, indica que se está cerrando.
ABN_POSCHANGED	Notifica a la barra de aplicación de un evento que puede afectar su tamaño y posición, como puede ser la adición, eliminación o cambio de tamaño de otra barra de aplicación, o el cambio de posición o estado de la barra de tareas de Windows. Al recibir este mensaje, la barra de aplicación debe enviar el mensaje ABM_QUERYPOS seguido del mensaje ABM_SETPOS para determinar si su posición ha cambiado. La función MoveWindow es entonces llamada para mover físicamente la ventana de la barra de aplicación a su nueva posición.

Valor	Descripción
ABN_STATECHANGE	Notifica a la barra de aplicación que el estado de ocultamiento automático o de siempre visible de la barra de tareas ha cambiado.
ABN_WINDOWARRANGE	Notifica a la barra de aplicación que el usuario ha seleccionado el comando Cascada, Mosaico Horizontal, o Mosaico Vertical desde el menú de contexto de la barra de tareas de Windows. Si el parámetro IParam es un valor distinto de cero, entonces la ejecución del comando de organización de ventanas ha comenzado y ninguna ventana ha sido aún movida. Un valor cero indica que el comando de organización ha finalizado y que todas las ventanas están en sus posiciones finales. La barra de aplicación recibe este mensaje dos veces, una antes de que la operación comience y otra cuando ésta ya ha terminado.

Tabla II-6: Valores del campo pData.uEdge de SHAppBarMessage

Valor	Descripción
ABE_BOTTOM	El borde inferior de la pantalla.
ABE_LEFT	El borde izquierdo de la pantalla.
ABE_RIGHT	El borde derecho de la pantalla.
ABE_TOP	El borde superior de la pantalla.

## SHBrowseForFolder      ShlObj.Pas

### Sintaxis

```
SHBrowseForFolder(
    var lpbi: TBrowseInfo           {puntero a registro TBrowseInfo}
): PItemIDList;                   {devuelve un puntero a una lista de identificadores
                                   de elementos}
```

### Descripción

Esta función muestra una caja de diálogo que permite al usuario seleccionar una carpeta del *shell*.

### Parámetros

*lpbi*: Puntero a un registro *TBrowseInfo*. Este registro almacena información que es usada para mostrar el cuadro de diálogo de selección de carpeta, y recibe información del cuadro de diálogo que indica la selección del usuario. Este registro se define como:

```
TBrowseInfo = packed record
    hwndOwner: HWND;           {manejador de ventana}
    pidlRoot: PItemIDList;     {puntero a lista de id. de elementos}
```

pszDisplayName: PAnsiChar;	{puntero a cadena de caracteres}
lpzTitle: PAnsiChar;	{puntero a cadena de caracteres}
ulFlags: UINT;	{opciones de control}
lpfn: TFNBFFCallback;	{dirección de función de respuesta}
lParam: LPARAM;	{valor definido por la aplicación}
iImage: Integer;	{índice de imagen de la lista de imágenes del sistema}

**end;**

*hwndOwner*: Manejador de la ventana propietaria del cuadro de diálogo.

*pidlRoot*: Puntero a una lista de identificadores de elementos que especifica la carpeta raíz a partir de la que el usuario comienza la exploración. Si este campo es **nil**, la raíz del espacio de nombres es usada como punto de partida.

*pszDisplayName*: Puntero a un *buffer* que recibe una cadena de caracteres terminada en nulo que contiene el nombre de la carpeta seleccionada. El tamaño de este *buffer* se asume que es *MAX\_PATH* bytes.

*lpzTitle*: Puntero a una cadena de caracteres terminada en nulo que contiene el texto a mostrar en la barra de título del cuadro de diálogo.

*ulFlags*: Combinación de valores que especifica los tipos de carpetas a mostrar y otras opciones. Este campo puede tomar uno o más valores de la Tabla 11-7.

*lpfn*: Puntero a una función de respuesta. Esta función es llamada siempre que una acción del usuario genere un evento en el cuadro de diálogo, tal como seleccionar una carpeta. A este campo se le puede asignar **nil**. La sintaxis de la función de respuesta se describe más adelante.

*lParam*: Un valor definido por la aplicación que es pasado a la función de respuesta, si hay una definida.

*iImage*: Recibe el índice en la lista de imágenes del sistema de la imagen que representa la carpeta seleccionada.

### Valor que devuelve

Si la función tiene éxito, devuelve un puntero a una lista de identificadores de elementos que identifica únicamente a la carpeta seleccionada. La ubicación de la carpeta es relativa a la raíz del espacio de nombres del *shell*. Si la función falla o el usuario pulsa el botón **Cancelar**, la función devuelve **nil**.

### Sintaxis de la función de respuesta

BrowseCallbackProc(	
hWnd: HWND;	{manejador de la ventana del cuadro de diálogo}
uMsg: UINT;	{mensaje de evento del cuadro de diálogo}
lParam: LPARAM;	{valor específico del mensaje}
lpData: LPARAM	{valor definido por la aplicación}
): Integer;	{devuelve un valor entero}

**Descripción**

La función de respuesta se ejecuta cada vez que el usuario provoca que un evento tenga lugar en el cuadro de diálogo de exploración de carpetas. Esta función de respuesta puede ejecutar cualquier tarea que se estime necesaria.

**Parámetros**

*hWnd*: Manejador de la ventana del cuadro de diálogo. La función de respuesta puede usar este parámetro para enviar un mensaje especial a la ventana del cuadro de diálogo. Los mensajes disponibles se listan en la Tabla 11-8.

*uMsg*: Un valor que indica el tipo de evento que ha ocurrido. Este parámetro puede tomar cualquier valor de la Tabla 11-9.

*lParam*: Un valor específico del mensaje, dependiente del parámetro *uMsg*.

*lpData*: El valor definido por la aplicación que fue pasado en el campo *lParam* del registro *TBrowseInfo*.

**Valor que devuelve**

La función de respuesta debe devolver siempre cero.

**Véase además**

*FindExecutable*, *ShellExecute*, *ShellExecuteEx*, *SHFileOperation*

**Ejemplo****Listado 11-9: Seleccionando una carpeta**

{Función de respuesta usada por el cuadro de diálogo de exploración de carpetas. Observe la directiva `export`.}

```
function BrowseCallback(hWnd: HWND; uMsg: UINT; lParam: LPARAM;
                        lpData: LPARAM): Integer; stdcall; export;
```

```
var
    Form1: TForm1;
```

**implementation**

```
{ $R *.DFM }
```

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
    IDList: PItemIDList;           // lista de ids. de elementos
    BrowseInfo: TBrowseInfo;       // registro de información
    PathName: array[0..MAX_PATH] of Char; // ruta
    DisplayName: array[0..MAX_PATH] of Char; // nombre del fichero a mostrar
```

```
begin
    {Inicializa el registro de exploración}
    BrowseInfo.hwndOwner := Form1.Handle;
    BrowseInfo.pidlRoot := nil;
    BrowseInfo.pszDisplayName := DisplayName;
    BrowseInfo.lpszTitle := 'Choose a file or folder';
```

```

BrowseInfo.ulFlags      := BIF_STATUSTEXT;           // muestra línea de estado
BrowseInfo.lpfncb       := @BrowseCallback;
BrowseInfo.lParam       := 0;

{Muestra cuadro de diálogo de Explorar carpetas}
IDList := SHBrowseForFolder(BrowseInfo);

{Recupera la ruta a partir de lista de identificadores de elementos devuelta}
SHGetPathFromIDList(IDList, @PathName);

{Muestra el nombre de la ruta y de la carpeta seleccionada}
Label2.Caption := PathName;
Label4.Caption := BrowseInfo.pszDisplayName;
end;

{Esta función de respuesta es llamada siempre que una acción tiene lugar en el
cuadro de diálogo de Explorar carpetas}
function BrowseCallback(hWnd: HWND; uMsg: UINT; lParam: LPARAM;
lpData: LPARAM): Integer;
var
  PathName: array[0..MAX_PATH] of Char; // almacena la ruta
begin
  {Si la selección del cuadro de diálogo de Explorar carpetas ha cambiado...}
  if uMsg = BFFM_SELCHANGED then
  begin
    {...recupera la ruta a partir de la lista de identificadores de elementos}
    SHGetPathFromIDList(PItemIDList(lParam), @PathName);

    {Muestra esta ruta en la línea de estado del cuadro de diálogo}
    SendMessage(hWnd, BFFM_SETSTATUSTEXT, 0, Longint(PChar(@PathName)));

    Result := 0;
  end;
end;
end;

```

Figura 11-8:  
El cuadro de  
diálogo de  
explorar  
carpetas

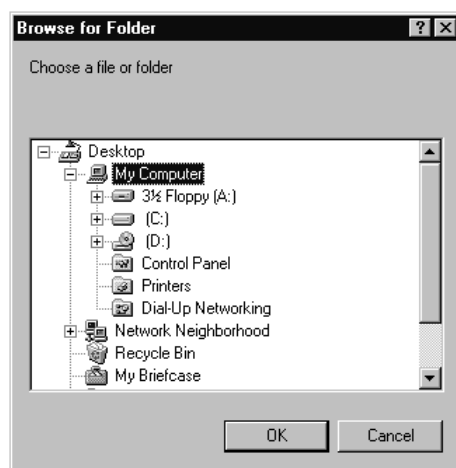


Tabla II-7: Valores del campo `lpbi.ulFlags` de `SHBrowseForFolder`

Valor	Descripción
<code>BIF_BROWSEFORCOMPUTER</code>	Permite al usuario seleccionar sólo ordenadores.
<code>BIF_BROWSEFORPRINTER</code>	Permite al usuario seleccionar sólo impresoras.
<code>BIF_DONTGOBELOWDOMAIN</code>	El cuadro de diálogo no contendrá carpetas de red bajo el nivel de dominio.
<code>BIF_RETURNFSANCESTORS</code>	Permite al usuario seleccionar sólo los ancestros del sistema de ficheros.
<code>BIF_RETURNONLYFSDIRS</code>	Permite al usuario seleccionar sólo los directorios del sistema de ficheros.
<code>BIF_STATUSTEXT</code>	Incluye una línea de estado en el cuadro de diálogo. La función de respuesta puede enviar un mensaje al cuadro de diálogo especificando qué mostrar en esa línea.

Tabla II-8: Mensajes del cuadro de diálogo para `BrowseCallbackProc`

Valor	Descripción
<code>BFFM_ENABLEOK</code>	Habilita el botón Aceptar si el parámetro <code>wParam</code> del mensaje contiene un valor distinto de cero. Si el parámetro <code>wParam</code> contiene cero, el botón Aceptar es deshabilitado.
<code>BFFM_SETSELECTION</code>	Selecciona una carpeta específica. Si el parámetro <code>wParam</code> del mensaje contiene <code>TRUE</code> , el parámetro <code>lParam</code> tiene que contener un puntero a una cadena que describe la ruta de la carpeta. Si el parámetro <code>wParam</code> es <code>FALSE</code> , el parámetro <code>lParam</code> tiene que contener un puntero a una lista de identificadores de elementos que especifica la carpeta seleccionada.
<code>BFFM_SETSTATUSTEXT</code>	Asigna el texto de la línea de estado del cuadro de diálogo. El parámetro <code>lParam</code> del mensaje tiene que contener un puntero a una cadena de caracteres terminada en nulo para la línea de estado. Este mensaje es válido sólo si la opción <code>BIF_STATUSTEXT</code> fue especificada en el campo <code>ulFlags</code> del registro <code>TBrowseInfo</code> .

Tabla II-9: Valores del parámetro `uMsg` de `BrowseCallbackProc`

Valor	Descripción
<code>BFFM_INITIALIZED</code>	El cuadro de diálogo de exploración de carpetas ha terminado su inicialización. El parámetro <code>lParam</code> contiene cero.
<code>BFFM_SELCHANGED</code>	El usuario ha seleccionado una carpeta. El parámetro <code>lParam</code> contiene un puntero a una lista de identificadores de elementos que especifica la carpeta elegida.

**ShellAbout****ShellAPI.Pas****Sintaxis**

ShellAbout(	
Wnd: HWND;	{manejador de la ventana madre}
szApp: PChar;	{puntero al texto de la barra de título}
szOtherStuff: PChar;	{puntero al texto descriptivo}
Icon: HICON	{manejador de un icono}
); Integer;	{devuelve un valor entero}

**Descripción**

Esta función muestra el cuadro de diálogo de **Acerca de...** del *shell*. Este es el cuadro de diálogo de identificación que es mostrado cuando se selecciona 'Acerca de Windows' en el Explorador de Windows. Este cuadro de diálogo muestra un icono y un texto que son específicos para Windows 95/98 o Windows NT.

**Parámetros**

*Wnd*: Manejador de la ventana madre. Si este parámetro es cero, el cuadro **Acerca de...** se comporta como un cuadro de diálogo no modal. Si se especifica un manejador, el cuadro de diálogo será modal.

*szApp*: Puntero al texto que se va a mostrar en la barra de título y a la primera línea del diálogo.

*szOtherStuff*: Puntero al texto que se va a mostrar después de la información de *copyright*.

*Icon*: Manejador del icono a mostrar en el cuadro de diálogo. Si este parámetro es cero, el diálogo mostrará el icono de Windows 95/98 ó Windows NT.

**Valor que devuelve**

Si la función tiene éxito, devuelve un valor distinto de cero; en caso contrario, devuelve cero.

**Véase además**

*GetSystemInfo*

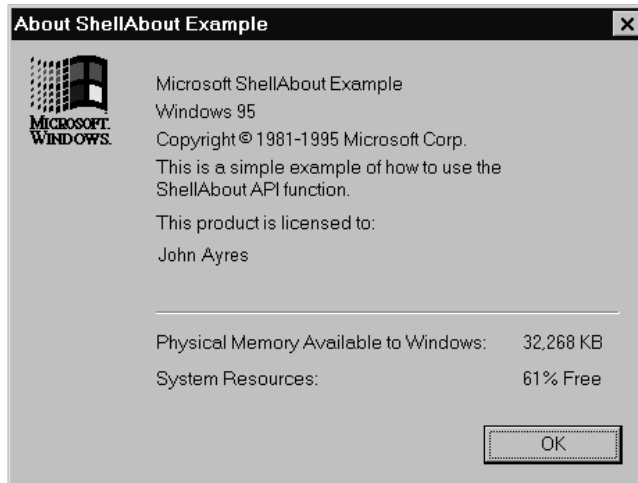
**Ejemplo****Listado II-10: Mostrando el cuadro de diálogo de ShellAbout**

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    {Lanza el diálogo 'Acerca de...' del shell de Microsoft}
    ShellAbout(Form1.Handle, 'ShellAbout Example',
        'This is a simple example of how to use the ShellAbout API function.', 0);
end;

```

Figura 11-9:  
La caja de  
diálogo del  
shell



## ShellExecute ShellAPI.Pas

### Sintaxis

```
ShellExecute(
  hWnd: HWND;           {manejador de la ventana madre}
  Operation: PChar;      {puntero a una cadena que describe la acción}
  FileName: PChar;       {puntero a un nombre de fichero o nombre de carpeta}
  Parameters: PChar;     {puntero a parámetros de un fichero ejecutable}
  Directory: PChar;      {puntero a nombre del directorio por defecto}
  ShowCmd: Integer      {modo de presentación}
): HINST;               {devuelve un manejador de instancia}
```

### Descripción

Esta función ejecuta la acción especificada sobre el fichero especificado y puede ser usada para imprimir un documento, lanzar un fichero ejecutable o abrir un carpeta de directorio.

### Parámetros

*hWnd*: Manejador de la ventana madre sobre la que se mostrarán los cuadros de mensajes en caso de error.

*Operation*: Puntero a una cadena de caracteres terminada en nulo que especifica la acción a ejecutar sobre el fichero o carpeta indicado por el parámetro *FileName*. La Tabla 11-10 lista las acciones estándar que pueden ser ejecutadas sobre un fichero o carpeta. Sin embargo, estas acciones no están limitadas a aquellas que se listan en la tabla. Este parámetro depende de las acciones registradas para el documento o la aplicación en el registro de Windows, y es posible crear nuevas acciones a través del menú ‘*Opciones de carpeta*’ del Explorador de Windows. Si este parámetro es **nil**, la operación “open” (abrir) es utilizada por defecto.

*FileName*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre de un documento, fichero ejecutable o carpeta.

*Parameters*: Si el parámetro *FileName* indica un fichero ejecutable, este parámetro contiene un puntero a una cadena de caracteres terminada en nulo que especifica los parámetros de línea de comandos a pasar al fichero ejecutable. Los parámetros deben estar separados por espacios. Si el parámetro *FileName* especifica un documento o carpeta, este parámetro debe ser **nil**.

*Directory*: Puntero a una cadena de caracteres terminada en nulo que contiene la ruta del directorio por defecto. Si el parámetro es **nil**, el directorio actual es usado como directorio de trabajo.

*ShowCmd*: Valor que determina cómo el fichero ejecutable indicado por el parámetro *FileName* va a ser será mostrado cuando sea lanzado. Este parámetro puede tomar un valor de la Tabla 11-11.

#### Valor que devuelve

Si la función tiene éxito, devuelve el manejador de instancia de la aplicación que fue lanzada o el manejador de un servidor de intercambio dinámico de datos. Si la función falla, devuelve un valor de la Tabla 11-12. Este valor será menor que 32.

#### Véase además

*FindExecutable*, *ShellExecuteEx*

#### Ejemplo

##### Listado II-II: Visualizando ficheros de texto

```
procedure TForm1.FileListBox1DbClick(Sender: TObject);
begin
    {Abre el fichero sobre el que se hizo doble clic en el cuadro de lista de
    ficheros. Si el fichero es demasiado grande para el Bloc de Notas, Windows
    preguntará si se desea lanzar Wordpad}
    ShellExecute(Form1.Handle, 'open', PChar(FileListBox1.FileName), nil, nil,
        SW_SHOWNORMAL);
end;
```

Tabla II-10: Valores del parámetro Operation de ShellExecute

Valor	Descripción
'open'	Abre el fichero o carpeta, o lanza el fichero ejecutable identificado por el parámetro <i>FileName</i> .
'print'	Imprime el documento identificado por el parámetro <i>FileName</i> . Si este parámetro identifica un fichero ejecutable, este es lanzado como si 'open' hubiera sido especificado.
'explore'	Abren una ventana del Explorador de Windows sobre la carpeta identificada por el parámetro <i>FileName</i> .

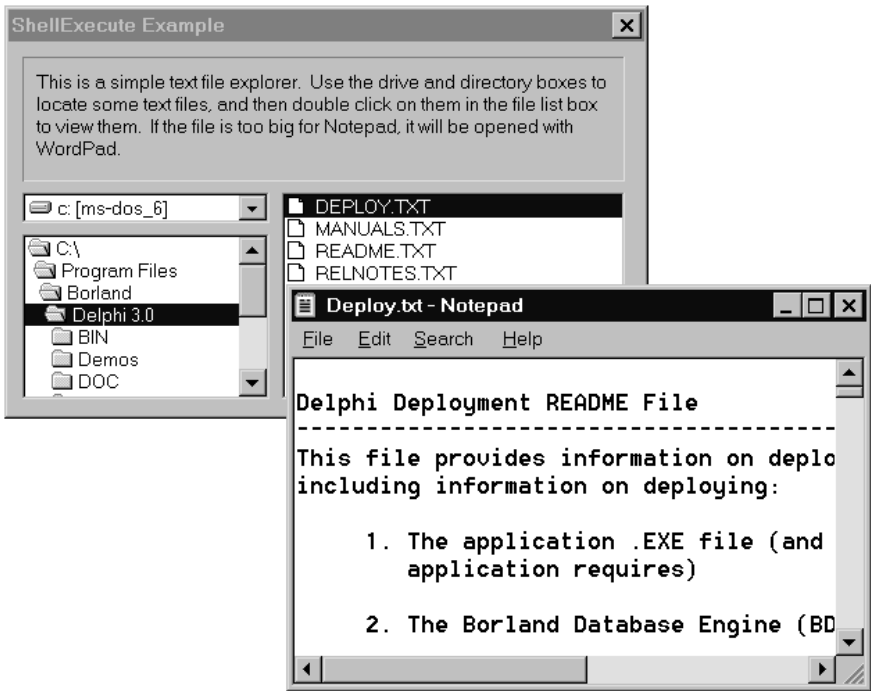


Figura 11-10:  
Visualizando  
un fichero de  
texto en el  
Bloc de Notas

Tabla II-II: Valores del parámetro ShowCmd de ShellExecute

Valor	Descripción
SW_HIDE	La ventana es ocultada y otra ventana es activada.
SW_MINIMIZE	La ventana es minimizada y la próxima ventana de nivel superior en el orden Z relativo es activada.
SW_RESTORE	La ventana es activada y mostrada en su tamaño y posición originales.
SW_SHOW	La ventana es activada y mostrada en su tamaño y posición actuales.
SW_SHOWDEFAULT	La ventana es mostrada según se indica en el campo wShowWindow del registro TStartupInfo pasado a la función CreateProcess cuando arrancó la aplicación. Esto es usado para seleccionar el estado de presentación inicial de la ventana principal de la aplicación. Esta opción debe ser usada cuando se muestra la ventana por primera vez, si la aplicación puede ser lanzada desde un acceso directo. Esta opción provocará que la ventana sea mostrada usando los atributos de ejecución indicados en las propiedades del acceso directo.
SW_SHOWMAXIMIZED	La ventana es activada y mostrada en estado maximizado.
SW_SHOWMINIMIZED	La ventana es activada y mostrada como un icono.

Valor	Descripción
SW_SHOWMINNOACTIVE	La ventana es mostrada como un icono. La ventana que estaba activa permanecerá activa.
SW_SHOWNORMAL	La ventana es mostrada en su estado actual. La ventana que estaba activa permanecerá activa.
SW_SHOWNOACTIVE	La ventana es mostrada en su estado más reciente. La ventana que estaba activa permanecerá activa.
SW_SHOWNORMAL	Equivalente a SW_RESTORE.

Tabla II-12: Códigos de error devueltos por ShellExecute

Valor	Descripción
0	No hay suficiente memoria o recursos.
ERROR_FILE_NOT_FOUND	El fichero especificado por el parámetro FileName no pudo ser encontrado.
ERROR_PATH_NOT_FOUND	El directorio especificado por el parámetro Directory no pudo ser encontrado.
ERROR_BAD_FORMAT	El fichero ejecutable no es válido o está corrupto.
SE_ERR_ACCESSDENIED	El acceso al fichero especificado fue denegado.
SE_ERR_ASSOCINCOMPLETE	Hay una asociación incompleta o no válida entre el fichero especificado y el fichero ejecutable.
SE_ERR_DDEBUSY	La transacción DDE solicitada no pudo ser completada debido a que otra transacción DDE se está realizando.
SE_ERR_DDEFAIL	La transacción DDE solicitada falló.
SE_ERR_DDETIMEOUT	La transacción DDE solicitada falló debido a que su tiempo de espera máximo ha finalizado.
SE_ERR_DLLNOTFOUND	Una biblioteca de enlace dinámico requerida no fue encontrada.
SE_ERR_FNF	El fichero especificado por el parámetro FileName no fue encontrado.
SE_ERR_NOASSOC	No hay fichero ejecutable asociado con la extensión del nombre de fichero especificado.
SE_ERR_OOM	La operación no pudo ser completada debido a insuficiente memoria.
SE_ERR_PNF	El directorio especificado por el parámetro Directory no fue encontrado.
SE_ERR_SHARE	Ha ocurrido una violación de compartición de recursos.

**ShellExecuteEx****ShellAPI.Pas****Sintaxis**

```
ShellExecuteEx(
    lpExecInfo: PShellExecuteInfo {puntero a un registro de información de ejecución}
):BOOL;                          {devuelve TRUE o FALSE}
```

**Descripción**

Similar a *ShellExecute*, esta función ejecuta una acción sobre un fichero y puede ser utilizada para imprimir un documento, lanzar un fichero ejecutable o abrir una carpeta de directorio.

**Parámetros**

*lpExecInfo*: Puntero a un registro *TShellExecuteInfo*. Este registro contiene información sobre la acción a ejecutar sobre un fichero particular y recibirá a su vez información una vez que la acción sea completada. El registro *TShellExecuteInfo* se define como:

*TShellExecuteInfo* = **record**

<i>cbSize</i> : DWORD;	{el tamaño del registro, en bytes}
<i>fMask</i> : ULONG;	{valor que indica cómo usar los otros campos}
<i>Wnd</i> : HWND;	{manejador de la ventana madre}
<i>lpVerb</i> : PAnsiChar;	{puntero a una cadena que describe la acción}
<i>lpFile</i> : PAnsiChar;	{puntero a nombre de fichero o de carpeta}
<i>lpParameters</i> : PAnsiChar;	{puntero a parámetros de un fichero ejecutable}
<i>lpDirectory</i> : PAnsiChar;	{puntero al nombre del directorio por defecto}
<i>nShow</i> : Integer;	{modo de presentación}
<i>hInstApp</i> : HINST;	{manejador de instancia de aplicación}
{*** los siguientes campos son opcionales ***}	
<i>lpIDList</i> : Pointer;	{puntero a lista de identificadores de elementos}
<i>lpClass</i> : PAnsiChar;	{puntero al nombre de clase de fichero o GUID}
<i>hkeyClass</i> : HKEY;	{manejador de la entrada en el registro de la clase del fichero}
<i>dwHotKey</i> : DWORD;	{tecla 'caliente' asociada con la aplicación}
<i>hIcon</i> : THandle;	{manejador de icono para la clase del fichero}
<i>hProcess</i> : THandle;	{manejador de proceso de la aplicación lanzada}

**end;**

*cbSize*: El tamaño del registro *TShellExecuteInfo*, en bytes. A este campo se le debe asignar *SizeOf(TShellExecuteInfo)*.

*fMask*: Una serie de valores que indican si los campos opcionales del registro deben ser usados. Este campo puede ser uno o más valores de la Tabla 11-13. Dos o más valores pueden ser especificados usando el operador booleano **or** (por ejemplo, *SEE\_MASK\_CLASSKEY or SEE\_MASK\_CLASSNAME*).

*Wnd*: Manejador de la ventana madre sobre la que se mostrarán los cuadros de mensaje en caso de error.

*lpVerb*: Puntero a una cadena de caracteres terminada en nulo que especifica la acción a ejecutar sobre el fichero o carpeta indicado en el campo *lpFile*. La Tabla 11-14 lista las acciones estándar que pueden ser ejecutadas sobre un fichero o carpeta. Sin embargo, estas acciones no están limitadas a aquellas que se listan en la tabla. Este parámetro depende de las acciones registradas para el documento o la aplicación en el registro de Windows, y es posible crear nuevas acciones a través del menú ‘*Opciones de carpeta*’ del Explorador de Windows. Si este parámetro es **nil**, se usa por defecto la operación ‘*open*’.

*lpFile*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre de un documento, fichero ejecutable o carpeta.

*lpParameters*: Si el campo *lpFile* indica un fichero ejecutable, este campo contiene un puntero a una cadena de caracteres terminada en nulo que especifica los parámetros de línea de comandos a pasar al fichero ejecutable. Los parámetros deben ser separados por espacios. Si el campo *lpFile* especifica un documento o carpeta, este parámetro debe ser **nil**.

*lpDirectory*: Puntero a una cadena de caracteres terminada en nulo que contiene la ruta del directorio por defecto. Si este parámetro es **nil**, el directorio actual es usado como directorio de trabajo.

*nShow*: Valor que determina cómo el fichero ejecutable indicado por el campo *lpFile* será mostrado cuando sea lanzado. Este parámetro puede tomar un valor de la Tabla 11-15.

*hInstApp*: Si la función tiene éxito, al retornar este campo contiene el manejador del fichero ejecutable que fue lanzado o el manejador de la aplicación servidora DDE. Si la función falla, este campo contendrá un valor de la Tabla 11-16.

Los siguientes campos son opcionales. Estos miembros no tienen que ser asignados para que la función *ShellExecuteEx* trabaje correctamente.

*lpIDList*: Puntero a una lista de identificadores de elementos que identifican de modo único al fichero ejecutable a lanzar. Este campo es ignorado si el campo *fMask* no contiene *SEE\_MASK\_IDLIST*.

*lpClass*: Puntero a una cadena de caracteres terminada en nulo que contiene el nombre de una clase de fichero o un identificador global único (GUID). Este campo es ignorado si el campo *fMask* no contiene *SEE\_MASK\_CLASSNAME*.

*hkeyClass*: Manejador de la clave del registro para la clase de fichero. Este campo es ignorado si el campo *fMask* no contiene *SEE\_MASK\_CLASSKEY*.

*dwHotKey*: La ‘tecla caliente’ a asociar con el fichero ejecutable lanzado. La palabra menos significativa contiene el código virtual de la tecla y la más significativa las opciones modificadoras. Las opciones modificadoras pueden ser uno o más valores de la Tabla 11-17 combinados mediante el operador booleano **or** (por ejemplo, *HOTKEYF\_ALT or HOTKEYF\_SHIFT*). Este campo es ignorado si el campo *fMask* no contiene *SEE\_MASK\_HOTKEY*.

*hIcon*: Manejador del icono a utilizar para la clase del fichero. Este campo es ignorado si el campo *fMask* no contiene *SEE\_MASK\_ICON*.

*hProcess*: Si la función tiene éxito, este campo contendrá al retornar el manejador de proceso de la aplicación que fue iniciada. Este campo es ignorado si el campo *fMask* no contiene *SEE\_MASK\_NOCLOSEPROCESS*.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE, y el campo *hInstApp* del registro *TShellExecuteInfo* contiene el manejador de instancia de la aplicación que fue iniciada. Si la función falla, devuelve FALSE y al campo *hInstApp* se le asignará un valor de la Tabla 11-16. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*ShellExecute*

#### Ejemplo

##### Listado 11-12: Otro modo de visualizar ficheros de texto

```
procedure TForm1.FileListBox1DbClick(Sender: TObject);
var
  ExecInfo: TShellExecuteInfo;
begin
  {Esto abrirá el fichero sobre el que se hizo doble clic en el cuadro de lista.
   Si este fichero es demasiado grande para el Bloc de Notas, Windows preguntará
   si se desea lanzar Wordpad}

  {Rellena el registro TShellExecuteInfo}
  ExecInfo.cbSize      := SizeOf(TShellExecuteInfo);
  ExecInfo.fMask       := SEE_MASK_NOCLOSEPROCESS;
  ExecInfo.Wnd         := Form1.Handle;
  ExecInfo.lpVerb       := 'Open';
  ExecInfo.lpFile       := PChar(FileListBox1.FileName);
  ExecInfo.lpParameters := '';
  ExecInfo.lpDirectory  := '';
  ExecInfo.nShow        := SW_SHOWNORMAL;

  {Abre el fichero especificado}
  ShellExecuteEx(@ExecInfo);
end;
```

Tabla 11-13: Valores del campo *lpExecInfo.fMask* de *ShellExecuteEx*

Valor	Descripción
SEE_MASK_CLASSKEY	Utiliza la clave de clase especificada en el campo <i>hkeyClass</i> .
SEE_MASK_CLASSNAME	Utiliza el nombre de clase especificado en el campo <i>lpClass</i> .
SEE_MASK_CONNECTNETDRV	El campo <i>lpFile</i> especifica una ruta UNC.
SEE_MASK_DOENVSUBST	Expande las variables de entorno incluidas en los campos <i>lpFile</i> o <i>lpDirectory</i> .

SEE_MASK_FLAG_DDEWAIT	Si un intercambio DDE ha comenzado, espera a que termine antes de retornar.
SEE_MASK_FLAG_NO_UI	No muestra cuadro de mensaje alguno en caso de que ocurran errores.
SEE_MASK_HOTKEY	Utiliza la tecla caliente especificada en el campo dwHotKey.
SEE_MASK_ICON	Utiliza el icono especificado en el campo hIcon.
SEE_MASK_IDLIST	Utiliza la lista de identificadores de elementos especificada en el campo lpIDList.
SEE_MASK_INVOKEIDLIST	Utiliza la lista de identificadores de elementos especificada en el campo lpIDList. Si el campo lpIDList es <b>nil</b> , la función crea una lista de identificadores de elementos y lanza la aplicación. Esta opción precede a SEE_MASK_IDLIST.
SEE_MASK_NOCLOSEPROCESS	Hace que el campo hProcess reciba un manejador del proceso iniciado. El proceso continúa ejecutándose después de que la función ShellExecuteEx termine.

Tabla II-14: Valores del campo lpExecInfo.lpVerb de ShellExecuteEx

Valor	Descripción
'open'	Abre el fichero o carpeta, o lanza el fichero ejecutable identificado por el parámetro FileName.
'print'	Imprime el documento identificado por el parámetro FileName. Si este parámetro identifica un fichero ejecutable, este es lanzado como si 'open' hubiera sido especificado.
'explore'	Abren una ventana del Explorador de Windows sobre la carpeta identificada por el parámetro FileName.

Tabla II-15: Valores del campo lpExecInfo.nShow de ShellExecuteEx

Valor	Descripción
SW_HIDE	La ventana es ocultada y otra ventana es activada.
SW_MINIMIZE	La ventana es minimizada y la próxima ventana de nivel superior en el orden Z relativo es activada.
SW_RESTORE	La ventana es activada y mostrada en su tamaño y posición originales.
SW_SHOW	La ventana es activada y mostrada en su tamaño y posición actuales.

Valor	Descripción
SW_SHOWDEFAULT	La ventana es mostrada según se indica en el campo <code>wShowWindow</code> del registro <code>TStartupInfo</code> pasado a la función <code>CreateProcess</code> cuando arrancó la aplicación. Esto es usado para seleccionar el estado de presentación inicial de la ventana principal de la aplicación. Esta opción debe ser usada cuando se muestra la ventana por primera vez, si la aplicación puede ser lanzada desde un acceso directo. Esta opción provocará que la ventana sea mostrada usando los atributos de ejecución indicados en las propiedades del acceso directo.
SW_SHOWMAXIMIZED	La ventana es activada y mostrada en estado maximizado.
SW_SHOWMINIMIZED	La ventana es activada y mostrada como un icono.
SW_SHOWMINNOACTIVE	La ventana es mostrada como un icono. La ventana que estaba activa permanecerá activa.
SW_SHOWNORMAL	La ventana es mostrada en su estado actual. La ventana que estaba activa permanecerá activa.
SW_SHOWNOACTIVE	La ventana es mostrada en su estado más reciente. La ventana que estaba activa permanecerá activa.
SW_SHOWNORMAL	Equivalente a <code>SW_RESTORE</code> .

Tabla II-16: Códigos de error de `ShellExecuteEx`

Valor	Descripción
SE_ERR_ACCESSDENIED	El acceso al fichero especificado fue denegado.
SE_ERR_ASSOCINCOMPLETE	Hay una asociación incompleta o no válida entre el fichero especificado y el fichero ejecutable.
SE_ERR_DDEBUSY	La transacción DDE solicitada no pudo ser completada debido a que otra transacción DDE se está realizando.
SE_ERR_DDEFAIL	La transacción DDE solicitada falló.
SE_ERR_DDETIMEOUT	La transacción DDE solicitada falló debido a que su tiempo de espera máximo ha finalizado.
SE_ERR_DLLNOTFOUND	Una biblioteca de enlace dinámico requerida no fue encontrada.
SE_ERR_FNF	El fichero especificado por el parámetro <code>FileName</code> no fue encontrado.
SE_ERR_NOASSOC	No hay ningún fichero ejecutable asociado con la extensión del nombre de fichero especificado.
SE_ERR_OOM	La operación no pudo ser completada debido a memoria insuficiente.
SE_ERR_PNF	El directorio especificado por el parámetro <code>Directory</code> no fue encontrado.
SE_ERR_SHARE	Ha ocurrido una violación de compartición de recursos.

Tabla 11-17: Modificadores para el campo `lpExecInfo.dwHotKey` de `ShellExecuteEx`

Valor	Descripción
HOTKEYF_ALT	La tecla <b>Alt</b> debe ser pulsada.
HOTKEYF_CONTROL	La tecla <b>Ctrl</b> debe ser pulsada.
HOTKEYF_SHIFT	La tecla <b>Shift</b> debe ser pulsada.

**Shell\_NotifyIcon****ShellAPI.Pas***Sintaxis*

```
Shell_NotifyIcon(
    dwMessage: DWORD;           {mensaje de icono de notificación}
    lpData: PNotifyIconData     {puntero a un registro de icono de notificación}
): BOOL;                       {devuelve TRUE o FALSE}
```

*Descripción*

Esta función añade, modifica o quita un icono de notificación de la bandeja de iconos de la barra de tareas del sistema.

*Parámetros*

*dwMessage*: Identificador del mensaje de icono de notificación. Puede ser cualquier valor de la Tabla 11-18.

*lpData*: Puntero a un registro *TNotifyIconData*. Este registro se define como:

*TNotifyIconData* = **record**

```
    cbSize: DWORD;               {el tamaño del registro TNotifyIconData}
    Wnd: HWND;                   {manejador de ventana}
    uID: UINT;                    {identificador definido por la aplicación}
    uFlags: UINT;                 {opciones de modificación}
    uCallbackMessage: UINT;       {id. de mensaje definido por la aplicación}
    hIcon: HICON;                 {manejador de icono}
    szTip: array [0..63] of AnsiChar; {cadena de indicaciones}
```

**end;**

*cbSize*: El tamaño del registro *TNotifyIconData*, en bytes. A este campo se le debe asignar *SizeOf(TNotifyIconData)*.

*Wnd*: Manejador de la ventana que recibe mensajes de notificación cuando un evento ocurre sobre el icono situado en la bandeja de iconos del sistema.

*uID*: Un identificador definido por la aplicación para el icono de notificación.

*uFlags*: Combinación de valores que indica qué otros campos del registro *TNotifyIconData* son válidos y deben ser usados. A este campo se debe asignar una combinación de valores de la Tabla 11-19.

*uCallbackMessage*: Un mensaje definido por la aplicación. Este mensaje es enviado a la ventana asociada con el manejador de la ventana especificado en el campo *Wnd* cada vez que un evento de ratón ocurra sobre el icono situado en la bandeja de iconos del sistema.

*hIcon*: Manejador del icono a mostrar en la bandeja de iconos del sistema.

*szTip*: Puntero a una cadena de caracteres terminada en nulo usada como texto de indicación para el icono de notificación.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

#### Véase además

*SHAppBarMessage*

#### Ejemplo

##### Listado II-13: Añadiendo un icono a la bandeja del sistema

```
const
{El mensaje de notificación definido por la aplicación}
WM_TRAYICONCLICKED = WM_USER + 1;
DELTRAYICON        = 1;           // el id del icono de la bandeja

type
TForm1 = class(TForm)
  ListBox1: TListBox;
  Label1: TLabel;
  procedure FormCreate(Sender: TObject);
  procedure FormDestroy(Sender: TObject);
private
  {Declaraciones privadas}
  {El manejador del mensaje para la notificación del icono de la bandeja}
  procedure WMTrayIconClicked(var Msg: TMessage); message WM_TRAYICONCLICKED;
public
  {Declaraciones públicas}
end;

var
  Form1: TForm1;
  IconData: TNotifyIconData; // el registro del icono de notificación

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {Inicializa el registro del icono de notificación de la bandeja}
  with IconData do begin
    cbSize      := SizeOf(TNotifyIconData);
    Wnd         := Form1.Handle;
```

```

    uID          := DELTRAYICON;
    uFlags       := NIF_ICON or NIF_MESSAGE or NIF_TIP;
    uCallbackMessage := WM_TRAYICONCLICKED;
    hIcon        := Application.Icon.Handle;
    szTip        := 'Delphi Tray Icon';
end;

{Notifica al sistema que estamos añadiendo un icono a la bandeja}
Shell_NotifyIcon(NIM_ADD, @IconData);
end;

procedure TForm1.WMTrayIconClicked(var Msg: TMessage);
begin
    {El icono de la bandeja ha recibido un mensaje, lo muestra}
    case Msg.lParam of
        WM_LBUTTONDOWN: listbox1.Items.add('double click');
        WM_LBUTTONDOWN: listbox1.Items.add('mouse down');
        WM_LBUTTONUP:   listbox1.Items.add('mouse up');
    end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Quita el icono del sistema}
    Shell_NotifyIcon(NIM_DELETE, @IconData);
end;

```



Figura 11-11:  
El nuevo icono  
de notificación

Tabla 11-18: Valores del parámetro dwMessage de Shell\_NotifyIcon

Valor	Descripción
NIM_ADD	Añade un icono de notificación a la bandeja de iconos en la barra de tareas del sistema.
NIM_DELETE	Borra un icono de notificación de la bandeja de iconos en la barra de tareas del sistema.
NIM_MODIFY	Modifica un icono de notificación de la bandeja de iconos en la barra de tareas del sistema.

Tabla 11-19: Valores del campo `lpData.uFlags` de `Shell_NotifyIcon`

Valor	Descripción
NIF_ICON	El manejador de icono en el campo <code>hIcon</code> es válido.
NIF_MESSAGE	El identificador de mensaje en el campo <code>uCallbackMessage</code> es válido.
NIF_TIP	La cadena de indicaciones a la que apunta el campo <code>szTip</code> es válida.

**SHFileOperation****ShellAPI.Pas****Sintaxis**

```
SHFileOperation(
    const lpFileOp: TSHFileOpStruct    {puntero a registro de operación de ficheros}
): Integer;                           {devuelve un valor entero}
```

**Descripción**

Esta función permite copiar, eliminar, mover, o renombrar ficheros o carpetas.

**Parámetros**

*lpFileOp*: Un registro de tipo *TSHFileOpStruct* que contiene información sobre los ficheros implicados y las acciones a ejecutar. Este registro se define como:

```
TSHFileOpStruct = record
    Wnd: HWND;                {manejador de ventana}
    wFunc: UINT;              {indica el tipo de operación}
    pFrom: PAnsiChar;        {puntero a nombres de ficheros de origen}
    pTo: PAnsiChar;          {puntero a nombres de ficheros de destino}
    fFlags: FILEOP_FLAGS;    {opciones de control de operaciones}
    fAnyOperationsAborted: BOOL; {indicador de operación abortada}
    hNameMappings: Pointer;   {manejador de objeto de mapeado de nombre de fichero}
    lpszProgressTitle: PAnsiChar; {el título del cuadro de diálogo de progreso}
end;
```

*Wnd*: Manejador de la ventana que será usada como madre del cuadro de diálogo que muestra el progreso de la operación sobre ficheros.

*wFunc*: Valor que indica la operación a ejecutar. Este campo puede tomar un valor de la Tabla 11-20.

*pFrom*: Puntero a un *buffer* que contiene los nombres de los ficheros sobre los cuales se realizará la operación indicada. Si se especifican varios nombres de fichero, cada uno tiene que ser separado por un terminador nulo, y el *buffer* debe terminar con dos terminadores nulos. Si los nombres de fichero no contienen una

ruta, se asume que el directorio de origen es el directorio indicado por la función *GetCurrentDirectory*.

*pTo*: Puntero a un *buffer* que contiene el nombre del fichero o directorio de destino. Si el campo *fFlags* contiene *FOF\_MULTIDESTFILES*, este *buffer* puede contener múltiples nombres de ficheros de destino, uno para cada fichero de origen. Cada nombre de fichero de destino tiene que ser separado del siguiente mediante un terminador nulo, y el *buffer* tiene que terminar con dos terminadores nulos. Si los nombres de ficheros no contienen una ruta, se asume que el directorio de destino es el directorio indicado por la función *GetCurrentDirectory*.

*fFlags*: Combinación de valores que indican el tipo de operación a ejecutar sobre los ficheros especificados. Este campo puede contener uno o más valores de la Tabla 11-21.

*fAnyOperationsAborted*: Este campo recibirá el valor TRUE si alguna operación de fichero fue abortada por el usuario antes de finalizar. En caso contrario, recibirá FALSE.

*hNameMappings*: Si el campo *fFlags* incluye la opción *FOF\_WANTMAPPINGHANDLE*, este campo recibirá un puntero a un objeto de mapeado de nombres de fichero que contiene un *array* de registros *TSHNameMapping*. Estas estructuras contienen las rutas y nombres de fichero nuevo y anterior para cada fichero que haya sido movido, copiado o renombrado. El objeto de mapeado de nombres de ficheros deberá ser eliminado usando la función *SHFreeNameMappings*.

El registro *TSHNameMapping* se define como:

```
TSHNameMapping = record
    pszOldPath: PAnsiChar;           {puntero a cadena}
    pszNewPath: PAnsiChar;           {puntero a cadena}
    cchOldPath: Integer;              {longitud de cadena}
    cchNewPath: Integer;              {longitud de cadena}
end;
```

*pszOldPath*: Cadena de caracteres terminada en nulo que especifica la ruta y el nombre de un fichero antiguo.

*pszNewPath*: Cadena de caracteres terminada en nulo que especifica la ruta y el nombre de un fichero nuevo.

*cchOldPath*: La cantidad de caracteres de la cadena *pszOldPath*.

*cchNewPath*: La cantidad de caracteres de la cadena *pszNewPath*.

*lpszProgressTitle*: Cadena de caracteres terminada en nulo que será usada como título para el cuadro de diálogo de progreso. Este campo es usado sólo si el campo *fFlags* incluye la opción *FOF\_SIMPLEPROGRESS*.

Valor que devuelve

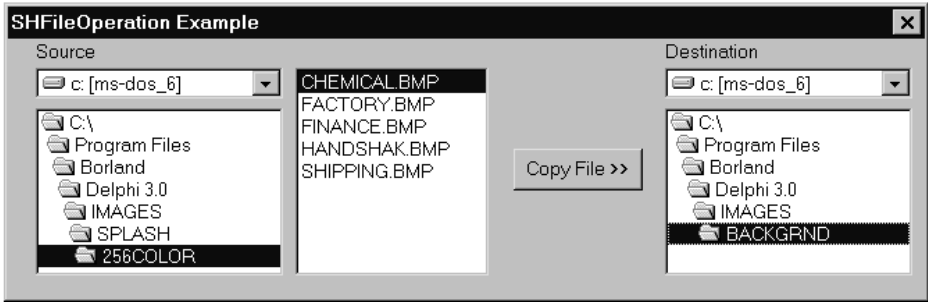
Si la función tiene éxito, devuelve un valor mayor que cero; en caso contrario, devuelve cero.

Véase además

*GetCurrentDirectory, SetCurrentDirectory, ShellExecute, SHFreeNameMappings*

Ejemplo

Figura 11-12:  
Copiando un  
fichero de  
mapa de bits



Listado II-14: Copiando un fichero

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileOpInfo: TSHFileOpStruct; // almacena información sobre el fichero
begin
  with FileOpInfo do begin
    Wnd      := Form1.Handle;
    wFunc    := FO_COPY;           // realiza una copia
    pFrom    := PChar(FileListBox1.FileName+#0+#0); // el fichero de origen
    pTo      := PChar(DirectoryListBox2.Directory); // el directorio de destino
    fFlags   := FOF_WANTMAPPINGHANDLE;
  end;

  {Ejecuta la operación del fichero}
  SHFileOperation(FileOpInfo);

  {El array de registros TSHNameMapping devuelto en FileOpInfo.hNameMappings debe
   ser liberado usando la función SHFreeNameMappings.}
  SHFreeNameMappings(GlobalHandle(FileOpInfo.hNameMappings));
end;
```

Tabla II-20: Valores del campo lpFileOp.wFunc de SHFileOperation

Valor	Descripción
FO_COPY	Copia los ficheros especificados en el campo pFrom a la ubicación especificada por el campo pTo.
FO_DELETE	Elimina (borra) los ficheros especificados en el campo pFrom. El campo pTo es ignorado.

Valor	Descripción
FO_MOVE	Mueve los ficheros especificados en el campo pFrom a la ubicación especificada por el campo pTo.
FO_RENAME	Renombra los ficheros especificados en el campo pFrom. El campo pTo es ignorado.

Tabla II-21: Valores del campo lpFileOp.fFlags de SHFileOperation

Valor	Descripción
FOF_ALLOWUNDO	El fichero especificado es borrado (enviado a la papelera de reciclaje).
FOF_FILESONLY	La operación es ejecutada solamente sobre ficheros solo si se ha especificado un nombre de fichero con comodines (por ejemplo, *.pas).
FOF_MULTIDESTFILES	El campo pTo contiene un fichero de destino para cada fichero de origen, en lugar de un directorio en el cual todos los ficheros serán depositados.
FOF_NOCONFIRMATION	No se pide la confirmación del usuario, y la operación se realiza como si se hubiera respondido 'Sí a Todo'.
FOF_NOCONFIRMMKDIR	Crea automáticamente un nuevo directorio si es necesario.
FOF_NOERRORUI	No hay indicación visual si ocurre un error.
FOF_RENAMEONCOLLISION	Al fichero de origen se le asigna automáticamente un nuevo nombre, tal como 'Copia #1 de...', en una operación de mover, copiar o renombrar, si en el directorio de destino ya existe un fichero con ese nombre.
FOF_SILENT	No muestra un cuadro de diálogo de progreso.
FOF_SIMPLEPROGRESS	Muestra un cuadro de diálogo de progreso, pero no muestra los nombres de ficheros.
FOF_WANTMAPPINGHANDLE	El campo hNameMappings recibe un manejador de un objeto de mapeado de nombres de ficheros.

**SHFreeNameMappings****ShellAPI.Pas****Sintaxis**

```
SHFreeNameMappings(
    hNameMappings: THandle    {manejador de objeto de mapeado de nombres
                               de ficheros}
);                             {esta función no devuelve ningún valor}
```

**Descripción**

Esta función libera un objeto de mapeado de nombres de ficheros devuelto por la función *SHFileOperation*.

**Parámetros**

*hNameMappings*: Manejador del objeto de mapeado de nombres de ficheros a liberar.

**Véase además**

*SHFileOperation*

**Ejemplo**

Vea el Listado 11-14 bajo *SHFileOperation*.

**SHGetFileInfo ShellAPI.Pas****Sintaxis**

```
SHGetFileInfo(
    pszPath: PAnsiChar;           {puntero a cadena de nombre de fichero}
    dwFileAttributes: DWORD;       {atributos de fichero}
    var psfi: TSHFileInfo;         {puntero al registro TSHFileInfo}
    cbFileInfo: UINT;             {tamaño del registro TSHFileInfo}
    uFlags: UINT                   {opciones de recuperación de información}
): DWORD;                        {devuelve un valor de doble palabra}
```

**Descripción**

Esta función recupera información sobre un fichero, carpeta, directorio, o unidad de disco.

**Parámetros**

*pszPath*: Puntero a una cadena de caracteres terminada en nulo que contiene la ruta y nombre del fichero cuya información será recuperada. Puede ser un nombre de fichero largo o un nombre de fichero en el formato de nombres de DOS (8.3). Si el parámetro *uFlags* contiene la opción *SHGFI\_PIDL*, este parámetro puede apuntar a una lista de identificadores de elementos que identifica al fichero.

*dwFileAttributes*: Combinación de valores que indican los atributos del fichero. Este parámetro puede contener uno o más valores de la Tabla 11-22. Si el parámetro *uFlags* no contiene la opción *SHGFI\_USEFILEATTRIBUTES*, este parámetro es ignorado.

*psfi*: Puntero a un registro de tipo *TSHFileInfo*. Este registro contiene la información requerida sobre el fichero especificado. El registro *TSHFileInfo* se define como:

*TSHFileInfo* = **record**

```
    hIcon: HICON;                 {manejador de icono}
    iIcon: Integer;               {índice de icono}
    dwAttributes: DWORD;          {atributos}
    szDisplayName: array [0..MAX_PATH-1] of AnsiChar;
                                {cadena de nombre a mostrar}
```

```

    szTypeName: array [0..79] of AnsiChar;           {cadena de tipo de fichero}
end;

    hIcon: Manejador del icono que representa al fichero especificado.
    iIcon: El índice del icono del fichero dentro de la lista de imágenes del sistema.
    dwAttributes: Combinación de valores que indican los atributos del fichero. Este
    campo puede tomar uno o más valores de la tabla dwFileAttributes (Tabla 11-22).
    szDisplayName: Cadena de caracteres terminada en nulo que indica el nombre a
    mostrar del fichero especificado, tal como aparece en el shell.
    szTypeName: Cadena de caracteres terminada en nulo que describe el tipo del
    fichero especificado.

    cbFileInfo: El tamaño, en bytes, del registro TSHFileInfo al que apunta el parámetro
    psfi. A este parámetro se le debe asignar SizeOf(TSHFileInfo).

    uFlags: Combinación de opciones que indican el tipo de información a recuperar. Este
    parámetro puede tomar uno o más valores de la Tabla 11-23.

```

#### Valor que devuelve

Si la función tiene éxito, devuelve un valor mayor que cero. En caso contrario, devuelve cero. Consulte la tabla de valores de *uFlags* (Tabla 11-23) para ver una descripción del valor devuelto.

#### Véase además

*ExtractAssociatedIcon*, *ExtractIcon*, *FindExecutable*

#### Ejemplo

##### Listado 11-15: Recuperando información sobre el fichero

```

procedure TForm1.FileListBox1DbClick(Sender: TObject);
var
    FileInfo: TSHFileInfo;           // almacena información sobre un fichero
    TempIcon: TIcon;                // un objeto de icono temporal
begin
    {Recupera información sobre el fichero seleccionado}
    SHGetFileInfo(PChar(FileListBox1.FileName), 0, FileInfo, SizeOf(TSHFileInfo),
        SHGFI_DISPLAYNAME or SHGFI_ICON or SHGFI_TYPENAME);

    {Muestra información sobre el fichero seleccionado}
    with ListBox1.Items do begin
        Clear;
        Add('Display Name: ' + FileInfo.szDisplayName);
        Add('Type Name: ' + FileInfo.szTypeName);
        Add('Icon index: ' + IntToStr(FileInfo.iIcon));
    end;

    {Crea un objeto de icono temporal, para que podemos mostrar el icono del fichero
    sobre el objeto imagen}
    TempIcon := TIcon.Create;

```

```
TempIcon.Handle := FileInfo.hIcon;  
Image1.Picture.Assign(TempIcon);  
TempIcon.Free;  
end;
```

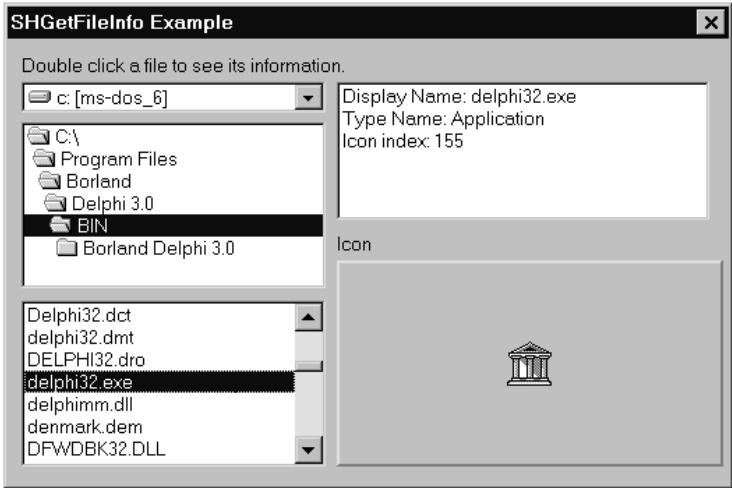


Figura 11-13:  
La información  
del fichero

Tabla II-22: Valores del parámetro dwFileAttributes de SHGetFileInfo

Valor	Descripción
FILE_ATTRIBUTE_READONLY	El fichero es de sólo-lectura.
FILE_ATTRIBUTE_HIDDEN	El fichero es oculto.
FILE_ATTRIBUTE_SYSTEM	El fichero es un fichero de sistema.
FILE_ATTRIBUTE_DIRECTORY	El fichero es un directorio.
FILE_ATTRIBUTE_ARCHIVE	El fichero es un archivo.
FILE_ATTRIBUTE_NORMAL	El fichero no tiene ningún atributo.
FILE_ATTRIBUTE_TEMPORARY	El fichero es un fichero temporal.
FILE_ATTRIBUTE_COMPRESSED	El fichero está comprimido.

Tabla II-23: Valores del parámetro uFlags de SHGetFileInfo

Valor	Descripción
SHGFI_ATTRIBUTES	Recupera los atributos del fichero especificado. Estos atributos son copiados en el campo dwAttributes del registro TSHFileInfo al que apunta el parámetro psfi.
SHGFI_DISPLAYNAME	Recupera el nombre a mostrar del fichero especificado. Esta cadena es copiada al campo szDisplayName del registro TSHFileInfo al que apunta el parámetro psfi.

Valor	Descripción
SHGFI_ICON	Recupera un manejador del icono que representa al fichero especificado. El manejador del icono es copiado en el campo <code>hIcon</code> de registro <code>TSHFileInfo</code> al que apunta el parámetro <code>psfi</code> . El índice del icono en la lista de imágenes del sistema es copiado al campo <code>ilcon</code> del registro <code>TSHFileInfo</code> al que apunta el parámetro <code>psfi</code> . La función devuelve el manejador de la lista de imágenes del sistema.
SHGFI_ICONLOCATION	Recupera el nombre del fichero que contiene el icono que representa al fichero especificado. Este nombre de fichero es copiado en el campo <code>szDisplayName</code> del registro <code>TSHFileInfo</code> al que apunta el parámetro <code>psfi</code> .
SHGFI_LARGEICON	Recupera el icono grande del fichero especificado. Esta opción debe utilizarse junto con la opción <code>SHGFI_ICON</code> .
SHGFI_LINKOVERLAY	Combina el gráfico de enlace con el icono del fichero especificado. Esta opción debe utilizarse junto con la opción <code>SHGFI_ICON</code> .
SHGFI_OPENICON	Recupera el icono de 'abrir' del fichero especificado. Esta opción debe utilizarse junto con la opción <code>SHGFI_ICON</code> .
SHGFI_PIDL	Indica que el parámetro <code>pszPath</code> apunta a una lista de identificadores de elementos y no a una ruta.
SHGFI_SELECTED	El icono del fichero es combinado con el color resaltado del sistema. Esta opción debe utilizarse junto con la opción <code>SHGFI_ICON</code> .
SHGFI_SHELLICONSIZE	Recupera el icono del fichero especificado, escalado al tamaño utilizado por el shell. Esta opción debe utilizarse junto con la opción <code>SHGFI_ICON</code> .
SHGFI_SMALLICON	Recupera el icono pequeño del fichero especificado. Esta opción debe utilizarse junto con la opción <code>SHGFI_ICON</code> .
SHGFI_SYSICONINDEX	Recupera el índice del icono del fichero especificado en la lista de imágenes del sistema. El índice del icono es copiado al campo <code>ilcon</code> del registro <code>TSHFileInfo</code> al que apunta el parámetro <code>psfi</code> . La función devuelve el manejador de la lista de imágenes del sistema.
SHGFI_TYPENAME	Recupera una cadena que describe el tipo de fichero especificado. Esta cadena es copiada al campo <code>szTypeName</code> del registro <code>TSHFileInfo</code> al que apunta el parámetro <code>psfi</code> .
SHGFI_USEFILEATTRIBUTES	Indica que la función debe recuperar información sólo sobre aquellos ficheros que tienen los atributos especificados por el parámetro <code>dwFileAttributes</code> .

**SHGetPathFromIDList****ShlObj.Pas****Sintaxis**

```
SHGetPathFromIDList(
    pidl: PItemIDList;           {puntero a lista de identificadores de elementos}
    pszPath: PChar               {puntero a buffer}
): BOOL;                       {devuelve TRUE o FALSE}
```

**Descripción**

Esta función recupera una cadena que contiene el nombre de ruta del fichero o carpeta identificado por la lista de identificadores de elementos.

**Parámetros**

*pidl*: Puntero a una lista de identificadores de elementos que identifica un fichero o directorio en el sistema de ficheros. Esta función fallará si la lista de identificadores de elementos especifica una carpeta que no pertenece al sistema de ficheros, como por ejemplo las carpetas de Impresoras o del Panel de Control.

*pszPath*: Puntero a un *buffer* que recibe la ruta asociada. Se asume que el tamaño de este *buffer* es de *MAX\_PATH* bytes.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*SHBrowseForFolder*, *SHGetFileInfo*

**Ejemplo**

Vea el Listado 11-9 bajo *SHBrowseForFolder*.

**SHGetSpecialFolderLocation****ShlObj.Pas****Sintaxis**

```
SHGetSpecialFolderLocation(
    hwndOwner: HWND;           {manejador de ventana}
    nFolder: Integer;           {indicador de carpeta}
    var ppidl: PItemIDList      {puntero a lista de identificadores de elementos}
): HRESULT;                   {devuelve un resultado OLE}
```

**Descripción**

Esta función recupera una lista de identificadores de elementos que especifica la ubicación de la carpeta especial. Observe que sólo aquellas carpetas que estén registradas bajo la clave:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\
Explorer\Shell Folders
```

devolverán una lista de identificadores de elementos que identifica a una carpeta del sistema de ficheros que *SHGetPathFromIDList* puede usar para recuperar una ruta física.

#### Parámetros

*hwndOwner*: Manejador de la ventana propietaria de posibles cuadros de diálogo o mensajes.

*nFolder*: Valor que indica la carpeta cuya ubicación se desea recuperar. Este parámetro puede tomar un valor de la Tabla 11-24.

*ppidl*: Puntero a una lista de identificadores de elementos que determina la ubicación de la carpeta indicada, relativa a la raíz del espacio de nombres.

#### Valor que devuelve

Si la función tiene éxito, devuelve *NOERROR*; en caso contrario, devuelve un error definido por el subsistema OLE.

#### Véase además

*SHBrowseForFolder*, *SHGetFileInfo*, *SHGetPathFromIDList*

#### Ejemplo

##### Listado 11-16: Recuperando la ubicación del directorio del escritorio de Windows

```
procedure TForm1.Button1Click(Sender: TObject);
var
  IDList: PItemIDList;           // lista de identificadores de elementos
  PathName: array[0..MAX_PATH] of Char; // la ruta de la carpeta especificada
begin
  {Recupera la lista de identificadores asociada a la ubicación del escritorio}
  SHGetSpecialFolderLocation(Form1.Handle, CSIDL_DESKTOPDIRECTORY, IDList);

  {Recupera la ruta a partir de la lista de identificadores}
  SHGetPathFromIDList(IDList, @PathName);

  {Muestra la ruta}
  Label1.Caption := PathName;
end;
```



Figura 11-14:  
Recuperando  
la ubicación  
del escritorio  
de Windows

Tabla II-24: Valores del parámetro nFolder de SHGetSpecialFolderLocation

Valor	Descripción
CSIDL_BITBUCKET	Recupera la ubicación de la papelera de reciclaje. Este directorio no está en el registro y tiene los atributos oculto y de sistema para evitar que los usuarios lo muevan o eliminen.
CSIDL_CONTROLS	Recupera la carpeta virtual que contiene los iconos para las aplicaciones (applets) del Panel de Control.
CSIDL_DESKTOP	Recupera la carpeta virtual para la raíz del espacio de nombres, el escritorio de Windows.
CSIDL_DESKTOPDIRECTORY	Recupera el directorio del sistema de ficheros donde se almacenan físicamente los elementos del escritorio.
CSIDL_DRIVES	Recupera la carpeta virtual Mi PC, que contiene dispositivos de almacenamiento, impresoras y el Panel de Control, y puede además contener unidades de red mapeadas.
CSIDL_FONTS	Recupera la carpeta virtual que contiene las fuentes.
CSIDL_NETHOOD	Recupera el directorio del sistema de ficheros que contiene objetos del Entorno de Red.
CSIDL_NETWORK	Recupera la carpeta virtual del Entorno de Red que representa el nivel más alto de la jerarquía de red.
CSIDL_PERSONAL	Recupera el directorio del sistema de ficheros que sirve como depósito común para documentos, que es generalmente el directorio C:\Mis Documentos.
CSIDL_PRINTERS	Recupera la carpeta virtual que contiene las impresoras instaladas.
CSIDL_PROGRAMS	Recupera el directorio del sistema de ficheros que contiene los grupos de programas.
CSIDL_RECENT	Recupera el directorio del sistema de ficheros donde se almacenan los enlaces a los documentos más recientes.
CSIDL_SENDTO	Recupera el directorio del sistema de ficheros que contiene las opciones del menú de contexto 'Enviar a'.
CSIDL_STARTMENU	Recupera el directorio del sistema de ficheros que contiene los accesos directos y ficheros ejecutables que aparecen en el menú Inicio.
CSIDL_STARTUP	Recupera el directorio del sistema de ficheros que contiene los accesos directos y ficheros ejecutables que son lanzados durante el arranque de Windows.
CSIDL_TEMPLATES	Recupera el directorio del sistema de ficheros que sirve como depósito común para las plantillas de documentos.



## Capítulo 12

# Funciones de menú

Todas las aplicaciones Windows complejas utilizan un menú como parte de las herramientas de navegación de la interfaz de usuario. Delphi ha hecho un excelente trabajo encapsulando la arquitectura de menús de Windows, eliminando así la necesidad por parte del desarrollador de definir una estructura de menú en un fichero de recursos e importarlo mediante código a la aplicación. Sin embargo, las funciones de bajo nivel del API de Windows para manipular menús ofrecen al desarrollador alguna funcionalidad adicional que no está incluida de modo nativo en Delphi.

## Información general sobre los menús

Todos los elementos de menú tienen un identificador único que la aplicación utiliza para determinar qué elemento del menú fue seleccionado. Este identificador de elemento de menú es asignado automáticamente por Delphi cuando un objeto *TMenu* es usado para crear la estructura de menú de la aplicación. Sin embargo, cuando se añaden por programa elementos de menú utilizando las funciones del API de Windows, el desarrollador tiene que asignar el identificador de menú explícitamente. Si este identificador coincide con algún otro identificador de menú ya existente, la aplicación puede confundirse, produciéndose resultados inesperados cuando se seleccionan elementos de menú. Utilice la función *GetMenuItemID* para recuperar los identificadores de elementos de menú de todos los elementos de menú creados por Delphi, para garantizar que un nuevo identificador de elemento de menú no provoque una colisión de identificadores.

Cada vez que un usuario selecciona un elemento de menú, éste envía un mensaje *WM\_COMMAND* a su ventana propietaria. El parámetro *wID* de este mensaje contiene el identificador del elemento de menú seleccionado. Cuando un elemento de menú es añadido usando el API de Windows, la aplicación debe ofrecer una función de redefinición de mensaje para el mensaje *WM\_COMMAND*. Esto permitirá a la aplicación responder tanto a los mensajes enviados por los elementos de menú añadidos mediante el API de Windows, como a los enviados por los objetos *TMenuItem* de Delphi.

## El menú de sistema

El uso más común de las funciones de menú del API de Windows en Delphi es añadir elementos a un menú de sistema de una ventana. El *menú de sistema* es un menú desplegable que se muestra cuando se hace clic sobre el icono situado en la esquina superior izquierda de una ventana.

Para añadir un elemento de menú al menú de sistema, una aplicación tiene primero que usar la función *GetSystemMenu* para recuperar un manejador del menú de sistema. Una vez que se ha hecho esto, las demás funciones de menú pueden ser llamadas utilizando este manejador, como si de un menú normal se tratara. Los elementos de menú añadidos al menú de sistema enviarán un mensaje *WM\_SYSCOMMAND* a la ventana, en lugar del mensaje *WM\_COMMAND*. Adicionalmente, Windows asigna identificadores de elementos de menú superiores a \$F000 a los elementos de menú por defecto del sistema. Por eso, cualquier elemento de menú que la aplicación añada al menú de sistema, tiene que tener un identificador de elemento de menú menor que \$F000. Cuando el manejador del mensaje *WM\_SYSCOMMAND* determina si el mensaje fue enviado por un elemento de menú añadido por la aplicación, tiene que combinar el valor del parámetro de mensaje *wParam* con el valor \$FFF0 usando el operador booleano **and**. Esto es necesario porque Windows utiliza los últimos cuatro bits del parámetro *wParam* internamente, y éstos deben ser eliminados de la máscara para comprobar su igualdad con un identificador de elemento de menú definido por la aplicación.

Observe que si la aplicación añade elementos al menú de sistema y cambia alguna propiedad de la ventana, como por ejemplo el estilo del borde, de manera que la ventana tenga que ser creada nuevamente, el menú del sistema regresa a su configuración por defecto y todos los cambios se pierden. Consulte la función *CreatePopupMenu* para ver un ejemplo de cómo añadir elementos al menú de sistema.

## Menús emergentes

Un menú puede tomar dos formas: la forma de una *barra de menú* y la forma de un *menú emergente*. Un barra de menú es un menú que ha sido asignado a una ventana, y sus elementos de menú son mostrados en una fila horizontal bajo la barra de título de la ventana. Un menú emergente es un menú que es mostrado dinámicamente, como es el caso cuando un elemento de menú en la barra de menú es seleccionado o cuando el usuario hace clic con el botón derecho sobre una ventana para mostrar su menú contextual. Todos los menús desplegables, los menús emergentes o los submenús caen en esta categoría. Cualquier menú desplegable o submenú puede ser mostrado como un menú emergente, recuperando su manejador por medio de la función *GetSubMenu* y mostrándolo mediante la función *TrackPopupMenu*. Consulte la función *CreatePopupMenu* para ver un ejemplo de esta técnica.

Los elementos de menú en un menú emergente envían un mensaje *WM\_COMMAND* cuando son seleccionados. Sin embargo, Delphi define un objeto privado conocido

como *PopupList* (lista emergente) cuando se utiliza la clase *TPopupMenu*. Todos los mensajes *WM\_COMMAND* enviados por *TPopupMenu* son manejados por este objeto *PopupList*. Por eso, el formulario nunca recibe estos mensajes. Cuando se usan las funciones *TrackPopupMenu* o *TrackPopupMenuEx* con un objeto *TPopupMenu* de Delphi, estos mensajes son enviados directamente a la ventana indicada, sobrepasando la funcionalidad nativa de Delphi. Esto requiere que el desarrollador gestione la selección de los elementos de menú como si éstos hubieran sido añadidos por programa usando las funciones del API de Windows. Si el programador envía estos mensajes de vuelta al menú emergente, utilizando el método *DispatchCommand* de la clase *TPopupMenu*, entonces podrá asignar de la manera habitual eventos *OnClick* a los elementos de menú situados en un *TPopupMenu*.

## Menús dibujados por el propietario

Los menús dibujados por el propietario (*owner-drawn menus*) permiten al desarrollador programar algunos efectos interesantes con menús. En este tipo de menús, el desarrollador tiene un control total sobre la apariencia del elemento de menú, dibujándolo de la manera deseada. Las funciones *AppendMenu*, *InsertMenu*, *InsertMenuItem*, *ModifyMenu* y *SetMenuItemInfo* permiten definir un menú como dibujado por el propietario. Cuando se utilizan menús dibujados por el propietario, la ventana propietaria recibe un mensaje *WM\_MEASUREITEM* por cada elemento de menú la primera vez que el menú es mostrado. Cuando el menú es dibujado en pantalla, la ventana propietaria recibe un mensaje *WM\_DRAWITEM* por cada elemento de menú dibujado por el propietario. Ambos mensajes contienen punteros a registros que contienen información sobre el elemento de menú que está siendo medido o dibujado. El siguiente ejemplo muestra cómo usar elementos de menús dibujados por el propietario para mostrar un mapa de bits en el fondo de un menú.

### Listado I2-1: Mostrando un mapa de bits en el fondo de un menú

```
procedure TForm1.FormCreate(Sender: TObject);
var
    iCount: Integer;           // una variable de control de bucle
    MenuInfo: TMenuItemInfo;   // almacena información de un elemento de menú
    ABitmap: TBitmap;          // almacena un mapa de bits
begin
    {Para cada elemento de menú en el menú desplegable...}
    for iCount := 0 to GetMenuItemCount(MainMenu2.Handle) - 1 do
        begin
            {...crea un mapa de bits que contendrá la imagen a mostrar cuando este
              elemento de menú es dibujado}
            ABitmap := TBitmap.Create;

            {Asigna las dimensiones del mapa de bits (18 píxeles de altura y el ancho de
              la imagen original)}
            ABitmap.Width := Image1.Width;
            ABitmap.Height := 18;
```

```

        {Copia la imagen original en el mapa de bits}
        ABitmap.Canvas.CopyRect(Rect(0, 0, ABitmap.Width, ABitmap.Height),
                                Imagel.Canvas, Rect(0, ABitmap.Height * iCount,
                                ABitmap.Width, (ABitmap.Height * iCount) +
                                ABitmap.Height));

        {Dibuja transparentemente la barra del elemento de menú sobre el mapa de bits}
        SetBkMode(ABitmap.Canvas.Handle, TRANSPARENT);
        ABitmap.Canvas.Font.Color := clRed;
        ABitmap.Canvas.TextOut(5, 2, MainMenu2.Items[iCount].Caption);

        {Inicializa el registro del elemento de menú para indicar que estamos
        definiendo un menú dibujado por el propietario. Asigna un puntero al mapa de
        bits para este elemento de menú en el campo dwItemData del elemento de menú.
        Este puntero será utilizado cuando se procese el mensaje WM_DRAWITEM}
        MenuInfo.cbSize := SizeOf(TMenuItemInfo);
        MenuInfo.fMask := MIIM_DATA or MIIM_TYPE;
        MenuInfo.fType := MFT_OWNERDRAW;
        MenuInfo.dwItemData := Longint(Pointer(ABitmap));

        {Modifica el elemento de menú, convirtiéndolo en un elemento de menú
        dibujado por el propietario}
        SetMenuItemInfo(MainMenu2.Handle, iCount, TRUE, MenuInfo);
    end;
end;

{El mensaje WM_MEASUREITEM redefinido}
procedure TForm1.WMMeasureItem(var Msg: TWMMeasureItem);
begin
    {Llama al método heredado}
    inherited;

    {Asigna las dimensiones del nuevo elemento de menú}
    Msg.MeasureItemStruct^.itemWidth := Imagel.Width;
    Msg.MeasureItemStruct^.itemHeight := 18;

    {Indica que el mensaje ha sido manejado}
    Msg.Result := 1;
end;

{El mensaje WM_DRAWITEM redefinido}
procedure TForm1.WMDrawItem(var Msg: TWMDrawItem);
begin
    {Llama al método heredado}
    inherited;

    {Asigna el manejador del área de dibujo al manejador del contexto de dispositivo
    provisto por el mensaje}
    Canvas.Handle := Msg.DrawItemStruct^.hDC;

    {Si el elemento es seleccionado o el cursor del ratón está sobre él...}
    if (Msg.DrawItemStruct^.itemAction and ODA_SELECT <> 0) and
        (Msg.DrawItemStruct^.itemState and ODS_SELECTED <> 0) then
        {...indica que el mapa de bits será dibujado en vídeo inverso...}

```

```

    Canvas.CopyMode := cmNotSrcCopy
else
    {...o normalmente}
    Canvas.CopyMode := cmSrcCopy;

{Dibuja el mapa de bits (obtenido del campo ItemData del mensaje)}
Canvas.Draw(Msg.DrawItemStruct^.rcItem.Left, Msg.DrawItemStruct^.rcItem.Top,
    TBitmap(Pointer(Msg.DrawItemStruct^.ItemData)));

{Reinicia el manejador del área de dibujo. Un nuevo manejador del área de dibujo
será automáticamente creado cuando sea necesario}
Canvas.Handle := 0;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
    iCount: Integer;          // variable de control de bucle
    MenuInfo: TMenuItemInfo;  // almacena información sobre el elemento de menú
begin
    {Inicializa el registro del elemento de menú, que indica que sólo deseamos
    recuperar un puntero al mapa de bits asociado al elemento de menú}
    MenuInfo.cbSize := SizeOf(TMenuItemInfo);
    MenuInfo.fMask := MIIM_DATA;

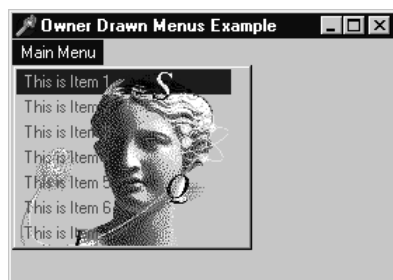
    {Para cada elemento en el menú...}
    for iCount := 0 to 6 do
    begin
        {...recupera información sobre el elemento de menú, específicamente el puntero
        asociado al mapa de bits}
        GetMenuItemInfo(MainMenu2.Handle, iCount, TRUE, MenuInfo);

        {Libera el mapa de bits}
        TBitmap(MenuInfo.dwItemData).Free;
    end;
end;

procedure TForm1.Item11Click(Sender: TObject);
begin
    {Indica qué elemento de menú fue seleccionado}
    Label2.Caption := TMenuItem(Sender).Caption;
end;

```

Figura 12-1:  
El elemento  
de menú  
dibujado por el  
propietario



## Funciones de menú

Las siguientes funciones son descritas en este capítulo:

**Tabla 12-1: Funciones de menú**

Valor	Descripción
AppendMenu	Añade un elemento de menú a un menú.
CheckMenuItem	Marca o desmarca un elemento de menú.
CheckMenuRadioItem	Marca un elemento de menú en un grupo de elementos de menú, y desmarca todos los demás. La marca de verificación es mostrada como una viñeta.
CreateMenu	Crea un menú nuevo y vacío.
CreatePopupMenu	Crea un menú emergente nuevo y vacío.
DeleteMenu	Elimina un elemento de menú y su submenú asociado.
DestroyMenu	Destruye un menú.
EnableMenuItem	Habilita o deshabilita un elemento de menú.
GetMenu	Recupera un manejador de un menú asignado a una ventana.
GetMenuDefaultItem	Recupera un elemento por defecto de un menú.
GetMenuItemCount	Recupera el número de elementos de menú en un menú.
GetMenuItemID	Recupera un identificador de un elemento de menú.
GetMenuItemInfo	Recupera información sobre un elemento de menú.
GetMenuItemRect	Recupera las coordenadas de pantalla de un elemento de menú.
GetMenuState	Recupera el estado de un elemento de menú.
GetMenuString	Recupera el texto de un elemento de menú.
GetSubMenu	Recupera un manejador del submenú activado por un elemento de menú.
GetSystemMenu	Recupera un manejador del menú de sistema.
HiliteMenuItem	Resalta u opaca un elemento de menú en una barra de menú.
InsertMenu	Inserta un elemento de menú.
InsertMenuItem	Inserta un elemento de menú. Ofrece más opciones que <i>InsertMenu</i> .
IsMenu	Indica si un manejador especificado es un manejador de un menú.
ModifyMenu	Modifica un elemento de menú existente.
RemoveMenu	Borra un elemento de menú de un menú.
SetMenu	Asigna un menú a una ventana.
SetMenuDefaultItem	Asigna el elemento por defecto del menú.
SetMenuItemBitmaps	Asigna el mapa de bits que indica que un elemento de menú está 'marcado' o 'no marcado'.
SetMenuItemInfo	Selecciona información de un elemento de menú.
TrackPopupMenu	Muestra un menú emergente.

Valor	Descripción
TrackPopupMenuEx	Muestra un menú emergente sin ocultar un área rectangular específica.

## AppendMenu Windows.Pas

### Sintaxis

```
AppendMenu(
    hMenu: HMenu;           {manejador del menú que está siendo añadido}
    uFlags: Integer;         {opciones del elemento de menú}
    uIDNewItem: Integer;     {identificador de elemento de menú o manejador de
                             submenú}
    lpNewItem: PChar        {datos del elemento de menú}
): BOOL;                   {devuelve TRUE o FALSE}
```

### Descripción

Esta función añade un nuevo elemento de menú al final de un menú, menú descendente, menú emergente, o submenú, asignando los atributos específicos del elemento de menú que se deseen.

### Parámetros

*hMenu*: Manejador del menú al cual será añadido el elemento de menú.

*uFlags*: Una combinación de opciones que indican ciertos atributos del nuevo elemento de menú. A este parámetro se le puede asignar uno o más valores de la Tabla 12-2.

*uIDNewItem*: Especifica el identificador del nuevo elemento de menú. Si el parámetro *uFlags* contiene la opción *MF\_POPUP*, a este parámetro se le asigna el manejador del menú desplegable o submenú.

*lpNewItem*: Puntero a datos adicionales asociados con el nuevo elemento de menú. El contenido de este parámetro depende de las opciones especificadas en el parámetro *uFlags*. Si el parámetro *uFlags* contiene la opción *MF\_BITMAP*, este parámetro contiene un manejador de un mapa de bits. Si el parámetro *uFlags* contiene la opción *MF\_STRING*, este parámetro apunta a una cadena terminada en nulo. Si el parámetro *uFlags* contiene la opción *MF\_OWNERDRAW*, este parámetro contiene un valor de 32 bits definido por la aplicación. Este valor estará contenido en el campo *itemData* del registro al que apuntan los mensajes *WM\_MEASUREITEM* y *WM\_DRAWITEM* que la ventana recibe cuando el menú es creado o dibujado.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*CreateMenu, CreatePopupMenu, DeleteMenu, DestroyMenu, InsertMenu, InsertMenuItem, ModifyMenu, RemoveMenu*

Ejemplo

#### Listado I2-2: Añadiendo un elemento de menú al menú del sistema

```

const
    MNUID_ABOUT = $1000;    // el identificador del nuevo elemento de menú

var
    Form1: TForm1;

implementation

uses AboutBoxU;

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Añade un menú Acerca de... al final del menú del sistema}
    AppendMenu(GetSystemMenu(Form1.Handle, FALSE), MF_STRING, MNUID_ABOUT,
        'About...');

    {Hace lo mismo para la aplicación, de manera que el elemento aparecerá cuando
    se haga clic con el botón derecho en el icono de la barra de tareas}
    AppendMenu(GetSystemMenu(Application.Handle, FALSE), MF_STRING, MNUID_ABOUT,
        'About...');

    {Debido a que estos elementos de menú son añadidos desde las funciones del API,
    debemos interceptar manualmente sus mensajes y procesarlos apropiadamente}
    Application.OnMessage := ApplicationMessage;
end;

procedure TForm1.ApplicationMessage(var Msg: TMsg; var Handled: Boolean);
begin
    {Si el mensaje proviene de un elemento de menú del sistema y específicamente
    de nuestro nuevo elemento de menú, muestra el Acerca de...}
    if (Msg.Message = WM_SYSCOMMAND) and (Msg.wParam and $FFFF = MNUID_ABOUT) then
        AboutBox.ShowModal;
end;

```

Figura I2-2:  
El elemento  
de menú  
añadido

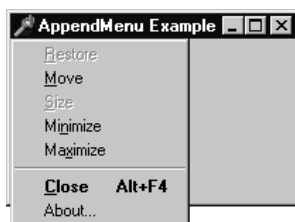


Tabla 12-2: Valores del parámetro uFlags de AppendMenu

Valor	Descripción
MF_BITMAP	Indica que se utiliza un mapa de bits para representar el elemento de menú. El manejador del mapa de bits es especificado en el parámetro lpNewItem. Esta opción no puede ser combinada con las opciones MF_STRING o MF_OWNERDRAW.
MF_CHECKED	Coloca el mapa de bits de la marca de verificación del menú junto al elemento de menú. Esta opción no puede ser combinada con la opción MF_UNCHECKED.
MF_DISABLED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado. Esta opción no puede ser combinada con las opciones MF_ENABLED o MF_GRAYED.
MF_ENABLED	Habilita el elemento de menú de manera que pueda ser seleccionado, y lo restaura si estaba sombreado. Esta opción no puede ser combinada con las opciones MF_DISABLED o MF_GRAYED.
MF_GRAYED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado y lo dibuja sombreado (en gris). Esta opción no puede ser combinada con las opciones MF_ENABLED o MF_DISABLED.
MF_MENUBARBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna y las columnas son separadas mediante una línea vertical. Esta opción no puede ser combinada con la opción MF_MENUBREAK.
MF_MENUBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna. No hay indicación visual de separación de columnas. Esta opción no puede ser combinada con la opción MF_MENUBARBREAK.
MF_OWNERDRAW	Indica un elemento de menú dibujado por el propietario. Cuando el elemento de menú es mostrado por primera vez, la ventana propietaria recibe un mensaje WM_MEASUREITEM, permitiendo a la aplicación especificar el ancho y altura del elemento de menú. La ventana propietaria recibe un mensaje WM_DRAWITEM cuando el elemento de menú va a ser dibujado, permitiendo a la aplicación dibujar el elemento de menú con la apariencia deseada. Esta opción no puede ser combinada con las opciones MF_BITMAP o MF_STRING.
MF_POPUP	Indica que el elemento de menú abre un menú desplegable o submenú. El manejador del menú desplegable o submenú es especificado en el parámetro ulDNewItem.

Valor	Descripción
MF_SEPARATOR	Indica un separador de menú. Este elemento de menú es dibujado como una línea horizontal y es válido solo cuando es usado con menús desplegables, menús emergentes o submenús. Este elemento de menú no puede ser resaltado, sombreado, o deshabilitado y los parámetros <code>uIDNewItem</code> y <code>lpNewItem</code> son ignorados.
MF_STRING	Indica que se utiliza una cadena de caracteres para representar el elemento de menú. El parámetro <code>lpNewItem</code> contiene un puntero a la cadena terminada en nulo mostrada en el elemento de menú. Esta opción no puede ser combinada con las opciones MF_BITMAP o MF_OWNERDRAW.
MF_UNCHECKED	Coloca el mapa de bits que indica que el elemento de menú está desmarcado junto al elemento de menú. Por defecto, ésta imagen es una imagen en blanco. Esta opción no puede ser combinada con la opción MF_CHECKED.

**CheckMenuItem****Windows.Pas****Sintaxis**

```

CheckMenuItem(
    hMenu: HMENU;           {manejador de menú}
    uIDCheckItem: UINT;      {identificador o posición del elemento de menú a marcar}
    uCheck: UINT             {opciones de elemento de menú}
): DWORD;                  {devuelve el estado anterior de la marca}

```

**Descripción**

Esta función asigna el estado de la marca al elemento de menú especificado. El elemento de menú no puede estar situado en una barra de menú; tiene que estar en un submenú, menú desplegable, o menú emergente. Si el elemento de menú especificado abre un submenú, el parámetro `uIDCheckItem` debe contener el identificador de elemento de menú.

**Parámetros**

**hMenu:** Manejador del menú que contiene el elemento que será marcado o desmarcado.

**uIDCheckItem:** Especifica el elemento de menú cuyo atributo será cambiado. Este parámetro contiene un identificador de elemento de menú o una posición (de base cero), según determina el parámetro `uCheck`.

**uCheck:** Una combinación de opciones que determinan cómo es interpretado el parámetro `uIDCheckItem` y si el elemento de menú es marcado o desmarcado. Este parámetro puede contener las opciones MF\_BYCOMMAND o MF\_BYPOSITION combinadas con MF\_CHECKED o MF\_UNCHECKED, según se describe en la Tabla 12-3.

Valor que devuelve

Si la función tiene éxito, devuelve un valor *MF\_CHECKED* o *MF\_UNCHECKED* correspondiente al estado anterior de la marca del elemento de menú; en caso contrario, devuelve \$FFFFFFFF.

Véase además

*CheckMenuRadioItem*, *GetMenuItemID*, *GetMenuItemInfo*, *GetMenuState*, *GetSystemMetrics*, *SetMenuItemBitmaps*, *SetMenuItemInfo*

Ejemplo

Vea el Listado 12-3 bajo *CreateMenu*.

Tabla 12-3: Valores del parámetro *uCheck* de *CheckMenuItem*

Valor	Descripción
MF_BYCOMMAND	Indica que el parámetro <i>uIDCheckItem</i> contiene el identificador del elemento de menú. Esta opción es utilizada por defecto si no hay ninguna especificada, y no puede ser combinada con MF_BYPOSITION.
MF_BYPOSITION	Indica que el parámetro <i>uIDCheckItem</i> contiene la posición del elemento de menú en el menú, contando a partir de cero. Esta opción no puede ser combinada con MF_BYCOMMAND.
MF_CHECKED	Coloca el mapa de bits de marca de verificación de elementos de menú junto al elemento de menú. Por defecto, ésta imagen tiene la forma de una marca de verificación. Esta opción no puede ser combinada con MF_UNCHECKED.
MF_UNCHECKED	Coloca el mapa de bits que indica que el elemento de menú está desmarcado junto al elemento de menú. Por defecto, ésta imagen es una imagen en blanco. Esta opción no puede ser combinada con la opción MF_CHECKED.

CheckMenuRadioItem Windows.Pas

Sintaxis

CheckMenuRadioItem(  
  hMenu: HMENU;  
  First: UINT;  
  
  Last: UINT;  
  
  Check: UINT;  
  Flags: UINT  
); BOOL;

{manejador de menú}  
{primer identificador de elemento de menú o posición en el grupo}  
{último identificador de elemento de menú o posición en el grupo}  
{identificador o posición del elemento de menú a marcar}  
{opciones del elemento de menú}  
{devuelve TRUE o FALSE}

**Descripción**

Esta función asigna la marca al elemento de menú identificado por el parámetro *Check*, mientras limpia las marcas de todos los demás elementos de menú en el grupo de elementos identificado por los parámetros *First* y *Last*. Específicamente, se asignan los atributos *MFT\_RADIOCHECK* y *MFS\_CHECKED* al elemento de menú identificado por el parámetro *Check* y se “limpian” los atributos de todos los demás. Consulte la función *GetMenuItemInfo* para ver una explicación más detallada de estas opciones. Por defecto, el elemento de menú marcado mostrará un pequeño círculo (una viñeta) en lugar de una marca de verificación.

**Parámetros**

*hMenu*: Manejador del menú que contiene el grupo de elementos de menú que será modificado.

*First*: Especifica el primer elemento de menú en el grupo. Este parámetro contiene un identificador de elemento de menú o una posición de base cero, según determina el valor del parámetro *Flags*.

*Last*: Especifica el último elemento de menú en el grupo. Este parámetro contiene un identificador de elemento de menú o una posición de base cero, según determina el valor del parámetro *Flags*.

*Check*: Especifica el elemento de menú que será marcado. Este parámetro contiene un identificador de elemento de menú o una posición de base cero, según determina el valor del parámetro *Flags*.

*Flags*: Valor que indica cómo deben ser interpretados los parámetros *First*, *Last* y *Check*. Si este parámetro contiene el valor *MF\_BYCOMMAND*, los parámetros *First*, *Last* y *Check* contienen identificadores de elementos de menú. Si este parámetro contiene el valor *MF\_BYPOSITION*, los demás parámetros contienen las posiciones de los elementos dentro del menú (contando a partir de cero).

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener más información sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CheckMenuItem*, *GetMenuItemInfo*, *GetMenuState*, *SetMenuItemBitmaps*, *SetMenuItemInfo*

**Ejemplo**

Vea el Listado 12-4 bajo *CreatePopupMenu*.

**CreateMenu**      **Windows.Pas****Sintaxis**

CreateMenu: HMENU;                      {devuelve un manejador de un nuevo menú}

**Descripción**

Esta función crea un menú vacío y devuelve su manejador. A este nuevo menú podrán añadirse luego elementos de menú y submenús mediante las funciones *AppendMenu*, *InsertMenu* e *InsertMenuItem*. Cuando la aplicación termina de utilizar el menú, debe eliminarlo usando la función *DestroyMenu*. Sin embargo, si el menú es asignado a una ventana mediante la función *SetMenu*, Windows liberará automáticamente los recursos de menú cuando la aplicación termine.



**Nota:** Windows 95/98 puede manipular un máximo de 16.364 manejadores de menú.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del menú recién creado; en caso contrario, devuelve cero.

**Véase además**

*AppendMenu*, *CreatePopupMenu*, *DestroyMenu*, *InsertMenu*, *InsertMenuItem*, *IsMenu*, *SetMenu*

**Ejemplo****Listado I2-3: Creando un menú a la vieja usanza**

```

const
  MNUID_MAINMENU = 1;           // identificadores de elementos de menú
  MNUID_ITEM1    = 2;
  MNUID_ITEM2    = 3;
  MNUID_ITEM3    = 4;
  MNUID_ITEM4    = 5;

var
  Form1: TForm1;
  MainMenu, SubMenu1: HMENU;    // manejadores de menú principal y desplegable

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  MenuInfo: TMenuItemInfo;      // almacena información del elemento de menú

```

```

begin
    {Comienza por crear el menú principal y el desplegable}
    MainMenu := CreateMenu;
    SubMenu1 := CreatePopupMenu;

    {Inicializa el registro del elemento de menú para indicar que el identificador
    de elemento de menú es válido y que el menú contendrá una cadena}
    MenuInfo.cbSize := SizeOf(TMenuItemInfo);
    MenuInfo.fMask := MIIM_TYPE or MIIM_ID;
    MenuInfo.fType := MFT_STRING;

    {Inserta el primer elemento de menú en el menú desplegable}
    MenuInfo.wID := MNUID_ITEM1;
    MenuInfo.dwTypeData := 'Item 1';
    InsertMenuItem(SubMenuItem, 0, TRUE, MenuInfo);

    {Inserta el segundo elemento de menú al final del menú desplegable}
    MenuInfo.wID := MNUID_ITEM2;
    MenuInfo.dwTypeData := 'Item 2';
    InsertMenuItem(SubMenuItem, $FFFF, TRUE, MenuInfo);

    {Inserta el tercer elemento de menú al final del menú desplegable}
    MenuInfo.wID := MNUID_ITEM3;
    MenuInfo.dwTypeData := 'Item 3';
    InsertMenuItem(SubMenuItem, $FFFF, TRUE, MenuInfo);

    {Inserta el cuarto elemento de menú al final del menú desplegable}
    MenuInfo.wID := MNUID_ITEM4;
    MenuInfo.dwTypeData := 'Item 4';
    InsertMenuItem(SubMenuItem, $FFFF, TRUE, MenuInfo);

    {Reinicializa el registro de elemento de menú para indicar que estamos
    insertando un elemento de menú que activará un menú desplegable}
    MenuInfo.wID := MNUID_MAINMENU;
    MenuInfo.fMask := MIIM_TYPE or MIIM_ID or MIIM_SUBMENU;
    MenuInfo.dwTypeData := 'Main Menu';
    MenuInfo.hSubMenu := SubMenuItem;

    {Inserta el elemento de menú en el menú principal}
    InsertMenuItem(MainMenu, 0, TRUE, MenuInfo);

    {Asigna el menú principal que fue creado como menú de la ventana. Esto hace que
    la aplicación dibuje la barra de menú que contiene el menú principal}
    SetMenu(Form1.Handle, MainMenu);
end;

{El manejador redefinido del mensaje WM_COMMAND}
procedure TForm1.WMCommand(var Msg: TWMCommand);
begin
    {Si este mensaje fue enviado desde un elemento de menú (cero indica un menú)...}
    if Msg.NotifyCode = 0 then
    begin
        {...recupera el estado del menú y lo marca}
        if (GetMenuState(MainMenu, Msg.ItemID, MF_BYCOMMAND) and MF_CHECKED) =

```

```

        MF_CHECKED then
            CheckMenuItem(MainMenu, Msg.ItemID, MF_BYCOMMAND or MF_UNCHECKED)
        else
            CheckMenuItem(MainMenu, Msg.ItemID, MF_BYCOMMAND or MF_CHECKED);

        {Indica que el mensaje fue manejado}
        Msg.Result := 0;
    end
else
    {En caso contrario, no fue ninguno de nuestros nuevos cuatro elementos de menú
    quien generó el mensaje, dejemos que Windows lo maneje}
    inherited;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    {Para llamar la atención sobre el menú, podemos asignar su estado resaltado.}
    if (GetMenuState(MainMenu, 0, MF_BYPOSITION) and MF_HILITE) = MF_HILITE then
        HiliteMenuItem(Form1.Handle, MainMenu, 0, MF_BYPOSITION or MF_UNHILITE)
    else
        HiliteMenuItem(Form1.Handle, MainMenu, 0, MF_BYPOSITION or MF_HILITE);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {Hemos terminado con los menús y los eliminamos a ambos: tanto al menú
    desplegable, como al principal}
    DeleteMenu(MainMenu, 0, MF_BYPOSITION);

    {Observe que debido a que el menú fue asignado a una ventana usando la función
    SetMenu, este menú será liberado automáticamente y la siguiente línea de código
    es innecesaria}
    DestroyMenu(MainMenu);
end;

```

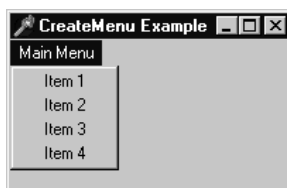


Figura 12-3:  
La estructura  
del menú

## CreatePopupMenu Windows.Pas

### Sintaxis

CreatePopupMenu: HMENU; {devuelve un manejador de un menú emergente}

### Descripción

Esta función crea un menú emergente vacío y devuelve su manejador. El menú resultante puede ser utilizado como menú emergente, submenú o menú desplegable. A

este nuevo menú podrán añadirse luego elementos de menú y submenús mediante las funciones *AppendMenu*, *InsertMenu* e *InsertMenuItem*. Si el menú va a ser usado como menú emergente, puede ser mostrado mediante las funciones *TrackPopupMenu* o *TrackPopupMenuEx*. Cuando la aplicación termina de utilizar el menú, debe eliminarlo usando la función *DestroyMenu*.



**Nota:** Windows 95/98 puede manipular un máximo de 16.364 manejadores de menú.

#### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del menú recién creado; en caso contrario, devuelve cero.

#### Véase además

*AppendMenu*, *CreateMenu*, *DestroyMenu*, *InsertMenu*, *InsertMenuItem*, *IsMenu*, *TrackPopupMenu*, *TrackPopupMenuEx*

#### Ejemplo

##### Listado I2-4: Creando un menú emergente y utilizándolo en el menú de sistema

```
const
  {Identificadores de elementos de menú. Debido a que estos elementos de menú
   serán situados en el menú de sistema, y el mensaje WM_SYSCOMMAND utiliza
   internamente los últimos 4 bits del parámetro wParam, los identificadores de
   elementos de menú deben ser mayores que 15}
  MNUID_BLACK = $1000;
  MNUID_NORMAL = $2000;

var
  Form1: TForm1;
  SysPopup: HMENU; // almacena un manejador de nuestro menú emergente

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  {Crea el menú emergente, que será usado como un menú desplegable en el menú
   de sistema}
  SysPopup := CreatePopupMenu;

  {Le añade los dos elementos de menú}
  AppendMenu(SysPopup, MF_STRING, MNUID_BLACK, 'Black');
  AppendMenu(SysPopup, MF_STRING, MNUID_NORMAL, 'Normal');
```

```

{Inserta primero un separador}
InsertMenu(GetSystemMenu(Form1.Handle, FALSE), 0,
           MF_BYPOSITION or MF_SEPARATOR, 0, nil);

{Inserta al principio del menú de sistema un elemento que mostrará el menú
desplegable creado}
InsertMenu(GetSystemMenu(Form1.Handle, FALSE), 0,
           MF_BYPOSITION or MF_POPUP or MF_STRING, SysPopup, 'Color');

{Marca de modo exclusivo el elemento de menú 'Normal'}
CheckMenuRadioItem(SysPopup, 0, 1, 1, MF_BYPOSITION);

{Instala un manejador de mensaje para recibir mensajes desde los elementos
del menú emergente}
Application.OnMessage := ApplicationMessage;
end;

procedure TForm1.ApplicationMessage(var Msg: TMsg; var Handled: Boolean);
begin
    {Si es un mensaje de menú y es el elemento de menú 'Black'...}
    if ((Msg.Message = WM_SYSCOMMAND) or (Msg.Message = WM_COMMAND)) and
        (Msg.WParam and $FFFF = MNUID_BLACK) then
        begin
            {...marca el elemento de menú 'Black' y cambia el color del formulario}
            CheckMenuRadioItem(SysPopup, 0, 1, 0, MF_BYPOSITION);
            Form1.Color := clBlack;
        end;
    end;

    {Si es un mensaje de menú y es el elemento de menú 'Normal'...}
    if ((Msg.Message = WM_SYSCOMMAND) or (Msg.Message = WM_COMMAND)) and
        (Msg.WParam and $FFFF = MNUID_NORMAL) then
        begin
            {...marca el elemento de menú 'Normal' y cambia el color del formulario}
            CheckMenuRadioItem(SysPopup, 0, 1, 1, MF_BYPOSITION);
            Form1.Color := clBtnFace;
        end;
    end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
var
    MenuPoint: TPoint; // usado para convertir las coordenadas del ratón
begin
    {Si el botón derecho fue pulsado...}
    if Button = mbRight then
        begin
            {...convierte las coordenadas del ratón a coordenadas de la pantalla}
            MenuPoint.X := X;
            MenuPoint.Y := Y;
            MenuPoint := ClientToScreen(MenuPoint);

            {Recupera el menú desplegable desde el menú del sistema y muestra el menú
            desplegable como un menú emergente}
            TrackPopupMenu(GetSubMenu(GetSystemMenu(Form1.Handle, FALSE), 0),

```

```

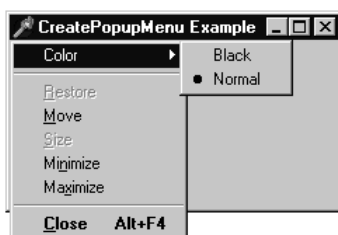
TPM_CENTERALIGN or TPM_LEFTBUTTON, MenuPoint.X, MenuPoint.Y,
0, Form1.Handle, nil);

end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {No necesitamos ya el menú emergente, lo borramos}
    DeleteMenu(GetSystemMenu(Form1.Handle, FALSE), 0, MF_BYPOSITION);
end;

```

Figura 12-4:  
El submenú en  
el menú de  
sistema



## DeleteMenu Windows.Pas

### Sintaxis

```

DeleteMenu(
    hMenu: HMENU;           {manejador de menú}
    uPosition: UINT;         {identificador o posición del elemento a eliminar}
    uFlags: UINT             {opciones de elemento de menú}
): BOOL;                   {devuelve TRUE o FALSE}

```

### Descripción

Esta función elimina un elemento de menú del menú especificado. Si el elemento de menú abre un menú desplegable o submenú, esta función elimina el manejador del menú desplegable o submenú, liberando sus recursos asociados.

### Parámetros

*hMenu*: Manejador del menú que contiene el elemento de menú que será eliminado.

*uPosition*: Especifica el elemento de menú que será eliminado. Este parámetro contiene un identificador de elemento de menú o una posición de base cero, según determina el valor del parámetro *uFlags*.

*uFlags*: Valor que indica cómo debe ser interpretado el parámetro *uPosition*. Si este parámetro contiene el valor *MF\_BYCOMMAND*, el parámetro *uPosition* contiene un identificador de elemento de menú. Si este parámetro contiene la opción *MF\_BYPOSITION*, el parámetro *uFlags* contiene una posición de base cero.

**Valor que devuelve**

Si esta función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*AppendMenu*, *CreatePopupMenu*, *DestroyMenu*, *InsertMenu*, *InsertMenuItem*, *RemoveMenu*

**Ejemplo**

Vea el Listado 12-4 bajo *CreatePopupMenu*.

**DestroyMenu      Windows.Pas****Sintaxis**

```
DestroyMenu(
    hMenu: HMENU           {manejador del menú que será destruido}
): BOOL;                  {devuelve TRUE o FALSE}
```

**Descripción**

Esta función destruye el menú especificado y libera todos sus recursos asociados. Cualquier menú que no haya sido asignado a una ventana mediante la función *SetMenu* debe ser destruido utilizando la función *DestroyMenu*. Los menús asignados a una ventana son automáticamente destruidos cuando la función termina y no necesitan ser destruidos explícitamente.

**Parámetros**

*hMenu*: Manejador del menú que será destruido.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreateMenu*, *CreatePopupMenu*, *DeleteMenu*, *RemoveMenu*

**Ejemplo**

Vea el Listado 12-3 bajo *CreateMenu*.

**EnableMenuItem****Windows.Pas****Sintaxis**

```
EnableMenuItem(
  hMenu: HMENU;           {manejador de menú}
  uIDEnableItem: UINT;    {identificador de elemento de menú o posición}
  uEnable: UINT           {opciones de elemento de menú}
): BOOL;                  {devuelve TRUE o FALSE}
```

**Descripción**

Esta función habilita, deshabilita, o sombrea (muestra en gris) un elemento de menú del menú especificado.

**Parámetros**

*hMenu*: Manejador del menú que contiene el elemento de menú cuyo estado será modificado.

*uIDEnableItem*: Especifica el elemento de menú que será habilitado, deshabilitado, o sombreado. Este parámetro contiene un identificador de elemento de menú o una posición de base cero, según determina el valor del parámetro *uEnable*.

*uEnable*: Combinación de opciones que determina cómo el parámetro *uIDEnableItem* debe ser interpretado y si el elemento de menú será habilitado, deshabilitado, o sombreado. Este parámetro puede contener los valores *MF\_BYCOMMAND* o *MF\_BYPOSITION* combinados con una de las siguientes opciones: *MF\_DISABLED*, *MF\_ENABLED*, o *MF\_GRAYED*. Estas opciones se describen en la Tabla 12-4. Observe que si este parámetro incluye la opción *MF\_BYCOMMAND*, todos los submenús alcanzables desde los elementos del menú identificado por el parámetro *hMenu* serán recorridos en busca del identificador de elemento de menú especificado. De esta manera, utilizando la opción *MF\_BYCOMMAND* (y asumiendo que no hay identificadores de menús duplicados), sólo es necesario especificar el manejador del menú principal.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*AppendMenu*, *CreateMenu*, *CreatePopupMenu*, *GetMenuItemID*, *GetMenuItemInfo*, *InsertMenu*, *InsertMenuItem*, *ModifyMenu*, *SetMenuItemInfo*

**Ejemplo****Listado 12-5: Habilitando y deshabilitando el elemento Cerrar del menú de sistema**

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SysMenu: HMENU;    // almacena un manejador del menú de sistema
```

```
begin
{Recupera un manejador del menú de sistema}
SysMenu := GetSystemMenu(Form1.Handle, FALSE);

{Si el elemento de menú Cerrar está deshabilitado...}
if (GetMenuState(SysMenu, 6, MF_BYPOSITION) and MF_GRAYED) = MF_GRAYED then
begin
{...habilitarlo...}
EnableMenuItem(SysMenu, 6, MF_BYPOSITION or MF_ENABLED);
Button1.Caption := 'Disable Close menu item'
end
else
begin
{...en caso contrario, deshabilitarlo y sombreadarlo}
EnableMenuItem(SysMenu, 6, MF_BYPOSITION or MF_GRAYED);
Button1.Caption := 'Enable Close menu item';
end;
end;
```

Figura 12-5:  
El elemento  
Cerrar del  
menú de  
sistema  
deshabilitado

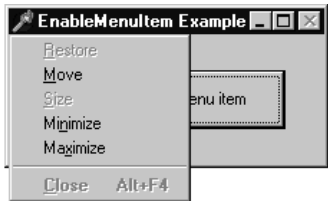


Tabla 12-4: Valores del parámetro uEnable de EnableMenuItem

Valor	Descripción
MF_BYCOMMAND	Indica que el parámetro ulDenableItem contiene el identificador de un elemento de menú. Esta opción es utilizada por defecto si no se especifica ninguna, y no puede ser combinada con la opción MF_BYPOSITION.
MF_BYPOSITION	Indica que el parámetro ulDenableItem contiene la posición de base cero del elemento de menú. Esta opción no puede ser combinada con MF_BYCOMMAND.
MF_DISABLED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado, sin sombreadarlo. Esta opción no puede ser combinada con MF_ENABLED o MF_GRAYED.
MF_ENABLED	Habilita el elemento de menú de manera que pueda ser seleccionado, restaurándolo si estaba sombreado. Esta opción no puede ser combinada con MF_DISABLED o MF_GRAYED.
MF_GRAYED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado y lo sombrea. Esta opción no puede ser combinada con MF_ENABLED o MF_DISABLED.

**GetMenu**      **Windows.Pas****Sintaxis**

```
GetMenu(
  hWnd: HWND;           {manejador de ventana cuyo menú será recuperado}
): HMENU;               {devuelve un manejador de menú}
```

**Descripción**

Esta función devuelve un manejador del menú asignado a la ventana especificada.

**Parámetros**

*hWnd*: Manejador de la ventana cuyo manejador de menú es recuperado.

**Valor que devuelve**

Si la función tiene éxito, devuelve un manejador del menú asignado a la ventana. Si la ventana no tiene menú o la función falla, devuelve cero. Si la ventana especificada es una ventana hija, el valor devuelto es indefinido.

**Véase además**

*CreateMenu*, *GetSubMenu*, *GetSystemMenu*, *SetMenu*

**Ejemplo****Listado I2-6: Controlando a Delphi a través de sus menús**

```
var
  Form1: TForm1;
  DelphiWindow: HWND;      // manejador de la ventana principal de Delphi
  ReopenMenu: HMENU;       // manejador de un submenú de Delphi

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  DelphiMainMenu,          // manejador del menú principal de Delphi
  DelphiFileMenu: HMENU;   // manejador del menú desplegable 'File' de Delphi
begin
  {Recupera un manejador de la ventana principal de Delphi}
  DelphiWindow := FindWindow('TAppBuilder', NIL);

  {Recupera un manejador del menú principal de Delphi}
  DelphiMainMenu := GetMenu(DelphiWindow);

  {Recupera un manejador del menú desplegable 'File' de Delphi}
  DelphiFileMenu := GetSubMenu(DelphiMainMenu, 0);

  {Recupera un manejador submenú 'Reopen' de Delphi}
  ReopenMenu := GetSubMenu(DelphiFileMenu, 5);
```

```

end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  MenuPos: TPoint; // las coordenadas de pantalla del ratón
begin
  {Si el botón derecho del ratón fue pulsado...}
  if Button = mbRight then
  begin
    {...convierte las coordenadas del ratón a coordenadas de la pantalla}
    MenuPos.X := X;
    MenuPos.Y := Y;
    MenuPos := ClientToScreen(MenuPos);

    {Muestra el menú 'Reopen' de Delphi como un menú emergente}
    TrackPopupMenu(ReopenMenu, TPM_RIGHTALIGN, MenuPos.X, MenuPos.Y, 0,
      Form1.Handle, nil);
  end;
end;

{El manejador redefinido del mensaje WM_COMMAND}
procedure TForm1.WMCommand(var Msg: TMessage);
begin
  {Redireccionando el mensaje desde el menú emergente hacia Delphi, podemos
  controlar a Delphi desde nuestra copia del menú de la misma manera que si
  se hiciera clic en la opción de menú del IDE de Delphi}
  PostMessage(DelphiWindow, Msg.Msg, Msg.WParam, Msg.LParam);
inherited;
end;

```

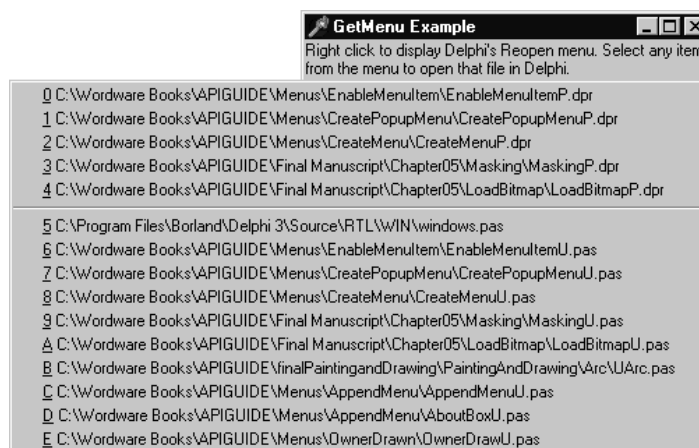


Figura 12-6:  
El menú  
'Reopen' de  
Delphi

**GetMenuDefaultItem      Windows.Pas****Sintaxis**

```

GetMenuDefaultItem(
    hMenu: HMENU;           {manejador de menú}
    fByPos: UINT;           {opción de identificador o posición}
    gmdiFlags: UINT         {opciones de búsqueda}
): UINT;                   {devuelve un identificador de elemento o posición}

```

**Descripción**

Esta función recupera la posición o identificador del elemento de menú predeterminado (por defecto) del menú especificado. A menos que se indique lo contrario, la función no devolverá el elemento de menú predeterminado si éste está deshabilitado.

**Parámetros**

*hMenu*: Manejador del menú cuyo elemento de menú predeterminado será recuperado.

*fByPos*: Valor que indica si se desea obtener el identificador de elemento de menú o su posición con base cero. Si este parámetro es cero, la función devuelve el identificador del elemento de menú predeterminado. Si es 1, la función devuelve la posición de base cero del elemento de menú predeterminado.

*gmdiFlags*: Combinación de valores que determinan cómo la función busca en el menú especificado para encontrar su elemento de menú predeterminado. Este parámetro puede ser cero o una combinación de uno o más valores de la Tabla 12-5.

**Valor que devuelve**

Si la función tiene éxito, devuelve el identificador o la posición del elemento de menú predeterminado; en caso contrario, devuelve -1.

**Véase además**

*GetMenuItemInfo*, *SetMenuDefaultItem*, *SetMenuItemInfo*

**Ejemplo**

Vea el Listado 12-10 bajo *SetMenuDefaultItem*.

Tabla 12-5: Valores del parámetro `gmdiFlags` de `GetMenuDefaultItem`

Valor	Descripción
<code>GMDI_GOINTOPOPUPS</code>	Indica que si el elemento predeterminado del menú despliega un submenú, la función debe buscar en el submenú un elemento predeterminado. Si el submenú no contiene un elemento predeterminado, será devuelto el elemento que despliega el submenú. Si el elemento predeterminado de un submenú abre a su vez otro submenú, la búsqueda continúa recursivamente hasta encontrar un elemento por defecto.
<code>GMDI_USEDISABLED</code>	Indica que la función devolverá el elemento de menú predeterminado incluso aunque estuviese deshabilitado.

**GetMenuItemCount** *Windows.Pas**Sintaxis*

```
GetMenuItemCount(
    hMenu: HMENU           {manejador de menú}
): Integer;               {devuelve la cantidad de elementos de menú en el menú}
```

*Descripción*

Esta función devuelve la cantidad de elementos de menú del menú especificado.

*Parámetros*

*hMenu*: Manejador del menú cuya cantidad de elementos de menú se desea obtener.

*Valor que devuelve*

Si la función tiene éxito, devuelve la cantidad de elementos de menú contenidos en el menú especificado; en caso contrario, devuelve -1. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

*Véase además*

*GetMenu*, *GetMenuItemID*, *GetSubMenu*, *GetMenuItemInfo*, *GetMenuString*

*Ejemplo*

Vea el Listado 12-10 bajo *SetMenuDefaultItem*.

**GetMenuItemID** *Windows.Pas**Sintaxis*

```
GetMenuItemID(
    hMenu: HMENU;          {manejador de menú}
    nPos: Integer           {posición del elemento de menú}
```

); UINT; {devuelve un identificador de elemento de menú}

### Descripción

Esta función devuelve el identificador del elemento de menú situado en la posición *nPos* del menú especificado.

### Parámetros

*hMenu*: Manejador del menú que contiene el elemento de menú cuyo identificador va a ser recuperado.

*nPos*: Especifica la posición (con base cero) del elemento de menú cuyo identificador va a ser recuperado.

### Valor que devuelve

Si la función tiene éxito, devuelve el identificador del elemento de menú especificado. Si la función falla, el identificador del elemento de menú especificado es nulo, el elemento de menú especificado no existe o el elemento de menú abre un submenú, la función devuelve \$FFFFFFFF.

### Véase además

*GetMenuItemCount*, *GetMenuItemID*, *GetMenuItemInfo*, *GetMenuString*

### Ejemplo

Vea el Listado 12-10 bajo *SetMenuDefaultItem*.

## GetMenuItemInfo

## Windows.Pas

### Sintaxis

```
GetMenuItemInfo(
  p1: HMENU;           {manejador de menú}
  p2: UINT;             {identificador de elemento de menú o posición}
  p3: BOOL;             {opción de identificador o posición}
  var p4: TMenuItemInfo {puntero a registro TMenuItemInfo}
): BOOL;               {devuelve TRUE o FALSE}
```

### Descripción

Esta función recupera información específica sobre el elemento de menú especificado por el parámetro *p2*, perteneciente al menú identificado por su manejador *p1*.

### Parámetros

*p1*: Manejador del menú que contiene el elemento de menú cuya información va a ser recuperada.

*p2*: Especifica el identificador de elemento de menú o la posición (de base cero) del elemento de menú cuya información va a ser recuperada. El valor del parámetro *p3* indica cómo la función debe interpretar el valor de este parámetro.

*p3*: Indica cómo la función debe interpretar el valor del parámetro *p2*. Si a este parámetro se le asigna TRUE, el valor del parámetro *p2* indica la posición del elemento de menú, contando a partir de cero. Si a este parámetro se le asigna FALSE, *p2* contiene un identificador de elemento de menú.

*p4*: Puntero a un registro *TMenuItemInfo* que contiene valores que especifican la información a recuperar. Este registro recibe la información solicitada cuando la función retorna. El registro *TMenuItemInfo* se define como:

*TMenuItemInfo* = **packed record**

<i>cbSize</i> : UINT;	{el tamaño del registro}
<i>fMask</i> : UINT;	{opción de asignación o recuperación}
<i>fType</i> : UINT;	{opciones de tipo del elemento de menú}
<i>fState</i> : UINT;	{opciones de estado del elemento de menú}
<i>wID</i> : UINT;	{el identificador del elemento de menú}
<i>hSubMenu</i> : HMENU;	{manejador de menú desplegable o submenú}
<i>hbmChecked</i> : HBITMAP;	{manejador del mapa de bits para el estado 'marcado'}
<i>hbmUnchecked</i> : HBITMAP;	{manejador del mapa de bits para el estado 'desmarcado'}
<i>dwItemData</i> : DWORD;	{valor definido por la aplicación}
<i>dwTypeData</i> : PAnsiChar;	{contenido del elemento de menú}
<i>cch</i> : UINT;	{longitud del texto del elemento de menú}

**end;**

*cbSize*: Especifica el tamaño del registro, en bytes. A este campo debe asignársele *SizeOf(TMenuItemInfo)*.

*fMask*: Valor que determina qué campos de la estructura se desea asignar o recuperar. Este campo puede contener una combinación de uno o más valores de la Tabla 12-6.

*fType*: Valor que indica el tipo del elemento de menú. Este campo puede contener una combinación de uno o más valores de la Tabla 12-7.

*fState*: Valor que indica el estado del elemento de menú. Este campo puede contener una combinación de uno o más valores de la Tabla 12-8.

*wID*: El identificador del elemento de menú.

*hSubMenu*: Manejador del menú desplegable o submenú que es activado cuando el elemento de menú es seleccionado. Si el elemento de menú no activa un submenú o menú desplegable, este campo tendrá valor cero.

*hbmChecked*: Manejador del mapa de bits que será mostrado cuando el elemento de menú sea marcado. Si a este campo se le asigna cero, se utilizará la imagen predeterminada de marca de verificación. Si el campo *fType* incluye la

opción *MFT\_RADIOCHECK*, la marca de verificación tendrá la apariencia de una viñeta.

*hbmUnchecked*: Manejador del mapa de bits que será mostrado cuando el elemento de menú sea desmarcado. Si a este campo se le asigna cero, no se utilizará ningún mapa de bits.

*dwItemData*: Un valor de 32 bits definido por la aplicación asociado con el elemento de menú especificado. Si el campo *fType* incluye la opción *MFT\_OWNERDRAW*, este valor estará contenido en el campo *itemData* del registro al que apuntan los mensajes *WM\_MEASUREITEM* y *WM\_DRAWITEM*. La ventana propietaria del menú recibe estos mensajes cuando el menú es creado o dibujado.

*dwTypeData*: Un puntero a datos adicionales asociados con el elemento de menú. El contenido de este parámetro depende de las opciones especificadas en el campo *fType*. Si el campo *fType* incluye la opción *MFT\_BITMAP*, este campo contiene un manejador de un mapa de bits. Si el campo *fType* contiene la opción *MFT\_STRING*, este campo apunta a una cadena de caracteres terminada en nulo. Si el campo *fType* contiene la opción *MFT\_OWNERDRAW*, este campo contiene un valor de 32 bits definido por la aplicación.

*cch*: La longitud del texto del elemento de menú, en caso de que el campo *fType* incluya la opción *MFT\_STRING* y la información esté siendo recuperada. Este campo es ignorado cuando se asigna el texto del elemento de menú.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*EnableMenuItem*, *GetMenuItemCount*, *GetMenuItemID*, *GetMenuItemRect*, *GetMenuState*, *GetMenuString*, *HiliteMenuItem*, *InsertMenuItem*, *ModifyMenu*, *SetMenuItemBitmaps*, *SetMenuItemInfo*

#### Ejemplo

Vea el Listado 12-1 bajo el ejemplo de menú dibujado por el propietario, en la introducción del capítulo.

**Tabla 12-6: Valores del campo *p4.fMask* de *GetMenuItemInfo***

Valor	Descripción
MIIM_CHECK MARKS	Recupera o asigna el valor de los campos <i>hbmChecked</i> y <i>hbmUnchecked</i> .
MIIM_DATA	Recupera o asigna el valor del campo <i>dwItemData</i> .
MIIM_ID	Recupera o asigna el valor del campo <i>wID</i> .
MIIM_STATE	Recupera o asigna el valor del campo <i>fState</i> .

Valor	Descripción
MIIM_SUBMENU	Recupera o asigna el valor del campo hSubMenu.
MIIM_TYPE	Recupera o asigna el valor de los campos fType y dwTypeData.

Tabla 12-7: Valores del campo p4.fType de GetMenuItemInfo

Valor	Descripción
MFT_BITMAP	Indica que se utiliza un mapa de bits para representar el elemento de menú. El manejador del mapa de bits es especificado en el parámetro dwTypeData. Esta opción no puede ser combinada con las opciones MFT_STRING o MFT_OWNERDRAW.
MFT_MENUBARBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna y las columnas son separadas mediante una línea vertical. Esta opción no puede ser combinada con la opción MFT_MENUBREAK.
MFT_MENUBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna. No hay indicación visual de separación de columnas. Esta opción no puede ser combinada con la opción MFT_MENUBARBREAK.
MFT_OWNERDRAW	Indica un elemento de menú dibujado por el propietario. Cuando el elemento de menú es mostrado por primera vez, la ventana propietaria recibe un mensaje WM_MEASUREITEM, permitiendo a la aplicación especificar el ancho y altura del elemento de menú. La ventana propietaria recibe un mensaje WM_DRAWITEM cuando el elemento de menú va a ser dibujado, permitiendo a la aplicación dibujar el elemento de menú con la apariencia deseada. Esta opción no puede ser combinada con las opciones MFT_BITMAP o MFT_STRING.
MFT_RADIOCHECK	Indica que la imagen de la marca de verificación usada para elementos de menú es una viñeta, si el campo hbmpChecked contiene cero.
MFT_RIGHTJUSTIFY	Justifica a la derecha el elemento de menú y los elementos de menú subsiguientes. Esta opción es válida sólo para elementos de menú situados en una barra de menú.
MFT_SEPARATOR	Indica un separador de menú. Este elemento de menú es dibujado como una línea horizontal y es válido solo cuando es usado con menús desplegables, menús emergentes, o submenús. Este elemento de menú no puede ser resaltado, sombreado, o deshabilitado y los parámetros dwTypeData y cch son ignorados.

Valor	Descripción
MFT_STRING	Indica que se utiliza una cadena de caracteres para representar el elemento de menú. El parámetro dwTypeData contiene un puntero a la cadena terminada en nulo mostrada en el elemento de menú. Esta opción no puede ser combinada con las opciones MFT_BITMAP o MFT_OWNERDRAW.

Tabla 12-8: Valores del campo p4.fState de GetMenuItemInfo

Valor	Descripción
MFS_CHECKED	Coloca el mapa de bits de la marca de verificación del menú junto al elemento de menú. Por defecto, es la imagen de una marca de verificación. Esta opción no puede ser combinada con la opción MFS_UNCHECKED.
MFS_DEFAULT	Indica que el elemento de menú es el elemento predeterminado del menú. Sólo puede haber un elemento predeterminado por menú.
MFS_DISABLED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado, pero no lo sombrea. Esta opción no puede ser combinada con las opciones MFS_ENABLED o MFS_GRAYED.
MFS_ENABLED	Habilita el elemento de menú de manera que pueda ser seleccionado, y lo restaura si estaba sombreado. Esta opción no puede ser combinada con las opciones MFS_DISABLED o MFS_GRAYED. Este es el estado por defecto de un nuevo elemento de menú.
MFS_GRAYED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado y lo dibuja sombreado (en gris). Esta opción no puede ser combinada con las opciones MFS_ENABLED o MFS_DISABLED.
MFS_HILITE	Resalta el elemento de menú.
MFS_UNCHECKED	Coloca el mapa de bits que indica que el elemento de menú está desmarcado junto al elemento de menú. Por defecto, esta imagen es una imagen en blanco. Esta opción no puede ser combinada con la opción MFS_CHECKED.
MFS_UNHILITE	Elimina el resaltado del elemento de menú. Los nuevos elementos de menú contienen este atributo de manera predeterminada.

**GetMenuItemRect****Windows.Pas****Sintaxis**

```
GetMenuItemRect(
    hWnd: HWND;           {manejador de la ventana que contiene un menú}
```

```

hMenu: HMENU;           {manejador de menú}
uItem: UINT;             {posición de un elemento de menú}
var lprcItem: TRect      {puntero a un registro TRect}
): BOOL;                {devuelve TRUE o FALSE}

```

### Descripción

Esta función recupera el rectángulo límite, en coordenadas de la pantalla, del elemento de menú especificado.

### Parámetros

*hWnd*: Manejador de la ventana propietaria del menú para uno de cuyos elementos se determina el rectángulo límite.

*hMenu*: Manejador del menú que contiene el elemento de menú cuyo rectángulo límite será recuperado.

*uItem*: La posición (contando a partir de cero) del elemento de menú.

*lprcItem*: Puntero al registro *TRect* que recibe las coordenadas del rectángulo límite del elemento de menú, en coordenadas de pantalla.

### Valor que devuelve

Si esta función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*GetMenu*, *GetMenuItemCount*, *GetMenuItemID*, *GetMenuItemInfo*, *GetMenuState*, *GetMenuString*, *GetSubMenu*, *GetSystemMenu*

### Ejemplo

Vea el Listado 12-10 bajo *SetMenuDefaultItem*.

## GetMenuState Windows.Pas

### Sintaxis

```

GetMenuState(
  hMenu: HMENU;           {manejador de menú}
  uId: UINT;              {identificador de un elemento de menú o posición}
  uFlags: UINT             {opción de identificador o posición}
): UINT;                 {devuelve las opciones de estado del elemento de menú}

```

**Descripción**

Esta función recupera un valor que indica el estado del elemento de menú especificado. Si el elemento de menú abre un submenú, la función también devuelve el número de elementos de menú en el submenú abierto.

**Parámetros**

*hMenu*: Manejador del menú que contiene el elemento de menú cuyo estado será recuperado.

*uId*: Especifica el identificador de elemento de menú o la posición de base cero del elemento de menú cuyo estado será recuperado. El valor del parámetro *uFlags* indica cómo la función debe interpretar el valor de este parámetro.

*uFlags*: Indica cómo la función debe interpretar el valor del parámetro *uId*. Si a este parámetro se le asigna *MF\_BYPOSITION*, el valor del parámetro *uId* indica la posición de base cero del elemento de menú. Si a este parámetro se le asigna *MF\_BYCOMMAND*, *uId* contiene el identificador del elemento de menú.

**Valor que devuelve**

Si la función tiene éxito, devuelve una combinación de valores de la Tabla 12-9. El valor devuelto por la función puede ser combinado con una de estas opciones usando el operador booleano **and** para determinar si la opción está presente en el valor devuelto (por ejemplo, **if (GetMenuState(SysMenu, 6, MF\_BYPOSITION) and MF\_GRAYED) = MF\_GRAYED then...**). Si el elemento de menú abre un submenú, la palabra menos significativa del valor devuelto contiene las opciones de estado para el elemento de menú indicado y la palabra más significativa contiene la cantidad de elementos de menú en el submenú abierto. Si la función falla, devuelve \$FFFFFFFF.

**Véase además**

*GetMenu*, *GetMenuDefaultItem*, *GetMenuItemCount*, *GetMenuItemID*, *GetMenuItemInfo*, *GetMenuItemRect*, *GetMenuString*, *GetSubMenu*

**Ejemplo**

Vea el Listado 12-3 bajo *CreateMenu* y el Listado 12-5 bajo *EnableMenuItem*.

**Tabla 12-9: Valores que devuelve GetMenuState**

Valor	Descripción
MF_CHECKED	Indica que el elemento de menú está marcado.
MF_DISABLED	Indica que el elemento de menú está deshabilitado.
MF_GRAYED	Indica que el elemento de menú está deshabilitado y sombreado.
MF_HILITE	Indica que el elemento de menú está resaltado.

Valor	Descripción
MF_MENUBARBREAK	Indica que el elemento de menú identifica un separador de menú. Para submenús o menús desplegables, las columnas estarán separadas por una barra vertical.
MF_MENUBREAK	Indica que el elemento de menú identifica un separador de menú. No habrá indicaciones visuales de separación de columna para menús desplegables o submenús.
MF_SEPARATOR	Indica que el elemento de menú es un separador de menú. Este elemento de menú es dibujado como una línea horizontal y es válido solo para menús desplegables, menús emergentes, o submenús.

## GetMenuString Windows.Pas

### Sintaxis

```
GetMenuString(
    hMenu: HMENU;           {manejador de menú}
    uIDItem: UINT;           {identificador de elemento de menú o posición}
    lpString: PChar;         {puntero a un buffer de cadena terminada en nulo}
    nMaxCount: Integer;      {longitud de la cadena que será recuperada}
    uFlag: UINT              {opción de identificador o posición}
): Integer;                 {devuelve la cantidad de caracteres copiados en el buffer}
```

### Descripción

Esta función recupera el texto asociado con el elemento de menú especificado, y lo almacena en una cadena terminada en nulo a la que apunta el parámetro *lpString*.

### Parámetros

*hMenu*: Manejador del menú que contiene el elemento de menú cuyo texto será recuperado.

*uIDItem*: Especifica el identificador de elemento de menú o la posición de base cero del elemento de menú cuyo texto será recuperado. El valor del parámetro *uFlag* indica cómo la función debe interpretar el valor de este parámetro.

*lpString*: Puntero a un *buffer* de cadena de caracteres terminada en nulo, reservado por la aplicación que hace la llamada, que recibirá el texto asociado al elemento de menú. Este *buffer* debe ser lo suficientemente grande para almacenar la cantidad de caracteres indicada por el parámetro *nMaxCount*. Si a este parámetro se le asigna **nil**, la función devuelve el tamaño de *buffer* necesario para almacenar el texto del elemento de menú, en caracteres.

*nMaxCount*: Especifica la cantidad máxima de caracteres a copiar en el *buffer* al que apunta el parámetro *lpString*. Si la cadena del elemento de menú es más larga que el

valor indicado, es truncada. Si a este parámetro se le asigna cero, la función devuelve la longitud de la cadena del elemento de menú.

*uFlag*: Indica cómo la función debe interpretar el valor del parámetro *uIDItem*. Si a este parámetro se le asigna *MF\_BYPOSITION*, el valor del parámetro *uIDItem* indica la posición de base cero del elemento de menú. Si a este parámetro se le asigna *MF\_BYCOMMAND*, *uIDItem* contiene el identificador del elemento de menú.

#### Valor que devuelve

Si la función tiene éxito, devuelve la cantidad de caracteres copiados al *buffer* al que apunta el parámetro *lpString*, sin incluir el terminador nulo. Si la función falla, devuelve cero.

#### Véase además

*GetMenuItemID*, *GetMenuItemInfo*, *GetMenuState*, *SetMenuItemInfo*

#### Ejemplo

Vea el Listado 12-10 bajo *SetMenuDefaultItem*.

## GetSubMenu Windows.Pas

### Sintaxis

```
GetSubMenu(
    hMenu: HMENU;           {manejador de menú}
    nPos: Integer            {posición de base cero del elemento de menú}
): HMENU;                  {devuelve un manejador de menú}
```

### Descripción

Esta función recupera un manejador del menú desplegable o un submenú activado por el elemento de menú especificado.

### Parámetros

*hMenu*: Manejador del menú que contiene el elemento de menú para el que se recuperará un manejador del menú desplegable o submenú.

*nPos*: Especifica la posición (contando a partir de cero) del elemento de menú que activa un menú desplegable o submenú.

### Valor que devuelve

Si la función tiene éxito, devuelve un manejador del menú desplegable o submenú activado por el elemento de menú indicado. Si la función falla o el elemento de menú indicado no activa un menú desplegable o submenú, la función devuelve cero.

Véase además

*CreatePopupMenu, GetMenu, GetMenuItemInfo, SetMenuItemInfo*

### Ejemplo

Vea el Listado 12-6 bajo *GetMenu*.

## GetSystemMenu

## Windows.Pas

### Sintaxis

```
GetSystemMenu(
    hWnd: HWND;           {manejador de ventana}
    bRevert: BOOL          {opción de reinicialización del menú de la ventana}
): HMENU;                {devuelve un manejador del menú de sistema}
```

### Descripción

Esta función recupera un manejador del menú de sistema de la ventana especificada (el menú mostrado cuando se hace clic en el icono situado en la esquina superior izquierda de la ventana). Una aplicación puede modificar este menú utilizando varias de las funciones que se describen en este capítulo. El elemento de menú añadido a este menú enviará a la ventana propietaria un mensaje *WM\_SYSCOMMAND* cuando se haga clic sobre él. Observe que todos los elementos de menú predefinidos en el menú de sistema tienen identificadores mayores que \$F000. Los elementos de menú que se añadan al menú de sistema deben tener identificadores menores que \$F000.

### Parámetros

*hWnd*: Manejador de la ventana cuyo manejador del menú de sistema será recuperado.

*bRevert*: Indica si el menú de sistema debe ser restaurado a su estado predeterminado. Si a este parámetro se le asigna FALSE, la función devuelve un manejador del menú de sistema actual. Si a este parámetro se le asigna TRUE, el menú del sistema es reinicializado a su estado por defecto y el menú de sistema anterior es destruido.

### Valor que devuelve

Si la función tiene éxito y al parámetro *bRevert* se le ha asignado FALSE, devuelve un manejador del menú de sistema actual. Si la función falla o al parámetro *bRevert* se le ha asignado TRUE, la función devuelve cero.

Véase además

*AppendMenu, GetMenu, GetSubMenu, InsertMenu, InsertMenuItem, ModifyMenu*

### Ejemplo

Vea el Listado 12-4 bajo *CreatePopupMenu*.

**HiliteMenuItem****Windows.Pas****Sintaxis**

```
HiliteMenuItem(
  hWnd: HWND;           {manejador de ventana}
  hMenu: HMENU;          {manejador de menú}
  uIDHiliteItem: UINT;   {identificador de elemento de menú o posición}
  uHilite: UINT           {opciones de elemento de menú}
): BOOL;                {devuelve TRUE o FALSE}
```

**Descripción**

Esta función resalta o quita el resaltado a un elemento de menú en una barra de menú.

**Parámetros**

*hWnd*: Manejador de la ventana propietaria del menú cuyo elemento de menú será resaltado.

*hMenu*: Manejador del menú que contiene el elemento de menú a resaltar. Observe que éste debe ser un manejador de una barra de menú, y no de un menú emergente, menú desplegable, o submenú.

*uIDHiliteItem*: Especifica el elemento de menú que será resaltado. Este parámetro contiene un identificador de elemento de menú o una posición de base cero, según determina el parámetro *uHilite*.

*uHilite*: Una combinación de valores que determina cómo el parámetro *uIDHiliteItem* es interpretado y si el elemento de menú debe ganar o perder el resaltado. Este parámetro puede contener una de las opciones *MF\_BYCOMMAND* o *MF\_BYPOSITION* combinada con *MF\_HILITE* o *MF\_UNHILITE*. El significado de estas opciones se describe en la Tabla 12-10.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE.

**Véase además**

*GetMenuItemInfo*, *GetMenuState*, *SetMenuItemInfo*

**Ejemplo**

Vea el Listado 12-3 bajo *CreateMenu*.

**Tabla 12-10: Valores del parámetro uHilite de HiliteMenuItem**

Valor	Descripción
MF_BYCOMMAND	Indica que el parámetro uIDHiliteItem contiene un identificador de elemento de menú. Esta opción se asume por defecto si ninguna es especificada, y no puede ser combinada con MF_BYPOSITION.

Valor	Descripción
MF_BYPOSITION	Indica que el parámetro <code>ulDHilitem</code> contiene una posición de elemento de menú (de base cero). Esta opción no puede ser combinada con MF_BYCOMMAND.
MF_HILITE	Resalta el elemento de menú.
MF_UNHILITE	Quita el resaltado al elemento de menú.

## InsertMenu Windows.Pas

### Sintaxis

```

InsertMenu(
    hMenu: HMENU;           {manejador de menú}
    uPosition: UINT;         {identificador de elemento de menú o posición}
    uFlags: UINT;            {opciones de elemento de menú}
    uIDNewItem: UINT;        {identificador de elemento de menú o manejador de
                             submenú}
    lpNewItem: PChar         {datos del elemento de menú}
): BOOL;                   {devuelve TRUE o FALSE}

```

### Descripción

Esta función inserta un nuevo elemento de menú antes del elemento de menú indicado, en el menú especificado. Todos los elementos de menú subsiguientes son desplazados hacia abajo.

### Parámetros

*hMenu*: Manejador del menú en el cual el nuevo elemento de menú será insertado.

*uPosition*: Especifica el elemento de menú antes del cual el nuevo elemento de menú será insertado. Este parámetro puede contener un identificador de elemento de menú o una posición de base cero, según determine el parámetro *uFlags*.

*uFlags*: Una combinación de opciones que determina cómo el parámetro *uPosition* es interpretado y cómo el nuevo elemento de menú aparecerá y se comportará. Este parámetro puede contener una de las opciones MF\_BYCOMMAND o MF\_BYPOSITION, combinada con una o más opciones de la Tabla 12-11.

*uIDNewItem*: Especifica el identificador del nuevo elemento de menú. Si el parámetro *uFlags* contiene la opción MF\_POPUP, este parámetro contiene el manejador del menú desplegable o submenú.

*lpNewItem*: Puntero a datos adicionales asociados con el nuevo elemento de menú. El contenido de este parámetro depende de las opciones especificadas en el parámetro *uFlags*. Si el parámetro *uFlags* incluye la opción MF\_BITMAP, este parámetro contiene un manejador de un mapa de bits. Si el parámetro *uFlags* incluye la opción MF\_STRING, este parámetro apunta a una cadena de caracteres terminada en nulo. Si el parámetro *uFlags* incluye la opción MF\_OWNERDRAW, este parámetro contiene un

valor de 32 bits definido por la aplicación. Este valor estará contenido en el campo *itemData* del registro al que apuntan los mensajes *WM\_MEASUREITEM* y *WM\_DRAWITEM*, que la ventana recibe cuando el menú es creado o dibujado.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*AppendMenu*, *CreateMenu*, *CreatePopupMenu*, *DeleteMenu*, *DestroyMenu*, *InsertMenuItem*, *ModifyMenu*, *RemoveMenu*

#### Ejemplo

Vea el Listado 12-4 bajo *CreatePopupMenu*.

**Tabla 12-II: Valores del parámetro uFlags de InsertMenu**

Valor	Descripción
MF_BITMAP	Indica que se utiliza un mapa de bits para representar el elemento de menú. El manejador del mapa de bits es especificado en el parámetro <i>lpNewItem</i> . Esta opción no puede ser combinada con las opciones MF_STRING o MF_OWNERDRAW.
MF_BYCOMMAND	Indica que el parámetro <i>uPosition</i> contiene un identificador de elemento de menú. Esta opción se asume por defecto si ninguna es especificada, y no puede ser combinada con MF_BYPOSITION.
MF_BYPOSITION	Indica que el parámetro <i>uPosition</i> contiene una posición de elemento de menú (de base cero). Esta opción no puede ser combinada con MF_BYCOMMAND. Observe que si el parámetro <i>uPosition</i> es \$FFFFFFFF; el elemento del menú se añade al final del menú.
MF_CHECKED	Coloca el mapa de bits de la marca de verificación del menú junto al elemento de menú. Por defecto, es la imagen de una marca de verificación. Esta opción no puede ser combinada con la opción MF_UNCHECKED.
MF_DISABLED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado, pero no lo sombrea. Esta opción no puede ser combinada con las opciones MF_ENABLED o MF_GRAYED.
MF_ENABLED	Habilita el elemento de menú de manera que pueda ser seleccionado, y lo restaura si estaba sombreado. Esta opción no puede ser combinada con las opciones MF_DISABLED o MF_GRAYED.

Valor	Descripción
MF_GRAYED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado y lo dibuja sombreado (en gris). Esta opción no puede ser combinada con las opciones MF_ENABLED o MF_DISABLED.
MF_MENUBARBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna y las columnas son separadas mediante una línea vertical. Esta opción no puede ser combinada con la opción MF_MENUBREAK.
MF_MENUBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna. No hay indicación visual de separación de columnas. Esta opción no puede ser combinada con la opción MF_MENUBARBREAK.
MF_OWNERDRAW	Indica un elemento de menú dibujado por el propietario. Cuando el elemento de menú es mostrado por primera vez, la ventana propietaria recibe un mensaje WM_MEASUREITEM, permitiendo a la aplicación especificar el ancho y altura del elemento de menú. La ventana propietaria recibe un mensaje WM_DRAWITEM cuando el elemento de menú va a ser dibujado, permitiendo a la aplicación dibujar el elemento de menú con la apariencia deseada. Esta opción no puede ser combinada con las opciones MF_BITMAP o MF_STRING.
MF_POPUP	Indica que el elemento de menú abre un menú desplegable o submenú. El manejador del menú desplegable o submenú es especificado en el parámetro ulDNNewItem.
MF_SEPARATOR	Indica un separador de menú. Este elemento de menú es dibujado como una línea horizontal y es válido solo cuando es usado con menús desplegables, menús emergentes, o submenús. Este elemento de menú no puede ser resaltado, sombreado, o deshabilitado y los parámetros ulDNNewItem y lpNewItem son ignorados.
MF_STRING	Indica que se utiliza una cadena de caracteres para representar el elemento de menú. El parámetro lpNewItem contiene un puntero a la cadena terminada en nulo mostrada en el elemento de menú. Esta opción no puede ser combinada con las opciones MF_BITMAP o MF_OWNERDRAW.
MF_UNCHECKED	Coloca el mapa de bits que indica que el elemento de menú está desmarcado junto al elemento de menú. Por defecto, esta imagen es una imagen en blanco. Esta opción no puede ser combinada con la opción MF_CHECKED.

**InsertMenuItem****Windows.Pas****Sintaxis**

```

InsertMenuItem(
  p1: HMENU;           {manejador de menú}
  p2: UINT;             {identificador de elemento de menú o posición}
  p3: BOOL;             {opción de identificador o posición}
  const p4: TMenuItemInfo {puntero a registro TMenuItemInfo}
): BOOL;               {devuelve TRUE o FALSE}

```

**Descripción**

Esta función inserta un nuevo elemento de menú antes del elemento de menú especificado en el menú indicado. Todos los elementos de menú subsiguientes son desplazados hacia abajo.

**Parámetros**

*p1*: Manejador del menú en el cual el nuevo elemento de menú será insertado.

*p2*: Especifica el identificador de elemento de menú o la posición de base cero del elemento de menú, delante del cual el nuevo elemento de menú será insertado. El valor del parámetro *p3* indica cómo la función debe interpretar el valor de este parámetro.

*p3*: Indica cómo la función debe interpretar el valor del parámetro *p2*. Si a este parámetro se le asigna TRUE, el valor del parámetro *p2* indica la posición de base cero del elemento de menú. Si a este parámetro se le asigna FALSE, *p2* contiene el identificador del elemento de menú.

*p4*: Puntero a un registro de tipo *TMenuItemInfo* que contiene información sobre el nuevo elemento de menú. El registro *TMenuItemInfo* se define como:

TMenuItemInfo = **packed record**

cbSize: UINT;	{el tamaño del registro}
fMask: UINT;	{opción de asignación o recuperación}
fType: UINT;	{opciones del tipo del elemento de menú}
fState: UINT;	{opciones del estado del elemento de menú}
wID: UINT;	{el identificador del elemento de menú}
hSubMenu: HMENU;	{manejador de menú desplegable o submenú}
hbmChecked: HBITMAP;	{manejador del mapa de bits para el estado 'marcado'}
hbmUnchecked: HBITMAP;	{manejador del mapa de bits para el estado 'desmarcado'}
dwItemData: DWORD;	{valor definido por la aplicación}
dwTypeData: PAnsiChar;	{contenido del elemento de menú}
cch: UINT;	{longitud del texto del elemento de menú}

**end;**

Consulte la función *GetMenuItemInfo* para ver una descripción completa de este registro.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*AppendMenu, CreateMenu, CreatePopupMenu, DeleteMenu, DestroyMenu, InsertMenu, ModifyMenu, RemoveMenu*

#### Ejemplo

Vea el Listado 12-3 bajo *CreateMenu*.

### **IsMenu**                      **Windows.Pas**

#### Sintaxis

```
IsMenu(
    hMenu: HMENU           {manejador de menú}
): BOOL;                  {devuelve TRUE o FALSE}
```

#### Descripción

Esta función indica si el manejador especificado es un manejador de menú.

#### Parámetros

*hMenu*: Especifica el manejador a comprobar.

#### Valor que devuelve

Si la función tiene éxito y el manejador identifica un menú, la función devuelve TRUE. Si la función falla o el manejador no identifica un menú, la función devuelve FALSE.

#### Véase además

*GetMenu, GetSubMenu, IsWindow*

#### Ejemplo

#### **Listado 12-7: Comprobando la validez del manejador**

```
procedure TForm1.TestHandle(Value: THandle);
begin
    if IsMenu(Value) then
        ShowMessage('This handle is a menu')
    else
        ShowMessage('This handle is not a menu');
end;
```

**ModifyMenu      Windows.Pas****Sintaxis**

```

ModifyMenu(
    hMnu: HMENU;           {manejador de menú}
    uPosition: UINT;        {identificador de elemento de menú o posición}
    uFlags: UINT;           {opciones de elemento de menú}
    uIDNewItem: UINT;       {identificador de elemento de menú o manejador de
                             submenú}
    lpNewItem: PChar        {datos del elemento de menú}
): BOOL;                  {devuelve TRUE o FALSE}

```

**Descripción**

Esta función modifica la apariencia y comportamiento del elemento de menú indicado en el menú especificado.

**Parámetros**

*hMnu*: Manejador del menú que contiene el elemento de menú que será modificado.

*uPosition*: Especifica el elemento de menú a modificar. Este parámetro contiene un identificador de elemento de menú o a una posición de base cero, según determine el parámetro *uFlags*.

*uFlags*: Una combinación de opciones que determinan cómo el parámetro *uPosition* debe ser interpretado y cómo el elemento de menú especificado será modificado. Este parámetro puede contener una de las opciones *MF\_BYCOMMAND* o *MF\_BYPOSITION*, combinadas con una o más opciones de la Tabla 12-12.

*uIDNewItem*: Especifica el identificador del nuevo elemento de menú. Si el parámetro *uFlags* contiene la opción *MF\_POPUP*, este parámetro contiene el manejador del menú desplegable o submenú.

*lpNewItem*: Puntero a datos adicionales asociados con el nuevo elemento de menú. El contenido de este parámetro depende de las opciones especificadas en el parámetro *uFlags*. Si el parámetro *uFlags* incluye la opción *MF\_BITMAP*, este parámetro contiene un manejador de un mapa de bits. Si el parámetro *uFlags* incluye la opción *MF\_STRING*, este parámetro apunta a una cadena de caracteres terminada en nulo. Si el parámetro *uFlags* incluye la opción *MF\_OWNERDRAW*, este parámetro contiene un valor de 32 bits definido por la aplicación. Este valor estará contenido en el campo *itemData* del registro al que apuntan los mensajes *WM\_MEASUREITEM* y *WM\_DRAWITEM*, que la ventana recibe cuando el menú es creado o dibujado.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*AppendMenu, CreateMenu, CreatePopupMenu, DeleteMenu, DestroyMenu, InsertMenu, InsertMenuItem, RemoveMenu*

### Ejemplo

#### Listado I2-8: Creando una herramienta de paleta en un menú

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    {Cambia los elementos de menú existentes en el menú desplegable. Esto dará al
    menú la apariencia de una barra de herramientas o un herramienta de paleta}
    ModifyMenu(Palette1.Handle, 0, MF_BYPOSITION or MF_BITMAP,
        GetMenuItemID(Palette1.Handle, 0),
        Pointer(Image1.Picture.Bitmap.Handle));
    ModifyMenu(Palette1.Handle, 1, MF_BYPOSITION or MF_BITMAP,
        GetMenuItemID(Palette1.Handle, 1),
        Pointer(Image2.Picture.Bitmap.Handle));
    ModifyMenu(Palette1.Handle, 2, MF_BYPOSITION or MF_BITMAP,
        GetMenuItemID(Palette1.Handle, 2),
        Pointer(Image3.Picture.Bitmap.Handle));

    {Queremos comenzar una nueva columna aquí}
    ModifyMenu(Palette1.Handle, 3, MF_BYPOSITION or MF_BITMAP or MF_MENUBREAK,
        GetMenuItemID(Palette1.Handle, 3),
        Pointer(Image4.Picture.Bitmap.Handle));
    ModifyMenu(Palette1.Handle, 4, MF_BYPOSITION or MF_BITMAP,
        GetMenuItemID(Palette1.Handle, 4),
        Pointer(Image5.Picture.Bitmap.Handle));
    ModifyMenu(Palette1.Handle, 5, MF_BYPOSITION or MF_BITMAP,
        GetMenuItemID(Palette1.Handle, 5),
        Pointer(Image6.Picture.Bitmap.Handle));
end;

procedure TForm1.Item11Click(Sender: TObject);
begin
    {Muestra en qué elemento se hizo clic}
    Label2.Caption := TMenuItem(Sender).Caption;
end;
```

Figura I2-7:  
El menú de  
paleta de  
herramientas

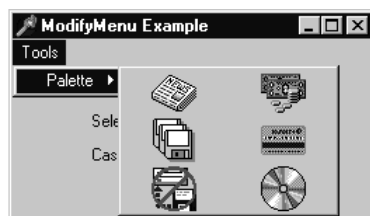


Tabla 12-12: Valores del parámetro uFlags de ModifyMenu

Valor	Descripción
MF_BITMAP	Indica que se utiliza un mapa de bits para representar el elemento de menú. El manejador del mapa de bits es especificado en el parámetro lpNewItem. Esta opción no puede ser combinada con las opciones MF_STRING o MF_OWNERDRAW.
MF_BYCOMMAND	Indica que el parámetro uPosition contiene un identificador de elemento de menú. Esta opción se asume por defecto si ninguna es especificada, y no puede ser combinada con MF_BYPOSITION.
MF_BYPOSITION	Indica que el parámetro uPosition contiene una posición de elemento de menú (de base cero). Esta opción no puede ser combinada con MF_BYCOMMAND. Observe que si el parámetro uPosition es \$FFFFFFFF, el elemento del menú se añade al final del menú.
MF_CHECKED	Coloca el mapa de bits de la marca de verificación del menú junto al elemento de menú. Por defecto, ésta es una imagen de marca de verificación. Esta opción no puede ser combinada con la opción MF_UNCHECKED.
MF_DISABLED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado, pero no lo muestra en gris. Esta opción no puede ser combinada con las opciones MF_ENABLED o MF_GRAYED.
MF_ENABLED	Habilita el elemento de menú de manera que pueda ser seleccionado, y lo restaura si estaba sombreado. Esta opción no puede ser combinada con las opciones MF_DISABLED o MF_GRAYED.
MF_GRAYED	Deshabilita el elemento de menú de manera que no pueda ser seleccionado y lo dibuja sombreado (en gris). Esta opción no puede ser combinada con las opciones MF_ENABLED o MF_DISABLED.
MF_MENUBARBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna y las columnas son separadas mediante una línea vertical. Esta opción no puede ser combinada con la opción MF_MENUBREAK.
MF_MENUBREAK	Coloca el elemento de menú en una nueva línea de la barra de menú. En el caso de submenús y menús desplegables, el elemento de menú es situado en una nueva columna. No hay indicación visual de separación de columnas. Esta opción no puede ser combinada con la opción MF_MENUBARBREAK.

Valor	Descripción
MF_OWNERDRAW	Indica un elemento de menú dibujado por el propietario. Cuando el elemento de menú es mostrado por primera vez, la ventana propietaria recibe un mensaje WM_MEASUREITEM, permitiendo a la aplicación especificar el ancho y altura del elemento de menú. La ventana propietaria recibe un mensaje WM_DRAWITEM cuando el elemento de menú va a ser dibujado, permitiendo a la aplicación dibujar el elemento de menú con la apariencia deseada. Esta opción no puede ser combinada con las opciones MF_BITMAP o MF_STRING.
MF_POPUP	Indica que el elemento de menú abre un menú desplegable o submenú. El manejador del menú desplegable o submenú es especificado en el parámetro <code>uIDNewItem</code> .
MF_SEPARATOR	Indica un separador de menú. Este elemento de menú es dibujado como una línea horizontal y es válido solo cuando es usado con menús desplegables, menús emergentes o submenús. Este elemento de menú no puede ser resaltado, sombreado o deshabilitado y los parámetros <code>uIDNewItem</code> y <code>lpNewItem</code> son ignorados.
MF_STRING	Indica que se utiliza una cadena de caracteres para representar el elemento de menú. El parámetro <code>lpNewItem</code> contiene un puntero a la cadena terminada en nulo mostrada en el elemento de menú. Esta opción no puede ser combinada con las opciones MF_BITMAP o MF_OWNERDRAW.
MF_UNCHECKED	Coloca el mapa de bits que indica que el elemento de menú está desmarcado junto al elemento de menú. Por defecto, ésta imagen es una imagen en blanco. Esta opción no puede ser combinada con la opción MF_CHECKED.

## RemoveMenu Windows.Pas

### Sintaxis

```
RemoveMenu(
    hMenu: HMENU;           {manejador de menú}
    uPosition: UINT;        {identificador de elemento de menú o posición}
    uFlags: UINT             {opción de identificador o posición}
): BOOL;                   {devuelve TRUE o FALSE}
```

### Descripción

Esta función elimina el elemento de menú indicado del menú especificado, desplazando todos los elementos subsiguientes hacia arriba. Si el elemento de menú abre un submenú o menú desplegable, el submenú o menú desplegable NO es destruido. Una aplicación debe llamar la función *GetSubMenu* para recuperar un manejador del submenú antes de eliminar el elemento de menú. Se puede utilizar la función

*DeleteMenu* para eliminar un elemento de menú y liberar su submenú o menú desplegable asociado.

#### Parámetros

*hMenu*: Manejador del menú del cual será eliminado el elemento de menú especificado.

*uPosition*: Especifica el identificador de elemento de menú o la posición de base cero del elemento de menú a eliminar. El valor del parámetro *uFlags* indica cómo la función debe interpretar el valor de este parámetro.

*uFlags*: Indica cómo la función debe interpretar el valor del parámetro *uPosition*. Si a este parámetro se le asigna *MF\_BYPOSITION*, el valor de *uPosition* indica la posición de base cero del elemento de menú. Si a este parámetro se le asigna *MF\_BYCOMMAND*, *uPosition* tendrá el identificador del elemento de menú.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

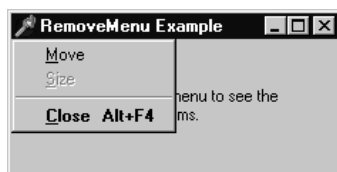
*CreatePopupMenu*, *DeleteMenu*, *DestroyMenu*, *GetMenu*, *GetSubMenu*

#### Ejemplo

##### Listado 12-9: Quitando elementos del menú de sistema

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    {Quita los comandos Maximizar, Minimizar y Restaurar del menú de sistema}
    RemoveMenu(GetSystemMenu(Form1.Handle, FALSE), 4, MF_BYPOSITION);
    RemoveMenu(GetSystemMenu(Form1.Handle, FALSE), 3, MF_BYPOSITION);
    RemoveMenu(GetSystemMenu(Form1.Handle, FALSE), 0, MF_BYPOSITION);
end;
```

Figura 12-8:  
El menú de  
sistema  
modificado



#### SetMenu

#### Windows.Pas

#### Sintaxis

```
SetMenu(
    hWnd: HWND;           {manejador de ventana}
    hMenu: HMENU           {manejador de menú})
```

); BOOL; {devuelve TRUE o FALSE}

### Descripción

Esta función asigna el menú especificado a la ventana especificada. El menú anterior, si existía, es reemplazado pero no destruido. Una aplicación debe llamar a alguna función de destrucción de menú para eliminar el menú anterior y sus recursos asociados.

### Parámetros

*hWnd*: Manejador de la ventana a la que se asigna el nuevo menú.

*hMenu*: Manejador del menú que será asignado a la ventana especificada. Este menú crea una barra de menú en la ventana. Si a este parámetro se le asigna cero, la función simplemente quita el menú actual.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

### Véase además

*CreateMenu*, *DeleteMenu*, *DestroyMenu*, *GetMenu*

### Ejemplo

Vea el Listado 12-3 bajo *CreateMenu*.

## SetMenuDefaultItem Windows.Pas

### Sintaxis

```
SetMenuDefaultItem(
    hMenu: HMENU;           {manejador de menú}
    uItem: UINT;             {identificador de elemento de menú o posición}
    fByPos: UINT             {opción de identificador o posición}
): BOOL;                   {devuelve TRUE o FALSE}
```

### Descripción

Esta función asigna el elemento de menú indicado como elemento predeterminado (por defecto) para el menú especificado. Un menú puede tener sólo un elemento por defecto y éste debe formar parte de un menú desplegable, submenú, o menú emergente.

### Parámetros

*hMenu*: Manejador del menú cuyo elemento por defecto va a ser asignado.

*uItem*: Especifica el identificador de elemento de menú o la posición de base cero del elemento de menú que será asignado como elemento por defecto del menú. El valor del

parámetro *fByPos* indica cómo la función debe interpretar el valor de este parámetro. Si a este parámetro se le asigna -1, la función quita el atributo de predeterminado al elemento por defecto actual, dejando al menú sin elemento por defecto.

*fByPos*: Indica cómo la función debe interpretar el valor del parámetro *uItem*. Si a este parámetro se le asigna *MF\_BYPOSITION*, el valor del parámetro *uItem* indica la posición de base cero del elemento de menú. Si a este parámetro se le asigna *MF\_BYCOMMAND*, *uItem* contiene el identificador del elemento de menú.

#### Valor que devuelve

Si esta función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetMenuDefaultItem*, *GetMenuItemID*, *GetMenuItemInfo*, *HiliteMenuItem*

#### Ejemplo

##### Listado I2-I0: Recuperando información sobre el elemento por defecto de un menú

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    {Asigna al máximo valor del editor la cantidad de elementos del menú}
    SpinEdit1.MaxValue := GetMenuItemCount(MainMenu2.Handle) - 1;
end;

procedure TForm1.SpinEdit1Change(Sender: TObject);
var
    DefaultItem: Integer;           // posición del elemento por defecto
    DefaultItemRect: TRect;         // coordenadas del elemento por defecto
    DefaultMenuString: array[0..255] of Char; // cadena del elemento por defecto
begin
    {Asigna como elemento por defecto el especificado por editor}
    SetMenuDefaultItem(MainMenu2.Handle, SpinEdit1.Value, 1);

    {Recupera el elemento por defecto recién asignado}
    DefaultItem := GetMenuDefaultItem(MainMenu2.Handle, 1, 0);

    {Recupera las coordenadas del elemento por defecto}
    GetMenuItemRect(Form1.Handle, MainMenu2.Handle, DefaultItem, DefaultItemRect);

    {Recupera el texto del elemento por defecto}
    GetMenuString(MainMenu2.Handle, DefaultItem, @DefaultMenuString, 255,
        MF_BYPOSITION);

    {Muestra la posición e identificador del elemento por defecto}
    Label2.Caption := 'Default Item Position: ' + IntToStr(DefaultItem);
    Label3.Caption := 'Default Item ID: ' + IntToStr(GetMenuItemID(MainMenu2.Handle,
        DefaultItem));

    {Muestra las coordenadas del elemento de menú por defecto}

```

```

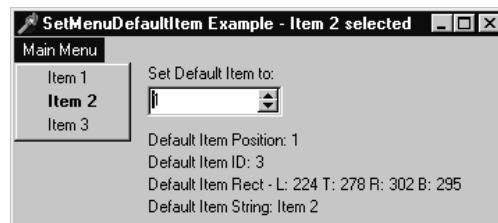
Label4.Caption := 'Default Item Rect - L: ' + IntToStr(DefaultItemRect.Left) +
  ' T: ' + IntToStr(DefaultItemRect.Top) +
  ' R: ' + IntToStr(DefaultItemRect.Right) +
  ' B: ' + IntToStr(DefaultItemRect.Bottom);

{Muestra el texto del elemento de menú por defecto}
Label5.Caption := 'Default Item String: ' + DefaultMenuString;
end;

procedure TForm1.Item1Click(Sender: TObject);
begin
  {Muestra el texto del elemento de menú seleccionado}
  Caption := 'SetMenuDefaultItem Example - ' + TMenuItem(Sender).Caption +
    ' selected';
end;

```

Figura 12-9:  
Los atributos  
del elemento  
por defecto  
del menú



## SetMenuItemBitmaps Windows.Pas

### Sintaxis

```

SetMenuItemBitmaps(
  hMenu: HMENU;           {manejador de menú}
  uPosition: UINT;        {identificador de elemento de menú o posición}
  uFlags: UINT;           {opción de identificador o posición}
  hBitmapUnchecked: HBITMAP; {manejador del mapa de bits del estado
                             'desmarcado'}
  hBitmapChecked: HBITMAP {manejador del mapa de bits del estado 'marcado'}
): BOOL;                 {devuelve TRUE o FALSE}

```

### Descripción

Esta función permite asignar el mapa de bits que será mostrado cuando el elemento de menú indicado, contenido en el menú especificado, esté marcado o desmarcado. La aplicación debe gestionar estos mapas de bits, ya que éstos no son destruidos cuando el menú o elemento de menú es eliminado o destruido. Observe que si a ambos parámetros, *hBitmapUnchecked* y *hBitmapChecked*, se les asigna cero (0), a partir de ese momento Windows mostrará una imagen de marca de verificación cuando el elemento de menú esté marcado y ninguna imagen cuando no esté marcado. La aplicación debe utilizar la función *GetSystemMetrics* con las opciones

*CX\_MENUCHECK* y *CY\_MENUCHECK* para determinar las dimensiones apropiadas para los mapas de bits de las marcas de verificación.

#### Parámetros

*hMenu*: Manejador del menú que contiene el elemento de menú cuyos mapas de bits de marcado y desmarcado serán asignados.

*uPosition*: Especifica el identificador de elemento de menú o la posición de base cero del elemento de menú cuyos mapas de bits de marcado y desmarcado serán asignados. El valor del parámetro *uFlags* indica cómo la función debe interpretar el valor de este parámetro.

*uFlags*: Indica cómo la función debe interpretar el valor del parámetro *uPosition*. Si a este parámetro se le asigna *MF\_BYPOSITION*, el valor de *uPosition* indica la posición (contando a partir de cero) del elemento de menú. Si a este parámetro se le asigna *MF\_BYCOMMAND*, *uPosition* contiene un identificador de elemento de menú.

*hBitmapUnchecked*: Manejador del mapa de bits que será mostrado cuando el elemento de menú no esté marcado. Si a este parámetro se le asigna cero, ninguna imagen será mostrada cuando el elemento de menú no esté marcado.

*hBitmapChecked*: Manejador del mapa de bits que será mostrado cuando el elemento de menú esté marcado. Si a este parámetro se le asigna cero, ninguna imagen será mostrada cuando el elemento de menú esté marcado.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*GetMenu*, *GetMenuItemInfo*, *GetSubMenu*, *GetSystemMetrics*, *SetMenuItemInfo*

#### Ejemplo

##### Listado I2-II: Especificando mapas de bits personalizados para marcas de verificación

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    {Especifica el nuevo mapa de bits de marca de verificación para todos los
    elementos de menú}
    SetMenuItemBitmaps(TheMainMenu1.Handle, 0, MF_BYPOSITION,
        Image2.Picture.Bitmap.Handle,
        Image1.Picture.Bitmap.Handle);
    SetMenuItemBitmaps(TheMainMenu1.Handle, 1, MF_BYPOSITION,
        Image2.Picture.Bitmap.Handle,
        Image1.Picture.Bitmap.Handle);
    SetMenuItemBitmaps(TheMainMenu1.Handle, 2, MF_BYPOSITION,
        Image2.Picture.Bitmap.Handle,
        Image1.Picture.Bitmap.Handle);
```

```

end;

procedure TForm1.ItemOneClick(Sender: TObject);

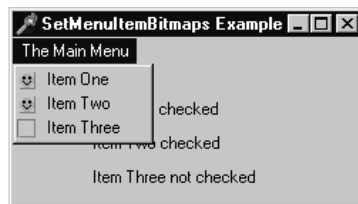
  procedure DisplayChecked(Item: TMenuItem; TheLabel: TLabel);
  begin
    {Indica si el elemento está marcado o no}
    if Item.Checked then
      TheLabel.Caption := Item.Caption + ' checked'
    else
      TheLabel.Caption := Item.Caption + ' not checked'
    end;
  end;

begin
  {Marca el elemento de menú seleccionado}
  TMenuItem(Sender).Checked := not TMenuItem(Sender).Checked;

  {Muestra qué elementos de menú están marcados}
  DisplayChecked(TheMainMenu.Items[0], Label1);
  DisplayChecked(TheMainMenu.Items[1], Label2);
  DisplayChecked(TheMainMenu.Items[2], Label3);
end;

```

Figura 12-10:  
La nueva  
marca de  
verificación del  
elemento de  
menu



## SetMenuItemInfo

## Windows.Pas

### Sintaxis

```

SetMenuItemInfo(
  p1: HMENU;           {manejador de menú}
  p2: UINT;             {identificador de elemento de menú o posición}
  p3: BOOL;             {opción de identificador o posición}
  const p4: TMenuItemInfo {puntero a registro TMenuItemInfo}
): BOOL;               {devuelve TRUE o FALSE}

```

### Descripción

Esta función asigna información específica sobre el elemento de menú del menú especificado.

### Parámetros

*p1*: Manejador del menú que contiene el elemento de menú cuya información es asignada.

*p2*: Especifica el identificador del elemento de menú o la posición de base cero del elemento de menú cuya información será asignada. El valor del parámetro *p3* determina cómo la función debe interpretar el valor de este parámetro.

*p3*: Indica cómo la función debe interpretar el valor del parámetro *p2*. Si a este parámetro se le asigna TRUE, el valor de *p2* indica la posición de base cero del elemento de menú. Si a este parámetro se le asigna FALSE, *p2* contiene el identificador del elemento de menú.

*p4*: Puntero a un registro de tipo *TMenuItemInfo* que contiene valores que especifican la información a asignar. El registro *TMenuItemInfo* se define como:

*TMenuItemInfo* = **packed record**

<i>cbSize</i> : UINT;	{tamaño del registro}
<i>fMask</i> : UINT;	{opción de asignación o recuperación}
<i>fType</i> : UINT;	{opciones de tipo del elemento de menú}
<i>fState</i> : UINT;	{opciones de estado del elemento de menú}
<i>wID</i> : UINT;	{identificador del elemento de menú}
<i>hSubMenu</i> : HMENU;	{manejador de menú desplegable o submenú}
<i>hbmChecked</i> : HBITMAP;	{manejador del mapa de bits para el estado 'marcado'}
<i>hbmUnchecked</i> : HBITMAP;	{manejador del mapa de bits para el estado 'desmarcado'}
<i>dwItemData</i> : DWORD;	{valor definido por la aplicación}
<i>dwTypeData</i> : PAnsiChar;	{contenido del elemento de menú}
<i>cch</i> : UINT;	{longitud del texto del elemento de menú}

**end;**

Consulte la función *GetMenuItemInfo* para ver una descripción detallada de este registro.

#### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

#### Véase además

*EnableMenuItem*, *GetMenuItemCount*, *GetMenuItemID*, *GetMenuItemInfo*, *GetMenuItemRect*, *GetMenuState*, *GetMenuString*, *HiliteMenuItem*, *InsertMenuItem*, *ModifyMenu*, *SetMenuItemBitmaps*

#### Ejemplo

Vea el Listado 12-1 bajo el ejemplo de menú dibujado por el propietario, en la introducción del capítulo.

**TrackPopupMenu****Windows.Pas****Sintaxis**

```

TrackPopupMenu(
    hMenu: HMENU;           {manejador de un menú emergente}
    uFlags: UINT;           {opciones de posición y seguimiento}
    x: Integer;             {coordenadas horizontales de pantalla}
    y: Integer;             {coordenadas verticales de pantalla}
    nReserved: Integer;     {parámetro reservado}
    hWnd: HWND;             {manejador de la ventana propietaria}
    prcRect: PRect          {puntero a registro TRect}
): BOOL;                  {devuelve TRUE o FALSE}

```

**Descripción**

Esta función muestra un menú emergente en las coordenadas especificadas de la pantalla, enviando un mensaje *WM\_COMMAND* a la ventana *hWnd* cuando el usuario selecciona un elemento de menú.

**Parámetros**

*hMenu*: Manejador del menú emergente que será mostrado. Este manejador es devuelto por la función *CreatePopupMenu* cuando se crea un nuevo menú emergente, o por la función *GetSubMenu* cuando se recupera un submenú o un menú desplegable.

*uFlags*: Una combinación de opciones que indican la alineación del menú emergente en relación a sus coordenadas y qué botón del ratón indica una selección. Este parámetro puede ser cero o una combinación de un elemento de la Tabla 12-13 y un elemento de la Tabla 12-14.

*x*: La coordenada horizontal en la cual se mostrará el menú emergente, relativa a la pantalla.

*y*: La coordenada vertical en la cual se mostrará el menú emergente, relativa a la pantalla.

*nReserved*: Este parámetro es reservado y se le debe asignar cero.

*hWnd*: Manejador de la ventana propietaria del menú emergente que será mostrado. Esta ventana recibirá un mensaje *WM\_COMMAND* cuando el usuario seleccione un elemento del menú. La ventana no recibirá el mensaje hasta que la función *TrackPopupMenu* retorne.

*prcRect*: Puntero al registro *TRect* que contiene las coordenadas, relativas a la pantalla, del rectángulo dentro del cual el usuario puede hacer clic sin que el menú emergente desaparezca. Si a este parámetro se le asigna **nil**, el menú emergente desaparece cuando el usuario hace clic fuera de las fronteras del menú emergente.

**Valor que devuelve**

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

**Véase además**

*CreatePopupMenu*, *GetSubMenu*, *TrackPopupMenuEx*

**Ejemplo**

Vea el Listado 12-4 bajo *CreatePopupMenu*.

**Tabla 12-13: Valores de alineación para el parámetro uFlags de TrackPopupMenu**

Valor	Descripción
TPM_CENTERALIGN	Centra el menú emergente con respecto a la coordenada horizontal.
TPM_LEFTALIGN	Ajusta el lado izquierdo del menú emergente con respecto a la coordenada horizontal.
TPM_RIGHTALIGN	Ajusta el lado derecho del menú emergente con respecto a la coordenada horizontal.

**Tabla 12-14: Valores de botón del ratón para el parámetro uFlags de TrackPopupMenu**

Valor	Descripción
TPM_LEFTBUTTON	Un elemento de menú será seleccionado cuando se haga clic sobre él con el botón izquierdo del ratón.
TPM_RIGHTBUTTON	Un elemento de menú será seleccionado cuando se haga clic sobre él con el botón derecho del ratón.

**TrackPopupMenuEx****Windows.Pas****Sintaxis**

```
TrackPopupMenuEx(
    hMenu: HMENU;           {manejador de un menú emergente}
    Flags: UINT;             {opciones de posición y seguimiento}
    x: Integer;              {coordenadas horizontales de pantalla}
    y: Integer;              {coordenadas verticales de pantalla}
    Wnd: HWND;              {manejador de la ventana propietaria}
    TPMPParams: PTPMPParams {puntero a registro TPMPParams}
); BOOL;                   {devuelve TRUE o FALSE}
```

**Descripción**

Esta función es similar a la función *TrackPopupMenu*, ya que muestra un menú emergente en las coordenadas especificadas de la pantalla y envía un mensaje

*WM\_COMMAND* a la ventana *Wnd* cuando el usuario selecciona un elemento de menú. Sin embargo, mediante esta función se puede especificar adicionalmente un área rectangular en la cual el menú emergente no puede ser mostrado. Si el menú emergente fuera mostrado en este área rectangular, se moverá de forma que no caiga en esta área. Esto es útil si un control en el formulario contiene información que no debe ser solapada por el menú emergente.

### Parámetros

*hMenu*: Manejador del menú emergente que será mostrado. Este manejador es devuelto por la función *CreatePopupMenu* cuando se crea un nuevo menú emergente, o por la función *GetSubMenu* cuando se recupera un submenú o menú desplegable.

*Flags*: Una combinación de opciones que indican la alineación del menú emergente respecto a sus coordenadas y qué botón del ratón indica una selección. Este parámetro puede ser cero o una combinación de un elemento de la Tabla 12-15, un elemento de la Tabla 12-16 y un elemento de la Tabla 12-17 (opcionalmente).

*x*: La coordenada horizontal en la cual se mostrará el menú emergente, relativa a la pantalla.

*y*: La coordenada vertical en la cual se mostrará el menú emergente, relativa a la pantalla.

*Wnd*: Manejador de la ventana propietaria del menú emergente que será mostrado. Esta ventana recibirá un mensaje *WM\_COMMAND* cuando el usuario seleccione un elemento del menú. La ventana no recibirá el mensaje hasta que la función *TrackPopupMenuEx* retorne.

*TPMParams*: Puntero a un registro de tipo *TTPMParams* que contiene las coordenadas de un área rectangular, en coordenadas de pantalla, en la que el menú emergente no aparecerá. Si este área rectangular no es necesaria, a este parámetro se le puede asignar **nil**. El registro *TTPMParams* se define como:

*TTPMParams* = **packed record**

<i>cbSize</i> : UINT;	{el tamaño del registro}
<i>rcExclude</i> : TRect;	{las coordenadas del rectángulo}

**end;**

*cbSize*: Especifica el tamaño del registro, en bytes. A este campo se le debe asignar *SizeOf(TPMParams)*.

*rcExclude*: Un registro *TRect* que contiene las coordenadas, relativas a la pantalla, del área rectangular en la que el menú emergente no debe aparecer.

### Valor que devuelve

Si la función tiene éxito, devuelve TRUE; en caso contrario, devuelve FALSE. Para obtener información más amplia sobre posibles errores, se debe llamar a la función *GetLastError*.

Véase además

*CreatePopupMenu, GetSubMenu, TrackPopupMenu*

Ejemplo

#### Listado I2-I2: Mostrando un menú emergente sin oscurecer un control

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  MenuPos: TPoint;           // la posición del menú en coordenadas de pantalla
  ExtraInfo: TTPMPParams;    // el área rectangular excluida
begin
  {Si el botón derecho fue pulsado, muestra el menú emergente}
  if Button = mbRight then
    begin
      {Convierte las coordenadas cliente en coordenadas de pantalla}
      MenuPos.X := X; MenuPos.Y := Y;
      MenuPos := ClientToScreen(MenuPos);

      {Inicializa el registro TTPMPParams, haciendo que el área rectangular excluida
       sea la ocupada por el cuadro de lista ListBox1}
      ExtraInfo.cbSize := SizeOf(TTPMPParams);
      ExtraInfo.rcExclude.TopLeft := ClientToScreen(ListBox1.BoundsRect.TopLeft);
      ExtraInfo.rcExclude.BottomRight := ClientToScreen(ListBox1.BoundsRect.
        BottomRight);

      {Muestra el menú emergente}
      TrackPopupMenuEx(PopupMenu1.Handle, 0, MenuPos.X, MenuPos.Y, Form1.Handle,
        @ExtraInfo);
    end;
  end;

  {El mensaje WM_COMMAND redefinido}
procedure TForm1.WMCommand(var Msg: TMessage);
begin
  {Esto es necesario para que el control de menú emergente de Delphi reciba
   los mensajes adecuados y la funcionalidad no se altere}
  PopupMenu1.DispatchCommand(Msg.wParam);
  inherited;
end;

procedure TForm1.Item11Click(Sender: TObject);
begin
  {Muestra qué elemento menú fue seleccionado}
  Label2.Caption := TMenuItem(Sender).Caption;
end;
```

Figura 12-11:  
El menú  
emergente  
mostrado

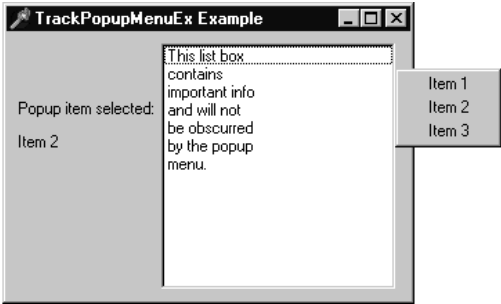


Tabla 12-15: Valores de alineación para el parámetro uFlags de TrackPopupMenuEx

Valor	Descripción
TPM_CENTERALIGN	Centra el menú emergente con respecto a la coordenada horizontal.
TPM_LEFTALIGN	Ajusta el lado izquierdo del menú emergente con respecto a la coordenada horizontal.
TPM_RIGHTALIGN	Ajusta el lado derecho del menú emergente con respecto a la coordenada horizontal.

Tabla 12-16: Valores de botón del ratón para el parámetro uFlags de TrackPopupMenuEx

Valor	Descripción
TPM_LEFTBUTTON	Un elemento de menú será seleccionado cuando se haga clic sobre él con el botón izquierdo del ratón.
TPM_RIGHTBUTTON	Un elemento de menú será seleccionado cuando se haga clic sobre él con el botón derecho del ratón.

Tabla 12-17: Valores de prioridad de alineación para el parámetro uFlags de TrackPopupMenuEx

Valor	Descripción
TPM_HORIZONTAL	Si el menú va a solapar el área rectangular excluida y tiene que ser desplazado, la alineación horizontal debe tener prioridad sobre la vertical.
TPM_VERTICAL	Si el menú va a solapar el área rectangular excluida y tiene que ser desplazado, la alineación vertical debe tener prioridad sobre la horizontal.



## Apéndice A

# Mensajes

Si las funciones de mensajes como *GetMessage*, *TranslateMessage* y *DispatchMessage* son los capilares y las arterias de Windows, los mensajes son las células sanguíneas. Hasta la más simple acción del usuario produce una docena o más de mensajes que son intercambiados entre la aplicación y el sistema. Aunque es posible desarrollar aplicaciones Delphi muy complejas que no tengan nada que ver con los mensajes, en ocasiones manipular o enviar un mensaje permite obtener una solución más elegante y simple de un problema.

Este apéndice no describe todos los mensajes disponibles en Windows. De una descripción completa resultaría un manual del tamaño de este libro. Este capítulo describe las estructuras de datos sobre los cuales Delphi ha definido un gran número de mensajes. Estos registros ayudan al desarrollador a tratar los mensajes de una manera eficiente, ya que son más intuitivos e informativos que los campos *wParam* y *lParam* de un mensaje tradicional de Windows.

## WM\_ACTIVATE

### Sintaxis

```
TWMActivate = record
    Msg: Cardinal;           {identificador del mensaje}
    Active: Word;            {opción de activación}
    Minimized: WordBool;     {estado minimizado}
    ActiveWindow: HWND;      {manejador de una ventana}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Una ventana recibe este mensaje cuando va a ser activada o desactivada. Este mensaje es enviado primero a la ventana que es desactivada y luego a la ventana que es activada. Si la ventana que va a ser activada no está minimizada, ésta recibe el foco del teclado. Si la ventana fue activada por un clic del ratón, también recibirá el mensaje *WM\_MOUSEACTIVATE*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ACTIVATE*.

*Active*: Indica si la ventana va a ser activada o desactivada y puede tomar un valor de la Tabla A-1.

*Minimized*: Indica el estado minimizado de la ventana que va a ser activada o desactivada. Si la ventana no está minimizada, este campo valdrá FALSE.

*ActiveWindow*: Contiene un manejador de la ventana que va a ser activada o desactivada. Este valor depende del valor del campo *Active*. Si el campo *Active* contiene *WA\_INACTIVE*, este campo contiene un manejador de la ventana que va a ser activada. Si el campo *Active* contiene *WA\_ACTIVE* o *WA\_CLICKACTIVE*, este campo contiene un manejador de la ventana que va a ser desactivada. Este campo puede contener cero.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*SetActiveWindow*, *WM\_MOUSEACTIVATE*, *WM\_NCACTIVATE*

**Tabla A-1: Valores de *TWMActivate.Active***

Valor	Descripción
<i>WA_ACTIVE</i>	La ventana fue activada por un mecanismo diferente al clic del ratón, ya sea mediante código o mediante órdenes de teclado.
<i>WA_CLICKACTIVE</i>	La ventana fue activada por un clic del ratón.
<i>WA_INACTIVE</i>	La ventana va a ser desactivada.

***WM\_ACTIVATEAPP*****Sintaxis**

```
TWMActivateApp = record
    Msg: Cardinal;           {identificador del mensaje}
    Active: BOOL;            {opción de activación}
    ThreadId: Longint;       {ID del hilo de ejecución}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Este mensaje es enviado cuando una ventana que no pertenece a la aplicación actual va a ser activada. Es enviado tanto a la aplicación propietaria de la ventana que va a ser activada, como a la aplicación propietaria de la ventana que va a ser desactivada.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ACTIVATEAPP*.

*Active*: Indica si la ventana va a ser activada o desactivada. Si la ventana va a ser activada, a este campo se le asigna TRUE; en caso contrario, se le asigna FALSE.

*ThreadId*: Un identificador de hilo de ejecución para la ventana que va a ser activada o desactivada. Si al campo *Active* se le asigna TRUE, este campo contiene el identificador del hilo que posee la ventana que va a ser desactivada. Si *Active* es FALSE, este campo contiene el identificador del hilo que posee la ventana que va a ser activada.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_ACTIVATE*

**WM\_ASKCBFORMATNAME****Sintaxis**

```
TWMAAskCBFormatName = record
    Msg: Cardinal;           {identificador del mensaje}
    NameLen: Word;          {tamaño del buffer en bytes}
    Unused: Word;           {no se utiliza}
    FormatName: PChar;       {buffer de salida}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Un visualizador del portapapeles envía el mensaje *WM\_ASKCBFORMATNAME* a un propietario del portapapeles para obtener el nombre de un formato de portapapeles *CF\_OWNERDISPLAY*. Como respuesta, el propietario del portapapeles coloca el nombre del formato en el *buffer* de salida suministrado por el visualizador.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ASKCBFORMATNAME*.

*NameLen*: Especifica el tamaño del *buffer* de salida al que apunta el campo *FormatName*, en bytes.

*Unused*: Este campo no es utilizado por este mensaje.

*FormatName*: Puntero a un *buffer* de cadena de caracteres terminada en nulo que almacenará el nombre del formato del portapapeles.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_DRAWCLIPBOARD*

## **WM\_CANCELMODE**

### **Sintaxis**

```
TWMCancelMode = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### **Descripción**

Una ventana con el foco de entrada recibe un mensaje *WM\_CANCELMODE* cuando un cuadro de diálogo o de mensaje es mostrado. La ventana que tenía el foco puede cancelar modos de operación tales como el modo de captura del ratón. Al recibir el mensaje, la función *DefWindowProc* cancelará el procesamiento interno, el procesamiento de menús y la captura del ratón.

### **Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CANCELMODE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *ReleaseCapture*

## **WM\_CHANGECHAIN**

### **Sintaxis**

```
TWMChangeCBChain = record
    Msg: Cardinal;           {identificador del mensaje}
    Remove: HWND;           {ventana que va a ser eliminada de la cadena}
    Next: HWND;             {próxima ventana en la cadena}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### **Descripción**

Cuando una ventana va a ser eliminada de la cadena de ventanas visualizadoras del portapapeles, un mensaje *WM\_CHANGECHAIN* es enviado a la primera ventana en la cadena. Cada ventana almacena un manejador de la próxima ventana en la cadena de

visualizadores del portapapeles. Cuando una nueva ventana visualizadora del portapapeles es creada, se convierte en la primera ventana en la cadena de visualizadores del portapapeles. Al recibir este mensaje, cada ventana visualizadora del portapapeles debe llamar a la función *SendMessage* para enviar el mensaje a la próxima ventana en la cadena, a menos que la próxima ventana sea la que va a ser eliminada. En este caso, la ventana debe guardar el manejador de ventana recibida en el campo *Next*, ya que ésta será la siguiente ventana en la cadena.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CHANGECHAIN*.

*Remove*: El manejador de la ventana que será eliminada de la cadena de ventanas visualizadoras del portapapeles.

*Next*: El manejador de la próxima ventana en la cadena de ventanas visualizadoras del portapapeles. Este campo contendrá cero si no hay más ventanas en la cadena de visualizadores del portapapeles.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*SendMessage*, *SetClipboardViewer*

## WM\_CHAR

### Sintaxis

```
TWMChar = record
    Msg: Cardinal;           {identificador del mensaje}
    CharCode: Word;         {código de carácter}
    Unused: Word;           {no se utiliza}
    KeyData: Longint;       {contiene información diversa}
    Result: Longint;        {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Cuando se pulsa una tecla, un mensaje *WM\_CHAR* es enviado a la ventana que tiene el foco del teclado. Este mensaje *WM\_CHAR* es el resultado de un mensaje *WM\_KEYDOWN* que es convertido por la función *TranslateMessage*.

Las teclas extendidas en los teclados de 101 ó 102 teclas son:

- En la sección principal del teclado, las teclas **Alt** y **Ctrl** de la derecha.
- A la izquierda del teclado numérico, las teclas **Ins**, **Supr**, **Inicio**, **Fin**, **Av.Pág.**, **Re.Pág.**, y las cuatro teclas de flechas.
- En el teclado numérico, las teclas **Bloq.Núm.**, dividir (/) y la tecla **Intro**.

- Las teclas **PrintScrn** y **Break**.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante del identificador de mensaje *WM\_CHAR*.

*CharCode*: El código de carácter de la tecla que fue pulsada.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado previo de la tecla e indicador de estado de transición. La Tabla A-2 muestra qué información se almacena en cada posición de bit, dentro de los 32 bits del campo *KeyData*. La palabra más significativa (bits del 16 al 31), pertenece al mensaje anterior *WM\_KEYDOWN*, que provocó el mensaje *WM\_CHAR* mediante la función *TranslateMessage*.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*TranslateMessage*, *WM\_KEYDOWN*

**Tabla A-2: Valores de TWMChar.KeyData**

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.
29	El código de contexto. Si la tecla Alt estaba presionada mientras otra tecla fue pulsada, este bit contiene 1; en caso contrario, contiene 0.
30	El estado previo de la tecla. Si la tecla estaba presionada antes de que se enviara el mensaje, este bit contiene 1; en caso contrario contiene 0.
31	El estado de transición. Si la tecla está siendo liberada, este bit contiene 1; en caso contrario, contiene 0.

### WM\_CHAR TO ITEM

#### Sintaxis

TWMCharToItem = **record**

Msg: Cardinal;	{identificador del mensaje}
Key: Word;	{valor de la tecla}
CaretPos: Word;	{posición del cursor de edición}
ListBox: HWND;	{manejador de cuadro de lista}

```
Result: Longint;      {devuelve -1 ó -2 si fue manejado}
end;
```

### Descripción

Cuando un cuadro de lista con estilo *LBS\_WANTKEYBOARDINPUT* recibe un mensaje *WM\_CHAR*, envía un mensaje *WM\_CHAROITEM* a su propietaria.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CHAROITEM*.

*Key*: El valor de la tecla, enviado por el mensaje *WM\_CHAR*.

*CaretPos*: La posición del cursor de edición en el cuadro de lista.

*ListBox*: El manejador del cuadro de lista.

*Result*: Si la aplicación maneja este mensaje, deberá asignar a este campo -1 ó -2. Si el cuadro de lista debe ejecutar su acción por defecto en adición a cualquier tratamiento del mensaje específico de la aplicación, se deberá asignar a este campo el índice de base cero del elemento del cuadro de lista.

### Véase además

*DefWindowProc*, *WM\_CHAR*

## WM\_CHILDACTIVATE

### Sintaxis

```
TWMChildActivate = record
    Msg: Cardinal;          {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;        {devuelve cero (0) si fue manejado}
end;
```

### Descripción

El mensaje *WM\_CHILDACTIVATE* es enviado a una ventana hija MDI cuando es activada.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CHILDACTIVATE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*MoveWindow, SetWindowPos*

## WM\_CLEAR

### Sintaxis

```
TWMClear = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {no se utiliza}
end;
```

### Descripción

El mensaje *WM\_CLEAR* es enviado a un cuadro de combinación o a un control de edición cuando la selección actual va a ser eliminada o vaciada.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CLEAR*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

Véase además

*WM\_COPY, WM\_CUT, WM\_PASTE*

## WM\_CLOSE

### Sintaxis

```
TWMClose = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Cuando una ventana o una aplicación va a ser cerrada o finalizada, recibe un mensaje *WM\_CLOSE*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CLOSE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*DefWindowProc*, *DestroyWindow*

**WM\_COMMAND****Sintaxis**

```
TWMCommand = record
    Msg: Cardinal;           {identificador del mensaje}
    ItemID: Word;            {identificador de elemento}
    NotifyCode: Word;        {código de notificación}
    Ctl: HWND;               {manejador del control}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

El mensaje *WM\_COMMAND* es enviado cuando el usuario pulsa una tecla aceleradora, entonces un control envía un mensaje de notificación a una ventana madre o el usuario hace una selección de menú. Este mensaje no es enviado cuando el usuario hace una selección de un menú de sistema. Al hacer una selección de un menú de sistema se envía un mensaje *WM\_SYSCOMMAND*. El menú de sistema es el menú que se presenta cuando se hace clic en el icono de la esquina superior izquierda de la ventana. Si la ventana es minimizada y se pulsa una tecla aceleradora, el mensaje *WM\_COMMAND* será enviado sólo si la tecla no se corresponde con ningún elemento de menú en el menú de la ventana o en el menú de sistema.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante del identificador de mensaje *WM\_COMMAND*.

*ItemID*: El identificador del elemento de menú, control o acelerador. Este campo tendrá valor cero si el usuario selecciona un separador de un menú.

*NotifyCode*: Indica si el mensaje se originó en un control, elemento de menú, o acelerador. Este campo tendrá valor cero si el mensaje proviene de un elemento de menú o uno si proviene de un acelerador. Cualquier otro valor indica un código de notificación específico proveniente de un control.

*Ctl*: El manejador del control que envía el mensaje. Este campo tendrá valor cero si el mensaje ha sido enviado como resultado de un acelerador o de una selección de menú.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_SYSCOMMAND*

## WM\_COMPACTING

### Sintaxis

```
TWMCompacting = record
    Msg: Cardinal;           {identificador del mensaje}
    CompactRatio: Longint;   {relación de tiempo de la CPU}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

El mensaje *WM\_COMPACTING* indica que hay poca memoria. Es enviado a todas las ventanas de nivel superior cuando más de un octavo del tiempo del sistema, en un intervalo de 30 a 60 segundos va a ser utilizado para compactar la memoria. La aplicación debe responder liberando la memoria que no utilice o no le sea necesaria.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_COMPACTING*.

*CompactRatio*: La relación entre el tiempo de la CPU dedicado a compactar memoria y el tiempo de la CPU dedicado a las demás operaciones.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_CLOSE*, *WM\_DESTROY*, *WM\_QUIT*

## WM\_COMPAREITEM

### Sintaxis

```
TWMCompareItem = record
    Msg: Cardinal;           {identificador del mensaje}
    Ctl: Hwnd;               {manejador del control}
    CompareItemStruct: PCompareItemStruct; {puntero al registro}
    Result: Longint;         {devuelve el resultado de la
```

comparación}  
**end;**

### Descripción

El mensaje *WM\_COMPAREITEM* es enviado a la ventana propietaria de un cuadro de lista o de combinación cuando una aplicación añade un elemento al objeto, si éste fue creado con un estilo *LBS\_SORT* o *CBS\_SORT*, respectivamente. El mensaje identifica dos elementos cuyas posiciones relativas en la lista de elementos será determinada. El valor devuelto indica el resultado de esta comparación. Para una lista que ya tenga varios elementos, Windows envía tantos mensajes *WM\_COMPAREITEM* como sea necesario para determinar la posición exacta del nuevo elemento.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_COMPAREITEM*.

*Ctl*: El manejador del control que envía el mensaje.

*CompareItemStruct*: Un puntero al registro *TCompareItemStruct* que contiene información sobre los dos elementos cuya posición en la lista va a ser determinada. El registro *TCompareItemStruct* se define como:

*TCompareItemStruct* = packed **record**

<i>CtlType</i> : UINT;	{tipo de control}
<i>CtlID</i> : UINT;	{identificador del control}
<i>hwndItem</i> : HWND;	{manejador del control}
<i>itemID1</i> : UINT;	{índice del primer elemento}
<i>itemData1</i> : DWORD;	{los datos que están siendo comparados}
<i>itemID2</i> : UINT;	{índice del segundo elemento}
<i>itemData2</i> : DWORD;	{los datos que están siendo comparados}
<i>dwLocaleId</i> : DWORD;	{identificador de lenguaje local}

**end;**

*CtlType*: Identifica el tipo de control que genera el mensaje, y puede contener *ODT\_LISTBOX* si los elementos a comparar pertenecen a un cuadro de lista dibujado por el propietario, u *ODT\_COMBOBOX* si los elementos a comparar pertenecen a un cuadro de combinación dibujado por el propietario.

*CtlID*: El identificador del cuadro de lista o de combinación.

*hwndItem*: El manejador del control que envía el mensaje.

*itemID1*: El índice del primer elemento a comparar.

*itemData1*: Los datos que serán comparados. Este es el parámetro *lParam* del mensaje enviado por la aplicación que añadió el primer elemento que va a ser comparado.

*itemID2*: El índice del segundo elemento a comparar.

*itemData2*: Los datos que serán comparados. Este es el parámetro *lParam* del mensaje enviado por la aplicación que añadió el segundo elemento que va a ser comparado.

*dwLocaleId*: El identificador de localidad que puede ser usado como base de la comparación.

*Result*: El mensaje devuelve un valor que indica el orden relativo de los dos elementos comparados. El mensaje debe devolver -1 si el elemento identificado por el campo *itemData1* debe preceder al elemento identificado por el campo *itemData2* en la lista, debe devolver 0 si los dos elementos son considerados iguales y debe devolver 1 si *itemData2* debe preceder a *itemData1* en la lista.

Véase además

*WM\_DELETEITEM*

## WM\_COPY

### Sintaxis

```
TWMCopy = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {no se utiliza}
end;
```

### Descripción

El mensaje *WM\_COPY* es enviado a un control de edición o cuadro de combinación cuando la aplicación o el usuario invocan un método de copiar (como **Ctrl+C**) que colocará el texto seleccionado en el portapapeles. El control que recibe el mensaje deberá colocar su texto seleccionado en el portapapeles en el formato *CF\_TEXT*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_COPY*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

Véase además

*WM\_CLEAR*, *WM\_CUT*, *WM\_PASTE*

**WM\_COPYDATA***Sintaxis*

```

TWMCopyData = record
    Msg: Cardinal;           {identificador del mensaje}
    From: HWND;             {manejador del emisor}
    CopyDataStruct: PCopyDataStruct; {puntero a los datos enviados}
    Result: Longint;         {devuelve uno (1) si fue manejado}
end;

```

*Descripción*

El mensaje *WM\_COPYDATA* permite a una aplicación enviar datos a otra aplicación. Este mensaje puede ser enviado sólo mediante la función *SendMessage*, y los datos identificados por el campo *CopyDataStruct* no pueden contener ningún puntero ni otra referencia a objetos o a memoria que no sean accesibles al proceso que los recibe. Durante el procesamiento, la aplicación que recibe los datos debe considerarlos como de sólo lectura. Si tiene que modificarlos, deberá copiarlos a un *buffer* local.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_COPYDATA*.

*From*: El manejador de la ventana que envía el mensaje.

*CopyDataStruct*: Un puntero a un registro *TCopyDataStruct* que contiene los datos que serán enviados a la otra aplicación. El registro *TCopyDataStruct* se define como:

```

TCopyDataStruct = packed record
    dwData: DWORD;          {valor de datos de 32 bits}
    cbData: DWORD;          {tamaño del buffer lpData}
    lpData: Pointer;        {puntero a datos adicionales}
end;

```

*dwData*: Un valor de datos de 32 bits que será pasado a la otra aplicación.

*cbData*: Especifica el tamaño del *buffer* de datos al que apunta el campo *lpData*.

*lpData*: Un puntero a datos adicionales.

*Result*: Si la aplicación maneja este mensaje, deberá asignar uno (1) a este campo.

*Véase además*

*PostMessage*, *SendMessage*

**WM\_CREATE***Sintaxis*

```

TWMLCreate = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: Integer;         {no se utiliza}
    CreateStruct: PCreateStruct; {puntero al registro creado}
    Result: Longint;         {devuelve un código de creación de ventana}
end;

```

*Descripción*

Cuando una aplicación crea una nueva ventana llamando a las funciones *CreateWindow* o *CreateWindowEx*, Windows envía un mensaje *WM\_CREATE* al procedimiento de ventana de la nueva ventana después de que ésta es creada, pero antes de que se haga visible y antes de que las funciones *CreateWindow* o *CreateWindowEx* retornen. El registro de datos enviado con el mensaje contiene especificaciones para la nueva ventana, y es en esencia la misma información pasada a la función *CreateWindowEx*.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CREATE*.

*Unused*: Este campo no es utilizado por este mensaje.

*CreateStruct*: Un puntero al registro *TCreateStruct* que contiene información sobre la nueva ventana. El registro *TCreateStruct* se define como:

```

TCreateStruct = packed record
    lpCreateParams: Pointer;   {puntero a datos definidos por la aplicación}
    hInstance: HINST;         {manejador de instancia del módulo}
    hMenu: HMENU;             {manejador de menú o ident. de ventana hija}
    hwndParent: HWND;         {manejador de la ventana madre}
    cy: Integer;              {altura inicial de la ventana}
    cx: Integer;              {ancho inicial de la ventana}
    y: Integer;               {posición vertical inicial}
    x: Integer;               {posición horizontal inicial}
    style: Longint;           {indicadores de estilo de ventana}
    lpszName: PAnsiChar;      {puntero a la cadena del nombre de la ventana}
    lpszClass: PAnsiChar;     {puntero a la cadena del nombre de la clase}
    dwExStyle: DWORD;         {indicadores de estilo de ventana extendidos}
end;

```

El campo *lpCreateParams* es un puntero a datos definidos por la aplicación. Los otros campos de este registro contienen la información pasada a través de los parámetros a las funciones *CreateWindow* o *CreateWindowEx*.

*Result:* Si la aplicación maneja este mensaje, deberá devolver cero para indicar que la ventana puede ser creada. Si la aplicación devuelve -1, la nueva ventana será destruida y las funciones *CreateWindow* o *CreateWindowEx* que generaron el mensaje devolverán el valor cero para el manejador de la ventana.

Véase además

*CreateWindow*, *CreateWindowEx*, *WM\_NCCREATE*

## WM\_CTLCOLORBTN

### Sintaxis

```
TWMCtlColorBtn = record
    Msg: Cardinal;           {identificador del mensaje}
    ChildDC: HDC;            {contexto de dispositivo del botón}
    ChildWnd: HWND;          {manejador de la ventana del botón}
    Result: Longint;         {devuelve un manejador de la brocha}
end;
```

### Descripción

El mensaje *WM\_CTLCOLORBTN* es enviado a la ventana madre de un botón cuando el botón va a ser dibujado. La ventana madre puede responder asignando el color del texto y del fondo del botón. Este mensaje es enviado únicamente dentro de un hilo de ejecución, y no puede ser enviado entre hilos.

### Campos

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CTLCOLORBTN*.

*ChildDC:* Manejador del contexto de dispositivo del botón que será dibujado.

*ChildWnd:* Manejador de ventana del botón que será dibujado.

*Result:* Si la aplicación maneja este mensaje, deberá devolver un manejador de brocha que Windows usará para dibujar el fondo del botón.

Véase además

*DefWindowProc*, *RealizePalette*, *SelectPalette*, *WM\_CTLCOLOREDIT*,  
*WM\_CTLCOLORLISTBOX*, *WM\_CTLCOLORMSGBOX*,  
*WM\_CTLCOLORSCROLLBAR*, *WM\_CTLCOLORSTATIC*

## WM\_CTLCOLOREDIT

### Sintaxis

```
TWMCtlColorEdit = record
```

Msg: Cardinal;	{identificador del mensaje}
ChildDC: HDC;	{contexto de dispositivo del cuadro de edición}
ChildWnd: HWND;	{manejador de la ventana del cuadro de edición}
Result: Longint;	{devuelve un manejador de brocha}

**end;**

#### Descripción

El mensaje *WM\_CTLCOLOREDIT* es enviado a la ventana madre de un control de edición cuando éste va a ser dibujado. La ventana madre puede responder asignando el color del texto y del fondo. Este mensaje es enviado únicamente dentro de un hilo de ejecución, y no puede ser enviado entre hilos.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CTLCOLOREDIT*.

*ChildDC*: Manejador del contexto de dispositivo del control de edición que va a ser dibujado.

*ChildWnd*: Manejador de ventana del control de edición que será dibujado.

*Result*: Si la aplicación maneja este mensaje, deberá devolver un manejador de brocha que Windows usará para dibujar el fondo del control de edición.

#### Véase además

*DefWindowProc*, *RealizePalette*, *SelectPalette*, *WM\_CTLCOLORBTN*,  
*WM\_CTLCOLORLISTBOX*, *WM\_CTLCOLORMSGBOX*,  
*WM\_CTLCOLORSCROLLBAR*, *WM\_CTLCOLORSTATIC*

### **WM\_CTLCOLORLISTBOX**

#### Sintaxis

TWMctlColorListbox = **record**

Msg: Cardinal;	{identificador del mensaje}
ChildDC: HDC;	{contexto de dispositivo del cuadro de lista}
ChildWnd: HWND;	{manejador la ventana del cuadro de lista}
Result: Longint;	{devuelve un manejador de brocha}

**end;**

#### Descripción

El mensaje *WM\_CTLCOLORLISTBOX* es enviado a la ventana madre de un cuadro de lista cuando éste va a ser dibujado. La ventana madre puede responder asignando los colores del texto y del fondo del cuadro de lista. Este mensaje es enviado únicamente dentro de un hilo de ejecución, y no puede ser enviado entre hilos.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CTLCOLORLISTBOX*.

*ChildDC*: Manejador del contexto de dispositivo del cuadro de lista que será dibujado.

*ChildWnd*: Manejador de ventana del cuadro de lista que será dibujado.

*Result*: Si la aplicación maneja este mensaje, deberá devolver un manejador de brocha que Windows usará para dibujar el fondo del cuadro de lista.

**Véase además**

*DefWindowProc*, *RealizePalette*, *SelectPalette*, *WM\_CTLCOLORBTN*,  
*WM\_CTLCOLOREDIT*, *WM\_CTLCOLORMSGBOX*, *WM\_CTLCOLORSCROLLBAR*,  
*WM\_CTLCOLORSTATIC*

**WM\_CTLCOLORMSGBOX****Sintaxis**

```
TWMCtlColorMsgbox = record
    Msg: Cardinal;           {identificador del mensaje}
    ChildDC: HDC;            {contexto de dispositivo del cuadro de mensaje}
    ChildWnd: HWND;         {manejador de la ventana del cuadro de mensaje}
    Result: Longint;         {devuelve un manejador de brocha}
end;
```

**Descripción**

El mensaje *WM\_CTLCOLORMSGBOX* es enviado a la ventana madre de un cuadro de mensaje cuando éste va a ser dibujado. La ventana madre puede responder asignando los colores del texto y de fondo del cuadro de mensaje. Este mensaje es enviado únicamente dentro de un hilo de ejecución, y no puede ser enviado entre hilos.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CTLCOLORMSGBOX*.

*ChildDC*: Manejador del contexto de dispositivo del cuadro de mensaje que será dibujado.

*ChildWnd*: Manejador de la ventana que será dibujada.

*Result*: Si la aplicación maneja este mensaje, deberá devolver un manejador de brocha que Windows usará para dibujar el fondo del cuadro de mensaje.

Véase además

*DefWindowProc, RealizePalette, SelectPalette, WM\_CTLCOLORBTN, WM\_CTLCOLOREDIT, WM\_CTLCOLORLISTBOX, WM\_CTLCOLORSCROLLBAR, WM\_CTLCOLORSTATIC*

## WM\_CTLCOLORSCROLLBAR

### Sintaxis

```
TWMctlColorScrollbar = record
    Msg: Cardinal;           {identificador del mensaje}
    ChildDC: HDC;            {contexto de disp. de barra de desplazamiento}
    ChildWnd: HWND;          {manejador de ventana de barra de desplazamiento}
    Result: Longint;         {devuelve un manejador de brocha}
end;
```

### Descripción

El mensaje *WM\_CTLCOLORSCROLLBAR* es enviado a la ventana madre de una barra de desplazamiento cuando ésta va a ser dibujada. La ventana madre puede responder asignando los colores del texto y del fondo de la barra de desplazamiento. Este mensaje es enviado únicamente dentro de un hilo de ejecución, y no puede ser enviado entre hilos.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CTLCOLORSCROLLBAR*.

*ChildDC*: Manejador del contexto de dispositivo de la barra de desplazamiento que será dibujada.

*ChildWnd*: Manejador de ventana de la barra de desplazamiento que será dibujada.

*Result*: Si la aplicación maneja este mensaje, deberá devolver un manejador de brocha que Windows usará para dibujar el fondo de la barra de desplazamiento.

Véase además

*DefWindowProc, RealizePalette, SelectPalette, WM\_CTLCOLORBTN, WM\_CTLCOLOREDIT, WM\_CTLCOLORLISTBOX, WM\_CTLCOLORMSGBOX, WM\_CTLCOLORSTATIC*

## WM\_CTLCOLORSTATIC

### Sintaxis

```
TWMctlColorStatic = record
```

```

Msg: Cardinal;           {identificador del mensaje}
ChildDC: HDC;            {contexto de dispositivo de control estático}
ChildWnd: HWND;          {manejador de ventana de control estático}
Result: Longint;         {devuelve un manejador de brocha}
end;

```

**Descripción**

El mensaje *WM\_CTLCOLORSTATIC* es enviado a la ventana madre de un control estático cuando éste va a ser dibujado. La ventana madre puede responder asignando los colores del texto y del fondo del control estático. Este mensaje es enviado únicamente dentro de un hilo de ejecución, y no puede ser enviado entre hilos.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CTLCOLORSTATIC*.

*ChildDC*: Manejador del contexto de dispositivo del control estático que será dibujado.

*ChildWnd*: Manejador de ventana del control estático que será dibujado.

*Result*: Si la aplicación maneja este mensaje, deberá devolver un manejador de brocha que Windows usará para dibujar el fondo del control estático.

**Véase además**

*DefWindowProc*, *RealizePalette*, *SelectPalette*, *WM\_CTLCOLORBTN*,  
*WM\_CTLCOLOREDIT*, *WM\_CTLCOLORLISTBOX*, *WM\_CTLCOLORMSGBOX*,  
*WM\_CTLCOLORSCROLLBAR*

**WM\_CUT****Sintaxis**

```

TWMCut = record
  Msg: Cardinal;           {identificador del mensaje}
  Unused: array[0..3] of Word; {no se utiliza}
  Result: Longint;         {no se utiliza}
end;

```

**Descripción**

El mensaje *WM\_CUT* es enviado a un control de edición o cuadro de combinación cuando la aplicación o el usuario invocan a un método de corte (como **Ctrl+X**) que coloca el texto seleccionado en el portapapeles. El control que recibe el mensaje debe borrar su texto seleccionado y colocarlo en el portapapeles en el formato *CF\_TEXT*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_CUT*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

**Véase además**

*WM\_CLEAR*, *WM\_COPY*, *WM\_PASTE*

**WM\_DEADCHAR****Sintaxis**

```
TWMDeadChar = record
    Msg: Cardinal;           {identificador del mensaje}
    CharCode: Word;          {código del carácter}
    Unused: Word;            {no se utiliza}
    KeyData: Longint;        {contiene información varia}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Cuando un mensaje *WM\_KEYDOWN* es traducido por una llamada a la función *TranslateMessage*, un mensaje *WM\_DEADCHAR* es enviado a la ventana que tiene el foco del teclado. Este mensaje es generado como resultado de pulsar una tecla “muerta”. Una tecla muerta es aquella que genera un carácter adicional que es usado en combinación con otra tecla, creando así un carácter combinado o compuesto. El ejemplo típico es el de un carácter con un acento o marca diacrítica. La tecla muerta que identifica el acento o marca diacrítica es introducida primero, seguida por la tecla que identifica el carácter que tendrá aplicada la marca.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DEADCHAR*.

*CharCode*: El código del carácter de la tecla que fue presionada.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado previo de la tecla e indicador de estado de transición. La Tabla A-3 muestra qué información se almacena en cada posición de bit, dentro de los 32 bits del campo *KeyData*. La palabra más significativa (bits del 16 al 31), pertenece al mensaje anterior *WM\_KEYDOWN*, que provocó el mensaje *WM\_DEADCHAR* mediante la función *TranslateMessage*.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*TranslateMessage*, *WM\_KEYDOWN*, *WM\_KEYUP*, *WM\_SYSDEADCHAR*,  
*WM\_SYSKEYDOWN*

**Tabla A-3: Valores de TWMDDeadChar.KeyData**

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.
29	El código de contexto. Si la tecla Alt estaba presionada mientras la tecla fue pulsada, este bit contiene 1; en caso contrario, contiene 0.
30	El estado previo de la tecla. Si la tecla estaba presionada antes de que se enviara el mensaje, este bit contiene 1; en caso contrario, contiene 0.
31	El estado de transición. Si la tecla está siendo liberada, este bit contiene 1; en caso contrario, contiene 0.

## **WM\_DELETEITEM**

### *Sintaxis*

```

TWMDDeleteItem = record
    Msg: Cardinal;           {identificador del mensaje}
    Ctl: HWND;               {manejador del control}
    DeleteItemStruct: PDeleteItemStruct; {puntero a un registro de elementos
                                         eliminados}
    Result: Longint;         {devuelve uno (1) si fue manejado}
end;

```

### *Descripción*

La ventana propietaria de un cuadro de lista o de combinación recibe un mensaje *WM\_DELETEITEM* cuando el control es destruido o cuando un elemento va a ser eliminado del control como resultado de un mensaje *LB\_DELETESTRING*, *LB\_RESETCONTENT*, *CB\_DELETESTRING*, o *CB\_RESETCONTENT*. Cuando se eliminan varios elementos, un mensaje *WM\_DELETEITEM* es enviado por cada elemento a eliminar. Bajo Windows NT, este mensaje es enviado sólo cuando se eliminan elementos de un cuadro de lista o de combinación dibujado por el propietario.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DELETEITEM*.

*Ctl*: Manejador del cuadro de lista o de combinación que envía el mensaje.

*DeleteItemStruct*: Puntero a un registro *TDeleteItemStruct* que contiene información sobre el elemento a ser eliminado. El registro *TDeleteItemStruct* se define como:

*TDeleteItemStruct* = **packed record**

<i>CtlType</i> : UINT;	{tipo de control}
<i>CtlID</i> : UINT;	{identificador del control}
<i>itemID</i> : UINT;	{índice del elemento dentro del control}
<i>hwndItem</i> : HWND;	{manejador del control}
<i>itemData</i> : UINT;	{elemento de datos}

**end;**

*CtlType*: Contiene un valor que especifica el tipo de control en el que el elemento ha sido eliminado. Este campo contendrá *ODT\_LISTBOX*, que indica un cuadro de lista, o *ODT\_COMBOBOX*, que indica un cuadro de combinación.

*CtlID*: El identificador del cuadro de lista o de combinación.

*itemID*: El índice del elemento a ser eliminado del control.

*hwndItem*: Manejador del control del que se elimina el elemento.

*itemData*: Los datos definidos por la aplicación para el elemento especificado.

*Result*: Si la aplicación maneja este mensaje, deberá asignar uno (1) a este campo.

Véase además

*WM\_COMPAREITEM*

## WM\_DESTROY

### Sintaxis

*TWMDestroy* = **record**

<i>Msg</i> : Cardinal;	{identificador del mensaje}
<i>Unused</i> : <b>array</b> [0..3] <b>of</b> Word;	{no se utiliza}
<i>Result</i> : Longint;	{devuelve cero (0) si fue manejado}

**end;**

### Descripción

El mensaje *WM\_DESTROY* es enviado a una ventana que va a ser destruida, después que es borrada de la pantalla, pero antes de que el manejador de la ventana sea invalidado. Durante el proceso de destrucción, el mensaje *WM\_DESTROY* es pasado a todas las ventanas hijas. Si la ventana es parte de la cadena de visualizadores del

portapapeles, deberá quitarse a sí misma de esa cadena llamando a la función *ChangeClipboardChain* antes de retornar del mensaje *WM\_DESTROY*.

#### **Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DESTROY*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

#### **Véase además**

*ChangeClipboardChain*, *DestroyWindow*, *PostQuitMessage*, *SetClipboardViewer*, *WM\_CLOSE*

### **WM\_DESTROYCLIPBOARD**

#### **Sintaxis**

```
TWMDestroyClipboard = record
    Msg: Cardinal;                {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;              {devuelve cero (0) si fue manejado}
end;
```

#### **Descripción**

Cuando la función *EmptyClipboard* es llamada para vaciar el portapapeles, el mensaje *WM\_DESTROYCLIPBOARD* es enviado a la ventana propietaria del portapapeles.

#### **Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DESTROYCLIPBOARD*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

#### **Véase además**

*EmptyClipboard*

### **WM\_DEVMODECHANGE**

#### **Sintaxis**

```
TWMDevModeChange = record
    Msg: Cardinal;                {identificador del mensaje}
```

Unused: Integer;	{no se utiliza}
Device: PChar;	{nombre del dispositivo}
Result: Longint;	{devuelve cero (0) si fue manejado}

**end;**

#### Descripción

Todas las ventanas de nivel superior reciben un mensaje *WM\_DEVMODECHANGE* cuando los atributos del modo de dispositivo son cambiados por el usuario.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DEVMODECHANGE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Device*: Un puntero a una cadena de caracteres terminada en nulo que contiene el nombre del dispositivo que va a ser cambiado, tal como está almacenado en el fichero **Win.ini** o en el registro de Windows.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

#### Véase además

*RegSetValue*, *RegSetValueEx*

### WM\_DISPLAYCHANGE

#### Sintaxis

```
TWMDisplayChange = record
  Msg: Cardinal;           {identificador del mensaje}
  BitsPerPixel: Integer;   {profundidad de color}
  Width: Word;             {tamaño horizontal de la pantalla}
  Height: Word;            {tamaño vertical de la pantalla}
end;
```

#### Descripción

El mensaje *WM\_DISPLAYCHANGE* es enviado a todas las ventanas de nivel superior cuando cambia la resolución de la pantalla.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DISPLAYCHANGE*.

*BitsPerPixel*: La profundidad de color de la pantalla, en bits por píxel.

*Width*: El tamaño horizontal de la pantalla, en píxeles.

*Height*: El tamaño vertical de la pantalla, en píxeles.

Véase además

*ChangeDisplaySettings*

## WM\_DRAWCLIPBOARD

### Sintaxis

```
TWMDrawClipboard = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {no se utiliza}
end;
```

### Descripción

El mensaje *WM\_DRAWCLIPBOARD* es enviado a la primera ventana en la cadena de visualizadores del portapapeles cuando el contenido del portapapeles cambia, indicando que la ventana del visualizador debe refrescar su contenido para reflejar el nuevo contenido del portapapeles. Sólo las ventanas añadidas a la cadena de visualizadores del portapapeles mediante una llamada a la función *SetClipboardViewer* recibirán este mensaje. Cada ventana en la cadena de visualizadores del portapapeles debe pasar este mensaje a la próxima ventana en la cadena, mediante la función *SendMessage*, de manera que todas las ventanas visualizadoras sean notificadas.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DRAWCLIPBOARD*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

Véase además

*SendMessage*, *SetClipboardViewer*, *WM\_CHANGECHAIN*

## WM\_DRAWITEM

### Sintaxis

```
TWMDrawItem = record
    Msg: Cardinal;           {identificador del mensaje}
    Ctl: HWND;               {manejador del control que envía el mensaje}
    DrawItemStruct: PDrawItemStruct; {puntero al registro DrawItemStruct}
    Result: Longint;         {devuelve uno (1) si fue manejado}
end;
```

### Descripción

La ventana propietaria de un cuadro de lista, de combinación, un elemento de menú o botón dibujado por el propietario recibe un mensaje *WM\_DRAWITEM* cuando algún aspecto visual del control cambia, como por ejemplo el foco, estado de habilitación, etc. Para crear elementos de menú dibujados por el propietario, utilice las funciones *InsertMenuItem* o *SetMenuItemInfo*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DRAWITEM*.

*Ctl*: Manejador de la ventana que envía el mensaje *WM\_DRAWITEM*.

*DrawItemStruct*: Puntero a un registro *TDrawItemStruct* que contiene la información sobre el control que necesita la ventana propietaria para dibujar. El registro *TDrawItemStruct* se define como:

*TDrawItemStruct* = **packed record**

<i>CtlType</i> : UINT;	{indicador de tipo de control}
<i>CtlID</i> : UINT;	{identificador del control}
<i>itemID</i> : UINT;	{identificador de elemento de menú o índice}
<i>itemAction</i> : UINT;	{opción de dibujo}
<i>itemState</i> : UINT;	{estado visual del elemento}
<i>hwndItem</i> : HWND;	{manejador del control}
<i>hDC</i> : HDC;	{contexto de dispositivo}
<i>rcItem</i> : TRect;	{rectángulo del elemento}
<i>itemData</i> : DWORD;	{los datos de 32 bits del elemento}

**end;**

*CtlType*: Un valor que especifica el tipo de control. Este campo puede contener un valor de la Tabla A-4.

*CtlID*: Especifica el identificador del control. Este campo es ignorado si el control es un elemento de menú.

*itemID*: Este campo contiene el identificador del elemento de menú o el índice del elemento seleccionado en el cuadro de lista o de combinación. Un valor -1 indica un cuadro de lista o de combinación vacío. Esto permite a la ventana propietaria del control redibujar sólo la porción del control especificada en el campo *rcItem* y es útil para dibujar el rectángulo de foco.

*itemAction*: Valor que indica qué acciones deben tomarse cuando se dibuja el elemento. Este campo puede ser un valor de la Tabla A-5.

*itemState*: Indica el nuevo estado visual del control. Este campo puede contener uno o más valores de la Tabla A-6.

*hwndItem*: El manejador de ventana del control, para cuadros de combinación, cuadros de lista, botones y controles estáticos. Para elementos de menú, este campo contiene el manejador de ventana del menú que contiene el elemento.

*hDC*: El contexto de dispositivo en el cual se realizará el dibujo.

*rcItem*: El rectángulo dentro del contexto de dispositivo especificado donde se realizará el dibujo. Para todos los controles, excepto los elementos de menú, las funciones de dibujo son restringidas a este área rectangular.

*itemData*: Especifica el valor de 32 bit que fue suministrado en el parámetro *lParam* de los mensajes *CB\_ADDSTRING*, *CB\_INSERTSTRING*, *LB\_ADDSTRING*, o *LB\_INSERTSTRING*, o el último valor asignado al control por los mensajes *LB\_SETITEMDATA* o *CB\_SETITEMDATA*. Para un cuadro de lista o cuadro de combinación con estilo *LBS\_HASSTRINGS* o *CBS\_HASSTRINGS*, este campo vale inicialmente cero. Si el campo *ctlType* contiene las opciones *ODT\_BUTTON* u *ODT\_STATIC*, este campo vale cero.

*Result*: Si la aplicación maneja este mensaje, deberá asignar uno (1) a este campo.

Véase además

*WM\_MEASUREITEM*

**Tabla A-4: Valores de *TWMDrawItem.TDrawItemStruct.CtlType***

Valor	Descripción
ODT_BUTTON	Indica un botón dibujado por el propietario.
ODT_COMBOBOX	Indica un cuadro de combinación dibujado por el propietario.
ODT_LISTBOX	Indica un cuadro de lista dibujado por el propietario.
ODT_LISTVIEW	Indica una vista de lista dibujada por el propietario.
ODT_MENU	Indica un elemento de menú dibujado por el propietario.
ODT_STATIC	Indica un control estático dibujado por el propietario.
ODT_TAB	Indica un control de tabulación dibujado por el propietario.

**Tabla A-5: Valores de *TWMDrawItem.TDrawItemStruct.itemAction***

Valor	Descripción
ODA_DRAWENTIRE	El control completo debe ser dibujado.
ODA_FOCUS	El foco del teclado ha cambiado. Verifique el valor del campo <i>itemState</i> para conocer el estado del foco.
ODA_SELECT	El estado de la selección ha cambiado. Verifique el valor del campo <i>itemState</i> para conocer el estado de selección.

**Tabla A-6: Valores de *TWMDrawItem.TDrawItemStruct.ItemState***

Valor	Descripción
ODS_CHECKED	El elemento de menú está marcado.
ODS_COMBOBOXEDIT	La acción de dibujo se efectuará en el cuadro de edición de un cuadro de combinación.
ODS_DEFAULT	Indica el elemento por defecto.
ODS_DISABLED	El elemento debe ser dibujado en estado deshabilitado.

ODS_FOCUS	El elemento tiene el foco del teclado.
ODS_GRAYED	El elemento de menú está sombreado.
ODS_SELECTED	El elemento de menú está seleccionado.

---

## WM\_DROPFILES

### Sintaxis

```

TWMDropFiles = record
    Msg: Cardinal;           {identificador del mensaje}
    Drop: THANDLE;          {manejador del registro de ficheros soltados}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

### Descripción

Si una ventana está registrada para recibir archivos arrastrados y soltados desde el Explorador o el Administrador de Ficheros, recibirá el mensaje *WM\_DROPFILES* cuando el usuario suelte el botón del ratón sobre la ventana para completar una operación de arrastrar y soltar. Utilice la función *DragAcceptFiles* para registrar una ventana como receptora de ficheros mediante ‘arrastrar y soltar’.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_DROPFILES*.

*Drop*: El manejador del registro interno que contiene los ficheros que han sido arrastrados y soltados.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*DragAcceptFiles*, *DragFinish*, *DragQueryFile*, *DragQueryPoint*

## WM\_ENABLE

### Sintaxis

```

TWMEEnable = record
    Msg: Cardinal;           {identificador del mensaje}
    Enabled: LongBool;       {opción de habilitación}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}

```

**end;**

### Descripción

Cuando una aplicación cambia el estado de habilitación de una ventana, un mensaje *WM\_ENABLE* es enviado a la ventana cuyo estado está cambiando. El mensaje es enviado después que el estado haya cambiado, pero antes que la función *EnableWindow* retorne.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ENABLE*.

*Enabled*: Este campo contiene TRUE si la ventana pasa a estar habilitada, o FALSE si la ventana pasa a estar deshabilitada.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*EnableWindow*

## WM\_ENDSESSION

### Sintaxis

```
TWMEndSession = record
    Msg: Cardinal;           {identificador del mensaje}
    EndSession: LongBool;    {indicador de fin de sesión}
    Unused: Longint;         {indicador de cerrar sesión}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Después que el mensaje *WM\_QUERYENDSESSION* es enviado a todas las ventanas y los resultados son procesados, Windows envía un mensaje *WM\_ENDSESSION* a todas las aplicaciones indicando si la sesión Windows está finalizando.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ENDSESSION*.

*EndSession*: Indica si la sesión Windows está terminando. Si este campo contiene TRUE, la sesión está terminando; en caso contrario, contiene FALSE.

*Unused*: Indica si el usuario está simplemente cerrando la sesión actual o está apagando el sistema. Si está cerrando la sesión, este campo contiene *ENDSESSION\_LOGOFF*. Si está apagando el sistema, este campo contiene cero.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DestroyWindow*, *PostQuitMessage*, *WM\_QUERYENDSESSION*

## WM\_ENTERIDLE

### Sintaxis

```
TWMEnterIdle = record
    Msg: Cardinal;           {identificador del mensaje}
    Source: Longint;         {tipo de control}
    IdleWnd: HWND;          {manejador del control}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Si un cuadro de diálogo modal o un menú están siendo mostrados y todos los mensajes en la cola han sido procesados, el menú o cuadro de diálogo modal entran en estado ocioso y el mensaje *WM\_ENTERIDLE* es enviado a la ventana propietaria del menú o al cuadro de diálogo.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ENTERIDLE*.

*Source*: Indica si un cuadro de diálogo o un menú están siendo mostrados. Este campo puede ser un valor de la Tabla A-7.

*IdleWnd*: El manejador del cuadro de diálogo o el manejador de la ventana que contiene el menú.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_ACTIVATE*, *WM\_MOUSEACTIVATE*

**Tabla A-7: Valores de TWMEnterIdle.Source**

Valor	Descripción
MSGF_DIALOGBOX	Indica que un cuadro de diálogo es mostrado.
MSGF_MENU	Indica que un menú es mostrado.

**WM\_ENTERMENULOOP***Sintaxis*

```

TWMEnterMenuLoop = record
    Msg: Cardinal;           {identificador del mensaje}
    IsTrackPopupMenu: LongBool; {opción de atajo de menú}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

El mensaje *WM\_ENTERMENULOOP* es enviado a la ventana principal de una aplicación cuando un menú va a ser mostrado y se ha entrado en el bucle modal del menú.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ENTERMENULOOP*.

*IsTrackPopupMenu*: Indica si el menú es un menú emergente o normal. Si a este campo se le asigna TRUE, el menú es emergente; en caso contrario, es un menú corriente.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

*Véase además*

*DefWindowProc*, *WM\_EXITMENULOOP*

**WM\_ERASEBKGND***Sintaxis*

```

TWMEraseBkgnd = record
    Msg: Cardinal;           {identificador del mensaje}
    DC: HDC;                 {contexto de dispositivo}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve uno (1) si fue manejado}
end;

```

*Descripción*

Este mensaje es enviado a una ventana cuyo fondo ha sido invalidado a causa de acciones tales como el redimensionamiento, y que necesita ser borrada. Típicamente el fondo de la ventana es borrado cuando se redibuja la región no válida usando la brocha de fondo de la clase de la ventana.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ERASEBKGND*.

*DC*: El contexto de dispositivo de la ventana cuyo fondo será borrado.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja este mensaje, deberá asignar uno (1) a este campo.

**Véase además**

*BeginPaint*, *DefWindowProc*, *WM\_ICONERASEBKGND*

**WM\_EXITMENULOOP****Sintaxis**

```
TWMExitMenuLoop = record
    Msg: Cardinal;           {identificador del mensaje}
    IsTrackPopupMenu: LongBool; {opción de atajo del menú}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

El mensaje *WM\_EXITMENULOOP* es enviado a la ventana principal de una aplicación cuando un menú es cerrado y se ha salido de su bucle modal de menú.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_EXITMENULOOP*.

*IsTrackPopupMenu*: Indica si el menú es un menú emergente o normal. Si a este campo se le asigna TRUE, el menú es emergente; en caso contrario, es un menú corriente.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*DefWindowProc*, *WM\_ENTERMENULOOP*

**WM\_FONTCHANGE****Sintaxis**

```
TWMFontChange = record
```

```

    Msg: Cardinal;                {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;              {no se utiliza}
end;

```

**Descripción**

Si una aplicación cambia los recursos de fuente globales al sistema mediante las funciones *AddFontResource* o *RemoveFontResource*, deberá enviar un mensaje *WM\_FONTCHANGE* a todas las ventanas de nivel superior. Este mensaje puede ser enviado a esas ventanas usando la función *SendMessage* con el parámetro *hWnd* puesto a *HWND\_BROADCAST*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_FONTCHANGE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

**Véase además**

*AddFontResource*, *RemoveFontResource*, *SendMessage*

**WM\_GETFONT****Sintaxis**

```

TWMGetFont = record
    Msg: Cardinal;                {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;              {devuelve un manejador de fuente}
end;

```

**Descripción**

Una aplicación puede recuperar la fuente que un control está usando para dibujar texto, enviándole un mensaje *WM\_GETFONT* a ese control.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_GETFONT*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este mensaje devuelve cero si el control utiliza la fuente de sistema. En caso contrario, el mensaje devuelve el manejador de la fuente actualmente seleccionada.

Véase además

*WM\_SETFONT*

## **WM\_GETHOTKEY**

### *Sintaxis*

```
TWMGetHotKey = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve información de la 'tecla caliente'}
end;
```

### *Descripción*

El mensaje *WM\_GETHOTKEY* es enviado a una ventana para recuperar su 'tecla caliente' asociada.

### *Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_GETHOTKEY*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este mensaje devuelve cero si no hay ninguna 'tecla caliente' asociada a la ventana. Si la ventana tiene una 'tecla caliente' asociada, el código virtual de la misma estará en el byte menos significativo del valor resultante y el modificador estará en el byte más significativo. El modificador de la 'tecla caliente' puede ser una combinación de valores de la Tabla A-8.

Véase además

*WM\_SETHOTKEY*, *WM\_SYSCOMMAND*

**Tabla A-8: Valores del modificador de la tecla caliente TWMGetHotKey.Result**

Valor	Descripción
HOTKEYF_ALT	La tecla Alt.
HOTKEYF_CONTROL	La tecla Ctrl.
HOTKEYF_EXT	Una tecla extendida.
HOTKEYF_SHIFT	La tecla May.

## **WM\_GETICON**

### *Sintaxis*

```
TWMGetIcon = record
```

Msg: Cardinal;	{identificador del mensaje}
BigIcon: Longbool;	{tipo de icono}
Unused: Longint;	{no se utiliza}
Result: Longint;	{devuelve un manejador de icono}

**end;**

**Descripción**

El mensaje *WM\_GETICON* es enviado a una ventana para recuperar su icono asociado, grande o pequeño.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_GETICON*.

*BigIcon*: Valor que indica el tipo de icono a recuperar. Este campo puede contener *ICON\_BIG* para recuperar el icono grande o *ICON\_SMALL* para recuperar el pequeño.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si el mensaje tiene éxito, devuelve un manejador del icono grande o pequeño.

**Véase además**

*DefWindowProc*, *WM\_SETICON*

**WM\_GETMINMAXINFO****Sintaxis**

TWMGetMinMaxInfo = **record**

Msg: Cardinal;	{identificador del mensaje}
Unused: Integer;	{no se utiliza}
MinMaxInfo: PMinMaxInfo;	{puntero a un registro}
Result: Longint;	{devuelve cero (0) si fue manejado}

**end;**

**Descripción**

Cuando el tamaño o la posición de una ventana va a cambiar, el mensaje *WM\_GETMINMAXINFO* es enviado a la ventana. Esto puede ser usado para redefinir los valores predeterminados para el tamaño maximizado, la posición maximizada, el tamaño mínimo de seguimiento o el tamaño máximo de seguimiento de la ventana.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_GETMINMAXINFO*.

*Unused*: Este campo no es utilizado por este mensaje.

*MinMaxInfo*: Puntero a un registro *TMinMaxInfo* que describe el tamaño y la posición máximas y mínimas de la ventana. El registro *TMinMaxInfo* se define como:

*TMinMaxInfo* = **packed record**

<i>ptReserved</i> : TPoint;	{reservado}
<i>ptMaxSize</i> : TPoint;	{tamaño máximo de la ventana}
<i>ptMaxPosition</i> : TPoint;	{ubicación de la ventana maximizada}
<i>ptMinTrackSize</i> : TPoint;	{tamaño mínimo de seguimiento}
<i>ptMaxTrackSize</i> : TPoint;	{tamaño máximo de seguimiento}

**end;**

*ptReserved*: Este campo está reservado para uso futuro y hay que asignarle cero.

*ptMaxSize*: El nuevo ancho y altura de la ventana maximizada, en píxeles.

*ptMaxPosition*: La nueva posición de la ventana maximizada, en píxeles, relativa a la esquina superior izquierda de la pantalla.

*ptMinTrackSize*: El nuevo tamaño mínimo de seguimiento de la ventana, en píxeles.

*ptMaxTrackSize*: El nuevo tamaño máximo de seguimiento de la ventana, en píxeles.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*MoveWindow*, *SetWindowPos*

## WM\_GETTEXT

*Sintaxis*

*TWMGetText* = **record**

<i>Msg</i> : Cardinal;	{identificador del mensaje}
<i>TextMax</i> : Integer;	{cantidad de caracteres a copiar}
<i>Text</i> : PChar;	{puntero al <i>buffer</i> de salida}
<i>Result</i> : Longint;	{devuelve la cantidad de caracteres copiados}

**end;**

*Descripción*

El mensaje *WM\_GETTEXT* es enviado para recuperar el texto o el título de una ventana. El texto o título es copiado en un *buffer* reservado por el usuario al que apunta el campo *Text*. Para un control de edición, el texto recuperado es el contenido completo del control de edición. Para un cuadro de combinación, es el contenido de su control de edición. Para un botón, es el título mostrado por el botón. Para las demás ventanas, es el título de la ventana. Para recuperar el texto de un elemento de un cuadro de lista, se debe usar el mensaje *LB\_GETTEXT*. Para un control de edición con más de 64K de texto, se deben usar los mensajes *EM\_STREAMOUT* o *EM\_GETSELTEXT*. Cuando un

control estático tiene el estilo *SS\_ICON* y el icono fue asignado por un mensaje *WM\_SETTEXT*, el mensaje *WM\_GETTEXT* devolverá el manejador del icono en los primeros cuatro bytes del *buffer* de salida.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_GETTEXT*.

*TextMax*: Especifica la cantidad máxima de caracteres a copiar, incluyendo el terminador nulo.

*Text*: Un puntero al *buffer* de salida que recibe el texto.

*Result*: Si el mensaje tiene éxito, devuelve el número de caracteres copiados en el *buffer* de salida.

### Véase además

*DefWindowProc*, *GetWindowText*, *GetWindowTextLength*, *WM\_GETTEXTLENGTH*, *WM\_SETTEXT*

## WM\_GETTEXTLENGTH

### Sintaxis

```
TWMGetTextLength = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve la longitud del texto en caracteres}
end;
```

### Descripción

El mensaje *WM\_GETTEXTLENGTH* es enviado a una ventana para determinar la longitud del texto asociado con esa ventana, excluyendo el terminador nulo. Este mensaje puede ser usado para determinar el tamaño del *buffer* de salida requerido por el mensaje *WM\_GETTEXT*. Cuando el sistema contiene una mezcla de cadenas ANSI y Unicode, el resultado puede ser mayor que el número de bytes requerido, pero nunca será menor que el número de bytes requerido por el *buffer* de salida usado por el mensaje *WM\_GETTEXT*. Vea el mensaje *WM\_GETTEXT* para ver una descripción del texto asociado con los diferentes tipos de controles.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_GETTEXTLENGTH*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result:* Si el mensaje tiene éxito, devuelve la longitud del texto asociado con la ventana a la cual el mensaje fue enviado, en caracteres.

Véase además

*DefWindowProc, GetWindowText, GetWindowTextLength, WM\_GETTEXT*

## WM\_HELP

### Sintaxis

```
TWMHelp = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: Integer;         {no se utiliza}
    HelpInfo: PHelpInfo;     {puntero al registro THelpInfo}
    Result: Longint;         {siempre devuelve uno (1)}
end;
```

### Descripción

El mensaje *WM\_HELP* es enviado cuando el usuario presiona la tecla **F1** en busca de ayuda. Si un elemento de menú está abierto, el mensaje es enviado a la ventana asociada con el menú; en caso contrario, el mensaje es enviado a la ventana que tiene el foco del teclado o a la ventana actualmente activa, si ninguna tiene el foco del teclado.

### Campos

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_HELP*.

*Unused:* Este campo no es utilizado por este mensaje.

*HelpInfo:* Un puntero al registro *THelpInfo* que contiene información sobre la ventana o el control para el cual la ayuda fue solicitada. El registro *THelpInfo* se define como:

```
THelpInfo = packed record
    cbSize: UINT;           {tamaño del registro THelpInfo}
    iContextType: Integer;   {tipo de ayuda}
    iCtrlId: Integer;        {identificador del control o del elemento de menú}
    hItemHandle: THandle;    {manejador del control o menú}
    dwContextId: DWORD;      {identificador del contexto de la ayuda}
    MousePos: TPoint;        {las coordenadas de pantalla del ratón}
end;
```

*cbSize:* Especifica el tamaño del registro *THelpInfo*, en bytes.

*iContextType:* Identifica el tipo de control para el cual la ayuda fue solicitada y puede contener un valor de la Tabla A-9.

*iCtrlId:* Especifica el identificador del elemento de menú, control o ventana.

*hItemHandle*: Identifica el manejador de la ventana o el manejador del menú de la ventana, del control o del elemento de menú.

*dwContextId*: Especifica el identificador del contexto de la ayuda para la ventana, control, o menú. El sistema de ayuda de Windows recupera el tópico de ayuda asociado con el identificador del contexto de la ayuda, desde el fichero de ayuda asociado.

*MousePos*: Especifica la posición del ratón en el momento en que la ayuda se solicita, en coordenadas de la pantalla.

*Result*: Este mensaje siempre devuelve uno.

Véase además

DefWindowProc, *WM\_SYSCOMMAND*

**Tabla A-9: Valores de TWMHelp.THelpInfo.iContextType**

Valor	Descripción
HELPIFNO_MENUITEM	Indica una solicitud de ayuda para un elemento de menú.
HELPIFNO_WINDOW	Indica una solicitud de ayuda para una ventana o control.

## WM\_HSCROLL

### Sintaxis

TWMHScroll = **record**

Msg: Cardinal;	{identificador del mensaje}
ScrollCode: Smallint;	{código de solicitud de desplazamiento}
Pos: Smallint;	{posición del botón de la barra de desplazamiento}
Scroll bar: HWND;	{manejador de la barra de desplazamiento}
Result: Longint;	{devuelve cero (0) si fue manejado}

**end;**

### Descripción

El mensaje *WM\_HSCROLL* es enviado a una ventana cuando un evento de desplazamiento horizontal ocurre, si la ventana tiene una barra de desplazamiento horizontal estándar. También es enviado cuando un evento de desplazamiento ocurre en un control de barra de desplazamiento horizontal. Si la aplicación cambia la posición de los datos en una ventana como resultado de un evento de desplazamiento horizontal, debe reinicializar la posición del botón de la barra de desplazamiento, llamando a la función *SetScrollPos*. Los mensajes *WM\_VSCROLL* y *WM\_HSCROLL* tienen un valor de 16 bits para la posición de los desplazamientos, restringiendo así la posición máxima a 65.535.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_HSCROLL*.

*ScrollCode*: Especifica el tipo de desplazamiento solicitado como resultado de un evento de desplazamiento. Este campo puede contener un valor de la Tabla A-10.

*Pos*: Si el campo *ScrollCode* contiene *SB\_THUMBPOSITION* o *SB\_THUMBTRACK*, este campo especifica la posición actual del botón de la barra de desplazamiento. En caso contrario, este campo no es usado.

*Scroll bar*: Especifica el manejador de la barra de desplazamiento, si es un control de barra de desplazamiento el que está enviando el mensaje *WM\_HSCROLL*. En caso contrario, este campo no es usado.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*WM\_VSCROLL*

**Tabla A-10: Valores de *TWMHScroll.ScrollCode***

Valor	Descripción
<i>SB_BOTTOM</i>	Indica un desplazamiento al extremo derecho.
<i>SB_ENDSCROLL</i>	Indica el fin de la operación de desplazamiento.
<i>SB_LINELEFT</i>	Indica un desplazamiento a la izquierda de una unidad.
<i>SB_LINERIGHT</i>	Indica un desplazamiento a la derecha de una unidad.
<i>SB_PAGELEFT</i>	Indica un desplazamiento a la izquierda del ancho de la ventana.
<i>SB_PAGERIGHT</i>	Indica un desplazamiento a la derecha del ancho de la ventana.
<i>SB_THUMBPOSITION</i>	Indica un desplazamiento a la posición absoluta que especifica el campo <i>Pos</i> .
<i>SB_THUMBTRACK</i>	Arrastra el botón de la barra de desplazamiento a la posición que indica el campo <i>Pos</i> . Esto se usa para obtener retroalimentación del control.
<i>SB_TOP</i>	Indica un desplazamiento al extremo izquierda.

***WM\_HSCROLLCLIPBOARD*****Sintaxis**

*TWMHScrollClipboard* = **record**

<i>Msg</i> : Cardinal;	{identificador del mensaje}
<i>Viewer</i> : HWND;	{manejador del visualizador del portapapeles}
<i>ScrollCode</i> : Word;	{código del desplazamiento solicitado}

ThumbPos: Word;                    {posición del botón de la barra de desplazamiento}  
 Result: Longint;                {devuelve cero (0) si fue manejado}  
**end;**

### Descripción

El mensaje *WM\_HSCROLLCLIPBOARD* es enviado por una ventana visualizadora del portapapeles a la propietaria del portapapeles cuando se produce un evento en la barra de desplazamiento horizontal del visor del portapapeles y el portapapeles contiene datos en el formato *CF\_OWNERDISPLAY*. La propietaria del portapapeles deberá desplazar la imagen y reinicializar el valor de la barra de desplazamiento horizontal.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_HSCROLLCLIPBOARD*.

*Viewer*: Especifica el manejador de la ventana visualizadora del portapapeles.

*ScrollCode*: Especifica el tipo de petición de desplazamiento como resultado de un evento de desplazamiento. Este campo puede contener un valor de la Tabla A-11.

*ThumbPos*: Si el campo *ScrollCode* contiene *SB\_THUMBPOSITION*, este campo especifica la posición actual del botón de la barra de desplazamiento. En caso contrario, este campo no es usado.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*WM\_VSCROLLCLIPBOARD*

**Tabla A-II: Valores de *TWMHScrollClipboard.ScrollCode***

Valor	Descripción
SB_BOTTOM	Indica un desplazamiento al extremo derecho.
SB_ENDSCROLL	Indica el final de la operación de desplazamiento.
SB_LINELEFT	Indica un desplazamiento a la izquierda de una unidad.
SB_LINERIGHT	Indica un desplazamiento a la derecha de una unidad.
SB_PAGELEFT	Indica un desplazamiento a la izquierda del ancho de la ventana.
SB_PAGERIGHT	Indica un desplazamiento a la derecha del ancho de la ventana.
SB_THUMBPOSITION	Indica un desplazamiento a la posición absoluta que especifica el campo Pos.
SB_TOP	Indica un desplazamiento al extremo izquierdo.

**WM\_ICONERASEBKGND***Sintaxis*

```
TWMIconEraseBkgnd = record
    Msg: Cardinal;           {identificador del mensaje}
    DC: HDC;                 {contexto de dispositivo}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve uno (1) si fue manejado}
end;
```

*Descripción*

Una ventana que ha sido minimizada recibe un mensaje *WM\_ICONERASEBKGND* cuando el fondo del icono tiene que ser rellenado antes de que sea dibujado. Este mensaje es enviado sólo si un icono de clase ha sido definido para la ventana; en caso contrario, es enviado el mensaje *WM\_ERASEBKGND*.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_ICONERASEBKGND*.

*DC*: El contexto de dispositivo sobre el cual el icono será dibujado.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja este mensaje, deberá asignar uno (1) a este campo.

*Véase además*

*DefWindowProc*, *WM\_ERASEBKGND*

**WM\_INITMENU***Sintaxis*

```
TWMInitMenu = record
    Msg: Cardinal;           {identificador del mensaje}
    Menu: HMENU;             {manejador del menú}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

*Descripción*

Cuando un menú va a activarse, por ejemplo cuando el usuario hace clic sobre un elemento del menú o pulsa una tecla de atajo al menú, un mensaje *WM\_INITMENU* es enviado a la ventana propietaria del menú. Esto permite que la aplicación modifique el menú antes de mostrarlo. Este mensaje es enviado sólo cuando un menú es accedido por primera vez, y solamente un mensaje es enviado por cada acceso. Mover el ratón

por los elementos del menú mientras se mantiene pulsado el botón no generará nuevos mensajes.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_INITMENU*.

*Menu*: Identifica el manejador del menú.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*WM\_ENTERMENULOOP*, *WM\_EXITMENULOOP*, *WM\_INITMENUPOPUP*

## WM\_INITMENUPOPUP

### Sintaxis

```
TWMInitMenuPopup = record
    Msg: Cardinal;           {identificador del mensaje}
    MenuPopup: HMENU;       {manejador del menú}
    Pos: Smallint;          {posición del elemento del submenú}
    SystemMenu: WordBool;   {opción de menú de sistema}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Cuando un menú desplegable o submenú se va a activar, por ejemplo cuando el usuario hace clic sobre un elemento de menú que contiene un submenú, un mensaje *WM\_INITMENUPOPUP* es enviado a la ventana propietaria del menú. Esto permite a la aplicación modificar el menú antes de mostrarlo.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_INITMENUPOPUP*.

*MenuPopup*: Identifica el manejador del menú desplegable o submenú.

*Pos*: Especifica el índice de base cero del elemento de menú que abrió el menú desplegable o submenú.

*SystemMenu*: Indica si el menú desplegable activado es el menú de sistema o de ventana. Si a este campo se le asigna TRUE, el menú desplegable activado es el de sistema; en caso contrario, es un menú desplegable o submenú.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_ENTERMENULOOP, WM\_EXITMENULOOP, WM\_INITMENU*

## WM\_KEYDOWN

### Sintaxis

```
TWMKeyDown = record
    Msg: Cardinal;           {identificador del mensaje}
    CharCode: Word;          {código de tecla virtual}
    Unused: Word;             {no se utiliza}
    KeyData: Longint;         {contiene información diversa}
    Result: Longint;          {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Cuando el usuario pulsa una tecla que no es de sistema, un mensaje *WM\_KEYDOWN* es enviado a la ventana que tiene el foco del teclado. Una tecla ‘no de sistema’ es una combinación de teclas que no incluye la tecla **Alt**. Si la tecla pulsada es **F10**, *DefWindowProc* asigna 1 a una opción interna. Cuando el mensaje *WM\_KEYUP* es recibido, *DefWindowProc* verifica el valor de esa opción interna y si ésta tiene valor 1, envía un mensaje *WM\_SYSCOMMAND* a la ventana de nivel superior con el campo *CmdType* del mensaje puesto a *SC\_KEYMENU*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_KEYDOWN*.

*CharCode*: Especifica el código de tecla virtual de la tecla presionada.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado anterior de la tecla e indicador del estado de transición. La Tabla A-12 muestra qué información se almacena en cada bit del valor de 32 bits del campo *KeyData*.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc, WM\_CHAR, WM\_KEYUP, WM\_SYSCOMMAND*

Tabla A-12: valores de TWMKeyDown.KeyData

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.
29	El código de contexto. Este bit siempre es 0.
30	El estado previo de la tecla. Si la tecla estaba presionada antes de que se enviara el mensaje, este bit contiene 1; en caso contrario contiene 0.
31	El estado de transición. Este bit siempre contiene 0.

**WM\_KEYUP***Sintaxis*

```

TWMKeyUp = record
    Msg: Cardinal;           {identificador del mensaje}
    CharCode: Word;          {código virtual de tecla}
    Unused: Word;            {no se utiliza}
    KeyData: Longint;        {contiene información diversa}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

Cuando el usuario libera una tecla que no es de sistema, un mensaje *WM\_KEYUP* es enviado a la ventana que posee el foco del teclado. Una tecla ‘no de sistema’ es una combinación de teclas que no incluye la tecla **Alt**. Si la tecla liberada es **F10**, *DefWindowProc* envía un mensaje *WM\_SYSCOMMAND* a la ventana de nivel superior con el campo *CmdType* del mensaje puesto a *SC\_KEYMENU*.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_KEYUP*.

*CharCode*: Especifica el código virtual de la tecla para la tecla que fue liberada.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado anterior de la tecla e indicador de estado de transición. La Tabla A-13 muestra qué información se almacena en cada bit del valor de 32 bits del campo *KeyData*.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_CHAR*, *WM\_KEYDOWN*, *WM\_SYSCOMMAND*

**Tabla A-13: Valores de *TWMKeyUp.KeyData***

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.
29	El código de contexto. Este bit siempre es 0.
30	El estado previo de la tecla. Este bit siempre es 1.
31	El estado de transición. Este bit siempre es 1.

## ***WM\_KILLFOCUS***

### *Sintaxis*

```
TWMKillFocus = record
    Msg: Cardinal;           {identificador del mensaje}
    FocusedWnd: HWND;        {manejador de la ventana que recibe el foco}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### *Descripción*

El mensaje *WM\_KILLFOCUS* es enviado a una ventana cuando va a perder el foco del teclado. La ventana recibirá el mensaje antes de que el foco del teclado cambie.

### *Campos*

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_KILLFOCUS*.

*FocusedWnd:* Especifica el manejador de la ventana que recibe el foco del teclado.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*SetFocus*, *WM\_SETFOCUS*

**WM\_LBUTTONDOWNBLCLK****Sintaxis**

```

TWMLButtonDownBlk = record
    Msg: Cardinal;           {identificador del mensaje}
    Keys: Longint;          {opciones de teclas virtuales}
    case Integer of
    0: (
        XPos: Smallint;      {coordenada horizontal del cursor}
        YPos: Smallint);     {coordenada vertical del cursor}
    1: (
        Pos: TSmallPoint;     {registro que contiene las coordenadas del cursor}
        Result: Longint);     {devuelve cero (0) si fue manejado}
    end;

```

**Descripción**

Si el cursor del ratón está situado dentro del área cliente de una ventana y se hace doble clic con el botón izquierdo, un mensaje *WM\_LBUTTONDOWNBLCLK* es enviado a la ventana que está bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón, mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana. Un doble clic se genera cuando el usuario pulsa y libera el botón del ratón y lo pulsa y libera de nuevo, en las mismas coordenadas y dentro del intervalo de doble clic del sistema. Este proceso genera una serie de cuatro mensajes, en el orden siguiente: *WM\_LBUTTONDOWN*, *WM\_LBUTTONUP*, *WM\_LBUTTONDOWNBLCLK* y *WM\_LBUTTONUP*. Sólo las ventanas cuyo estilo de clase incluye el atributo *CS\_DBLCLKS* recibirán el mensaje *WM\_LBUTTONDOWNBLCLK*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_LBUTTONDOWNBLCLK*.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento del doble clic. Este campo puede contener uno o más valores de la Tabla A-14.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón, relativas al área cliente.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*GetCapture*, *GetDoubleClickTime*, *SetCapture*, *SetDoubleClickTime*,  
*WM\_LBUTTONDOWN*, *WM\_LBUTTONUP*

Tabla A-14: Valores de `TWMLButtonDownClick.Keys`

Valor	Descripción
<code>MK_CONTROL</code>	Indica que la tecla Ctrl está presionada.
<code>MK_LBUTTON</code>	Indica que el botón izquierdo del ratón está pulsado.
<code>MK_MBUTTON</code>	Indica que el botón central del ratón está pulsado.
<code>MK_RBUTTON</code>	Indica que el botón derecho del ratón está pulsado.
<code>MK_SHIFT</code>	Indica que la tecla May. está presionada.

**`WM_LBUTTONDOWN`***Sintaxis*

```

TWMLButtonDown = record
    Msg: Cardinal;           {identificador del mensaje}
    Keys: Longint;          {opciones de teclas virtuales}
    case Integer of
    0: (
        XPos: Smallint;     {coordenada horizontal del cursor}
        YPos: Smallint;     {coordenada vertical del cursor}
    1: (
        Pos: TSmallPoint;    {registro que contiene las coordenadas del cursor}
        Result: Longint;     {devuelve cero (0) si fue manejado}
    end;

```

*Descripción*

Si el cursor del ratón está situado dentro del área cliente de una ventana y se pulsa el botón izquierdo, se envía un mensaje `WM_LBUTTONDOWN` a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje `WM_LBUTTONDOWN`.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento del clic. Este campo puede contener uno o más valores de la Tabla A-15.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón, relativas al área cliente.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*GetCapture, SetCapture, WM\_LBUTTONDOWN, WM\_LBUTTONUP*

**Tabla A-15: Valores de TWMLButtonDown.Keys**

Valor	Descripción
MK_CONTROL	Indica que la tecla Ctrl está presionada .
MK_LBUTTON	Indica que el botón izquierdo del ratón está pulsado.
MK_MBUTTON	Indica que el botón central del ratón está pulsado.
MK_RBUTTON	Indica que el botón derecho del ratón está pulsado.
MK_SHIFT	Indica que la tecla May. está presionada.

## WM\_LBUTTONUP

### Sintaxis

```

TWMLButtonUp = record
    Msg: Cardinal;           {identificador del mensaje}
    Keys: Longint;          {opciones de teclas virtuales}
    case Integer of
        0: (
            XPos: Smallint;   {coordenada horizontal del cursor}
            YPos: Smallint);  {coordenada vertical del cursor}
        1: (
            Pos: TSmallPoint; {registro que contiene las coordenadas del cursor}
            Result: Longint); {devuelve cero (0) si fue manejado}
    end;

```

### Descripción

Si el cursor del ratón está situado dentro del área cliente de la ventana y el botón izquierdo es liberado, un mensaje *WM\_LBUTTONUP* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_LBUTTONUP*.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento que el botón fue liberado. Este campo puede contener uno o más valores de la Tabla A-16.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón relativas al área cliente.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*GetCapture*, *SetCapture*, *WM\_LBUTTONDOWNBLCLK*, *WM\_LBUTTONDOWN*

**Tabla A-16: Valores de *TWMLButtonUp.Keys***

Valor	Descripción
MK_CONTROL	Indica que la tecla Ctrl está presionada.
MK_MBUTTON	Indica que el botón central del ratón está pulsado.
MK_RBUTTON	Indica que el botón derecho del ratón está pulsado.
MK_SHIFT	Indica que la tecla May. está presionada.

## ***WM\_MBUTTONDOWNBLCLK***

### *Sintaxis*

```

TWMLButtonDblClk = record
    Msg: Cardinal;           {identificador del mensaje}
    Keys: Longint;          {opciones de teclas virtuales}
    case Integer of
        0: (
            XPos: Smallint;   {coordenada horizontal del cursor}
            YPos: Smallint);  {coordenada vertical del cursor}
        1: (
            Pos: TSmallPoint; {registro que contiene las coordenadas del cursor}
            Result: Longint); {devuelve cero (0) si fue manejado}
    end;

```

### *Descripción*

Si el cursor del ratón está situado dentro del área cliente de una ventana y se hace doble clic con el botón central, un mensaje *WM\_MBUTTONDOWNBLCLK* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón, mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana. Un doble clic se genera cuando el usuario pulsa y libera el botón del ratón y lo pulsa y libera de nuevo, en las mismas coordenadas y dentro del intervalo de doble clic del sistema. Este proceso genera una serie de cuatro mensajes en el orden siguiente: *WM\_MBUTTONDOWN*, *WM\_MBUTTONUP*, *WM\_MBUTTONDOWNBLCLK* y *WM\_MBUTTONUP*. Sólo aquellas ventanas cuyo estilo de clase incluye el atributo *CS\_DBLCLKS* recibirán el mensaje *WM\_MBUTTONDOWNBLCLK*.

Campos

- Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MBUTTONDOWNCLK*.
- Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento del doble clic. Este campo puede contener uno o más valores de la Tabla A-17.
- XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.
- YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.
- Pos*: Un registro *TSmallPoint* que contiene las coordenadas del ratón, relativas al área cliente.
- Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*GetCapture, GetDoubleClickTime, SetCapture, SetDoubleClickTime, WM\_MBUTTONDOWN, WM\_MBUTTONUP*

Tabla A-17: Valores de *TWMMButtonDownClk.Keys*

Valor	Descripción
<i>MK_CONTROL</i>	Indica que la tecla Ctrl está presionada.
<i>MK_LBUTTON</i>	Indica que el botón izquierdo del ratón está pulsado.
<i>MK_MBUTTON</i>	Indica que el botón central del ratón está pulsado.
<i>MK_RBUTTON</i>	Indica que el botón derecho del ratón está pulsado.
<i>MK_SHIFT</i>	Indica que la tecla May. está presionada.

***WM\_MBUTTONDOWN***

Sintaxis

```
TWMMButtonDown = record
    Msg: Cardinal;           {identificador del mensaje}
    Keys: Longint;          {opciones de teclas virtuales}
    case Integer of
    0: (
        XPos: Smallint;      {coordenada horizontal del cursor}
        YPos: Smallint);     {coordenada vertical del cursor}
    1: (
        Pos: TSmallPoint;    {registro que contiene las coordenadas del cursor}
        Result: Longint);    {devuelve cero (0) si fue manejado}
    end;
```

**Descripción**

Si el cursor del ratón está situado dentro del área cliente de una ventana y se hace clic con el botón central, un mensaje *WM\_MBUTTONDOWN* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana.

**Campos**

*Msg*: El identificador del mensaje. Contiene la constante *WM\_MBUTTONDOWN*.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento del clic. Este campo puede contener uno o más valores de la Tabla A-18.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*GetCapture*, *SetCapture*, *WM\_MBUTTONDOWNBLCLK*, *WM\_MBUTTONUP*

**Tabla A-18: Valores de *TWMMButtonDown.Keys***

Valor	Descripción
<i>MK_CONTROL</i>	Indica que la tecla Ctrl está presionada.
<i>MK_LBUTTON</i>	Indica que el botón izquierdo del ratón está pulsado.
<i>MK_MBUTTON</i>	Indica que el botón central del ratón está pulsado.
<i>MK_RBUTTON</i>	Indica que el botón derecho del ratón está pulsado.
<i>MK_SHIFT</i>	Indica que la tecla May. está presionada.

***WM\_MBUTTONUP*****Sintaxis**

*TWMMButtonUp* = **record**

*Msg*: Cardinal;           {identificador del mensaje}  
*Keys*: Longint;           {opciones de teclas virtuales}

**case** Integer **of**

0: (  
     *XPos*: Smallint;       {coordenada horizontal del cursor}  
     *YPos*: Smallint);     {coordenada vertical del cursor}

1: (  
     *Pos*: TSmallPoint;     {registro que contiene las coordenadas del cursor}  
     *Result*: Longint);    {devuelve cero (0) si fue manejado}

**end;**

**Descripción**

Si el cursor del ratón está situado dentro del área cliente de una ventana y el botón central es liberado, un mensaje *WM\_MBUTTONDOWN* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MBUTTONDOWN*.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento en que el botón fue liberado. Este campo puede contener uno o más valores de la Tabla A-19.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*GetCapture*, *SetCapture*, *WM\_MBUTTONDOWNBLCLK*, *WM\_MBUTTONDOWN*

**Tabla A-19: Valores de *TWMButtonUp.Keys***

Valor	Descripción
<i>MK_CONTROL</i>	Indica que la tecla Ctrl está presionada.
<i>MK_LBUTTON</i>	Indica que el botón izquierdo del ratón está pulsado.
<i>MK_RBUTTON</i>	Indica que el botón derecho del ratón está pulsado.
<i>MK_SHIFT</i>	Indica que la tecla May. está presionada.

***WM\_MDIACTIVATE*****Sintaxis**

```

TWMMDIActivate = record
    Msg: Cardinal;           {identificador del mensaje}
    case Integer of
    0: (
        ChildWnd: HWND);    {manejador de la ventana hija que será activada}
    1: (
        DeactiveWnd: HWND;   {manejador de la ventana hija que será desactivada}
        ActiveWnd: HWND;     {manejador de la ventana hija que será activada}
        Result: Longint;     {devuelve cero (0) si fue manejado}
    end;

```

**Descripción**

Una ventana cliente de una aplicación MDI recibe el mensaje *WM\_MDIACTIVATE* cuando debe activar una nueva ventana hija. La ventana cliente MDI pasará el mensaje tanto a la ventana que va a ser desactivada como a la ventana que va a ser activada. Las ventanas hijas MDI son activadas independientemente de la ventana marco MDI. Cuando la ventana marco es activada, la última ventana hija MDI que fue activada mediante el mensaje *WM\_MDIACTIVATE* recibirá el mensaje *WM\_NCACTIVATE*, pero no recibirá otro mensaje *WM\_MDIACTIVATE*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDIACTIVATE*.

*ChildWnd*: El manejador de la ventana hija que va a ser activada. Este campo es válido solamente cuando el mensaje es recibido por una ventana cliente MDI.

*DeactiveWnd*: El manejador de la ventana hija que va a ser desactivada. Este campo es válido solamente cuando el mensaje es recibido por una ventana hija MDI.

*ActiveWnd*: El manejador de la ventana hija que va a ser activada. Este campo es válido solamente cuando el mensaje es recibido por una ventana hija MDI.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*WM\_MDIGETACTIVE*, *WM\_MDINEXT*, *WM\_NCACTIVATE*

**WM\_MDICASCADE****Sintaxis**

```
TWMMDICascade = record
    Msg: Cardinal;           {identificador del mensaje}
    Cascade: Longint;        {opción de comportamiento de la cascada}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve uno (1) si tiene éxito}
end;
```

**Descripción**

El mensaje *WM\_MDICASCADE* es enviado a la ventana cliente de una aplicación MDI para indicarle que disponga en cascada a todas las ventanas hijas abiertas.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDICASCADE*.

*Cascade*: Especifica un comportamiento alternativo para la cascada. Si a este campo se le asigna *MDITILE\_SKIPDISABLED*, todas las ventanas hijas MDI deshabilitadas serán excluidas de la cascada. Si a este campo se le asigna cero, todas las ventanas hijas MDI serán puestas en cascada.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: La ventana cliente MDI devuelve uno si el mensaje tiene éxito; en caso contrario, devuelve cero.

Véase además

*WM\_MDIICONARRANGE*, *WM\_MDITILE*

## WM\_MDICREATE

### Sintaxis

```
TWMMDICreate = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: Integer;         {no se utiliza}
    MDICreateStruct: PMDICreateStruct; {puntero a registro TMDICreateStruct}
    Result: Longint;         {devuelve el manejador de la nueva
                             ventana hija}
end;
```

### Descripción

El mensaje *WM\_MDICREATE* es enviado a una ventana cliente MDI cuando una aplicación quiere crear una ventana hija. La información para la nueva ventana es especificada por el registro al que apunta el campo *MDICreateStruct*. La ventana hija es creada con los estilos *WS\_CHILD*, *WS\_CLIPSIBLINGS*, *WS\_CLIPCHILDREN*, *WS\_SYSMENU*, *WS\_CAPTION*, *WS\_THICKFRAME*, *WS\_MINIMIZEBOX*, *WS\_MAXIMIZEBOX*, además de cualquier otro que sea especificado en el registro *TMDICreateStruct*. Una aplicación debe terminar el procesamiento del mensaje actual *WM\_MDICREATE* antes de que otro mensaje *WM\_MDICREATE* sea enviado. Cuando la ventana cliente MDI recibe el mensaje, envía un mensaje *WM\_CREATE* a la ventana hija. El campo *lpCreateParams* del registro *TCreateStruct* utilizado en el mensaje *WM\_CREATE* contendrá un puntero al registro *TMDICreateStruct*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDICREATE*.

*Unused*: Este campo no es utilizado por este mensaje.

*MDICreateStruct*: Un puntero al registro *TMDICreateStruct* que define los atributos de la nueva ventana hija. El registro *TMDICreateStruct* se define como:

TMDICreateStruct = **packed record**

```

szClass: PAnsiChar;      {clase de la ventana}
szTitle: PAnsiChar;      {título de la ventana}
hOwner: THandle;         {instancia de la propietaria}
x: Integer;              {posición horizontal de la ventana}
y: Integer;              {posición vertical de la ventana}
cx: Integer;             {ancho de la ventana}
cy: Integer;             {altura de la ventana}
style: DWORD;            {estilos de la ventana}
lParam: LPARAM;          {valor definido por la aplicación}

```

**end;**

*szClass*: Puntero a una cadena de caracteres terminada en nulo, sensible a mayúsculas que especifica la clase de ventana para una ventana hija MDI. Esta clase es registrada llamando a la función *RegisterClass*.

*szTitle*: Puntero a una cadena de caracteres terminada en nulo. Esta cadena es mostrada en la barra de título de la ventana hija MDI.

*hOwner*: Manejador de instancia de la aplicación propietaria de la ventana cliente MDI.

*x*: La coordenada horizontal de la ventana hija, relativa al área cliente.

*y*: La coordenada vertical de la ventana hija, relativa al área cliente.

*cx*: El ancho de la ventana hija, en píxeles.

*cy*: La altura de la ventana hija, en píxeles.

*style*: Un valor de 32 bits que especifica los estilos que utiliza esta ventana. Si la ventana cliente MDI utiliza la opción de estilo *MDIS\_ALLCHILDSTYLES*, este campo puede ser una combinación de estilos de la tabla de estilos de ventana en la función *CreateWindow*. (Esta función se describe en detalle en *Los tomos de Delphi: Núcleo del API Win32, editado en castellano también por Danysoft*). En caso contrario, puede ser una combinación de estilos de la Tabla A-20. Dos o más estilos se combinan mediante el operador booleano **or**, por ejemplo, *WS\_MINIMIZE or WS\_HSCROLL*.

*lParam*: Un valor de 32 bits definido por la aplicación.

*Result*: Si el mensaje tiene éxito, la ventana cliente devuelve un manejador de la ventana hija recién creada; en caso contrario, devuelve cero.

Véase además

*CreateMDIWindow*, *WM\_CREATE*, *WM\_MDIDESTROY*

**Tabla A-20: Valores de TWMMDICreate.TMDICreateStruct.style**

Valor	Descripción
WS_MINIMIZE	La ventana hija MDI está inicialmente minimizada.
WS_MAXIMIZE	La ventana hija MDI está inicialmente maximizada.

WS_HSCROLL	La ventana hija MDI tiene una barra de desplazamiento horizontal.
WS_VSCROLL	La ventana hija MDI tiene una barra de desplazamiento vertical.

**WM\_MDIDESTROY***Sintaxis*

```
TWMMDIDestroy = record
    Msg: Cardinal;           {identificador del mensaje}
    Child: HWND;             {manejador de ventana hija}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero}
end;
```

*Descripción*

Este mensaje es enviado a una ventana cliente MDI para cerrar una ventana hija. El título de la ventana hija es eliminado de la ventana marco MDI y la ventana hija es desactivada.

*Campos*

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDIDESTROY*.

*Child:* Especifica el manejador de la ventana hija MDI que será cerrada.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Este mensaje siempre devuelve cero.

*Véase además*

*WM\_MDICREATE*

**WM\_MDIGETACTIVE***Sintaxis*

```
TWMMDIGetActive = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve un manejador de ventana hija}
end;
```

*Descripción*

Este mensaje recupera el manejador de la ventana hija MDI activa.

**Campos**

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDIGETACTIVE*.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Si el mensaje tiene éxito, devuelve un manejador de la ventana hija MDI activa; en caso contrario, devuelve cero.

**Véase además**

*WM\_MDIIVATE, WM\_MDICREATE, WM\_MDIDESTROY*

**WM\_MDIICONARRANGE****Sintaxis**

```
TWMMDIIconArrange = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {no se utiliza}
end;
```

**Descripción**

Este mensaje es enviado a una ventana cliente MDI para alinear todos los iconos de las ventanas hijas. No tiene efecto sobre ventanas hijas que no estén minimizadas.

**Campos**

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDIICONARRANGE*.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Este campo no es utilizado por este mensaje.

**Véase además**

*WM\_MDICASCADE, WM\_MDITILE*

**WM\_MDIMAXIMIZE****Sintaxis**

```
TWMMDIMaximize = record
    Msg: Cardinal;           {identificador del mensaje}
    Maximize: HWND;         {manejador de ventana hija}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero}
```

**end;**

### Descripción

El mensaje *WM\_MDIMAXIMIZE* es enviado a una ventana cliente MDI para maximizar una ventana hija. Windows redimensiona la ventana hija para hacer que su área cliente llene la ventana cliente MDI. La barra de título de la ventana hija es añadida a la barra de título de la ventana marco y el icono del menú de sistema de la ventana hija y sus botones de ventana son situados en la barra de menú de la ventana marco. Si la ventana hija actualmente activa está maximizada y otra ventana hija MDI es activada, Windows restaura la ventana hija que estaba activa y maximiza la ventana hija recién activada.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDIMAXIMIZE*.

*Maximize*: Especifica el manejador de la ventana hija MDI que será maximizada.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este mensaje siempre devuelve cero.

### Véase además

*SM\_MDIICONARRANGE*, *WM\_MDIRESTORE*

## WM\_MDINEXT

### Sintaxis

TWMMDINext = **record**

Msg: Cardinal;	{identificador del mensaje}
Child: HWND;	{manejador de ventana hija}
Next: Longint;	{indicador de ventana anterior o siguiente}
Result: Longint;	{devuelve cero}

**end;**

### Descripción

El mensaje *WM\_MDINEXT* activa la ventana hija anterior o siguiente en una ventana cliente MDI. Si la ventana hija actualmente activa está maximizada y otra ventana hija MDI es activada, Windows restaura la ventana hija que estaba activa y maximiza la ventana hija recién activada.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDINEXT*.

*Child*: Especifica un manejador de una ventana hija. La ventana hija anterior o siguiente a esta ventana será activada en función del valor del campo *Next*. Si este campo es cero, la ventana hija anterior o siguiente a la ventana actualmente activa será activada.

*Next*: Valor que indica si se debe activar la ventana anterior o siguiente. Si este campo tiene valor cero, será activada la siguiente ventana hija. Si este campo vale uno, será activada la ventana hija anterior.

*Result*: Este mensaje siempre devuelve cero.

Véase además

*WM\_MDIACTIVATE*, *WM\_MDIGETACTIVE*

## **WM\_MDIREFRESHMENU**

### *Sintaxis*

```
TWMMDIRefreshMenu = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve un manejador de menú}
end;
```

### *Descripción*

Este mensaje es enviado a la ventana marco MDI para actualizar el menú de la ventana marco.

### *Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDIREFRESHMENU*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si este mensaje tiene éxito, devuelve un manejador del menú de la ventana marco; en caso contrario, devuelve cero.

Véase además

*WM\_MDISETMENU*

## **WM\_MDIRESTORE**

### *Sintaxis*

```
TWMMDIRestore = record
    Msg: Cardinal;           {identificador del mensaje}
```

```

IDChild: HWND;           {manejador de una ventana hija}
Unused: Longint;         {no se utiliza}
Result: Longint;         {devuelve cero}
end;
```

**Descripción**

Este mensaje restaura una ventana hija MDI desde un estado maximizado o minimizado.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDIRESTORE*.

*IDChild*: Especifica el manejador de la ventana hija MDI que será restaurada.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este mensaje siempre devuelve cero.

**Véase además**

*WM\_MDIMAXIMIZE*

**WM\_MDISETMENU****Sintaxis**

```

TWMMDISetMenu = record
    Msg: Cardinal;           {identificador del mensaje}
    MenuFrame: HMENU;        {manejador del menú de la ventana marco}
    MenuWindow: HMENU;       {manejador del menú de la ventana hija}
    Result: Longint;          {devuelve un manejador de menú}
end;
```

**Descripción**

Este mensaje es usado para reemplazar el menú completo o solamente la ventana de menú de una ventana marco MDI. Los elementos de menú de la ventana hija MDI son eliminados del menú de ventana anterior y añadidos al nuevo menú de ventana si este mensaje reemplaza el menú de la ventana. El icono del menú de sistema y los botones de la ventana son quitados del menú anterior de la ventana marco y añadidos al nuevo menú de la ventana marco, si una ventana hija MDI es maximizada y este mensaje reemplaza el menú de la ventana marco MDI.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDISETMENU*.

*MenuFrame*: Especifica el manejador del nuevo menú de la ventana marco. Si este campo es cero, el menú de la ventana marco no es modificado.

*MenuWindow*: Especifica el manejador del nuevo menú de la ventana. Si este campo es cero, el menú de la ventana no es modificado.

*Result*: Si el mensaje tiene éxito, devuelve el manejador del menú anterior de la ventana marco; en caso contrario, devuelve cero.

Véase además

*WM\_MDIREFRESHMENU*

## WM\_MDITILE

### Sintaxis

```
TWMMDITile = record
    Msg: Cardinal;      {identificador del mensaje}
    Tile: Longint;      {opción de mosaico}
    Unused: Longint;    {no se utiliza}
    Result: Longint;    {devuelve uno si tiene éxito}
end;
```

### Descripción

Este mensaje hace que una ventana cliente MDI disponga sus ventanas hijas en forma de mosaico.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MDITILE*.

*Tile*: Valor que especifica cómo las ventanas hijas serán dispuestas en el mosaico. A este campo se le puede asignar un valor de la Tabla A-21.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la función tiene éxito, devuelve uno; en caso contrario, devuelve cero.

Véase además

*WM\_MDICASCADE*, *WM\_MDIICONARRANGE*

**Tabla A-21: Valores de TWMDITile.Tile**

Valor	Descripción
MDITILE_HORIZONTAL	Las ventanas son dispuestas en mosaico horizontal.
MDITILE_SKIPDISABLED	Las ventanas deshabilitadas no son dispuestas en mosaico.
MDITILE_VERTICAL	Las ventanas son dispuestas en mosaico vertical.

**WM\_MEASUREITEM****Sintaxis**

```

TWMMMeasureItem = record
    Msg: Cardinal;                {identificador del mensaje}
    IDCtl: HWND;                  {ID del control}
    MeasureItemStruct: PMeasureItemStruct; {puntero a registro}
    Result: Longint;              {devuelve uno (1) si es manejado}
end;

```

**Descripción**

El mensaje *WM\_MEASUREITEM* es enviado a la ventana propietaria de un botón dibujado por el propietario, un cuadro de combinación, un cuadro de lista, una vista de lista o un elemento de menú, cuando ese control o elemento de menú es creado. Esto permite a la aplicación especificar el tamaño del control. El registro *TWMMMeasureItem* al que apunta el campo *MeasureItemStruct* contiene información que identifica al control que envía el mensaje. La aplicación deberá rellenar los campos *itemWidth* e *itemHeight* de este registro antes de retornar. Si un cuadro de lista o de combinación es creado con los estilos *LBS\_OWNERDRAWVARIABLE* o *CBS\_OWNERDRAWVARIABLE*, el mensaje es enviado a cada elemento de ese control.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MEASUREITEM*.

*IDCtl*: Especifica el identificador del control cuyas medidas serán determinadas, que se corresponde con el campo *CtlID* del registro al que apunta el campo *MeasureItemStruct*. Si este campo tiene valor cero, el mensaje fue enviado por un menú; en caso contrario, fue enviado por un botón, cuadro de combinación, vista de lista o cuadro de lista. Si este campo es distinto de cero y al campo *itemID* de la estructura a la que apunta el campo *MeasureItemStruct* se le asigna -1, el mensaje fue enviado por el control de edición de un cuadro de combinación desplegable.

*MeasureItemStruct*: Puntero a un registro *TMeasureItemStruct* que contiene información sobre el control que envía el mensaje. La aplicación debe rellenar los campos *itemWidth* e *itemHeight* de esta estructura antes de retornar. El registro *TMeasureItemStruct* se define como:

```

TMeasureItemStruct = packed record
    CtlType: UINT;                {indicador de tipo de control}
    CtlID: UINT;                  {identificador del control que envía}
    itemID: UINT;                 {identificador del elemento}
    itemWidth: UINT;              {ancho del elemento}
    itemHeight: UINT;             {altura del elemento}
    itemData: DWORD;              {datos asociados}
end;

```

*CtlType*: Valor que especifica el tipo de control cuyas medidas serán determinadas. Este campo puede contener un valor de la Tabla A-22.

*CtlID*: El identificador del control emisor, si éste es un cuadro de combinación, cuadro de lista o botón.

*itemID*: El identificador del elemento cuyas medidas serán determinadas si el control emisor es un elemento de menú, cuadro de lista, o cuadro de combinación.

*itemWidth*: El ancho de un elemento de menú, en píxeles.

*itemHeight*: La altura de un elemento de menú, de un cuadro de combinación, o de un cuadro de lista, en píxeles.

*itemData*: El valor de 32 bits que fue suministrado en el parámetro *lParam* a los mensajes *CB\_ADDSTRING*, *CB\_INSERTSTRING*, *LB\_ADDSTRING* o *LB\_INSERTSTRING*; o el último valor asignado al control por los mensajes *LB\_SETITEMDATA* o *CB\_SETITEMDATA*. Para un cuadro de lista o cuadro de combinación con los estilos *LBS\_HASSTRINGS* o *CBS\_HASSTRINGS*, este campo es inicialmente cero. Si el campo *ctlType* incluye las opciones *ODT\_BUTTON* o *ODT\_STATIC*, este campo tiene valor cero.

*Result*: Si la aplicación maneja este mensaje, deberá asignar uno (1) a este campo.

Véase además

*WM\_DRAWITEM*

**Tabla A-22: Valores de TWMMMeasureItem.TMeasureItemStruct.CtlType**

Valor	Descripción
ODT_BUTTON	Indica un botón dibujado por el propietario.
ODT_COMBOBOX	Indica un cuadro de combinación dibujado por el propietario.
ODT_LISTBOX	Indica un cuadro de lista dibujado por el propietario.
ODT_LISTVIEW	Indica una vista de lista dibujada por el propietario.
ODT_MENU	Indica un elemento de menú dibujado por el propietario.
ODT_STATIC	Indica un control estático dibujado por el propietario.
ODT_TAB	Indica un control de tabulación dibujado por el propietario.

## WM\_MENUCHAR

### Sintaxis

TWMMMenuChar = **record**

Msg: Cardinal;	{identificador del mensaje}
User: Char;	{valor ASCII del carácter}
Unused: Byte;	{no se utiliza}
MenuFlag: Word;	{opción de tipo de menú}
Menu: HMENU;	{manejador de menú}

```

        Result: Longint;      {devuelve el código de la acción}
    end;

```

### Descripción

Este mensaje es enviado cuando un menú está activo y el usuario pulsa una tecla que no es un acelerador o un atajo de teclado de alguno de los elementos del menú.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MENUCHAR*.

*User*: Especifica el valor ASCII del carácter que fue presionado.

*Unused*: Este campo no es usado por este mensaje.

*MenuFlag*: Valor que indica el tipo de menú activo. Este campo tiene el valor *MF\_POPUP* si un menú desplegable, submenú, o menú emergente está actualmente activo. Si este campo tiene el valor *MF\_SYSMENU*, el menú de sistema está actualmente activo.

*Menu*: Manejador del menú activo.

*Result*: Si la aplicación maneja este mensaje, deberá devolver un valor de la Tabla A-23 en la palabra más significativa de este campo.

### Véase además

*WM\_CHAR*, *WM\_MENUSELECT*

**Tabla A-23: Valores de TWMMenuChar.Result**

Valor	Descripción
0	Descarta el carácter y emite un pitido por el altavoz.
1	Cierra el menú activo.
2	Especifica que la palabra menos significativa del valor devuelto contiene el índice de base cero del elemento de menú a seleccionar.

### WM\_MENUSELECT

#### Sintaxis

```

TWMMenuSelect = record
    Msg: Cardinal;      {identificador del mensaje}
    IDItem: Word;       {identificador del elemento de menú}
    MenuFlag: Word;     {opciones de elemento de menú}
    Menu: HMENU;        {manejador de menú}
    Result: Longint;     {devuelve cero (0) si fue manejado}
end;

```

**Descripción**

Cuando el usuario hace una selección en un menú, un mensaje *WM\_MENUSELECT* es enviado a la ventana propietaria del menú.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MENUSELECT*.

*IDItem*: El identificador del elemento de menú seleccionado. Si el elemento de menú seleccionado abre un submenú, este campo contendrá el índice de base cero de ese elemento de menú.

*MenuFlag*: Combinación de valores que contienen información sobre el elemento de menú seleccionado. Este campo puede contener uno o más valores de la Tabla A-24.

*Menu*: Manejador del menú en el que se hizo clic.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*GetSubMenu*, *WM\_ENTERMENULOOP*, *WM\_EXITMENULOOP*

**Tabla A-24: Valores de TWMMenuSelect.MenuFlag**

Valor	Descripción
MF_BITMAP	El elemento de menú muestra un mapa de bits.
MF_CHECKED	El elemento de menú está marcado.
MF_DISABLED	El elemento de menú está deshabilitado.
MF_GRAYED	Elemento de menú está sombreado.
MF_HILITE	El elemento de menú está resaltado.
MF_MOUSESELECT	El elemento de menú está seleccionado con el ratón.
MF_OWNERDRAW	El elemento de menú es dibujado por el propietario.
MF_POPUP	El elemento de menú abre un menú desplegable o un submenú.
MF_SYSMENU	El elemento de menú está contenido en el menú de sistema.

**WM\_MOUSEACTIVATE****Sintaxis**

TWMMouseActivate = **record**

Msg: Cardinal;	{identificador del mensaje}
TopLevel: HWND;	{manejador de la ventana madre}
HitTestCode: Word;	{código de zona de pulsación}
MouseMsg: Word;	{mensaje del ratón}
Result: Longint;	{devuelve un código de acción}

**end;**

**Descripción**

El mensaje *WM\_MOUSEACTIVATE* es enviado cuando el usuario hace clic sobre una ventana inactiva. Si la ventana en la que se hizo clic es una ventana hija, su ventana madre recibirá el mensaje sólo si la ventana hija lo pasa a la función *DefWindowProc*. La función *DefWindowProc* pasará el mensaje a la ventana madre antes de procesarlo, permitiendo a la ventana madre determinar si la ventana hija debe ser activada.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MOUSEACTIVATE*.

*TopLevel*: Manejador de la ventana madre de nivel superior de la ventana que será activada.

*HitTestCode*: Valor que indica el área de la ventana en la que se hizo clic. Este campo puede contener un valor de la Tabla A-30. Consulte el mensaje *WM\_NCHITTEST* para ver una lista de las opciones disponibles.

*MouseMsg*: El mensaje del ratón que fue generado como resultado del clic. Este mensaje será procesado o descartado en dependencia del valor devuelto.

*Result*: Este mensaje devuelve un resultado que indica si la ventana debe ser activada y si el mensaje del ratón debe ser procesado. Será un valor de la Tabla A-25.

**Véase además**

*DefWindowProc*, *WM\_ACTIVATE*, *WM\_NCHITTEST*

**Tabla A-25: Valores de TWMMouseActivate.Result**

Valor	Descripción
MA_ACTIVATE	Activar la ventana y procesar el mensaje.
MA_ACTIVATEANDEAT	Activar la ventana y descartar el mensaje.
MA_NOACTIVATE	No activar la ventana y procesar el mensaje.
MA_NOACTIVATEANDEAT	No activar la ventana y descartar el mensaje.

**WM\_MOUSEMOVE****Sintaxis**

TWMMouseMove = **record**

Msg: Cardinal;      {identificador del mensaje}  
 Keys: Longint;      {opciones de teclas virtuales}

**case** Integer **of**

0: (

XPos: Smallint;      {coordenada horizontal del cursor}  
 YPos: Smallint);      {coordenada vertical del cursor}

```

1: (
  Pos: TSmallPoint;    {registro que contiene las coordenadas del cursor}
  Result: Longint);    {devuelve cero (0) si fue manejado}
end;

```

### Descripción

Este mensaje es enviado a una ventana cuando el usuario mueve el cursor del ratón dentro del área cliente de la ventana. Sin embargo, si otra ventana ha capturado el ratón, entonces el mensaje es enviado a esa ventana.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MOUSEMOVE*.

*Keys*: Indica si teclas virtuales específicas estaban presionadas en el momento en que el ratón se mueve. Este campo puede contener uno o más valores de la Tabla A-26.

*XPos*: La coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: La coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*GetCapture*, *SetCapture*, *WM\_NCMOUSEMOVE*

**Tabla A-26: Valores de TWMMouseMove.Keys**

Valor	Descripción
MK_CONTROL	Indica que la tecla Ctrl está presionada.
MK_LBUTTON	Indica que el botón izquierdo del ratón está pulsado.
MK_MBUTTON	Indica que el botón central del ratón está pulsado.
MK_RBUTTON	Indica que el botón derecho del ratón está pulsado.
MK_SHIFT	Indica que la tecla May. está presionada.

## WM\_MOVE

### Sintaxis

```

TWMMove = record
  Msg: Cardinal;        {identificador del mensaje}
  Unused: Longint;      {no se utiliza}
  case Integer of
    0: (
      XPos: Smallint;    {coordenada horizontal del cursor}

```

```

        YPos: Smallint);    {coordenada vertical del cursor}
1: (
    Pos: TSmallPoint;      {registro que contiene las coordenadas del cursor}
    Result: Longint);      {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Este mensaje es enviado cuando una ventana es movida. Las coordenadas son relativas a la pantalla si la ventana es una ventana de nivel superior, o relativas al área cliente de la ventana madre si es una ventana hija.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_MOVE*.

*Unused*: Este campo no es utilizado por este mensaje.

*XPos*: La coordenada horizontal de la esquina superior izquierda del área cliente de la ventana movida.

*YPos*: La coordenada vertical de la esquina superior izquierda del área cliente de la ventana movida.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas de la esquina superior izquierda del área cliente de la ventana movida.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*WM\_GETMINMAXINFO*

**WM\_NCACTIVATE**

TWMNCActivate = **record**

```

    Msg: Cardinal;          {identificador del mensaje}
    Active: BOOL;           {indicador de activación}
    Unused: Longint;        {no se utiliza}
    Result: Longint;        {devuelve cero ó uno}
```

**end;**

**Descripción**

Cuando el área no-cliente de una ventana necesita ser cambiada para indicar un estado activo o inactivo, el mensaje *WM\_NCACTIVATE* es enviado.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCACTIVATE*.

*Active*: Indica cuando una barra de título o icono necesita ser cambiado para indicar un estado activo o inactivo. Este campo vale TRUE cuando la barra de título y el icono van a ser dibujados como activos; o FALSE en caso contrario.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si al campo *Active* se le asigna TRUE, el valor devuelto es ignorado. Cuando al campo *Active* se le asigna FALSE, la aplicación debe devolver uno (1) si el mensaje es procesado normalmente; o cero (0) para evitar que la barra de título y el icono sean desactivados.

**Véase además**

*WM\_ACTIVATE*

**WM\_NCCALCSIZE****Sintaxis**

```
TWMNCCalcSize = record
    Msg: Cardinal;                {identificador del mensaje}
    CalcValidRects: BOOL;         {opción de información válida}
    CalcSize_Params: PNCCalcSizeParams; {puntero a la información de tamaño}
    Result: Longint;              {devuelve un código de acción}
end;
```

**Descripción**

Este mensaje es enviado cuando el tamaño y la posición del área cliente de una ventana son calculadas.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCCALCSIZE*.

*CalcValidRects*: Indica si la aplicación debe especificar dónde existe información válida en el área cliente de la ventana. Si este campo contiene TRUE, el campo *CalcSize\_Params* contendrá un puntero al registro *TNCCalcSizeParams* que contiene datos sobre el tamaño, posición y partes válidas del área cliente. Si a este campo se le asigna FALSE, la aplicación no necesita especificar las partes válidas del área cliente y el campo *CalcSize\_Params* apuntará a un registro *TRect*.

*CalcSize\_Params*: Si al campo *CalcValidRects* se le asigna FALSE, este campo apuntará a un registro *TRect* que contiene las nuevas coordenadas de la ventana. En caso contrario, este campo apunta a un registro *TNCCalcSizeParams* que contiene

información usada para calcular el nuevo tamaño y posición de la ventana cliente. El registro *TNCCalcSizeParams* se define como:

*TNCCalcSizeParams* = **packed record**

rgrc: **array**[0..2] of TRect;                    {array de rectángulos}  
lppos: PWindowPos;                    {puntero a los datos de posición de ventana}  
**end;**

*rgrc*: Apunta a un *array* de registros *TRect* que contiene coordenadas de posiciones de la ventana. La primera entrada contiene las nuevas coordenadas de la ventana. La segunda entrada contiene las coordenadas originales de la ventana antes que fuera movida o redimensionada. La tercera entrada contiene las coordenadas originales del área cliente de la ventana antes que fuera movida o redimensionada. Si la ventana es una ventana de nivel superior estas coordenadas son relativas a la pantalla. Si es una ventana hija, las coordenadas son relativas al área cliente de la ventana madre.

*lppos*: Puntero a un registro *TWindowPos* que contiene los valores de tamaño y posición especificados cuando la ventana fue movida o redimensionada. El registro *TWindowPos* se define como:

*TWindowPos* = **packed record**

hwnd: HWND;                    {manejador de ventana}  
hwndInsertAfter: HWND;        {manejador de ventana o indicador de posición}  
x: Integer;                    {posición horizontal}  
y: Integer;                    {posición vertical}  
cx: Integer;                    {ancho de la ventana}  
cy: Integer;                    {altura de la ventana}  
flags: UINT                    {opciones de tamaño y posición}  
**end;**

*hwnd*: Manejador de la ventana que será movida o redimensionada.

*hwndInsertAfter*: Identifica la ventana que precederá, en el orden Z, a la ventana reposicionada. Puede especificar un manejador de una ventana o un valor de la Tabla A-27. Este campo es ignorado si la opción *SWP\_NOZORDER* está incluida en el campo *flags*. Si a este campo se le asigna cero, la ventana será ubicada en la cima del orden Z. Si la posición del orden Z de una ventana es situado encima de todas las demás ventanas siempre visibles, esta ventana se convierte en una ventana siempre visible. Esto tendrá el mismo efecto que especificar la opción *HWND\_TOPMOST* para este campo.

*x*: La coordenada horizontal de la esquina superior izquierda. Si es una ventana hija, las coordenadas son relativas al área cliente de la ventana madre.

*y*: La coordenada vertical de la esquina superior izquierda. Si es una ventana hija, las coordenadas son relativas al área cliente de la ventana madre.

*cx*: El nuevo ancho de la ventana, en píxeles.

*cy*: La nueva altura de la ventana, en píxeles.

*flags*: Combinación de valores de la Tabla A-28 que afectará a la posición y el tamaño de la ventana. Dos o más valores pueden combinarse mediante el operador booleano **or**.

*Result*: Si al campo *CalcValidRects* se le asigna FALSE, la aplicación debe devolver cero. Si al campo *CalcValidRects* se le asigna TRUE, la aplicación puede devolver cero o una combinación de valores de la Tabla A-29.

Véase además

*DefWindowProc*, *MoveWindow*, *SetWindowPos*

**Tabla A-27: Valores de TWMNCCalcSize.CalcSize\_Params.lppos.hwndInsertAfter**

Valor	Descripción
HWND_BOTTOM	Coloca la ventana al final del orden Z. Si esta ventana era una ventana siempre visible, pierde su condición de siempre visible y es colocada debajo de todas las demás.
HWND_NOTOPMOST	Coloca la ventana encima de todas la ventanas que no son siempre visibles, pero detrás de las siempre visibles. Si la ventana ya no era siempre visible, esta opción no tiene efecto.
HWND_TOP	Coloca la ventana en la cima del orden Z.
HWND_TOPMOST	Coloca la ventana encima de todas las no siempre visibles; esta ventana retendrá su posición siempre visible incluso si es desactivada.

**Tabla A-28: Valores de TWMNCCalcSize.CalcSize\_Params.lppos.flags**

Valor	Descripción
SWP_DRAWFRAME	Dibuja el marco definido en la descripción de la clase de la ventana alrededor de la ventana.
SWP_FRAMECHANGED	Hace que un mensaje WM_NCCALCSIZE sea enviado a la ventana, incluso si su tamaño no está cambiando.
SWP_HIDEWINDOW	Oculto la ventana.
SWP_NOACTIVATE	No activa la ventana. Si esta opción no se especifica, la ventana es activada y movida a la cima del grupo de ventanas siempre visibles o no siempre visibles, en dependencia del valor del campo hwndInsertAfter.

Valor	Descripción
SWP_NOCOPYBITS	Descarta el área cliente completa. Si esta opción no se especifica, el área válida del área cliente es guardada y copiada de nuevo en el área cliente después que todos los movimientos y posicionamientos se completen.
SWP_NOMOVE	Mantiene la posición actual, ignorando los campos X e Y.
SWP_NOOWNERZORDER	No cambia la posición de la ventana propietaria en el orden Z.
SWP_NOREDRAU	Cuando se especifica esta opción, no se redibuja la ventana y la aplicación deberá invalidar explícitamente o redibujar cualquier parte de la ventana que necesite ser redibujada, incluyendo el área no-cliente y las barras de desplazamiento.
SWP_NOREPOSITION	Equivalente a SWP_NOOWNERZORDER.
SWP_NOSENDCHANGING	La ventana no recibirá mensajes WM_WINDOWPOSCHANGING.
SWP_NOSIZE	Mantiene el tamaño actual, ignorando los campos CX y CY.
SWP_NOZORDER	Mantiene el orden Z actual, ignorando el campo hwndInsertAfter.
SWP_SHOWWINDOW	Muestra la ventana.

Tabla A-29: Valores de TWMNCCalcSize.Result

Valor	Descripción
WVR_ALIGNBOTTOM	Ajusta el área cliente a la parte inferior de la ventana.
WVR_ALIGNLEFT	Ajusta el área cliente al borde izquierdo de la ventana.
WVR_ALIGNRIGHT	Ajusta el área cliente al borde derecho de la ventana.
WVR_ALIGNTOP	Ajusta el área cliente a la parte superior de la ventana.
WVR_HREDRAW	Redibuja la ventana completa si el área cliente es redimensionada horizontalmente.
WVR_VREDRAW	Redibuja la ventana completa si el área cliente es redimensionada verticalmente.
WVR_REDRAU	Redibuja la ventana completa si el área cliente es redimensionada horizontal o verticalmente.
WVR_VALIDRECTS	Indica que los rectángulos especificados en rgrc[1] y rgrc[2] son rectángulos de origen y destino válidos. Estos determinan qué parte del área cliente va a ser conservada. El rectángulo de origen es copiado, y cualquier parte que pertenezca al rectángulo destino será recortada.

**WM\_NCCREATE***Sintaxis*

```

TWMNCCreate = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: Integer;         {no se utiliza}
    CreateStruct: PCreateStruct; {puntero a registro TCreateStruct}
    Result: Longint;         {devuelve cero (0) ó uno(1)}
end;

```

*Descripción*

El mensaje *WM\_NCCREATE* es enviado a una ventana justo antes de que el mensaje *WM\_CREATE* sea enviado cuando se crea una ventana.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCCREATE*.

*Unused*: Este campo no es utilizado por este mensaje.

*CreateStruct*: Puntero a un registro *TCreateStruct* que contiene datos para inicializar la ventana. El registro *TCreateStruct* se define como:

```

TCreateStruct = packed record
    lpCreateParams: Pointer;   {puntero a datos definidos por la aplicación}
    hInstance: HINST;         {manejador de instancia del módulo}
    hMenu: HMENU;             {manejador del menú o identificador de una
                               ventana hija}
    hwndParent: HWND;         {manejador de la ventana madre}
    cy: Integer;              {altura inicial de la ventana}
    cx: Integer;              {ancho inicial de la ventana}
    y: Integer;               {posición vertical inicial}
    x: Integer;               {posición horizontal inicial}
    style: Longint;           {opciones de estilo de ventana}
    lpszName: PAnsiChar;      {puntero a cadena del nombre de la ventana}
    lpszClass: PAnsiChar;     {puntero a cadena del nombre de la clase}
    dwExStyle: DWORD;         {opciones de estilo extendido de ventanas}
end;

```

Consulte el mensaje *WM\_CREATE* para ver una descripción de este registro.

*Result*: Si la aplicación procesa este mensaje, deberá devolver uno (1) para indicar que la ventana puede ser creada. En caso contrario, puede devolver cero (0), bloqueando la creación de la ventana y causando que las funciones *CreateWindow* o *CreateWindowEx*, que fueron llamadas para crear la ventana devuelvan cero.

Véase además

*CreateWindow, CreateWindowEx, DefWindowProc, WM\_CREATE*

## WM\_NCDESTROY

### Sintaxis

```
TWMNCDESTROY = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

El mensaje *WM\_NCDESTROY* es enviado a una ventana por la función *DestroyWindow*. Es recibido a continuación del mensaje *WM\_DESTROY* e informa a la ventana que su área no-cliente va a ser destruida. Este mensaje hace que se libere cualquier memoria reservada para la ventana.

### Campos

*Msg*: El identificador del mensaje. A este campo se le asigna la constante *WM\_NCDESTROY*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DestroyWindow, WM\_CREATE, WM\_DESTROY, WM\_NCCREATE*

## WM\_NCHITTEST

### Sintaxis

```
TWMNCHITTEST = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: Longint;         {no se utiliza}
    case Integer of
        0: (
            XPos: Smallint;   {coordenada horizontal del cursor del ratón}
            YPos: Smallint);   {coordenada vertical del cursor del ratón}
        1: (
            Pos: TSmallPoint;  {registro que contiene las coordenadas del cursor}
            Result: Longint);  {devuelve la posición del clic}
```

**end;**

### Descripción

El mensaje *WM\_NCHITTEST* es enviado a una ventana cuando el cursor se mueve sobre la ventana o un botón del ratón es pulsado. Si el ratón ha sido capturado, el mensaje es enviado a la ventana que hace la captura.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCHITTEST*.

*Unused*: Este campo no es utilizado por este mensaje.

*XPos*: La coordenada *x* del cursor del ratón, relativa a la pantalla.

*YPos*: La coordenada *y* del cursor del ratón, relativa a la pantalla.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas del cursor del ratón, relativas a la pantalla.

*Result*: Este mensaje devuelve un valor que indica la posición del cursor del ratón dentro de la ventana y puede ser un valor de la Tabla A-30.

### Véase además

*DefWindowProc*, *WM\_MOUSEACTIVATE*, *WM\_MOUSEMOVE*,  
*WM\_NCMOUSEMOVE*

**Tabla A-30: Valores de *TWMNCHitTest.Result***

Valor	Descripción
HTBORDER	El borde de una ventana sin un borde de redimensionamiento.
HTBOTTOM	El borde horizontal inferior.
HTBOTTOMLEFT	La esquina inferior izquierda del borde de la ventana.
HTBOTTOMRIGHT	La esquina inferior derecha del borde de la ventana.
HTCAPTION	La barra de título.
HTCLIENT	El área cliente.
HTERROR	El fondo de la pantalla o una línea divisoria entre ventanas. Esta opción es equivalente a HTNOWHERE, con la diferencia de que es tratada como un error y la función <i>DefWindowProc</i> produce un pitido por el altavoz del sistema.
HTGROWBOX	El cuadro de redimensionamiento. Equivalente a HTSIZE.
HTHSCROLL	La barra de desplazamiento horizontal.
HTLEFT	El borde izquierdo.
HTMENU	El menú.
HTNOWHERE	El fondo de pantalla o una línea divisoria entre ventanas.
HTREDUCE	El botón de Minimizar.
HTRIGHT	El borde derecho.

HTSIZE	El cuadro de redimensionamiento.
HTSYSTEMMENU	El menú de sistema o el botón de Cerrar en una ventana hija.
HTTOP	El borde superior horizontal.
HTTOPLEFT	La esquina superior izquierda del borde de la ventana.
HTTOPRIGHT	La esquina superior derecha del borde de la ventana.
HTTRANSPARENT	En una ventana actualmente cubierta por otra.
HTVSCROLL	La barra de desplazamiento vertical.
HTZOOM	El botón de Maximizar.

## WM\_NCLBUTTONDBLCLK

### Sintaxis

```
TWMNCLButtonDblClk = record
    Msg: Cardinal;           {identificador del mensaje}
    HitTest: Longint;        {indicador de posición del clic}
    XCursor: Smallint;       {coordenada horizontal del cursor}
    YCursor: Smallint;       {coordenada vertical del cursor}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Si el cursor del ratón está situado en el área no cliente de una ventana y se hace doble clic con el botón izquierdo del ratón, un mensaje *WM\_NCLBUTTONDBLCLK* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado. Un doble clic se genera cuando el usuario pulsa y libera el botón del ratón y lo pulsa y libera de nuevo, en las mismas coordenadas y dentro del intervalo de doble clic del sistema. Este proceso genera una serie de cuatro mensajes, en el orden siguiente: *WM\_NCLBUTTONDOWN*, *WM\_NCLBUTTONUP*, *WM\_NCLBUTTONDBLCLK* y *WM\_NCLBUTTONUP*. Una ventana no tiene que tener el estilo de clase *CS\_DBLCLKS* para recibir el mensaje *WM\_NCLBUTTONDBLCLK*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCLBUTTONDBLCLK*.

*HitTest*: Valor que indica el área de la ventana en la que el doble clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCLBUTTONDOWN*,  
*WM\_NCLBUTTONUP*, *WM\_SYSCOMMAND*

## **WM\_NCLBUTTONDOWN**

### **Sintaxis**

```
TWMNCLButtonDown = record
    Msg: Cardinal;           {identificador del mensaje}
    HitTest: Longint;        {indicador de posición del clic}
    XCursor: Smallint;       {coordenada horizontal del cursor}
    YCursor: Smallint;       {coordenada vertical del cursor}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### **Descripción**

Si el cursor del ratón está situado en el área no cliente de una ventana y el botón izquierdo del ratón es pulsado, un mensaje *WM\_NCLBUTTONDOWN* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado.

### **Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCLBUTTONDOWN*.

*HitTest*: Valor que indica el área de la ventana en la que el clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCLBUTTONDBLCLK*,  
*WM\_NCLBUTTONUP*, *WM\_SYSCOMMAND*

## **WM\_NCLBUTTONUP**

### **Sintaxis**

```
TWMNCLButtonUp = record
    Msg: Cardinal;           {identificador del mensaje}
```

```

HitTest: Longint;      {indicador de posición del clic}
XCursor: Smallint;    {coordenada horizontal del cursor}
YCursor: Smallint;    {coordenada vertical del cursor}
Result: Longint;      {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Si el cursor del ratón está situado en el área no cliente de una ventana y el botón izquierdo del ratón es liberado, un mensaje *WM\_NCLBUTTONUP* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCLBUTTONUP*.

*HitTest*: Valor que indica el área de la ventana en la que el clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCLBUTTONDBLCLK*,  
*WM\_NCLBUTTONDOWN*, *WM\_SYSCOMMAND*

**WM\_NCMBUTTONDBLCLK****Sintaxis**

```

TWMNCMButtonDbcClk = record
Msg: Cardinal;      {identificador del mensaje}
HitTest: Longint;   {indicador de posición del clic}
XCursor: Smallint;  {coordenada horizontal del cursor}
YCursor: Smallint;  {coordenada vertical del cursor}
Result: Longint;    {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Si el cursor del ratón está situado en el área no cliente de una ventana y se hace doble clic con el botón central del ratón, un mensaje *WM\_NCMBUTTONDBLCLK* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada

del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado. Un doble clic se genera cuando el usuario pulsa y libera el botón del ratón y lo pulsa y libera de nuevo, en las mismas coordenadas y dentro del intervalo de doble clic del sistema. Este proceso genera una serie de cuatro mensajes, en el orden siguiente: *WM\_NCMBUTTONDOWN*, *WM\_NCMBUTTONUP*, *WM\_NCMBUTTONDBLCLK* y *WM\_NCMBUTTONUP*. Una ventana no tiene que tener el estilo de clase *CS\_DBLCLKS* para recibir el mensaje *WM\_NCMBUTTONDBLCLK*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCMBUTTONDBLCLK*.

*HitTest*: Valor que indica el área de la ventana en la que el doble clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCMBUTTONDOWN*, *WM\_NCMBUTTONUP*, *WM\_SYSCOMMAND*

## WM\_NCMBUTTONDOWN

### Sintaxis

```
TWMNCMButtonDown = record
    Msg: Cardinal;           {identificador del mensaje}
    HitTest: Longint;        {indicador de posición del clic}
    XCursor: Smallint;       {coordenada horizontal del cursor}
    YCursor: Smallint;       {coordenada vertical del cursor}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Si el cursor del ratón está situado en el área no cliente de una ventana y el botón central del ratón es pulsado, un mensaje *WM\_NCMBUTTONDOWN* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCMBUTTONDOWN*.

*HitTest*: Valor que indica el área de la ventana en la que el clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCMBUTTONDOWNBLCLK*,  
*WM\_NCMBUTTONUP*, *WM\_SYSCOMMAND*

**WM\_NCMBUTTONUP****Sintaxis**

```
TWMNCMButtonUp = record
    Msg: Cardinal;           {identificador del mensaje}
    HitTest: Longint;        {indicador de posición del clic}
    XCursor: Smallint;       {coordenada horizontal del cursor}
    YCursor: Smallint;       {coordenada vertical del cursor}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Si el cursor del ratón está situado en el área no cliente de una ventana y el botón central del ratón es liberado, un mensaje *WM\_NCMBUTTONUP* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCMBUTTONUP*.

*HitTest*: Valor que indica el área de la ventana en la que el clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCMBUTTONDBLCLK*,  
*WM\_NCMBUTTONDOWN*, *WM\_SYSCOMMAND*

## **WM\_NCMOUSEMOVE**

*Sintaxis*

```
TWMNCMouseMove = record
    Msg: Cardinal;           {identificador del mensaje}
    HitTest: Longint;        {indicador de posición del clic}
    XCursor: Smallint;       {coordenada horizontal del cursor}
    YCursor: Smallint;       {coordenada vertical del cursor}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

*Descripción*

Este mensaje es enviado a una ventana cuando el usuario mueve el cursor del ratón dentro del área no cliente de la ventana. Sin embargo, si otra ventana ha capturado la entrada del ratón, el mensaje no es enviado.

*Campos*

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCMOUSEMOVE*.

*HitTest:* Valor que indica el área de la ventana en la que se produjo el movimiento. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor:* Especifica la coordenada horizontal del cursor, relativa a la pantalla.

*YCursor:* Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_SYSCOMMAND*

## **WM\_NCPAINT**

*Sintaxis*

```
TWMNCPaint = record
    Msg: Cardinal;           {identificador del mensaje}
```

```

        Unused: array[0..3] of Word;      {no se utiliza}
        Result: Longint;                    {devuelve cero (0) si fue manejado}
    end;

```

### Descripción

Este mensaje es enviado a una ventana cuando las áreas no clientes, tales como la barra de título y el marco, necesitan ser dibujadas.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCPAINT*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*DefWindowProc*, *GetWindowDC*, *WM\_PAINT*

## WM\_NCRBUTTONDBLCLK

### Sintaxis

```

TWMNCRButtonDblClk = record
    Msg: Cardinal;          {identificador del mensaje}
    HitTest: Longint;       {indicador de posición del clic}
    XCursor: Smallint;      {coordenada horizontal del cursor}
    YCursor: Smallint;      {coordenada vertical del cursor}
    Result: Longint;        {devuelve cero (0) si fue manejado}
end;

```

### Descripción

Si el cursor del ratón está situado en el área no cliente de una ventana y se hace doble clic con el botón derecho del ratón, un mensaje *WM\_NCRBUTTONDBLCLK* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado. Un doble clic se genera cuando el usuario pulsa y libera el botón del ratón y lo pulsa y libera de nuevo, en las mismas coordenadas y dentro del intervalo de doble clic del sistema. Este proceso genera una serie de cuatro mensajes, en el orden siguiente: *WM\_NCRBUTTONDOWN*, *WM\_NCRBUTTONUP*, *WM\_NCRBUTTONDBLCLK* y *WM\_NCRBUTTONUP*. Una ventana no tiene que tener el estilo de clase *CS\_DBLCLKS* para recibir el mensaje *WM\_NCRBUTTONDBLCLK*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCRBUTTONDBLCLK*.

*HitTest*: Valor que indica el área de la ventana en la que el doble clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCRBUTTONDOWN*,  
*WM\_NCRBUTTONUP*, *WM\_SYSCOMMAND*

**WM\_NCRBUTTONDOWN****Sintaxis**

```
TWMNCRButtonDown = record
    Msg: Cardinal;           {identificador del mensaje}
    HitTest: Longint;        {indicador de posición del clic}
    XCursor: Smallint;       {coordenada horizontal del cursor}
    YCursor: Smallint;       {coordenada vertical del cursor}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Si el cursor del ratón está situado dentro del área no cliente de una ventana y se hace clic con el botón derecho del ratón, un mensaje *WM\_NCRBUTTONDOWN* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje no es enviado.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCRBUTTONDOWN*.

*HitTest*: Valor que indica el área de la ventana en la que el clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor*: Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor*: Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCRBUTTONDBLCLK*,  
*WM\_NCRBUTTONUP*, *WM\_SYSCOMMAND*

## **WM\_NCRBUTTONUP**

### *Sintaxis*

```
TWMNCRButtonUp = record
    Msg: Cardinal;           {identificador del mensaje}
    HitTest: Longint;        {indicador de posición del clic}
    XCursor: Smallint;       {coordenada horizontal del cursor}
    YCursor: Smallint;       {coordenada vertical del cursor}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### *Descripción*

Si el cursor del ratón está situado dentro del área no cliente de una ventana y se libera el botón derecho del ratón, un mensaje *WM\_NCRBUTTONUP* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón, mediante una llamada a la función *SetCapture*, el mensaje no es enviado.

### *Campos*

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NCRBUTTONUP*.

*HitTest:* Valor que indica el área de la ventana en la que el clic ocurrió. Consulte los posibles valores de este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*XCursor:* Especifica la coordenada horizontal del cursor del ratón, relativa a la pantalla.

*YCursor:* Especifica la coordenada vertical del cursor del ratón, relativa a la pantalla.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_NCHITTEST*, *WM\_NCRBUTTONDBLCLK*,  
*WM\_NCRBUTTONDOWN*, *WM\_SYSCOMMAND*

**WM\_NEXTDLGCTRL***Sintaxis*

```

TWMNextDlgCtrl = record
    Msg: Cardinal;           {identificador del mensaje}
    CtlFocus: Longint;       {manejador del control o indicador de dirección}
    Handle: WordBool;        {opción de foco}
    Unused: Word;            {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

Este mensaje indica a una ventana desplazar el foco del teclado al control anterior o siguiente.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NEXTDLGCTRL*.

*CtlFocus*: El manejador del control que recibirá el foco del teclado, o un indicador de la dirección en que el foco del teclado debe moverse, dependiendo del valor del campo *Handle*. Si al campo *Handle* se le asigna TRUE, este campo contiene el manejador de la ventana que debe recibir el foco del teclado. Si al campo *Handle* se le asigna FALSE, este campo contiene un valor que indica si el control anterior o el siguiente en la secuencia de tabulación debe recibir el foco. En ese caso, si a este campo se le asigna cero, el control siguiente recibirá el foco; de lo contrario, lo recibirá el control anterior.

*Handle*: Indica si el campo *CtlFocus* contiene un manejador de una ventana que va a recibir el foco o un indicador de la dirección de movimiento del foco. Si a este campo se le asigna TRUE, el campo *CtlFocus* contiene un manejador del control que recibirá el foco. Si a este campo se le asigna FALSE, el campo *CtlFocus* contiene un valor que indica la dirección de movimiento del foco del teclado.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

*Véase además*

*PostMessage*, *SendMessage*, *SetFocus*

**WM\_NOTIFY***Sintaxis*

```

TWMNotify = record
    Msg: Cardinal;           {identificador del mensaje}

```

```

IDCtrl: Longint;           {identificador del control}
NMHdr: PNMHdr;            {puntero a un registro}
Result: Longint;          {no se utiliza}
end;
```

### Descripción

El mensaje *WM\_NOTIFY* alerta a una ventana madre de que un evento ha ocurrido en el control que envía el mensaje.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NOTIFY*.

*IDCtrl*: El identificador del control que envía el mensaje.

*NMHdr*: Puntero a un registro *TNMHdr* que contiene información sobre el control que envía el mensaje y el evento que ocurrió dentro de ese control. El registro *TNMHdr* se define como:

*TNMHdr* = **packed record**

```

  hwndFrom: HWND;          {manejador del control}
  idFrom: UINT;             {identificador del control}
  code: Integer;           {código de notificación}
end;
```

*hwndFrom*: Manejador del control que envía el mensaje.

*idFrom*: Identificador del control que envía el mensaje.

*code*: Valor que indica el tipo de notificación. A este campo se le puede asignar un código específico de notificación de control o puede contener un valor de la Tabla A-31.

*Result*: Este campo no es utilizado por este mensaje.

### Véase además

*WM\_KILLFOCUS*, *WM\_LBUTTONDOWN*, *WM\_LBUTTONDOWNBLCLK*,  
*WM\_SETFOCUS*, *WM\_RBUTTONDOWN*, *WM\_RBUTTONDOWNBLCLK*

**Tabla A-31: Valores de *TWMNotify.NMHdr.code***

Valor	Descripción
NM_CLICK	Indica un clic del botón izquierdo del ratón.
NM_DBLCLK	Indica un doble clic del botón izquierdo del ratón.
NM_KILLFOCUS	Indica que el control ha perdido el foco de entrada.
NM_OUTOFMEMORY	Indica un error de falta de memoria.
NM_RCLICK	Indica un clic del botón derecho del ratón.
NM_RDBLCLK	Indica un doble clic del botón derecho del ratón.

Valor	Descripción
NM_RETURN	Indica que el control tiene el foco y que la tecla Intro fue presionada.
NM_SETFOCUS	Indica que el control ha recibido el foco de entrada

## WM\_NOTIFYFORMAT

### Sintaxis

```
TWMNotifyFormat = record
    Msg: Cardinal;           {identificador del mensaje}
    From: HWND;              {manejador de ventana}
    Command: Longint;        {indicador de comando}
    Result: Longint;         {devuelve un código de formato}
end;
```

### Descripción

El mensaje *WM\_NOTIFYFORMAT* es enviado entre ventanas madre e hija para determinar si se debe utilizar una estructura ANSI o Unicode en el mensaje *WM\_NOTIFY*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_NOTIFYFORMAT*.

*From*: Especifica el manejador del control que envía el mensaje.

*Command*: Valor que especifica una consulta de estado. Este campo puede contener un valor de la Tabla A-32.

*Result*: Este mensaje devuelve un valor que indica el formato de la estructura de datos usada en el mensaje *WM\_NOTIFY* y puede ser un valor de la Tabla A-33.

### Véase además

*DefWindowProc*, *WM\_NOTIFY*

**Tabla A-32: Valores de TWMNotifyFormat.Command**

Valor	Descripción
NF_QUERY	Usado por un control cuando envía este mensaje a su ventana madre para determinar si usar un formato de estructura ANSI o Unicode. Se utiliza durante la creación de un control como respuesta al mensaje NF_REQUERY.
NF_REQUERY	Usado para solicitar un mensaje NF_QUERY. En este caso el mensaje es enviado desde una ventana a un control del cual es propietaria.

Tabla A-33: Valores de `TWMNotifyFormat.Result`

Valor	Descripción
NFR_ANSI	WM_NOTIFY debe usar estructuras de datos ANSI.
NFR_UNICODE	WM_NOTIFY debe usar estructuras de datos Unicode.
0	Se ha producido un error.

## WM\_PAINT

### Sintaxis

```

TWMPaint = record
    Msg: Cardinal;           {identificador del mensaje}
    DC: HDC;                 {manejador de contexto de dispositivo}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si es manejado}
end;
```

### Descripción

El mensaje *WM\_PAINT* es enviado para solicitar que una aplicación redibuje su ventana. La solicitud puede ser hecha por el sistema o por otra aplicación. Puede también ser generada cuando se hace una llamada a las funciones *UpdateWindow* o *RedrawWindow*. El mensaje *WM\_PAINT* es enviado cuando una ventana tiene una región invalidada y no hay otros mensajes en la cola de mensajes de la aplicación. La función *DefWindowProc* validará las regiones no válidas después de que se efectúe el dibujo. La función *DefWindowProc* puede también enviar un mensaje *WM\_NCPAINT* si el marco de la ventana necesita ser redibujado, o un mensaje *WM\_ERASEBKGD* si el fondo necesita ser redibujado. Observe que el mensaje *WM\_PAINT* es enviado sólo una vez para cada evento del sistema que requiera el redibujo.

Si la aplicación llama a la función *RedrawWindow* usando la opción *RDW\_INTERNALPAINT*, la aplicación recibirá un mensaje de dibujo interno. En este caso, la función *GetUpdateRect* debe ser llamada para determinar si hay una región no válida que deba ser actualizada. Si no hay región no válida, la aplicación no debe llamar a las funciones *BeginPaint* o *EndPaint*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_PAINT*.

*DC*: Especifica el contexto de dispositivo sobre el cual se debe dibujar. A este campo se le puede asignar cero si el dibujo va a ser realizado sobre el contexto de dispositivo por defecto de la ventana.

*Unused*: Este campo no es utilizado por este mensaje.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*BeginPaint, DefWindowProc, DispatchMessage, EndPaint, GetMessage, GetUpdateRect, PeekMessage, RedrawWindow, UpdateWindow, WM\_ERASEBKGND, WM\_NCPAINT*

## WM\_PAINTCLIPBOARD

### Sintaxis

```
TWMPaintClipboard = record
    Msg: Cardinal;           {identificador del mensaje}
    Viewer: HWND;            {manejador del visualizador del portapapeles}
    PaintStruct: THandle     {manejador del registro TPaintStruct}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Un visualizador del portapapeles envía el mensaje *WM\_PAINTCLIPBOARD* a un propietario del portapapeles cuando el portapapeles contiene datos del tipo *CF\_OWNERDISPLAY* y el área cliente del visualizador necesita ser redibujada. El propietario debe comparar el campo *rcPaint* del objeto *PaintStruct* con las dimensiones especificadas en el último mensaje *WM\_SIZECLIPBOARD* recibido, para determinar qué parte del área cliente necesita ser redibujada. El propietario del portapapeles debe llamar a la función *GlobalLock* para acceder al registro *PaintStruct* y desbloquearlo mediante una llamada a la función *GlobalUnlock* antes de retornar.

### Campos

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_PAINTCLIPBOARD*.

*Viewer:* Especifica el manejador de la ventana del visualizador del portapapeles.

*PaintStruct:* Manejador de un objeto *DDESHARE* que contiene un registro *TPaintStruct*. Este registro define qué parte del área cliente necesita ser redibujada. El registro *TPaintStruct* se define como:

```
TPaintStruct = packed record
    hdc: HDC;                {manejador de contexto de dispositivo}
    fErase: BOOL;            {opción de borrado de fondo}
    rcPaint: Trect;          {área que será dibujada}
    fRestore: BOOL;          {reservado}
    fIncUpdate: BOOL;        {reservado}
    rgbReserved: array[0..31] of Byte; {reservado}
end;
```

*hdc*: Manejador del contexto de dispositivo que necesita ser redibujado.

*fErase*: Valor que indica si el fondo del contexto de dispositivo debe ser borrado. Si a este campo se le asigna TRUE, el fondo debe ser borrado. Esto podría ser útil en el caso de que una aplicación crease una clase de ventana que no tuviera una brocha de fondo. Esta información se especifica en el campo *hbrBackground* del registro *TWndClass*.

*rcPaint*: Un registro *TRect* que define las coordenadas rectangulares de la región que será redibujada.

*fRestore*: Este campo es usado internamente por Windows y debe ser ignorado.

*fIncUpdate*: Este campo es usado internamente por Windows y debe ser ignorado.

*rgbReserved*: Este campo es usado internamente por Windows y debe ser ignorado.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*GlobalLock*, *GlobalUnlock*, *WM\_SIZECLIPBOARD*

## WM\_PALETTECHANGED

### Sintaxis

```
TWMPaletteChanged = record
    Msg: Cardinal;           {identificador del mensaje}
    PalChg: HWND;           {manejador de ventana}
    Unused: Longint;        {no se utiliza}
    Result: Longint;        {no se utiliza}
end;
```

### Descripción

El mensaje *WM\_PALETTECHANGED* es enviado a todas las ventanas de nivel superior y solapadas cuando la ventana enfocada cambie la paleta del sistema realizando una paleta lógica. Esto permite a las otras ventanas realizar una paleta lógica como paleta de fondo. Este mensaje tiene que ser enviado a todas las ventanas de nivel superior y a las ventanas solapadas, incluyendo a aquella que cambió la paleta del sistema, y a todas las ventanas hijas que usan una paleta. Para evitar caer en un bucle infinito, una ventana que reciba este mensaje no debe activar su paleta si el manejador identificado por el campo *PalChg* se corresponde con su propio manejador de ventana.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_PALETTECHANGED*.

*PalChg*: Especifica el manejador de la ventana que cambió la paleta del sistema.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Este campo no es utilizado por este mensaje.

Véase además

*WM\_PALETTEISCHANGING*, *WM\_QUERYNEWPALETTE*

## **WM\_PALETTEISCHANGING**

### **Sintaxis**

```
TWMPaletteIsChanging = record
    Msg: Cardinal;           {identificador del mensaje}
    Realize: HWND;          {manejador de ventana}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### **Descripción**

El mensaje *WM\_PALETTEISCHANGING* es enviado cuando una aplicación va a realizar una paleta lógica. Cuando el mensaje *WM\_PALETTECHANGED* es enviado, la aplicación que realiza la paleta lógica no espera a que otras aplicaciones procesen este mensaje. Por lo tanto, la paleta del sistema puede ya contener nuevos colores en el momento en que una aplicación recibe este mensaje.

### **Campos**

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_PALETTEISCHANGING*.

*Realize:* Especifica el manejador de la ventana que va a realizar su paleta lógica.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_PALETTECHANGED*, *WM\_QUERYNEWPALETTE*

## **WM\_PARENTNOTIFY**

### **Sintaxis**

```
TWMParentNotify = record
    Msg: Cardinal;           {identificador del mensaje}
    case Event: Word of      {indicador de tipo de evento}
        WM_CREATE, WM_DESTROY: (
```

```

ChildID: Word;           {identificador de la ventana hija}
ChildWnd: Hwnd;          {manejador de la ventana hija}
WM_LBUTTONDOWN,
WM_MBUTTONDOWN,
WM_RBUTTONDOWN: (
    Value: Word;          {indicador de tipo de evento}
    XPos: Smallint;       {posición horizontal del cursor del ratón}
    YPos: Smallint;       {posición vertical del cursor del ratón}
0: (
    Value1: Word;          {indicador de tipo de evento}
    Value2: Longint;       {identificador de la ventana hija o posición del cursor}
    Result: Longint;       {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Cuando una ventana hija es creada o destruida, o cuando se hace clic con un botón del ratón dentro de una ventana hija, un mensaje *WM\_PARENTNOTIFY* es enviado a la madre de la ventana hija y a todas las ventanas anteriores. Si una ventana hija va a ser creada, el mensaje *WM\_PARENTNOTIFY* es enviado antes que las funciones *CreateWindow* o *CreateWindowEx* retornen. Si una ventana hija va a ser destruida, el mensaje es enviado antes de que cualquier acción para destruir la ventana haya tenido lugar. Sin embargo, las ventanas hijas que tienen el estilo extendido *WS\_EX\_NOPARENTNOTIFY* no enviarán este mensaje a sus ventanas madres.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_PARENTNOTIFY*.

*Event*: Valor que indica el tipo de evento que ha ocurrido. Este campo puede contener un valor de la Tabla A-34.

*ChildID*: Identificador de la ventana hija.

*ChildWnd*: Manejador de la ventana hija.

*Valor*: Valor que indica el tipo de evento que ha ocurrido. Este campo puede contener un valor de la Tabla A-34.

*XPos*: Especifica la posición horizontal del cursor del ratón. Este campo es válido sólo cuando el campo *Value* identifica eventos *WM\_LBUTTONDOWN*, *WM\_MBUTTONDOWN* o *WM\_RBUTTONDOWN*.

*YPos*: Especifica la posición vertical del cursor del ratón. Este campo es válido sólo cuando el campo *Value* identifica eventos *WM\_LBUTTONDOWN*, *WM\_MBUTTONDOWN* o *WM\_RBUTTONDOWN*.

*Value1*: Valor que indica el tipo de evento que ha ocurrido. Este campo puede contener un valor de la Tabla A-34.

*Value2*: Si el campo *Value1* identifica eventos *WM\_CREATE* o *WM\_DESTROY*, este campo contendrá un manejador de la ventana hija que va a ser creada o destruida. Si el campo *Value1* identifica eventos *WM\_LBUTTONDOWN*, *WM\_MBUTTONDOWN* o *WM\_RBUTTONDOWN*, este campo contiene las coordenadas del cursor del ratón, con la coordenada horizontal en la palabra menos significativa y la coordenada vertical en la palabra más significativa.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*CreateWindow*, *CreateWindowEx*, *WM\_CREATE*, *WM\_DESTROY*,  
*WM\_LBUTTONDOWN*, *WM\_MBUTTONDOWN*, *WM\_RBUTTONDOWN*

**Tabla A-34: Valores de TWMParentNotify.Event**

Valor	Descripción
WM_CREATE	Indica que una ventana hija va a ser creada.
WM_DESTROY	Indica que una ventana hija va a ser destruida.
WM_LBUTTONDOWN	Indica que el cursor del ratón está sobre la ventana hija y se ha pulsado el botón izquierdo.
WM_MBUTTONDOWN	Indica que el cursor del ratón está sobre la ventana hija y se ha pulsado el botón central.
WM_RBUTTONDOWN	Indica que el cursor del ratón está sobre la ventana hija y se ha pulsado el botón derecho.

## WM\_PASTE

### Sintaxis

```

TWMPaste = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {no se utiliza}
end;
```

### Descripción

Este mensaje es enviado a un control de edición o cuadro de combinación para copiar el texto ubicado en el portapapeles a dicho control, en la posición actual del cursor de edición. El texto será copiado dentro del control sólo si el portapapeles contiene datos en formato *CF\_TEXT*. Este mensaje fallará si se envía a un cuadro de combinación con el estilo *CBS\_DROPDOWNLIST*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_PASTE*.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Este campo no es utilizado por este mensaje.

Véase además

*WM\_CLEAR, WM\_COPY, WM\_CUT*

## WM\_QUERYDRAGICON

### Sintaxis

```
TWMQueryDragIcon = record
    Msg: Cardinal;                {identificador del mensaje}
    Unused: array[0..3] of Word;  {no se utiliza}
    Result: Longint;              {devuelve un manejador de cursor o icono}
end;
```

### Descripción

Cuando una ventana minimizada va a ser arrastrada y no hay icono definido para su clase de ventana, el mensaje *WM\_QUERYDRAGICON* es enviado a la ventana para determinar qué imagen mostrar durante el arrastre. La aplicación debe devolver el manejador de un icono o cursor que sea compatible con el controlador de vídeo actual del sistema. Si la aplicación no devuelve un manejador de un icono o cursor compatible, el sistema utilizará el cursor por defecto. Si el icono devuelto por la aplicación utiliza colores, será convertido a blanco y negro.

### Campos

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_QUERYDRAGICON*.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Si la aplicación maneja este mensaje deberá devolver el manejador del cursor o icono que será mostrado cuando se arrastre la ventana.

Véase además

*DefWindowProc, LoadCursor, LoadIcon*

## WM\_QUERYENDSESSION

### Sintaxis

```
TWMQueryEndSession = record
    Msg: Cardinal;                {identificador del mensaje}
    Source: Longint;              {manejador de la ventana que está finalizando}
```

Unused: Longint;	{no se utiliza}
Result: Longint;	{devuelve un código de finalización}

**end;**

### Descripción

Este mensaje es enviado a todos los procesos activos cuando Windows se está cerrando o una aplicación ha llamado a la función *ExitWindows*. Esto permite a una aplicación cancelar una solicitud de terminación de sesión. Si alguna aplicación devuelve cero, Windows no finalizará y dejará de enviar mensajes *WM\_QUERYENDSESSION*. Después de procesar este mensaje, el sistema enviará el mensaje *WM\_ENDSESSION* a todos los procesos activos, con el resultado del mensaje *WM\_QUERYENDSESSION* en el campo *EndSession*. Por defecto, la función *DefWindowProc* devuelve uno (1) para este mensaje. Bajo Windows NT, cuando una aplicación devuelve 1 para este mensaje recibe el mensaje *WM\_ENDSESSION* y es finalizada, independientemente de cómo las otras aplicaciones respondan al mensaje *WM\_QUERYENDSESSION*. Bajo Windows 95/98, las aplicaciones recibirán el mensaje *WM\_ENDSESSION* sólo después de que todas las aplicaciones hayan devuelto 1 en el mensaje *WM\_QUERYENDSESSION*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_QUERYENDSESSION*.

*Source*: Especifica el manejador de la ventana que solicita la finalización.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Una aplicación debe devolver uno (1) si quiere terminar; en caso contrario, debe devolver cero (0).

### Véase además

*DefWindowProc*, *ExitWindows*, *WM\_ENDSESSION*

## WM\_QUERYNEWPALETTE

### Sintaxis

TWMQueryNewPalette = <b>record</b>	
Msg: Cardinal;	{identificador del mensaje}
Unused: <b>array</b> [0..3] of Word;	{no se utiliza}
Result: Longint;	{devuelve un código de realización de paleta}
<b>end;</b>	

### Descripción

Una ventana recibe el mensaje *WM\_QUERYNEWPALETTE* cuando está recibiendo el foco del teclado, lo que indica que debe realizar su paleta lógica.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_QUERYNEWPALETTE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la ventana realiza su paleta lógica debe devolver uno; en caso contrario, debe devolver cero.

**Véase además**

*WM\_PALETTECHANGED*, *WM\_PALETTEISCHANGING*

**WM\_QUERYOPEN****Sintaxis**

```
TWMQueryOpen = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve un código de restauración}
end;
```

**Descripción**

Este mensaje es enviado a una ventana iconizada cuando va a ser restaurada. La aplicación no debe ejecutar ninguna acción que produzca un cambio de foco mientras se procesa este mensaje.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_QUERYOPEN*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: La aplicación debe devolver uno (1) para permitir a la ventana que sea restaurada, o cero (0) para evitar que la ventana sea restaurada.

**Véase además**

*DefWindowProc*, *WM\_SHOWWINDOW*, *WM\_SIZE*

**WM\_QUIT****Sintaxis**

```
TWMQuit = record
    Msg: Cardinal;           {identificador del mensaje}
    ExitCode: Longint;       {código de finalización}
```

```

        Unused: Longint;           {no se utiliza}
        Result: Longint;          {no se utiliza}
    end;

```

#### Descripción

Este mensaje es enviado como resultado de una llamada a la función *PostQuitMessage*. Hace que la función *GetMessage* devuelva cero, saliendo así del bucle de mensajes y terminando la aplicación.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_QUIT*.

*ExitCode*: Especifica el código de finalización. Consulte la función *PostQuitMessage* para ver una descripción de este campo. (Esta función está descrita en *Los Tomos de Delphi: Núcleo del API Win32*, editado en castellano también por Danysoft).

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

#### Véase además

*GetMessage*, *PostQuitMessage*

### WM\_RBUTTONDOWNBLCLK

#### Sintaxis

```

TWMRButtonDownBlcK = record
    Msg: Cardinal;           {identificador del mensaje}
    Keys: Longint;          {opciones de teclas virtuales}
    case Integer of
        0: (
            XPos: Smallint;   {coordenada horizontal del cursor}
            YPos: Smallint);  {coordenada vertical del cursor}
        1: (
            Pos: TSmallPoint;  {registro que contiene las coordenadas del cursor}
            Result: Longint);  {devuelve cero (0) si fue manejado}
    end;

```

#### Descripción

Si el cursor del ratón está situado dentro del área cliente de una ventana y se hace doble clic con el botón derecho, un mensaje *WM\_RBUTTONDOWNBLCLK* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana. Un doble clic se genera cuando el usuario pulsa y libera el botón del ratón y lo

pulsa y libera de nuevo, en las mismas coordenadas y dentro del intervalo de doble clic del sistema. Este proceso genera una serie de cuatro mensajes, en el orden siguiente: *WM\_RBUTTONDOWN*, *WM\_RBUTTONUP*, *WM\_RBUTTONDOWNBLCLK* y *WM\_RBUTTONUP*. Sólo una ventana con el estilo de clase *CS\_DBLCLKS* recibirá el mensaje *WM\_RBUTTONDOWNBLCLK*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_RBUTTONDOWNBLCLK*.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento del doble clic. Este campo puede contener uno o más valores de la Tabla A-35.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón, relativas al área cliente.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*GetCapture*, *GetDoubleClickTime*, *SetCapture*, *SetDoubleClickTime*,  
*WM\_RBUTTONDOWN*, *WM\_RBUTTONUP*

**Tabla A-35: Valores de TWRButtonDblClk.Keys**

Valor	Descripción
<i>MK_CONTROL</i>	Indica que la tecla Ctrl está presionada.
<i>MK_LBUTTON</i>	Indica que el botón izquierdo del ratón está pulsado.
<i>MK_MBUTTON</i>	Indica que el botón central del ratón está pulsado.
<i>MK_RBUTTON</i>	Indica que el botón derecho del ratón está pulsado.
<i>MK_SHIFT</i>	Indica que la tecla May. está presionada.

## **WM\_RBUTTONDOWN**

### Sintaxis

TWMRButtonDown = **record**

*Msg*: Cardinal; {identificador del mensaje}  
*Keys*: Longint; {opciones de teclas virtuales}

**case** Integer **of**

0: (

*XPos*: Smallint; {coordenada horizontal del cursor}

*YPos*: Smallint); {coordenada vertical del cursor}

1: (

```

        Pos: TSmallPoint;    {registro que contiene las coordenadas del cursor}
        Result: Longint;     {devuelve cero (0) si fue manejado}
    end;

```

### Descripción

Si el cursor del ratón está situado dentro del área cliente de una ventana y el botón derecho es pulsado, un mensaje *WM\_RBUTTONDOWN* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_RBUTTONDOWN*.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento del clic. Este campo puede contener uno o más valores de la Tabla A-36.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón, relativas al área cliente.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*GetCapture*, *SetCapture*, *WM\_RBUTTONDOWNBLCLK*, *WM\_RBUTTONUP*

**Tabla A-36: Valores de *TWMRButtonDown.Keys***

Valor	Descripción
<i>MK_CONTROL</i>	Indica que la tecla Ctrl está presionada.
<i>MK_LBUTTON</i>	Indica que el botón izquierdo del ratón está pulsado.
<i>MK_MBUTTON</i>	Indica que el botón central del ratón está pulsado.
<i>MK_RBUTTON</i>	Indica que el botón derecho del ratón está pulsado.
<i>MK_SHIFT</i>	Indica que la tecla May. está presionada.

## ***WM\_RBUTTONUP***

### Sintaxis

```

TWMRButtonUp = record
    Msg: Cardinal;    {identificador del mensaje}
    Keys: Longint;    {opciones de teclas virtuales}
    case Integer of
        0: (

```

```

XPos: Smallint;      {coordenada horizontal del cursor}
YPos: Smallint);     {coordenada vertical del cursor}
1: (
Pos: TSmallPoint;    {registro que contiene las coordenadas del cursor}
Result: Longint);    {devuelve cero (0) si fue manejado}

```

**end;**

### Descripción

Si el cursor del ratón está situado dentro del área cliente de una ventana y se libera el botón derecho, un mensaje *WM\_RBUTTONDOWN* es enviado a la ventana situada bajo el cursor. Sin embargo, si otra ventana ha capturado la entrada del ratón mediante una llamada a la función *SetCapture*, el mensaje es enviado a esa ventana.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_RBUTTONDOWN*.

*Keys*: Indica si ciertas teclas virtuales estaban presionadas en el momento en que el botón es liberado. Este campo puede contener uno o más valores de la Tabla A-37.

*XPos*: Especifica la coordenada horizontal del cursor del ratón, relativa al área cliente.

*YPos*: Especifica la coordenada vertical del cursor del ratón, relativa al área cliente.

*Pos*: Un registro *TSmallPoint* que contiene las coordenadas actuales del ratón, relativas al área cliente.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*GetCapture*, *SetCapture*, *WM\_RBUTTONDOWNBLCLK*, *WM\_RBUTTONDOWNDOWN*

**Tabla A-37: Valores de *TWMRButtonUp.Keys***

Valor	Descripción
<i>MK_CONTROL</i>	Indica que la tecla Ctrl está presionada.
<i>MK_LBUTTON</i>	Indica que el botón izquierdo del ratón está pulsado.
<i>MK_MBUTTON</i>	Indica que el botón central del ratón está pulsado.
<i>MK_SHIFT</i>	Indica que la tecla May. está presionada.

## WM\_RENDERALLFORMATS

### Sintaxis

```

TWMRenderAllFormats = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}

```

```

        Result: Longint;           {devuelve cero (0) si fue manejado}
    end;

```

#### Descripción

Este mensaje es enviado a la ventana propietaria del portapapeles si va a ser destruida y ha demorado la entrega de uno o más de los formatos del portapapeles. Con vistas a que los datos permanezcan accesibles para otras aplicaciones, la propietaria del portapapeles debe colocar sus datos en el portapapeles en todos los formatos en que sea capaz. Cualquier formato del portapapeles no entregado es eliminado de la lista de formatos disponibles en el portapapeles antes que el mensaje retorne.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_RENDERALLFORMATS*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

#### Véase además

*EmptyClipboard*, *OpenClipboard*, *SetClipboardData*, *WM\_RENDERFORMAT*

### WM\_RENDERFORMAT

#### Sintaxis

```

TWMMRenderFormat = record
    Msg: Cardinal;           {identificador del mensaje}
    Format: Longint;         {formato de portapapeles}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

#### Descripción

Este mensaje es enviado a la ventana propietaria del portapapeles si está usando entrega demorada para un formato específico y una aplicación ha solicitado datos del portapapeles en este formato. La ventana propietaria del portapapeles no debe abrir el portapapeles antes de llamar a la función *SetClipboardData* para colocar los datos solicitados en el portapapeles.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_RENDERFORMAT*.

*Format*: Indica qué formato del portapapeles va a ser entregado.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*SetClipboardData, WM\_RENDERALLFORMATS*

## WM\_SETCURSOR

### Sintaxis

```
TWMSetCursor = record
    Msg: Cardinal;           {identificador del mensaje}
    CursorWnd: HWND;         {manejador de la ventana que tiene el cursor}
    HitTest: Word;           {indicador de posición del clic}
    MouseMsg: Word;          {identificador de mensaje de ratón}
    Result: Longint;         {devuelve un código de procesamiento de mensaje}
end;
```

### Descripción

Si el cursor del ratón se mueve dentro de una ventana y el ratón no está capturado, el mensaje *WM\_SETCURSOR* es enviado a esa ventana. Antes de que este mensaje sea enviado a una ventana, la función *DefWindowProc* lo envía a la ventana madre. Si la ventana madre devuelve uno (1), este mensaje no es enviado a la ventana de destino. La función *DefWindowProc* procesa este mensaje cambiando el cursor a una flecha, si el cursor no está situado dentro del área cliente, o al cursor registrado, si el cursor está situado dentro del área cliente. Si el campo *HitTest* incluye el atributo *HTERROR* y el campo *MouseMsg* contiene un identificador de un mensaje de clic de botón, la función *MessageBeep* es llamada. Si un menú ha sido activado, el campo *MouseMsg* será cero.

### Campos

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SETCURSOR*.

*CursorWnd:* Manejador de la ventana que contiene el cursor del ratón.

*HitTest:* Valor que indica el área de la ventana en la que el cursor del ratón está situado. Consulte los posibles valores para este campo en la Tabla A-30 bajo el mensaje *WM\_NCHITTEST*.

*MouseMsg:* El identificador del mensaje del ratón que generó el mensaje *WM\_SETCURSOR*.

*Result:* Este mensaje devuelve el resultado del tratamiento del mensaje en la ventana madre.

Véase además

*DefWindowProc*, *MessageBeep*, *WM\_MOUSEMOVE*, *WM\_NCHITTEST*

## **WM\_SETFOCUS**

### *Sintaxis*

```
TWMSetFocus = record
    Msg: Cardinal;           {identificador del mensaje}
    FocusedWnd: HWND;        {manejador de ventana}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### *Descripción*

Este mensaje es enviado a una ventana después de que ésta recibe el foco del teclado.

### *Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SETFOCUS*.

*FocusedWnd*: Especifica el manejador de la ventana que perdió el foco del teclado. Este campo puede valer cero si ninguna ventana tenía el foco del teclado.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*SetFocus*, *WM\_KILLFOCUS*

## **WM\_SETFONT**

### *Sintaxis*

```
TWMSetFont = record
    Msg: Cardinal;           {identificador del mensaje}
    Font: HFONT;             {manejador de fuente}
    Redraw: WordBool;        {opción de control de redibujado}
    Unused: Word;            {no se utiliza}
    Result: Longint;         {no se utiliza}
end;
```

**Descripción**

El mensaje *WM\_SETFONT* es enviado a un control para especificar qué fuente utilizar cuando se dibuja texto. Este mensaje no alterará el tamaño del control. Un control debe hacer cualquier cambio que sea necesario en su tamaño antes de cambiar la fuente.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SETFONT*.

*Font*: Manejador de la fuente que será usada en el control. Si a este campo se le asigna cero, el control utilizará la fuente por defecto del sistema.

*Redraw*: Valor que indica si el control debe ser redibujado después de seleccionar la fuente. Si a este campo se le asigna TRUE, el control será redibujado.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

**Véase además**

*CreateFont*, *CreateFontIndirect*, *WM\_GETFONT*

**WM\_SETHOTKEY****Sintaxis**

```
TWMSetHotKey = record
    Msg: Cardinal;           {identificador del mensaje}
    Key: Longint;            {código y modificador de la tecla virtual}
    Unused: Longint;         {no se utiliza}
    Result: Longint;         {devuelve un código de éxito}
end;
```

**Descripción**

El mensaje *WM\_SETHOTKEY* es enviado a una ventana para asociar la ventana con una ‘tecla caliente’. Cuando esta combinación sea pulsada, el sistema activará la ventana y le enviará un mensaje *WM\_SYSCOMMAND* con el valor *SC\_HOTKEY* asignado al campo *CmdType*. Una ventana puede tener solamente una tecla caliente asociada en cada momento. Si este mensaje es enviado a una ventana que ya tiene una ‘tecla caliente’ asociada, la nueva ‘tecla caliente’ reemplaza a la anterior. Sin embargo, la misma combinación de ‘tecla caliente’ puede ser asociada con más de una ventana. En ese caso, cualquiera de las ventanas podrá ser activada cuando la ‘tecla caliente’ sea pulsada.

El mensaje *WM\_SETHOTKEY* no tiene relación alguna con la función *RegisterHotKey*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SETHOTKEY*.

*Key*: Especifica el código virtual de la tecla y las opciones de modificación de la ‘tecla caliente’ que será asociada con la ventana a la que se le envió el mensaje. Asignando cero a este campo la ventana dejará de tener una ‘tecla caliente’ asociada. Al byte menos significativo de este campo se le debe asignar el código virtual de la tecla, y al más significativo las opciones de modificación, que se listan en la Tabla A-38. Observe que los códigos virtuales de teclas *VK\_ESCAPE*, *VK\_SPACE* y *VK\_TAB* no pueden ser utilizados como ‘tecla caliente’.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este mensaje devuelve un código de éxito de la Tabla A-39.

Véase además

*WM\_GETHOTKEY*, *WM\_SYSCOMMAND*

**Tabla A-38: Valores de *TWMSetHotKey.Key***

Valor	Descripción
HOTKEYF_ALT	La tecla Alt.
HOTKEYF_CONTROL	La tecla Ctrl.
HOTKEYF_EXT	Una tecla extendida.
HOTKEYF_SHIFT	La tecla May.

**Tabla A-39: Valores de *TWMSetHotKey.Result***

Valor	Descripción
-1	Error debido a una tecla caliente no válida.
0	Error debido a una ventana no válida.
1	La ‘tecla caliente’ ha sido asignada.
2	La ‘tecla caliente’ ha sido asignada, pero más de una ventana tiene asignada esa misma combinación.

***WM\_SETICON*****Sintaxis**

*TWMSetIcon* = **record**

<i>Msg</i> : Cardinal;	{identificador del mensaje}
<i>BigIcon</i> : Longbool;	{indicador de tamaño de icono}
<i>Icon</i> : HICON;	{manejador de icono}
<i>Result</i> : Longint;	{devuelve el manejador de icono anterior}

**end;**

**Descripción**

El mensaje *WM\_SETICON* asocia un nuevo icono grande o pequeño con la ventana de destino.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SETICON*.

*BigIcon*: Valor que indica si el icono es grande o pequeño. Este campo contiene TRUE si el icono a ser asociado con la ventana es un icono grande; FALSE si es un icono pequeño.

*Icon*: Manejador del icono que será asociado con la ventana.

*Result*: Este mensaje devuelve el manejador del icono que estaba anteriormente asociado con la ventana, dependiendo del valor del campo *BigIcon*. Devuelve cero si la ventana no tenía asociado un icono del tamaño indicado.

**Véase además**

*DefWindowProc*, *WM\_GETICON*

**WM\_SETREDRAW****Sintaxis**

```
TWMSetRedraw = record
    Msg: Cardinal;      {identificador del mensaje}
    Redraw: Longint;    {valor para la opción de redibujado}
    Unused: Longint;    {no se utiliza}
    Result: Longint;    {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

El mensaje *WM\_SETREDRAW* es enviado a una ventana para asignar el estado de su opción de redibujado. Si el valor de esta opción es cero, la ventana no será redibujada después de cualquier función de dibujo, comportándose igual que la función *LockWindowUpdate*. La opción de redibujado debe ser asignada antes de que se produzca cualquier dibujo sobre la ventana.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SETREDRAW*.

*Redraw*: Especifica el valor a asignar a la opción de redibujado. Si a este campo se le asigna uno (1), la opción de redibujado será asignada y se podrá comenzar a dibujar. Si

a este campo se le asigna cero (0), la ventana no se actualizará hasta que a la opción de redibujado se le asigne 1.

*Unused:* Este campo no es utilizado por este mensaje.

*Result:* Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*InvalidateRect, LockWindowUpdate*

## WM\_SETTEXT

### Sintaxis

```
TWMSetText = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: Longint;         {no se utiliza}
    Text: PChar;             {puntero a una cadena de caracteres terminada en nulo}
    Result: Longint;         {devuelve un código de éxito}
end;
```

### Descripción

Este mensaje es enviado a una ventana para cambiar su texto.

### Campos

*Msg:* El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SETTEXT*.

*Unused:* Este campo no es utilizado por este mensaje.

*Text:* Puntero a una cadena de caracteres terminada en nulo que contiene el texto que será asociado a la ventana.

*Result:* Este mensaje devuelve un valor de la Tabla A-40.

Véase además

*DefWindowProc, WM\_GETTEXT*

**Tabla A-40: Valores de TWMSetText.Result**

Valor	Descripción
1	El texto fue asignado con éxito.
0 (sólo controles de edición)	El texto no fue asignado debido a espacio insuficiente.
LB_ERRSPACE (sólo cuadros de lista)	El texto no fue asignado debido a espacio insuficiente.
CB_ERRSPACE (sólo cuadros de combinación)	El texto no fue asignado debido a espacio insuficiente.

Valor	Descripción
CB_ERR (sólo cuadros de combinación)	El texto no fue asignado porque el cuadro de combinación no tiene un control de edición.

## WM\_SHOWWINDOW

### Sintaxis

```

TWMSHOWWINDOW = record
    Msg: Cardinal;           {identificador del mensaje}
    Show: BOOL;              {opción de mostrar/ocultar}
    Status: Longint;         {código de estado}
    Result: Longint;         {devuelve cero (0) si es manejado}
end;
```

### Descripción

Una ventana recibirá un mensaje *WM\_SHOWWINDOW* cuando va a ser mostrada u ocultada. Si a la ventana se le asignó el estilo *WS\_VISIBLE* cuando fue creada, el mensaje *WM\_SHOWWINDOW* es enviado después que la ventana es creada, pero antes de que sea mostrada. Las funciones *ShowWindow* y *ShowOwnedPopups* también generarán este mensaje. Este mensaje no es enviado cuando una ventana solapada de nivel superior es creada con los estilos *WS\_MAXIMIZE* o *WS\_MINIMIZE*, o si la opción *SW\_SHOWNORMAL* es especificada en una llamada a la función *ShowWindow*.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SHOWWINDOW*.

*Show*: Valor que indica si la ventana va a ser mostrada u ocultada. A este campo se le asigna FALSE si la ventana va a ser ocultada o TRUE si va a ser mostrada.

*Status*: Valor que indica el estado de la ventana que va a ser mostrada. A este campo se le asigna cero si el mensaje fue enviado como resultado de una llamada a la función *ShowWindow*. En caso contrario, a este campo se le asigna un valor de la Tabla A-41.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*DefWindowProc*, *ShowOwnedPopups*, *ShowWindow*

**Tabla A-41: Valores de TWMSHOWWINDOW.Status**

Valor	Descripción
SW_OTHERUNZOOM	La ventana va a ser descubierta porque una ventana previamente maximizada va a ser restaurada o minimizada.

Valor	Descripción
SW_OTHERZOOM	La ventana va a ser cubierta porque otra ventana va a ser maximizada.
SW_PARENTCLOSING	La propietaria de la ventana ha sido minimizada.
SW_PARENTOPENING	La propietaria de la ventana ha sido restaurada.

**WM\_SIZE***Sintaxis*

```

TWMSize = record
    Msg: Cardinal;           {identificador del mensaje}
    SizeType: Longint;       {indicador de tipo de tamaño}
    Width: Word;             {nuevo ancho del área cliente, en píxeles}
    Height: Word;            {nueva altura del área cliente, en píxeles}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

El mensaje *WM\_SIZE* es enviado a una ventana después de que su tamaño ha cambiado. Si el mensaje *WM\_SIZE* hace que se llame a las funciones *SetScrollPos* o *MoveWindow*, al parámetro *redraw* de estas funciones se le debe asignar TRUE para forzar que la ventana sea redibujada.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SIZE*.

*SizeType*: Valor que indica el tipo de operación de redimensionamiento que ha ocurrido. Este campo puede contener un valor de la Tabla A-42.

*Width*: Especifica el nuevo ancho del área cliente, en píxeles.

*Height*: Especifica la nueva altura del área cliente, en píxeles.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

*Véase además*

*MoveWindow*

**Tabla A-42: Valores de TWMSize.SizeType**

Valor	Descripción
SIZE_MAXHIDE	Indica que otra ventana ha sido maximizada.
SIZE_MAXIMIZED	Indica que la ventana ha sido maximizada.
SIZE_MAXSHOW	Indica que otra ventana ha sido restaurada.

SIZE_MINIMIZED	Indica que la ventana ha sido minimizada.
SIZE_RESTORED	La ventana ha sido redimensionada, pero no maximizada o minimizada.

**WM\_SIZECLIPBOARD***Sintaxis*

```

TWMSizeClipboard = record
    Msg: Cardinal;           {identificador del mensaje}
    Viewer: HWND;           {manejador de visualizador del portapapeles}
    RC: THandle;            {manejador de un objeto rectangular}
    Result: Longint;        {no se utiliza}
end;

```

*Descripción*

El mensaje *WM\_SIZECLIPBOARD* es enviado por una ventana visualizadora del portapapeles a la propietaria del portapapeles cuando el tamaño de su área cliente ha cambiado. Este mensaje sólo será enviado si el portapapeles contiene datos en el formato *CF\_OWNERDISPLAY*. Cuando la ventana visualizadora del portapapeles va a ser destruida o redimensionada, enviará este mensaje a la propietaria del portapapeles, especificando un rectángulo vacío. La propietaria del portapapeles debe usar las funciones *GlobalLock* y *GlobalUnlock* para recuperar el rectángulo identificado por el manejador *DDESHARE* situado en el campo *RC*.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SIZECLIPBOARD*.

*Viewer*: Manejador de la ventana visualizadora del portapapeles.

*RC*: Manejador de un objeto *DDESHARE* que contiene un registro *TRect*. Este registro *TRect* contiene las nuevas coordenadas del área cliente de la ventana visualizadora del portapapeles.

*Result*: Este campo no es utilizado por este mensaje.

*Véase además*

*GlobalLock*, *GlobalUnlock*, *WM\_RENDERALLFORMATS*, *WM\_RENDERFORMAT*

**WM\_SPOOLERSTATUS***Sintaxis*

```

TWMSpoolerStatus = record

```

Msg: Cardinal;	{identificador del mensaje}
JobStatus: Longint;	{contiene la constante PR_JOBSTATUS}
JobsLeft: Word;	{cantidad de trabajos de impresión pendientes}
Unused: Word;	{no se utiliza}
Result: Longint;	{devuelve cero (0) si fue manejado}

**end;**

#### Descripción

El mensaje *WM\_SPOOLERSTATUS* es enviado por el Administrador de Impresión cuando un trabajo de impresión es añadido o quitado de la cola de impresión.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SPOOLERSTATUS*.

*JobStatus*: Este campo contiene la constante *PR\_JOBSTATUS*.

*JobsLeft*: Especifica la cantidad de trabajos de impresión que quedan en el *spooler* de la impresora, gestionado por el Administrador de Impresión.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

#### Véase además

*WM\_NOTIFY*, *WM\_NOTIFYFORMAT*

## WM\_STYLECHANGED

#### Sintaxis

TWMStyleChanged = **record**

Msg: Cardinal;	{identificador del mensaje}
StyleType: Longint;	{opción de tipo de estado}
StyleStruct: PStyleStruct;	{puntero a un registro TStyleStruct}
Result: Longint;	{devuelve cero (0) si fue manejado}

**end;**

#### Descripción

El mensaje *WM\_STYLECHANGED* es enviado a una ventana cuando sus estilos han cambiado como resultado de una llamada a la función *SetWindowLong*.

#### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_STYLECHANGED*.

*StyleType*: Combinación de valores que especifica si han cambiado los estilos comunes o extendidos de la ventana. Este campo puede contener uno o más valores de la Tabla A-43.

*StyleStruct*: Puntero a un registro *TStyleStruct*. Este registro contiene los atributos de estilo de la ventana antes y después de la llamada a la función *SetWindowLong*. El registro *TStyleStruct* se define como:

*TStyleStruct* = **packed record**

*styleOld*: DWORD;                      {estilos anteriores de la ventana}  
    *styleNew*: DWORD;                    {nuevos estilos de la ventana}

**end;**

*styleOld*: Especifica los atributos de estilo presentes antes de que la función *SetWindowLong* fuera llamada.

*styleNew*: Especifica los atributos de estilo asignados por la llamada a la función *SetWindowLong*.

Consulte las funciones *CreateWindow* y *CreateWindowEx* para ver una lista de las opciones de estilo de ventanas disponibles. Estas funciones se analizan en detalle en *Los Tomos de Delphi: Núcleo del API Win32, editado en castellano también por Danysoft*).

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*CreateWindow*, *CreateWindowEx*, *SetWindowLong*, *WM\_STYLECHANGING*

**Tabla A-43: Valores de *TWMStyleChanged.StyleType***

Valor	Descripción
GWL_EXSTYLE	Indica que los atributos de estilo extendidos de la ventana han cambiado.
GWL_STYLE	Indica que los atributos de estilo comunes de la ventana han cambiado.

## **WM\_STYLECHANGING**

*Sintaxis*

*TWMStyleChanging* = **record**

*Msg*: Cardinal;                      {identificador del mensaje}  
    *StyleType*: Longint;                {indicador de tipo de estilo}  
    *StyleStruct*: *PStyleStruct*;        {puntero a un registro *TStyleStruct*}  
    *Result*: Longint;                    {devuelve cero (0) si fue manejado}

**end;**

**Descripción**

El mensaje *WM\_STYLECHANGING* es enviado a una ventana cuando sus estilos van a cambiar como resultado de una llamada a la función *SetWindowLong*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_STYLECHANGING*.

*StyleType*: Combinación de valores que especifica si han cambiado los estilos comunes o extendidos de la ventana. Este campo puede contener uno o más valores de la Tabla A-44.

*StyleStruct*: Puntero a un registro *TStyleStruct*. Este registro contiene los atributos de estilo de la ventana antes y después de la llamada a la función *SetWindowLong*. El registro *TStyleStruct* se define como:

```
TStyleStruct = packed record
    styleOld: DWORD;           {estilos anteriores de la ventana}
    styleNew: DWORD;           {nuevos estilos de la ventana}
end;
```

Consulte el mensaje *WM\_STYLECHANGED* para ver una descripción de este registro.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*CreateWindow*, *CreateWindowEx*, *SetWindowLong*, *WM\_STYLECHANGED*

**Tabla A-44: Valores de *TWMStyleChanging.StyleType***

Valor	Descripción
GWL_EXSTYLE	Indica que los atributos de estilo extendidos de la ventana han cambiado.
GWL_STYLE	Indica los los atributos de estilo comunes de la ventana han cambiado.

**WM\_SYSCHAR****Sintaxis**

```
TWMSysChar = record
    Msg: Cardinal;           {identificador del mensaje}
    CharCode: Word;          {código de carácter}
    Unused: Word;            {no se utiliza}
    KeyData: Longint;        {contiene información diversa}
    Result: Longint;          {devuelve cero (0) si fue manejado}
end;
```

**Descripción**

Cuando una tecla es pulsada mientras se mantiene presionada la tecla **Alt** (indicando una tecla de sistema), un mensaje *WM\_SYSCHAR* es enviado a la ventana que tiene el foco del teclado. Este mensaje *WM\_SYSCHAR* es el resultado de un mensaje *WM\_SYSKEYDOWN* traducido por la función *TranslateMessage*.

Las teclas extendidas en un teclado de 101 ó 102 teclas son:

- En la sección principal del teclado, las teclas **Alt** y **Ctrl** de la derecha.
- A la izquierda del teclado numérico, las teclas **Ins**, **Supr**, **Inicio**, **Fin**, **Av.Pág.**, **Re.Pág.**, y las cuatro teclas de flechas.
- En el teclado numérico, las teclas **Bloq.Núm.**, dividir (/) y la tecla **Intro**.
- Las teclas **PrintScrn** y **Break**.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SYSCHAR*.

*CharCode*: El código del carácter de la tecla de menú de la ventana.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado anterior de la tecla e indicador de estado de transición. La Tabla A-45 muestra qué información se almacena en cada posición de bit, dentro de los 32 bits del campo *KeyData*. La palabra más significativa (bits del 16 al 31), pertenece al mensaje inmediatamente precedente *WM\_SYSKEYDOWN*, que provocó el mensaje *WM\_SYSCHAR* mediante la función *TranslateMessage*.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

**Véase además**

*TranslateMessage*, *WM\_SYSKEYDOWN*

**Tabla A-45: Valores de TWMSysChar.KeyData**

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.
29	El código de contexto. Si la tecla Alt estaba presionada mientras la tecla fue pulsada, este bit contiene 1; en caso contrario, contiene 0.

30	El estado previo de la tecla. Si la tecla estaba presionada antes de que se enviara el mensaje, este bit contiene 1; en caso contrario, contiene 0.
31	El estado de transición. Si la tecla está siendo liberada, este bit contiene 1; en caso contrario, contiene 0.

---

**WM\_SYSCOLORCHANGE***Sintaxis*

```

TWMSysColorChange = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {no se utiliza}
end;

```

*Descripción*

Cuando los atributos de un color del sistema son cambiados, el mensaje *WM\_SYSCOLORCHANGE* es enviado a todas las ventanas de nivel superior. Un mensaje *WM\_PAINT* será enviado a cualquier ventana afectada por el cambio de color. Este mensaje tiene que ser reenviado a cualquier control común usado por la ventana.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SYSCOLORCHANGE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este campo no es utilizado por este mensaje.

*Véase además*

*SetSysColors*, *WM\_PAINT*

**WM\_SYSCOMMAND***Sintaxis*

```

TWMSysCommand = record
    Msg: Cardinal;           {identificador del mensaje}
    case CmdType: Longint of {indicador de tipo de comando}
        SC_HOTKEY: (
            ActivateWnd: HWND); {manejador de ventana}
        SC_KEYMENU: (
            Key: Word);          {código de carácter}
    SC_CLOSE, SC_HSCROLL, SC_MAXIMIZE, SC_MINIMIZE,
    SC_MOUSEMENU,

```

```

SC_MOVE, SC_NEXTWINDOW, SC_PREVWINDOW, SC_RESTORE,
SC_SCREENSAVE, SC_SIZE, SC_TASKLIST, SC_VSCROLL: (
  XPos: Smallint;           {posición horizontal del cursor del ratón}
  YPos: Smallint;           {posición vertical del cursor del ratón}
  Result: Longint);         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Este mensaje es enviado a una ventana cuando un elemento del menú de sistema es seleccionado o se hace clic en el botón de minimizar o maximizar. Este mensaje puede ser también enviado a la función *DefWindowProc* para ejecutar cualquier acción identificada por los valores en la Tabla A-46. Observe que si el menú del sistema es modificado mediante las funciones *AppendMenu*, *InsertMenu*, *ModifyMenu*, *InsertMenuItem*, o *SetMenuItemInfo*, un valor de comando de sistema definido por el usuario tiene que ser especificado para el nuevo elemento. En este caso, la aplicación debe manejar el mensaje *WM\_SYSCOMMAND* para procesar los comandos procedentes de los nuevos elementos de menú.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SYSCOMMAND*.

*CmdType*: Valor que indica el tipo de comando de sistema. Este campo puede contener un valor de comando de sistema definido por el usuario (usado cuando se añaden elementos de menú al menú de sistema de la ventana) o puede contener un valor de la Tabla A-46. Observe que los cuatro bits menos significativos de este valor son usados internamente. Si una aplicación está comprobando un valor de comando de sistema definido por el usuario, debe combinar el valor de este campo con \$FFF0 mediante el operador booleano **and**.

*ActivateWnd*: Si una tecla caliente asociada con una ventana fue pulsada, este campo contendrá el manejador de la ventana que será activada por esa tecla caliente.

*Key*: Contiene el código del carácter de la tecla de sistema que fue pulsada. Este no es otro que el valor ASCII de cualquier tecla pulsada mientras la tecla **Alt** estaba presionada. Por defecto, la combinación **Alt + espacio** activará el menú de sistema.

*XPos*: Especifica la posición horizontal del cursor del ratón, en coordenadas de pantalla, cuando un elemento del menú de sistema es seleccionado con el ratón.

*YPos*: Especifica la posición vertical del cursor del ratón, en coordenadas de pantalla, cuando un elemento del menú de sistema es seleccionado con el ratón.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*AppendMenu*, *DefWindowProc*, *GetSystemMenu*, *InsertMenu*, *ModifyMenu*, *WM\_COMMAND*, *WM\_SETHOTKEY*

Tabla A-46: Valores de `TWMSysCommand.CmdType`

Valor	Descripción
SC_CLOSE	Cierra la ventana.
SC_CONTEXTHELP	El cursor del ratón es cambiado a un puntero con forma de signo de interrogación. Cualquier control sobre el que se haga clic con este cursor del ratón recibirá el mensaje WM_HELP.
SC_DEFAULT	El elemento por defecto del menú es seleccionado.
SC_HOTKEY	Indica que la tecla caliente de una ventana fue pulsada.
SC_HSCROLL	Desplaza la ventana horizontalmente.
SC_KEYMENU	Indica que el menú de sistema fue activado por teclado.
SC_MAXIMIZE	Maximiza la ventana.
SC_MINIMIZE	Minimiza la ventana.
SC_MONITORPOWER	Sólo Windows 95/98: Asigna el estado del dispositivo de video para aquellos que soporten ahorro de energía.
SC_MOUSEMENU	Indica que el menú de sistema fue activado por un clic del ratón.
SC_MOVE	Mueve la ventana.
SC_NEXTWINDOW	Activa la ventana siguiente.
SC_PREVWINDOW	Activa la ventana anterior.
SC_RESTORE	Restaura la ventana.
SC_SCREENSAVE	Ejecuta el protector de pantalla.
SC_SIZE	Redimensiona la ventana.
SC_TASKLIST	Activa y muestra la lista de tareas del sistema.
SC_VSCROLL	Desplaza la ventana verticalmente.

**WM\_SYSDEADCHAR***Sintaxis*

```

TWMSysDeadChar = record
    Msg: Cardinal;      {identificador del mensaje}
    CharCode: Word;     {código de carácter}
    Unused: Word;       {no se utiliza}
    KeyData: Longint;   {contiene información diversa}
    Result: Longint;    {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

Cuando un mensaje `WM_SYSKEYDOWN` es traducido por una llamada a la función `TranslateMessage`, un mensaje `WM_SYSDEADCHAR` es enviado a la ventana con el foco del teclado. Este mensaje es generado como resultado de pulsar una tecla muerta

mientras se mantiene presionada la tecla **Alt**. Una tecla muerta es aquella que genera un carácter adicional que es usado en combinación con otra tecla, creando así un carácter combinado o compuesto. El ejemplo típico es el de un carácter con un acento o marca diacrítica. La tecla muerta que identifica el acento o marca diacrítica es introducida primero, seguida por la tecla que identifica el carácter que tendrá aplicada la marca.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SYSDEADCHAR*.

*CharCode*: El código del carácter de la tecla que fue presionada.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado previo de la tecla e indicador de estado de transición. La Tabla A-47 muestra qué información se almacena en cada posición de bit, dentro de los 32 bits del campo *KeyData*. La palabra más significativa (bits del 16 al 31), pertenece al mensaje inmediatamente precedente *WM\_SYSKEYDOWN*, que provocó el mensaje *WM\_SYSDEADCHAR* mediante la función *TranslateMessage*.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*TranslateMessage*, *WM\_DEADCHAR*, *WM\_KEYDOWN*, *WM\_SYSKEYDOWN*, *WM\_SYSKEYUP*

**Tabla A-47: Valores de TWMSysDeadChar.KeyData**

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.
29	El código de contexto. Si la tecla Alt estaba presionada mientras la tecla fue pulsada, este bit contiene 1; en caso contrario, contiene 0.
30	El estado previo de la tecla. Si la tecla estaba presionada antes de que se enviara el mensaje, este bit contiene 1; en caso contrario, contiene 0.
31	El estado de transición. Si la tecla está siendo liberada, este bit contiene 1; en caso contrario, contiene 0.

**WM\_SYSKEYDOWN***Sintaxis*

```

TWMSysKeyDown = record
    Msg: Cardinal;           {identificador del mensaje}
    CharCode: Word;          {código de la tecla virtual}
    Unused: Word;            {no se utiliza}
    KeyData: Longint;        {contiene información diversa}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

El mensaje *WM\_SYSKEYDOWN* es enviado cuando el usuario pulsa una tecla con la tecla **Alt** presionada. El mensaje será enviado a la ventana que tiene el foco del teclado o a la ventana actualmente activa, si ninguna ventana tiene el foco del teclado. El indicador de contexto en el campo *KeyData* diferencia entre estos dos eventos.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_SYSKEYDOWN*.

*CharCode*: Especifica el código virtual de la tecla que fue pulsada.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado previo de la tecla e indicador de estado de transición. La Tabla A-48 muestra qué información se almacena en cada posición de bit, dentro de los 32 bits del campo *KeyData*.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

*Véase además*

*DefWindowProc*, *WM\_SYSCHAR*, *WM\_SYSCOMMAND*, *WM\_SYSKEYUP*

**Tabla A-48: Valores de TWMSysKeyDown.KeyData**

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.

29	El código de contexto. Si la tecla <b>Alt</b> estaba presionada mientras la tecla fue pulsada, este bit contiene 1. Este bit no se asignará si el mensaje <b>WM_SYSKEYDOWN</b> fue enviado a la ventana activa porque ninguna ventana tenía el foco del teclado.
30	El estado previo de la tecla. Si la tecla estaba presionada antes de que se enviara el mensaje, este bit contiene 1; en caso contrario, contiene 0.
31	El estado de transición. Este bit siempre contendrá 0.

**WM\_SYSKEYUP***Sintaxis*

```

TWMSysKeyUp = record
    Msg: Cardinal;           {identificador del mensaje}
    CharCode: Word;          {código de la tecla virtual}
    Unused: Word;            {no se utiliza}
    KeyData: Longint;         {contiene información diversa}
    Result: Longint;          {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

El mensaje **WM\_SYSKEYUP** es enviado cuando el usuario libera una tecla que fue pulsada mientras la tecla **Alt** estaba presionada. El mensaje será enviado a la ventana que tenga el foco del teclado o a la ventana activa, si ninguna ventana tiene el foco del teclado. El código de contexto en el campo *KeyData* distingue entre estos dos eventos.

En teclados ampliados de 102 teclas no-U.S., la tecla **Alt** de la derecha se interpreta como una combinación **Ctrl+Alt**. Cuando esta tecla es pulsada, produce los siguientes mensajes, en orden: **WM\_KEYDOWN** (con el código de tecla virtual **VK\_CONTROL**), **WM\_KEYDOWN** (con el código de tecla virtual **VK\_MENU**), **WM\_KEYUP** (con el código de tecla virtual **VK\_CONTROL**) y **WM\_SYSKEYUP** (con el código de tecla virtual **VK\_MENU**).

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje **WM\_SYSKEYUP**.

*CharCode*: Especifica el código de la tecla virtual de la tecla que fue liberada.

*Unused*: Este campo no es utilizado por este mensaje.

*KeyData*: Especifica la cantidad de repeticiones, código de barrido, indicador de tecla extendida, código de contexto, indicador de estado previo de la tecla e indicador de estado de transición. La Tabla A-49 muestra qué información se almacena en cada posición de bit, dentro de los 32 bits del campo *KeyData*.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*DefWindowProc*, *WM\_SYSCHAR*, *WM\_SYSCOMMAND*, *WM\_SYSKEYDOWN*

**Tabla A-49: Valores de TWMSysKeyUP.KeyData**

Valor	Descripción
0-15	La cantidad de repeticiones resultante de que el usuario haya mantenido presionada la tecla.
16-23	El código de barrido, cuyo valor depende del fabricante OEM del teclado.
24	El indicador de tecla extendida. Si la tecla es extendida (Alt y Ctrl derechos), este bit contiene 1; en caso contrario, contiene 0.
25-28	No usados.
29	El código de contexto. Si la tecla Alt estaba presionada mientras la tecla fue pulsada, este bit contiene 1. Este bit no se asignará si el mensaje <i>WM_SYSKEYDOWN</i> fue enviado a la ventana activa porque ninguna ventana tenía el foco del teclado.
30	El estado previo de la tecla. Este bit siempre contendrá 1.
31	El estado de transición. Este bit siempre contendrá 0.

## **WM\_TIMECHANGE**

### **Sintaxis**

```

TWMTTimeChange = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

### **Descripción**

Cualquier aplicación que modifica la hora del sistema debe enviar este mensaje a todas las ventanas de nivel superior, usando la función *SendMessage* y asignando *HWND\_TOPMOST* al parámetro *hWnd*.

### **Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_TIMECHANGE*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*SendMessage*

**WM\_TIMER***Sintaxis*

```

TWMTimer = record
    Msg: Cardinal;           {identificador del mensaje}
    TimerID: Longint;        {identificador del temporizador}
    TimerProc: TFarProc;     {dirección de la función de respuesta opcional}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;

```

*Descripción*

Este mensaje es enviado a la cola de mensajes del hilo de ejecución que instaló un temporizador cada vez que expire el intervalo de tiempo establecido para el temporizador. Si una función de respuesta fue especificada en la llamada a la función *SetTimer*, este mensaje será pasado a la función de respuesta y no a la cola de mensajes del hilo de ejecución.

*Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_TIMER*.

*TimerID*: El identificador del temporizador creado cuando la función *SetTimer* fue llamada.

*TimerProc*: Especifica la dirección de la función de respuesta opcional. Cuando este campo no es **nil**, el mensaje *WM\_TIMER* es enviado a la función de respuesta; en caso contrario, es enviado a la cola de mensajes del hilo de ejecución que instaló el temporizador.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

*Véase además*

*DispatchMessage*, *SetTimer*

**WM\_UNDO***Sintaxis*

```

TWMTimer = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: array[0..3] of Word; {no se utiliza}
    Result: Longint;         {si tiene éxito, devuelve uno}
end;

```

**Descripción**

El mensaje *WM\_UNDO* es enviado a un control de edición como un comando para deshacer la última operación. El nuevo texto es borrado y el texto anteriormente borrado es restaurado.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_UNDO*.

*Unused*: Este campo no es utilizado por este mensaje.

*Result*: Este mensaje devolverá uno, si tiene éxito; en caso contrario, devolverá cero.

**Véase además**

*WM\_CLEAR*, *WM\_COPY*, *WM\_CUT*, *WM\_PASTE*

**WM\_VKEYTOITEM****Sintaxis**

```
TWMVKeyToItem = record
    Msg: Cardinal;           {identificador del mensaje}
    Key: Word;               {código de la tecla virtual}
    CaretPos: Word;          {posición del cursor de edición}
    ListBox: HWND;           {manejador del cuadro de lista}
    Result: Longint;          {devuelve -1 ó -2 si fue manejado}
end;
```

**Descripción**

Un cuadro de lista con el estilo *LBS\_WANTKEYBOARDINPUT* envía un mensaje *WM\_VKEYTOITEM* a su propietaria en respuesta a un mensaje *WM\_KEYDOWN*.

**Campos**

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_VKEYTOITEM*.

*Key*: El código de la tecla virtual de la tecla que generó el mensaje *WM\_KEYDOWN*.

*CaretPos*: La posición del cursor de edición en el cuadro de lista.

*ListBox*: Manejador del cuadro de lista.

*Result*: Este mensaje debe devolver -2 para indicar que el mensaje fue manejado y que ningún procesamiento posterior por parte del cuadro de lista es necesario. Un valor de retorno de -1 indica que el cuadro de lista debe ejecutar su acción por defecto para la tecla indicada. Un valor cero o mayor especifica el índice de base cero de un elemento

en el cuadro de lista e indica al cuadro de lista que debe ejecutar su acción por defecto para la tecla indicada sobre el elemento especificado.

Véase además

*DefWindowProc*, *WM\_CHARTOITEM*, *WM\_KEYDOWN*

## WM\_VSCROLL

### Sintaxis

```
TWMVScroll = record
    Msg: Cardinal;           {identificador del mensaje}
    ScrollCode: Smallint;    {código del desplazamiento solicitado}
    Pos: Smallint;           {posición del botón de la barra de desplazamiento}
    Scroll bar: HWND;        {manejador de la barra de desplazamiento}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### Descripción

El mensaje *WM\_VSCROLL* es enviado a una ventana cuando se produce un evento de desplazamiento vertical, si la ventana tiene una barra de desplazamiento vertical estándar. También es enviado cuando un desplazamiento ocurre en un control de la barra de desplazamiento. Si la aplicación cambia la posición de los datos en una ventana como resultado de un desplazamiento vertical, debe reinicializar la posición del botón de la barra de desplazamiento llamando a la función *SetScrollPos*. Los mensajes *WM\_VSCROLL* y *WM\_HSCROLL* tienen valores de 16 bits para posiciones de desplazamiento, restringiendo la posición máxima a 65.535.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_VSCROLL*.

*ScrollCode*: Especifica el tipo de desplazamiento solicitado como resultado del evento de desplazamiento. Este campo puede contener un valor de la Tabla A-50.

*Pos*: Si el campo *ScrollCode* contiene *SB\_THUMBPOSITION* o *SB\_THUMBTRACK*, este campo especifica la posición actual del botón de la barra de desplazamiento. En caso contrario, este campo no es usado.

*Scroll bar*: Manejador de un control de barra de desplazamiento, si es un control de barra de desplazamiento el que está enviando el mensaje *WM\_VSCROLL*. En caso contrario, este campo no es usado.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_HSCROLL*

**Tabla A-50: Valores de *TWMVScroll.ScrollCode***

Valor	Descripción
SB_BOTTOM	Indica un desplazamiento total hacia abajo.
SB_ENDSCROLL	Indica el final de la operación de desplazamiento.
SB_LINEDOWN	Indica un desplazamiento hacia abajo de una unidad.
SB_LINEUP	Indica un desplazamiento hacia arriba de una unidad.
SB_PAGEDOWN	Indica un desplazamiento hacia abajo del ancho de la ventana.
SB_PAGEUP	Indica un desplazamiento hacia arriba del ancho de la ventana.
SB_THUMBPOSITION	Indica un desplazamiento a la posición absoluta especificada en el campo Pos.
SB_THUMBTRACK	Arrastra el botón de la barra de desplazamiento a la posición indicada por el campo Pos. Esto es normalmente utilizado para ofrecer retroalimentación.
SB_TOP	Indica un desplazamiento total hacia arriba.

## ***WM\_VSCROLLCLIPBOARD***

### *Sintaxis*

```
TWMVScrollClipboard = record
    Msg: Cardinal;           {identificador del mensaje}
    Viewer: HWND;           {manejador del visualizador del portapapeles}
    ScrollCode: Word;       {código del desplazamiento solicitado}
    ThumbPos: Word;         {posición del botón de la barra de desplazamiento}
    Result: Longint;        {devuelve cero (0) si fue manejado}
end;
```

### *Descripción*

El mensaje *WM\_VSCROLLCLIPBOARD* es enviado por una ventana visualizadora del portapapeles a la propietaria del portapapeles, cuando se produce un evento en la barra de desplazamiento vertical del visualizador y el portapapeles contiene datos en formato *CF\_OWNERDISPLAY*. La propietaria del portapapeles tiene que desplazar la imagen y entonces reinicializar el valor de la barra de desplazamiento vertical.

### *Campos*

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_VSCROLLCLIPBOARD*.

*Viewer*: Especifica el manejador de la ventana visualizadora del portapapeles.

*ScrollCode*: Especifica el tipo de desplazamiento requerido, como resultado de un desplazamiento. Este campo puede contener un valor de la Tabla A-51.

*ThumbPos*: Si el campo *ScrollCode* contiene *SB\_THUMBPOSITION*, este campo especifica la posición actual del botón de la barra de desplazamiento. En caso contrario, este campo no es usado.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

Véase además

*WM\_HSCROLLCLIPBOARD*

**Tabla A-51: Valores de *TWMVScrollClipboard.ScrollCode***

Valor	Descripción
SB_BOTTOM	Indica un desplazamiento total hacia abajo.
SB_ENDSCROLL	Indica el final de la operación de desplazamiento.
SB_LINEDOWN	Indica un desplazamiento hacia abajo de una unidad.
SB_LINEUP	Indica un desplazamiento hacia arriba de una unidad.
SB_PAGEDOWN	Indica un desplazamiento hacia abajo del ancho de la ventana.
SB_PAGEUP	Indica un desplazamiento hacia arriba del ancho de la ventana.
SB_THUMBPOSITION	Indica un desplazamiento a la posición absoluta especificada en el campo Pos.
SB_TOP	Indica un desplazamiento total hacia arriba.

## ***WM\_WINDOWPOSCHANGED***

### *Sintaxis*

```
TWMWindowPosChanged = record
    Msg: Cardinal;           {identificador del mensaje}
    Unused: Integer;         {no se utiliza}
    WindowPos: PWindowPos;  {puntero a un registro TWindowPos}
    Result: Longint;         {devuelve cero (0) si fue manejado}
end;
```

### *Descripción*

Este mensaje es enviado cuando el tamaño de una ventana, su posición o el orden Z han sido reajustados como resultado de una llamada a una función de movimiento o posicionamiento de ventana, como *SetWindowPos* o *EndDeferWindowPos*. Los mensajes *WM\_SIZE* y *WM\_MOVE* no serán enviados si la función *DefWindowProc* no es llamada por una ventana en respuesta a este mensaje.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_WINDOWPOSCHANGED*.

*Unused*: Este campo no es utilizado por este mensaje.

*WindowPos*: Especifica un puntero a un registro *TWindowPos* que contiene la nueva información sobre el tamaño y posición de la ventana. El registro *TWindowPos* se define como:

```
TWindowPos = packed record
    hwnd: HWND;           {manejador de ventana}
    hwndInsertAfter: HWND; {manejador de ventana o indicador de posición}
    x: Integer;            {posición horizontal}
    y: Integer;            {posición vertical}
    cx: Integer;           {ancho de la ventana}
    cy: Integer;           {altura de la ventana}
    flags: UINT            {opciones de tamaño y posición}
end;
```

Vea el mensaje *WM\_NCCALCSIZE* para una descripción de este registro.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*DefWindowProc*, *EndDeferWindowPos*, *SetWindowPos*, *WM\_MOVE*,  
*WM\_NCCALCSIZE*, *WM\_SIZE*, *WM\_WINDOWPOSCHANGING*

## WM\_WINDOWPOSCHANGING

### Sintaxis

```
TWMWindowPosChanging = record
    Msg: Cardinal;         {identificador del mensaje}
    Unused: Integer;       {no se utiliza}
    WindowPos: PWindowPos; {puntero a un registro TWindowPos}
    Result: Longint;       {devuelve cero (0) si fue manejado}
end;
```

### Descripción

Este mensaje es enviado cuando el tamaño de una ventana, su posición o el orden Z van a ser reajustados como resultado de una llamada a una función de movimiento o posicionamiento de ventana, como *SetWindowPos* o *EndDeferWindowPos*. Si la ventana tiene los estilos *WS\_OVERLAPPED* o *WS\_THICKFRAME* y envía este mensaje a la función *DefWindowProc*, un mensaje *WM\_GETMINMAXINFO* será enviado a la ventana. Este valida el nuevo tamaño y posición de la ventana, de manera

que los estilos de clase de ventana *CS\_BYTEALIGNCLIENT* y *CS\_BYTEALIGNWINDOW* sean respetados.

### Campos

*Msg*: El identificador del mensaje. Este campo contiene la constante de identificador de mensaje *WM\_WINDOWPOSCHANGED*.

*Unused*: Este campo no es utilizado por este mensaje.

*WindowPos*: Especifica un puntero a un registro *TWindowPos* que contiene la nueva información sobre el tamaño y posición de la ventana. El registro *TWindowPos* se define como:

*TWindowPos* = **packed record**

<i>hwnd</i> : HWND;	{manejador de ventana}
<i>hwndInsertAfter</i> : HWND;	{manejador de ventana o indicador de posición}
<i>x</i> : Integer;	{posición horizontal}
<i>y</i> : Integer;	{posición vertical}
<i>cx</i> : Integer;	{ancho de la ventana}
<i>cy</i> : Integer;	{altura de la ventana}
<i>flags</i> : UINT	{opciones de tamaño y posición}

**end;**

Vea *WM\_NCCALCSIZE* para una descripción de este registro.

*Result*: Si la aplicación maneja el mensaje, deberá asignar cero a este campo.

### Véase además

*DefWindowProc*, *EndDeferWindowPos*, *SetWindowPos*, *WM\_GETMINMAXINFO*, *WM\_MOVE*, *WM\_NCCALCSIZE*, *WM\_SIZE*, *WM\_WINDOWPOSCHANGING*



## Apéndice B

## Códigos de operaciones de barrido terciarias

## B

## Apéndice

Código ROP	Operación booleana
\$00000042	El resultado es todo negro
\$00010289	<b>NOT</b> (brocha <b>OR</b> origen <b>OR</b> destino)
\$00020C89	<b>NOT</b> (brocha <b>OR</b> origen) <b>AND</b> destino
\$000300AA	<b>NOT</b> (brocha <b>OR</b> origen)
\$00040C88	<b>NOT</b> (brocha <b>OR</b> destino) <b>AND</b> origen
\$000500A9	<b>NOT</b> (brocha <b>OR</b> destino)
\$00060865	<b>NOT</b> (brocha <b>OR NOT</b> (origen <b>XOR</b> destino))
\$000702C5	<b>NOT</b> (brocha <b>OR</b> (origen <b>AND</b> destino))
\$00080F08	<b>NOT</b> brocha <b>AND</b> origen <b>AND</b> destino
\$00090245	<b>NOT</b> (brocha <b>OR</b> (origen <b>XOR</b> destino))
\$000A0329	<b>NOT</b> brocha <b>AND</b> destino
\$000B0B2A	<b>NOT</b> (brocha <b>OR</b> (origen <b>AND NOT</b> destino))
\$000C0324	<b>NOT</b> brocha <b>AND</b> origen
\$000D0B25	<b>NOT</b> (brocha <b>OR</b> ( <b>NOT</b> origen <b>AND</b> destino))
\$000E08A5	<b>NOT</b> (brocha <b>OR NOT</b> (origen <b>OR</b> destino))
\$000F0001	<b>NOT</b> brocha
\$00100C85	brocha <b>AND NOT</b> (origen <b>OR</b> destino)
\$001100A6	<b>NOT</b> (origen <b>OR</b> destino)
\$00120868	<b>NOT</b> (origen <b>OR NOT</b> (brocha <b>XOR</b> destino))
\$001302C8	<b>NOT</b> (origen <b>OR</b> (brocha <b>AND</b> destino))
\$00140869	<b>NOT</b> (destino <b>OR NOT</b> (brocha <b>XOR</b> origen))
\$001502C9	<b>NOT</b> (destino <b>OR</b> (brocha <b>AND</b> origen))
\$00165CCA	brocha <b>XOR</b> (origen <b>XOR</b> (destino <b>AND NOT</b> (brocha <b>AND</b> origen)))
\$00171D54	<b>NOT</b> (origen <b>XOR</b> ((origen <b>XOR</b> brocha) <b>AND</b> (origen <b>XOR</b> destino)))
\$00180D59	(brocha <b>XOR</b> origen) <b>AND</b> (brocha <b>XOR</b> destino)

Código ROP	Operación booleana
\$00191CC8	<b>NOT</b> (origen <b>XOR</b> (destino <b>AND NOT</b> (brocha <b>AND</b> origen)))
\$001A06C5	brocha <b>XOR</b> (destino <b>OR</b> (origen <b>AND</b> brocha))
\$001B0768	<b>NOT</b> (origen <b>XOR</b> (destino <b>AND</b> (brocha <b>XOR</b> origen)))
\$001C06CA	brocha <b>XOR</b> (origen <b>OR</b> (brocha <b>AND</b> destino))
\$001D0766	<b>NOT</b> (destino <b>XOR</b> (origen <b>AND</b> (brocha <b>XOR</b> destino)))
\$001E01A5	brocha <b>XOR</b> (origen <b>OR</b> destino)
\$001F0385	<b>NOT</b> (brocha <b>AND</b> (origen <b>OR</b> destino))
\$00200F09	brocha <b>AND NOT</b> origen <b>AND</b> destino
\$00210248	<b>NOT</b> (origen <b>OR</b> (brocha <b>XOR</b> destino))
\$00220326	<b>NOT</b> origen <b>AND</b> destino
\$00230B24	<b>NOT</b> (origen <b>OR</b> (brocha <b>AND NOT</b> destino))
\$00240D55	(origen <b>XOR</b> brocha) <b>AND</b> (origen <b>XOR</b> destino)
\$00251CC5	<b>NOT</b> (brocha <b>XOR</b> (destino <b>AND NOT</b> (origen <b>AND</b> brocha)))
\$002606C8	origen <b>XOR</b> (destino <b>OR</b> (brocha <b>AND</b> origen))
\$00271868	origen <b>XOR</b> (destino <b>OR NOT</b> (brocha <b>XOR</b> origen))
\$00280369	destino <b>AND</b> (brocha <b>XOR</b> origen)
\$002916CA	<b>NOT</b> (brocha <b>XOR</b> (origen <b>XOR</b> (destino <b>OR</b> (brocha <b>AND</b> origen))))
\$002A0CC9	destino <b>AND NOT</b> (brocha <b>AND</b> origen)
\$002B1D58	<b>NOT</b> (origen <b>XOR</b> ((origen <b>XOR</b> brocha) <b>AND</b> (brocha <b>AND</b> destino)))
\$002C0784	origen <b>XOR</b> (brocha <b>AND</b> (origen <b>OR</b> destino))
\$002D060A	brocha <b>XOR</b> (origen <b>OR NOT</b> destino)
\$002E064A	brocha <b>XOR</b> (origen <b>OR</b> (brocha <b>XOR</b> destino))
\$002F0E2A	<b>NOT</b> (brocha <b>AND</b> (origen <b>OR NOT</b> destino))
\$0030032A	brocha <b>AND NOT</b> origen
\$00310B28	<b>NOT</b> (origen <b>OR</b> ( <b>NOT</b> brocha <b>AND</b> destino))
\$00320688	origen <b>XOR</b> (brocha <b>OR</b> origen <b>OR</b> destino)
\$00330008	<b>NOT</b> origen
\$003406C4	origen <b>XOR</b> (brocha <b>OR</b> (origen <b>AND</b> destino))
\$00351864	origen <b>XOR</b> (brocha <b>OR NOT</b> (origen <b>XOR</b> destino))
\$003601A8	origen <b>XOR</b> (brocha <b>OR</b> destino)
\$00370388	<b>NOT</b> (origen <b>AND</b> (brocha <b>OR</b> destino))
\$0038078A	brocha <b>XOR</b> (origen <b>AND</b> (brocha <b>OR</b> destino))
\$00390604	origen <b>XOR</b> (brocha <b>OR NOT</b> destino)
\$003A0644	origen <b>XOR</b> (brocha <b>XOR</b> (origen <b>XOR</b> destino))
\$003B0E24	<b>NOT</b> (origen <b>AND</b> (brocha <b>OR NOT</b> destino))
\$003C004A	brocha <b>XOR</b> origen
\$003D18A4	origen <b>XOR</b> (brocha <b>OR NOT</b> (origen <b>OR</b> destino))
\$003E1B24	origen <b>XOR</b> (brocha <b>OR</b> ( <b>NOT</b> origen <b>AND</b> destino))
\$003F00EA	<b>NOT</b> (brocha <b>AND</b> origen)

Código ROP	Operación booleana
\$00400F0A	brocha <b>AND</b> origen <b>AND NOT</b> destino
\$00410249	<b>NOT</b> (destino <b>OR</b> (brocha <b>XOR</b> origen))
\$00420D5D	(origen <b>XOR</b> destino) <b>AND</b> (brocha <b>XOR</b> destino)
\$00431CC4	<b>NOT</b> (origen <b>XOR</b> (brocha <b>AND NOT</b> (origen <b>AND</b> destino)))
\$00440328	origen <b>AND NOT</b> destino
\$00450B29	<b>NOT</b> (destino <b>OR</b> (brocha <b>AND NOT</b> origen))
\$004606C6	destino <b>XOR</b> (origen <b>OR</b> (brocha <b>AND</b> destino))
\$0047076A	<b>NOT</b> (brocha <b>XOR</b> (origen <b>AND</b> (brocha <b>XOR</b> destino)))
\$00480368	origen <b>AND</b> (brocha <b>XOR</b> destino)
\$004916C5	<b>NOT</b> (brocha <b>XOR</b> (destino <b>XOR</b> (origen <b>OR</b> (brocha <b>AND</b> destino))))
\$004A0789	destino <b>XOR</b> (brocha <b>AND</b> (origen <b>OR</b> destino))
\$004B0605	brocha <b>XOR</b> ( <b>NOT</b> origen <b>OR</b> destino)
\$004C0CC8	origen <b>AND NOT</b> (brocha <b>AND</b> destino)
\$004D1954	<b>NOT</b> (origen <b>XOR</b> ((brocha <b>XOR</b> origen) <b>OR</b> (origen <b>XOR</b> destino)))
\$004E0645	brocha <b>XOR</b> (destino <b>OR</b> (brocha <b>XOR</b> origen))
\$004F0E25	<b>NOT</b> (brocha <b>AND</b> ( <b>NOT</b> origen <b>OR</b> destino))
\$00500325	brocha <b>AND NOT</b> destino
\$00510B26	<b>NOT</b> (destino <b>OR</b> ( <b>NOT</b> brocha <b>AND</b> origen))
\$005206C9	destino <b>XOR</b> (brocha <b>OR</b> (origen <b>AND</b> destino))
\$00530764	<b>NOT</b> (origen <b>XOR</b> (brocha <b>AND</b> (origen <b>XOR</b> destino)))
\$005408A9	<b>NOT</b> (destino <b>OR NOT</b> (brocha <b>OR</b> origen))
\$00550009	<b>NOT</b> destino
\$005601A9	destino <b>XOR</b> (brocha <b>OR</b> origen)
\$00570389	<b>NOT</b> (destino <b>AND</b> (brocha <b>OR</b> origen))
\$00580785	brocha <b>XOR</b> (destino <b>AND</b> (brocha <b>OR</b> origen))
\$00590609	destino <b>XOR</b> (brocha <b>OR NOT</b> origen)
\$005A0049	brocha <b>XOR</b> destino
\$005B18A9	destino <b>XOR</b> (brocha <b>OR NOT</b> (origen <b>OR</b> destino))
\$005C0649	destino <b>XOR</b> (brocha <b>OR</b> (origen <b>XOR</b> destino))
\$005D0E29	<b>NOT</b> (destino <b>AND</b> (brocha <b>OR NOT</b> origen))
\$005E1B29	destino <b>XOR</b> (brocha <b>OR</b> (origen <b>AND NOT</b> destino))
\$005F00E9	<b>NOT</b> (brocha <b>AND</b> destino)
\$00600365	brocha <b>AND</b> (origen <b>XOR</b> destino)
\$006116C6	<b>NOT</b> (destino <b>XOR</b> (origen <b>XOR</b> (brocha <b>OR</b> (origen <b>AND</b> destino))))
\$00620786	destino <b>XOR</b> (origen <b>AND</b> (brocha <b>OR</b> destino))
\$00630608	origen <b>XOR</b> ( <b>NOT</b> brocha <b>OR</b> destino)
\$00640788	origen <b>XOR</b> (destino <b>AND</b> (brocha <b>OR</b> origen))
\$00650606	destino <b>XOR</b> ( <b>NOT</b> brocha <b>OR</b> origen)
\$00660046	origen <b>XOR</b> destino

Código ROP	Operación booleana
\$006718A8	origen <b>XOR</b> (destino <b>OR NOT</b> (brocha <b>OR</b> origen))
\$006858A6	<b>NOT</b> (destino <b>XOR</b> (origen <b>XOR</b> (brocha <b>OR NOT</b> (origen <b>OR</b> destino))))
\$00690145	<b>NOT</b> (brocha <b>XOR</b> (origen <b>XOR</b> destino))
\$006A01E9	destino <b>XOR</b> (brocha <b>AND</b> origen)
\$006B178A	<b>NOT</b> (brocha <b>XOR</b> (origen <b>XOR</b> (destino <b>AND</b> (origen <b>OR</b> brocha))))
\$006C01E8	origen <b>XOR</b> (brocha <b>AND</b> destino)
\$006D1785	<b>NOT</b> (brocha <b>XOR</b> (destino <b>XOR</b> (origen <b>AND</b> (brocha <b>OR</b> destino))))
\$006E1E28	origen <b>XOR</b> (destino <b>AND</b> (brocha <b>OR NOT</b> origen))
\$006F0C65	<b>NOT</b> (brocha <b>AND NOT</b> (origen <b>XOR</b> destino))
\$00700CC5	brocha <b>AND NOT</b> (origen <b>AND</b> destino)
\$00711D5C	<b>NOT</b> (origen <b>XOR</b> ((origen <b>XOR</b> destino) <b>AND</b> (brocha <b>XOR</b> destino)))
\$00720648	origen <b>XOR</b> (destino <b>OR</b> (brocha <b>XOR</b> origen))
\$00730E28	<b>NOT</b> (origen <b>AND</b> ( <b>NOT</b> brocha <b>OR</b> destino))
\$00740646	destino <b>XOR</b> (origen <b>OR</b> (brocha <b>XOR</b> destino))
\$00750E26	<b>NOT</b> (destino <b>AND</b> ( <b>NOT</b> brocha <b>OR</b> origen))
\$00761B28	origen <b>XOR</b> (destino <b>OR</b> (brocha <b>AND NOT</b> origen))
\$007700E6	<b>NOT</b> (origen <b>AND</b> destino)
\$007801E5	brocha <b>XOR</b> (origen <b>AND</b> destino)
\$00791786	<b>NOT</b> (destino <b>XOR</b> (origen <b>XOR</b> (brocha <b>AND</b> (origen <b>OR</b> destino))))
\$007A1E29	destino <b>XOR</b> (brocha <b>AND</b> (origen <b>OR NOT</b> destino))
\$007B0C68	<b>NOT</b> (origen <b>AND NOT</b> (brocha <b>XOR</b> destino))
\$007C1E24	origen <b>XOR</b> (brocha <b>AND</b> ( <b>NOT</b> origen <b>OR</b> destino))
\$007D0C69	<b>NOT</b> (destino <b>AND NOT</b> (origen <b>XOR</b> brocha))
\$007E0955	(brocha <b>XOR</b> origen) <b>OR</b> (origen <b>XOR</b> destino)
\$007F03C9	<b>NOT</b> (brocha <b>AND</b> origen <b>AND</b> destino)
\$008003E9	brocha <b>AND</b> origen <b>AND</b> destino
\$00810975	<b>NOT</b> ((brocha <b>XOR</b> origen) <b>OR</b> (origen <b>XOR</b> destino))
\$00820C49	<b>NOT</b> (brocha <b>XOR</b> origen) <b>AND</b> destino
\$00831E04	<b>NOT</b> (origen <b>XOR</b> (brocha <b>AND</b> ( <b>NOT</b> origen <b>OR</b> destino)))
\$00840C48	origen <b>AND NOT</b> (brocha <b>XOR</b> destino)
\$00851E05	<b>NOT</b> (brocha <b>XOR</b> (destino <b>AND</b> ( <b>NOT</b> brocha <b>OR</b> origen)))
\$008617A6	destino <b>XOR</b> (origen <b>XOR</b> (brocha <b>AND</b> (origen <b>OR</b> destino)))
\$008701C5	<b>NOT</b> (brocha <b>XOR</b> (origen <b>AND</b> destino))
\$00800C6	origen <b>AND</b> destino
\$00891B08	<b>NOT</b> (origen <b>XOR</b> (destino <b>OR</b> (brocha <b>AND NOT</b> origen)))
\$008A0E06	( <b>NOT</b> brocha <b>OR</b> origen) <b>AND</b> destino
\$008B0666	<b>NOT</b> (destino <b>XOR</b> (origen <b>OR</b> (brocha <b>OR</b> destino)))
\$008C0E08	origen <b>AND</b> ( <b>NOT</b> brocha <b>OR</b> destino)

Código ROP	Operación booleana
\$008D0668	<b>NOT</b> (origen <b>XOR</b> (destino <b>OR</b> (brocha <b>XOR</b> origen)))
\$008E1D7C	origen <b>XOR</b> (origen <b>XOR</b> destino <b>AND</b> (brocha <b>XOR</b> destino))
\$008F0CE5	<b>NOT</b> (brocha <b>AND NOT</b> (origen <b>AND</b> destino))
\$00900C45	brocha <b>AND NOT</b> (origen <b>XOR</b> destino)
\$00911E08	<b>NOT</b> (origen <b>XOR</b> (destino <b>AND</b> (brocha <b>OR NOT</b> origen)))
\$009217A9	destino <b>XOR</b> (brocha <b>XOR</b> (origen <b>AND</b> (brocha <b>OR</b> destino)))
\$009301C4	<b>NOT</b> (origen <b>XOR</b> (brocha <b>AND</b> destino))
\$009417AA	brocha <b>XOR</b> (origen <b>XOR</b> (destino <b>AND</b> (brocha <b>OR</b> origen)))
\$009501C9	<b>NOT</b> (destino <b>XOR</b> (brocha <b>AND</b> origen))
\$00960169	brocha <b>XOR</b> origen <b>XOR</b> destino
\$0097588A	brocha <b>XOR</b> (origen <b>XOR</b> (destino <b>OR NOT</b> (brocha <b>OR</b> origen)))
\$00981888	<b>NOT</b> (origen <b>XOR</b> (destino <b>OR NOT</b> (brocha <b>OR</b> origen)))
\$00990066	<b>NOT</b> (origen <b>XOR</b> destino)
\$009A0709	(brocha <b>AND NOT</b> origen) <b>XOR</b> destino
\$009B07A8	<b>NOT</b> (origen <b>XOR</b> (destino <b>AND</b> (brocha <b>OR</b> origen)))
\$009C0704	origen <b>XOR</b> (brocha <b>AND NOT</b> destino)
\$009D07A6	<b>NOT</b> (destino <b>XOR</b> (origen <b>AND</b> (brocha <b>OR</b> destino)))
\$009E16E6	(origen <b>XOR</b> (brocha <b>OR</b> (origen <b>AND</b> destino))) <b>XOR</b> destino
\$009F0345	<b>NOT</b> (brocha <b>AND</b> (origen <b>XOR</b> destino))
\$00A000C9	brocha <b>AND</b> destino
\$00A11B05	<b>NOT</b> (brocha <b>XOR</b> (destino <b>OR</b> ( <b>NOT</b> brocha <b>AND</b> origen)))
\$00A20E09	(brocha <b>OR NOT</b> origen) <b>AND</b> destino
\$00A30699	<b>NOT</b> (destino <b>XOR</b> (brocha <b>OR</b> (origen <b>XOR</b> destino)))
\$00A41885	<b>NOT</b> (brocha <b>XOR</b> (destino <b>OR NOT</b> (brocha <b>OR</b> origen)))
\$00A50065	<b>NOT</b> (brocha <b>XOR</b> destino)
\$00A60706	( <b>NOT</b> brocha <b>AND</b> origen) <b>XOR</b> destino
\$00A707A5	<b>NOT</b> (brocha <b>XOR</b> (destino <b>AND</b> (brocha <b>OR</b> origen)))
\$00A803A9	(brocha <b>OR</b> origen) <b>AND</b> destino
\$00A90189	<b>NOT</b> ((brocha <b>OR</b> origen) <b>XOR</b> destino)
\$00AA0029	destino
\$00AB0889	<b>NOT</b> (brocha <b>OR</b> origen) <b>OR</b> destino
\$00AC0744	origen <b>XOR</b> (brocha <b>AND</b> (origen <b>XOR</b> destino))
\$00AD06E9	<b>NOT</b> (destino <b>XOR</b> (brocha <b>OR</b> (origen <b>AND</b> destino)))
\$00AE0B06	( <b>NOT</b> brocha <b>AND</b> origen) <b>OR</b> destino
\$00AF0229	<b>NOT</b> brocha <b>OR</b> destino
\$00B00E05	brocha <b>AND</b> ( <b>NOT</b> origen <b>OR</b> destino)
\$00B10665	<b>NOT</b> (brocha <b>OR</b> (destino <b>OR</b> (brocha <b>XOR</b> origen)))
\$00B12974	origen <b>XOR</b> ((brocha <b>XOR</b> origen) <b>OR</b> (origen <b>XOR</b> destino))
\$00B03CE8	<b>NOT</b> (origen <b>AND NOT</b> (brocha <b>AND</b> destino))

Código ROP	Operación booleana
\$00B4070A	brocha <b>XOR</b> (origen <b>AND NOT</b> destino)
\$00B507A9	<b>NOT</b> (destino <b>XOR</b> (brocha <b>AND</b> (origen <b>OR</b> destino)))
\$00B616E9	destino <b>XOR</b> (brocha <b>XOR</b> (origen <b>OR</b> (brocha <b>AND</b> destino)))
\$00B70348	<b>NOT</b> (origen <b>AND</b> (brocha <b>XOR</b> destino))
\$00B8074A	brocha <b>XOR</b> (origen <b>AND</b> (brocha <b>XOR</b> destino))
\$00B906E6	<b>NOT</b> (destino <b>XOR</b> (origen <b>OR</b> (brocha <b>AND</b> destino)))
\$00BA0B09	(brocha <b>AND NOT</b> origen) <b>OR</b> destino
\$00BB0226	<b>NOT</b> origen <b>OR</b> destino
\$00BC1CE4	origen <b>XOR</b> (brocha <b>AND NOT</b> (origen <b>AND</b> destino))
\$00BD0D7D	<b>NOT</b> ((brocha <b>XOR</b> destino) <b>AND</b> (origen <b>XOR</b> destino))
\$00BE0269	(brocha <b>XOR</b> origen) <b>OR</b> destino
\$00BF08C9	<b>NOT</b> (brocha <b>AND</b> origen) <b>OR</b> destino
\$00C000CA	brocha <b>AND</b> origen
\$00C11B04	<b>NOT</b> (origen <b>XOR</b> (brocha <b>OR</b> ( <b>NOT</b> origen <b>AND</b> destino)))
\$00C21884	<b>NOT</b> (origen <b>XOR</b> (brocha <b>OR NOT</b> (origen <b>OR</b> destino)))
\$00C3006A	<b>NOT</b> (brocha <b>XOR</b> origen)
\$00C40E04	origen <b>AND</b> (brocha <b>OR NOT</b> destino)
\$00C50664	<b>NOT</b> (origen <b>XOR</b> (brocha <b>OR</b> (origen <b>XOR</b> destino)))
\$00C60708	origen <b>XOR</b> ( <b>NOT</b> brocha <b>AND</b> destino)
\$00C707AA	<b>NOT</b> (brocha <b>XOR</b> (origen <b>AND</b> (brocha <b>OR</b> destino)))
\$00C803A8	origen <b>AND</b> (brocha <b>OR</b> destino)
\$00C90184	<b>NOT</b> (origen <b>XOR</b> (brocha <b>OR</b> destino))
\$00CA0749	destino <b>XOR</b> (brocha <b>AND</b> (origen <b>XOR</b> destino))
\$00CB06E4	<b>NOT</b> (origen <b>XOR</b> (brocha <b>OR</b> (origen <b>AND</b> destino)))
\$00CC0020	origen
\$00CD0888	origen <b>OR NOT</b> (brocha <b>OR</b> destino)
\$00CE0B08	origen <b>OR</b> ( <b>NOT</b> brocha <b>AND</b> destino)
\$00CF0224	origen <b>OR NOT</b> brocha
\$00D00E0A	brocha <b>AND</b> (origen <b>OR NOT</b> destino)
\$00D1066A	<b>NOT</b> (brocha <b>XOR</b> (origen <b>OR</b> (brocha <b>XOR</b> destino)))
\$00D20705	brocha <b>XOR</b> ( <b>NOT</b> origen <b>AND</b> destino)
\$00D307A4	<b>NOT</b> (origen <b>XOR</b> (brocha <b>AND</b> (origen <b>OR</b> destino)))
\$00D41D78	origen <b>XOR</b> (brocha <b>XOR</b> origen <b>AND</b> (brocha <b>XOR</b> destino))
\$00D50CE9	<b>NOT</b> (destino <b>AND NOT</b> (brocha <b>AND</b> origen))
\$00D616EA	brocha <b>XOR</b> (origen <b>XOR</b> (destino <b>OR</b> (brocha <b>AND</b> origen)))
\$00D70349	<b>NOT</b> (destino <b>AND</b> (brocha <b>XOR</b> origen))
\$00D80745	brocha <b>XOR</b> (destino <b>AND</b> (brocha <b>XOR</b> origen))
\$00D906E8	<b>NOT</b> (origen <b>XOR</b> (destino <b>OR</b> (brocha <b>AND</b> origen)))
\$00DA1CE9	destino <b>XOR</b> (brocha <b>AND NOT</b> (origen <b>XOR</b> destino))

Código ROP	Operación booleana
\$00DB0D75	<b>NOT</b> ((brocha <b>XOR</b> origen) <b>AND</b> (origen <b>XOR</b> destino))
\$00DC0B04	origen <b>OR</b> (brocha <b>AND NOT</b> destino)
\$00DD0228	origen <b>OR NOT</b> destino
\$00DE0268	origen <b>OR</b> (brocha <b>XOR</b> destino)
\$00DF08C8	origen <b>OR NOT</b> (brocha <b>AND</b> destino)
\$00E003A5	brocha <b>AND</b> (destino <b>OR</b> origen)
\$00E10185	<b>NOT</b> (brocha <b>XOR</b> (origen <b>OR</b> destino))
\$00E20746	destino <b>XOR</b> (origen <b>AND</b> (brocha <b>XOR</b> destino))
\$00E306EA	<b>NOT</b> (brocha <b>XOR</b> (origen <b>OR</b> (brocha <b>AND</b> destino)))
\$00E40748	origen <b>XOR</b> (destino <b>AND</b> (brocha <b>XOR</b> origen))
\$00E506E5	<b>NOT</b> (brocha <b>XOR</b> (destino <b>OR</b> (brocha <b>AND</b> origen)))
\$00E61CE8	origen <b>XOR</b> (destino <b>AND NOT</b> (brocha <b>AND</b> origen))
\$00E70D79	<b>NOT</b> ((brocha <b>XOR</b> origen) <b>AND</b> (brocha <b>XOR</b> destino))
\$00E81D74	origen <b>XOR</b> ((brocha <b>XOR</b> origen) <b>AND</b> (origen <b>XOR</b> destino))
\$00E95CE6	<b>NOT</b> (destino <b>XOR</b> (origen <b>XOR</b> (brocha <b>AND NOT</b> (origen <b>AND</b> destino))))
\$00EA02E9	(brocha <b>AND</b> origen) <b>OR</b> destino
\$00EB0849	<b>NOT</b> (brocha <b>XOR</b> origen) <b>OR</b> destino
\$00EC02E8	origen <b>OR</b> (brocha <b>AND</b> destino)
\$00ED0848	origen <b>OR NOT</b> (brocha <b>XOR</b> destino)
\$00EE0086	origen <b>OR</b> destino
\$00EF0A08	<b>NOT</b> brocha <b>OR</b> origen <b>OR</b> destino
\$00F00021	brocha
\$00F10885	brocha <b>OR NOT</b> (origen <b>OR</b> destino)
\$00F20B05	brocha <b>OR</b> ( <b>NOT</b> origen <b>AND</b> destino)
\$00F3022A	brocha <b>OR NOT</b> origen
\$00F40B0A	brocha <b>OR</b> (origen <b>AND NOT</b> destino)
\$00F50225	brocha <b>OR NOT</b> destino
\$00F60265	brocha <b>OR</b> (origen <b>XOR</b> destino)
\$00F708C5	brocha <b>OR NOT</b> (origen <b>AND</b> destino)
\$00F802E5	brocha <b>OR</b> (origen <b>AND</b> destino)
\$00F90845	brocha <b>OR NOT</b> (origen <b>XOR</b> destino)
\$00FA0089	brocha <b>OR</b> destino
\$00FB0A09	brocha <b>OR NOT</b> origen <b>OR</b> destino
\$00FC008A	brocha <b>OR</b> origen
\$00FD0A0A	brocha <b>OR</b> origen <b>OR NOT</b> destino
\$00FE02A9	brocha <b>OR</b> origen <b>OR</b> destino
\$00FF0062	El resultado es todo blanco



## Apéndice C

# Conjunto de caracteres ASCII

Dec.	Hex.	Car.	Descripción
0	00	NULL	Nulo
1	01	☺	Comienzo de encabezamiento
2	02	☹	Comienzo de texto
3	03	♥	Fin de texto
4	04	♦	Fin de la transmisión
5	05	♣	Pregunta
6	06	♠	Reconocimiento
7	07	•	Campana
8	08	▣	Retroceso
9	09	○	Tabulación horizontal
10	0A	◼	Nueva línea
11	0B	♂	Tabulación vertical
12	0C	♀	Nueva página
13	0D	♪	Retorno
14	0E	♫	Deplazamiento hacia afuera (Shift out)
15	0F	☼	Deplazamiento hacia dentro (Shift in)
16	10	▶	Escape de enlace de datos
17	11	◀	Control de dispositivo 1
18	12	↕	Control de dispositivo 2
19	13	!!	Control de dispositivo 3
20	14	¶	Control de dispositivo 4
21	15	§	Reconocimiento negativo
22	16	—	Tiempo ocioso síncrono
23	17	↕	Fin de la transmisión de bloque
24	18	↑	Cancelar
25	19	↓	Fin de medio
26	1A	→	Sustituir
27	1B	←	Escape

Dec.	Hex.	Car.	Descripción
28	1C	L	Separador de fichero
29	1D	↔	Separador de grupo
30	1E	▲	Separador de registro
31	1F	▼	Separador de unidad

Dec.	Hex.	Car.	Dec.	Hex.	Car.	Dec.	Hex.	Car.
32	20		65	41	A	98	62	b
33	21	!	66	42	B	99	63	c
34	22	"	67	43	C	100	64	d
35	23	#	68	44	D	101	65	e
36	24	\$	69	45	E	102	66	f
37	25	&	70	46	F	103	67	g
38	26	'	71	47	G	104	68	h
39	27	(	72	48	H	105	69	i
40	28	(	73	49	I	106	6A	j
41	29	)	74	4A	J	107	6B	k
42	2A	*	75	4B	K	108	6C	l
43	2B	+	76	4C	L	109	6D	m
44	2C	,	77	4D	M	110	6E	n
45	2D	-	78	4E	N	111	6F	o
46	2E	.	79	4F	O	112	70	p
47	2F	/	80	50	P	113	71	q
48	30	0	81	51	Q	114	72	r
49	31	1	82	52	R	115	73	s
50	32	2	83	53	S	116	74	t
51	33	3	84	54	T	117	75	u
52	34	4	85	55	U	118	76	v
53	35	5	86	56	V	119	77	w
54	36	6	87	57	W	120	78	x
55	37	7	88	58	X	121	79	y
56	38	8	89	59	Y	122	7A	z
57	39	9	90	5A	Z	123	7B	{
58	3A	:	91	5B	[	124	7C	
59	3B	;	92	5C	\	125	7D	}
60	3C	<	93	5D	]	126	7E	~
61	3D	=	94	5E	^	127	7F	^
62	3E	>	95	5F	_	128	80	Ç
63	3F	?	96	60	`	129	81	ù
64	40	@	97	61	a	130	82	é

Dec.	Hex.	Car.	Dec.	Hex.	Car.	Dec.	Hex.	Car.
131	83	â	170	AA	¬	209	D1	ƒ
132	84	ä	171	AB	½	210	D2	π
133	85	à	172	AC	¼	211	D3	ℓ
134	86	å	173	AD	ı	212	D4	ℓ
135	87	ç	174	AE	«	213	D5	ƒ
136	88	ê	175	AF	»	214	D6	π
137	89	ë	176	B0	☐	215	D7	†
138	8A	è	177	B1	☐	216	D8	‡
139	8B	ï	178	B2	☐	217	D9	ℓ
140	8C	î	179	B3		218	DA	ℓ
141	8D	ì	180	B4	†	219	DB	■
142	8E	Ä	181	B5	‡	220	DC	■
143	8F	Å	182	B6	‡	221	DD	■
144	90	É	183	B7	π	222	DE	■
145	91	æ	184	B8	ƒ	223	DF	■
146	92	Æ	185	B9	‡	224	E0	α
147	93	ô	186	BA		225	E1	β
148	94	ö	187	BB	¶	226	E2	Γ
149	95	ò	188	BC	¶	227	E3	π
150	96	û	189	BD	¶	228	E4	Σ
151	97	ù	190	BE	¶	229	E5	σ
152	98	ÿ	191	BF	¶	230	E6	μ
153	99	Ö	192	C0	⊥	231	E7	τ
154	9A	Ü	193	C1	⊥	232	E8	Φ
155	9B	Ç	194	C2	⊥	233	E9	⊙
156	9C	£	195	C3	⊥	234	EA	Ω
157	9D	¥	196	C4	—	235	EB	δ
158	9E	ℳ	197	C5	+	236	EC	∞
159	9F	ƒ	198	C6	†	237	ED	φ
160	A0	á	199	C7	‡	238	EE	ε
161	A1	í	200	C8	ℓ	239	EF	∩
162	A2	ó	201	C9	ℓ	240	F0	≡
163	A3	ú	202	CA	≡	241	F1	±
164	A4	ñ	203	CB	π	242	F2	≥
165	A5	Ñ	204	CC	‡	243	F3	≤
166	A6	ä	205	CD	=	244	F4	
167	A7	ö	206	CE	‡	245	F5	
168	A8	ç	207	CF	≡	246	F6	÷
169	A9	ℓ	208	D0	≡	247	F7	≈

## 894 ■ Apéndice C

Dec.	Hex.	Car.	Dec.	Hex.	Car.	Dec.	Hex.	Car.
248	F8	°	251	FB	√	254	FE	■
249	F9	•	252	FC	²	255	FF	
250	FA	•	253	FD	²			

## Apéndice D

# Tabla de códigos de teclas virtuales

Código de tecla virtual	Dec.	Hex.	Descripción
VK_LBUTTON	1	\$1	Botón izquierdo del ratón.
VK_RBUTTON	2	\$2	Botón derecho del ratón.
VK_CANCEL	3	\$3	Combinación de teclas Ctrl+Break.
VK_MBUTTON	4	\$4	Botón central del ratón.
VK_BACK	8	\$8	Retroceso.
VK_TAB	9	\$9	Tabulación.
VK_CLEAR	12	\$C	5 del teclado numérico, BloqNum apagado.
VK_RETURN	13	\$D	Intro.
VK_SHIFT	16	\$10	Mayúscula.
VK_CONTROL	17	\$11	Ctrl.
VK_MENU	18	\$12	Alt.
VK_PAUSE	19	\$13	Pausa.
VK_CAPITAL	20	\$14	BloqMayús.
VK_ESCAPE	27	\$1B	Esc.
VK_SPACE	32	\$20	Barra espaciadora.
VK_PRIOR	33	\$21	Re.Pág.
VK_NEXT	34	\$22	Av.Pág.
VK_END	35	\$23	Fin.
VK_HOME	36	\$24	Inicio.
VK_LEFT	37	\$25	Tecla de cursor a la izquierda.
VK_UP	38	\$26	Tecla de cursor arriba.
VK_RIGHT	39	\$27	Tecla de cursor a la derecha.
VK_DOWN	40	\$28	Tecla de cursor abajo.
VK_SNAPSHOT	44	\$2C	Impr.Pant.
VK_INSERT	45	\$2D	Ins.
VK_DELETE	46	\$2E	Supr.

Código de tecla virtual	Dec.	Hex.	Descripción
VK_LWIN	91	\$5B	Tecla de Windows izquierda en un teclado compatible Windows 95/98.
VK_RWIN	92	\$5C	Tecla de Windows derecha en un teclado compatible Windows 95/98.
VK_APPS	93	\$5D	Tecla de menú en un teclado compatible Windows 95/98.
VK_NUMPAD0	96	\$60	0 del teclado numérico.
VK_NUMPAD1	97	\$61	1 del teclado numérico.
VK_NUMPAD2	98	\$62	2 del teclado numérico.
VK_NUMPAD3	99	\$63	3 del teclado numérico.
VK_NUMPAD4	100	\$64	4 del teclado numérico.
VK_NUMPAD5	101	\$65	5 del teclado numérico.
VK_NUMPAD6	102	\$66	6 del teclado numérico.
VK_NUMPAD7	103	\$67	7 del teclado numérico.
VK_NUMPAD8	104	\$68	8 del teclado numérico.
VK_NUMPAD9	105	\$69	9 del teclado numérico.
VK_MULTIPLY	106	\$6A	Multiplicar (*) del teclado numérico.
VK_ADD	107	\$6B	Sumar (+) del teclado numérico.
VK_SUBTRACT	109	\$6D	Restar (-) del teclado numérico.
VK_DECIMAL	110	\$6E	Punto decimal (.) del teclado numérico.
VK_DIVIDE	111	\$6F	Dividir (/) del teclado numérico.
VK_F1	112	\$70	F1.
VK_F2	113	\$71	F2.
VK_F3	114	\$72	F3.
VK_F4	115	\$73	F4.
VK_F5	116	\$74	F5.
VK_F6	117	\$75	F6.
VK_F7	118	\$76	F7.
VK_F8	119	\$77	F8.
VK_F9	120	\$78	F9.
VK_F10	121	\$79	F10.
VK_F11	122	\$7A	F11.
VK_F12	123	\$7B	F12.
VK_F13	124	\$7C	F13.
VK_F14	125	\$7D	F14.
VK_F15	126	\$7E	F15.
VK_F16	127	\$7F	F16.
VK_F17	128	\$80	F17.
VK_F18	129	\$81	F18.
VK_F19	130	\$82	F19.

Código de tecla virtual	Dec.	Hex.	Descripción
VK_F20	131	\$83	F20.
VK_F21	132	\$84	F21.
VK_F22	133	\$85	F22.
VK_F23	134	\$86	F23.
VK_F24	135	\$87	F24.
VK_NUMLOCK	144	\$90	Bloq.Num.
VK_SCROLL	145	\$91	Bloq.Scroll.
VK_LSHIFT	160	\$A0	Tecla May. izquierda.
VK_RSHIFT	161	\$A1	Tecla May. derecha.
VK_LCONTROL	162	\$A2	Tecla Ctrl izquierda.
VK_RCONTROL	163	\$A3	Tecla Ctrl derecha.
VK_LMENU	164	\$A4	Tecla Alt izquierda.
VK_RMENU	165	\$A5	Tecla Alt derecha.



## Apéndice E

# Bibliografía

Se dispone de una base de conocimientos bastante amplia acerca de la programación para Windows en general y la programación en Delphi en particular. La información contenida en este libro se apoya parcialmente en nuestra investigación y en conocimientos extraídos de los siguientes libros:

Calvert, Charles, *Delphi 4 Unleashed* [Sams Publishing, 1998]

Pacheco y Teixeira, *Delphi 4 Developers Guide* [Sams Publishing, 1998]

Thorpe, Danny, *Delphi Component Design* [Addison-Wesley Developers Press, 1997]

Konopka, Ray, *Developing Custom Delphi 3 Components* [Coriolis Group Books, 1997]

Petzold y Yao, *Programming Windows 95* [Microsoft Press, 1996]

Rector y Newcomer, *Win32 Programming* [Addison-Wesley Developers Press, 1997]

Richter, Jeffrey, *Advanced Windows*, [Microsoft Press, 1997]

Beveridge and Wiener, *Multithreading Applications in Win32*, [Addison-Wesley Developers Press, 1997]



**Nota:** En la tienda de libros y software de Danysoft podrá encontrar una selección de títulos orientados a Delphi (en el momento de publicar este libro existen más de 60 títulos), tanto en inglés como en castellano, con una información muy detallada sobre cada uno. Más información en: <http://www.danyshop.com>. Asimismo, el equipo de Danysoft está fuertemente comprometido con Delphi; puede ver las novedades editoriales que publiquemos en <http://www.danypress.com>.



## Apéndice F

# Código fuente y servicios en la Web

Nuestro equipo ha revisado el texto y código fuente para evitar cualquier tipo de error, pero no podemos prometerle que éste está o estará siempre libre de errores. Por ello y para que esto no sea una molestia para usted, hemos habilitado en <http://www.danypress.com> un apartado especial para cada libro que publiquemos, con el objetivo de que el libro permanezca “vivo”.

En este apartado encontrará por cada título :

- Información detallada del libro.
- Una lista de las erratas.
- Comentarios de los lectores.
- Una lista de libros relacionados.

Asimismo, podrá enviarnos por email a [editorial@danysoft.com](mailto:editorial@danysoft.com), (le rogamos no se olvide de indicarnos el título del libro y sus datos):

- Sus comentarios sobre el libro.
- Añadir una errata a lista existente (le rogamos no se olvide de indicarnos la página).
- Decirnos exactamente lo que piensa, para que conozcamos sus inquietudes. Todo comentario será muy bien recibido y tenido en cuenta en futuras acciones de nuestro equipo para intentar satisfacer sus necesidades.

# Indice

## A

AbortPath - 206  
 AddFontResource - 484  
 AdjustWindowRect - 610  
 AdjustWindowRectEx - 612  
 AnimatePalette - 420  
 ANSI, funciones - 8  
 AppendMenu - 701  
 Arc - 85  
 ASCII conj.caracteres - 891

## B

barras de aplicación - 646  
 BeginDeferWindowPos - 614  
 BeginPaint - 87  
 BeginPath - 207  
 BitBlt - 290  
 BringWindowToTop - 616  
 Brochas - 82

## C

Cadenas de caracteres - 4  
 CascadeWindows - 617  
 ChangeDisplaySettings - 21  
 CheckMenuItem - 704  
 CheckMenuRadioItem - 705  
 Chord - 89  
 ClientToScreen - 26  
 CloseEnhMetaFile - 292

CloseFigure - 208  
 CloseWindow - 618  
 CombineRgn - 210  
 Código fuente - XXIX, 901  
 Constantes - 4  
 Contextos de Dispositivos - 11  
 Convenios - XXVIII  
 CopyEnhMetaFile - 293  
 CopyIcon - 373  
 CopyImage - 293  
 CopyRect - 213  
 CreateBitmap - 296  
 CreateBitmapIndirect - 299  
 CreateBrushIndirect - 91  
 CreateCaret - 374  
 CreateCompatibleBitmap - 302  
 CreateCompatibleDC - 28  
 CreateCursor - 376  
 CreateDIBitmap - 304  
 CreateDIBSection - 308  
 CreateEllipticRgn - 214  
 CreateEllipticRgnIndirect - 215  
 CreateEnhMetaFile - 316  
 CreateFont - 484  
 CreateFontIndirect - 492  
 CreateHalftonePalette - 424  
 CreateHatchBrush - 94

CreateIcon - 379  
 CreateIconFromResource - 382  
 CreateIconFromResourceEx - 384  
 CreateIconIndirect - 386  
 CreateMenu - 707  
 CreatePalette - 427  
 CreatePatternBrush - 96  
 CreatePen - 97  
 CreatePenIndirect - 99  
 CreatePolygonRgn - 216  
 CreatePolyPolygonRgn - 219  
 CreatePopupMenu - 709  
 CreateRectRgn - 222  
 CreateRectRgnIndirect - 223  
 CreateRoundRectRgn - 224  
 CreateScalableFontResource - 499  
 CreateSolidBrush - 102  
 cursores de edición - 367  
 cursores - 368

## D

DeferWindowPos - 619  
 DeleteDC - 31  
 DeleteEnhMetaFile - 319  
 DeleteMenu - 712  
 DestroyCaret - 388  
 DestroyCursor - 389  
 DestroyIcon - 390

DestroyMenu - 713  
 DPTOLP - 32  
 DragAcceptFiles - 650  
 DragFinish - 652  
 DragQueryFile - 653  
 DragQueryPoint - 654  
 DrawCaption - 103  
 DrawEdge - 105  
 DrawFocusRect - 108  
 DrawFrameControl - 109  
 DrawIcon - 390  
 DrawIconEx - 391  
 DrawState - 114  
 DrawText - 502  
 DrawTextEx - 506

**E**

Efectos Especiales - 202, 608  
 Ellipse - 117  
 EnableMenuItem - 714  
 EndDeferWindowPos - 622  
 EndPaint - 119  
 EndPath - 226  
 EnumDisplaySettings - 32  
 EnumFontFamilies - 511  
 EnumFontFamiliesEx - 516  
 EnumObjects - 120  
 EnumResourceLanguages - 584  
 EnumResourceNames - 589  
 EnumResourceTypes - 591  
 EqualRect - 226  
 EqualRgn - 227  
 ExcludeClipRect - 228

ExtCreatePen - 123  
 ExtCreateRegion - 231  
 ExtFloodFill - 127  
 ExtractAssociatedIcon - 393  
 ExtractIcon - 394  
 ExtractIconEx - 396  
 ExtSelectClipRgn - 234

**F**

FillPath - 128  
 FillRect - 129  
 FillRgn - 131  
 FindExecutable - 655  
 FindResource - 595  
 FindResourceEx - 597  
 FlattenPath - 235  
 FrameRgn - 133  
 Fuentes - 473  
 Funciones

- de cursores - 367
- de dibujo - 81
- de iconos - 367
- de Importación - 5
- de la interfaz gráfica - 20
- de mapas de bits - 279
- de menú - 695
- de metaarchivos - 279
- de mov. de ventanas - 607
- de paleta - 415
- de recursos - 579
- de regiones y rutas - 199
- de respuesta - 6
- de salida de texto - 473

del Shell - 641  
 Importadas incorrectamente - 5  
 Parámetros - 7

**G**

GetBitmapBits - 323  
 GetBitmapDimensionEx - 325  
 GetBkColor - 135  
 GetBkMode - 136  
 GetBoundsRect - 136  
 GetBrushOrgEx - 138  
 GetBValue - 432  
 GetCharABCWidths - 523  
 GetCharWidth - 525  
 GetClipBox - 236  
 GetClipRgn - 237  
 GetCurrentObject - 139  
 GetCurrentPositionEx - 140  
 GetCursor - 399  
 GetDC - 37  
 GetDCOrgEx - 37  
 GetDeviceCaps - 37  
 GetDIBColorTable - 433  
 GetDIBits - 325  
 GetEnhMetaFile - 329  
 GetEnhMetaFileDescription - 332  
 GetEnhMetaFileHeader - 333  
 GetEnhMetaFilePaletteEntries - 437  
 GetFontData - 526  
 GetGlyphOutline - 527  
 GetGValue - 440  
 GetIconInfo - 399  
 GetKerningPairs - 532

GetMapMode - 46	GetTabbedTextExtent - 552	InvertRgn - 246
GetMenu - 716	GetTextAlign - 553	IsMenu - 735
GetMenuDefaultItem - 718	GetTextCharacterExtra - 555	IsRectEmpty - 246
GetMenuItemCount - 719	GetTextColor - 555	<b>L</b>
GetMenuItemID - 719	GetTextExtentExPoint - 556	LineDDA - 163
GetMenuItemInfo - 720	GetTextExtentPoint32 - 559	LineTo - 165
GetMenuItemRect - 724	GetTextFace - 560	LoadBitmap - 336
GetMenuState - 725	GetTextMetrics - 561	LoadCursor - 403
GetMenuString - 727	GetUpdateRect - 154	LoadCursorFromFile - 405
GetMiterLimit - 141	GetUpdateRgn - 155	LoadIcon - 406
GetNearestColor - 441	GetViewportExtEx - 53	LoadImage - 340
GetNearestPaletteIndex - 442	GetViewportOrgEx - 54	LoadResource - 599
GetObject - 142	GetWindowExtEx - 56	LoadString - 599
GetObjectType - 147	GetWindowOrgEx - 57	LockResource - 601
GetOutlineTextMetrics - 534	GetWindowPlacement - 623	LockWindowUpdate - 166
GetPaletteEntries - 445	GrayString - 156	LookupIconIdFromDirectory - 407
GetPath - 238	<b>H</b>	LookupIconIdFromDirectoryEx - 408
GetPixel - 148	HideCaret - 403	LPTODP - 58
GetPolyFillMode - 148	HiliteMenuItem - 730	<b>M</b>
GetRasterizerCaps - 551	<b>I</b>	MakeIntResource - 602
GetRegionData - 241	iconos - 368	Manejadores - 4
GetRgnBox - 242	identificador de elemento - 645	Mapas de bits - 279
GetROP2 - 150	Importación de funciones de Windows - 5	MapWindowPoints - 59
GetRValue - 447	InflateRect - 243	Máscaras de cursores - 368
GetStockObject - 152	InsertMenu - 731	Máscaras de iconos - 368
GetStretchBltMode - 335	InsertMenuItem - 734	Mensajes - 753
GetSubMenu - 728	Interfaz Gráfica - 20	menú de sistema - 696
GetSysColor - 447	IntersectRect - 244	menú dibujado por propietario - 697
GetSystemMenu - 729	InvalidateRect - 159	menús emergentes - 696
GetSystemMetrics - 48	InvalidateRgn - 161	Metaficheros - 287
GetSystemPaletteEntries - 450	InvertRect - 244	ModifyMenu - 736
GetSystemPaletteUse - 454		

MoveToEx - 167

MoveWindow - 624

## O

OffsetClipRgn - 247

OffsetRect - 249

OffsetRgn - 253

OffsetViewportOrgEx - 61

OffsetWindowOrgEx - 62

OpenIcon - 625

Orden Z - 607

## P

PaintDesktop - 168

PaintRgn - 168

paletas de colores - 415

PaletteIndex - 455

PaletteRGB - 455

PatBlt - 343

PathToRegion - 255

Pie - 170

PlayEnhMetaFile - 345

PlayEnhMetaFileRecord - 346

Plumas - 82

PolyBezier - 172

PolyBezierTo - 173

Polygon - 175

Polyline - 175

PolylineTo - 177

PolyPolygon - 178

PolyPolyline - 179

PtInRect - 257

PtInRegion - 258

PtVisible - 259

## R

RealizePalette - 456

Rectangle - 181

RectInRegion - 260

RectVisible - 260

recursos - 579

Regiones - 199

ReleaseDC - 63

RemoveFontResource - 567

RemoveMenu - 739

ResizePalette - 458

RestoreDC - 64

RGB - 461

RoundRect - 183

Rutas - 202

## S

SaveDC - 64

ScaleViewportExtEx - 65

ScaleWindowExtEx - 70

ScreenToClient - 71

ScrollDC - 71

SelectClipPath - 261

SelectClipRgn - 266

SelectObject - 185

SelectPalette - 461

SetBitmapBits - 347

SetBitmapDimensionEx - 350

SetBkColor - 186

SetBkMode - 187

SetBoundsRect - 187

SetBrushOrgEx - 189

SetCursor - 409

SetDIBColorTable - 463

SetDIBits - 351

SetDIBitsToDevice - 355

SetMapMode - 74

SetMenu - 740

SetMenuDefaultItem - 741

SetMenuItemBitmaps - 743

SetMenuItemInfo - 745

SetMiterLimit - 190

SetPaletteEntries - 465

SetPixel - 191

SetPixelV - 192

SetPolyFillMode - 193

SetRect - 269

SetRectEmpty - 270

SetRectRgn - 271

SetROP2 - 195

SetStretchBltMode - 358

SetSysColors - 466

SetSystemCursor - 410

SetSystemPaletteUse - 468

SetTextAlign - 568

SetTextCharacterExtra - 572

SetTextColor - 572

SetTextJustification - 573

SetViewportExtEx - 76

SetViewportOrgEx - 77

SetWindowExtEx - 78

SetWindowOrgEx - 78

SetWindowPlacement - 626

SetWindowPos - 629

SetWindowRgn - 272

SHAddToRecentDocs - 658

SHAppBarMessage - 659

SHBrowseForFolder - 665

Shell\_NotifyIcon - 680

ShellAbout - 670

ShellExecute - 671

ShellExecuteEx - 675

SHFileOperation - 683

SHFreeNameMappings - 686

SHGetFileInfo - 687

SHGetPathFromIDList - 691

SHGetSpecialFolderLocation - 691

ShowCaret - 412

ShowCursor - 413

ShowOwnedPopups - 632

ShowWindow - 634

ShowWindowAsync - 636

Sistemas de Coordenadas - 14

SizeofResource - 604

StretchBlt - 359

StretchDIBits - 362

StrokeAndFillPath - 196

StrokePath - 197

SubtractRect - 275

**T**

TabbedTextOut - 574

Teclas virtuales - 895

TextOut - 577

TileWindows - 638

Tipos de datos de Windows - 2

TrackPopupMenu - 747

TrackPopupMenuEx - 748, 751

**U**

Unicode - 7

Unicode, funciones - 8

UnionRect - 276

**W**

WidenPath - 277

Windows

Cadenas de caracteres - 4

Constantes - 4

Importación de funciones - 5

WM\_ACTIVATE - 753

WM\_ACTIVATEAPP - 754

WM\_ASKCBFORMATNAME - 755

WM\_CANCELMODE - 756

WM\_CHANGECHAIN - 756

WM\_CHAR - 757

WM\_CHARTOITEM - 758

WM\_CHILDACTIVATE - 759

WM\_CLEAR - 760

WM\_CLOSE - 760

WM\_COMMAND - 761

WM\_COMPACTING - 762

WM\_COMPAREITEM - 762

WM\_COPY - 764

WM\_COPYDATA - 765

WM\_CREATE - 766

WM\_CTLCOLORBTN - 767

WM\_CTLCOLOREDIT - 767

WM\_CTLCOLORLISTBOX - 768

WM\_CTLCOLORMSGBOX - 769

WM\_CTLCOLORSCROLLBAR - 770

WM\_CTLCOLORSTATIC - 770

WM\_CUT - 771

WM\_DEADCHAR - 772

WM\_DELETEITEM - 773

WM\_DESTROY - 774

WM\_DESTROYCLIPBOARD - 775

WM\_DEVMODECHANGE - 775

WM\_DISPLAYCHANGE - 776

WM\_DRAWCLIPBOARD - 777

WM\_DRAWITEM - 777

WM\_DROPFILES - 780

WM\_ENABLE - 780

WM\_ENDSESSION - 781

WM\_ENTERIDLE - 782

WM\_ENTERMENULOOP - 783

WM\_ERASEBKGND - 783

WM\_EXITMENULOOP - 784

WM\_FONTCHANGE - 784

WM\_GETFONT - 785

WM\_GETHOTKEY - 786

WM\_GETICON - 786

WM\_GETMINMAXINFO - 787

WM\_GETTEXT - 788

WM\_GETTEXTLENGTH - 789

WM\_HELP - 790

WM\_HSCROLL - 791

WM\_HSCROLLCLIPBOARD - 792

WM\_ICONERASEBKGND - 794

WM\_INITMENU - 794

WM_INITMENUPOPUP - 795	WM_NCDESTROY - 827	WM_RBUTTONUP - 852
WM_KEYDOWN - 796	WM_NCHITTEST - 827	WM_RENDERALLFORMATS - 853
WM_KEYUP - 797	WM_NCLBUTTONDBLCLK - 829	WM_RENDERFORMAT - 854
WM_KILLFOCUS - 798	WM_NCLBUTTONDOWN - 830	WM_SETCURSOR - 855
WM_LBUTTONDBLCLK - 799	WM_NCLBUTTONUP - 830	WM_SETFOCUS - 856
WM_LBUTTONDOWN - 800	WM_NCMBUTTONDBLCLK - 831	WM_SETFONT - 856
WM_LBUTTONUP - 801	WM_NCMBUTTONDOWN - 832	WM_SETHOTKEY - 857
WM_MBUTTONDBLCLK - 802	WM_NCMBUTTONUP - 833	WM_SETICON - 858
WM_MBUTTONDOWN - 803	WM_NCMOUSEMOVE - 834	WM_SETREDRAW - 859
WM_MBUTTONUP - 804	WM_NCPAINT - 834	WM_SETTEXT - 860
WM_MDIACTIVATE - 805	WM_NCRBUTTONDBLCLK - 835	WM_SHOWWINDOW - 861
WM_MDICASCADE - 806	WM_NCRBUTTONDOWN - 836	WM_SIZE - 863
WM_MDICREATE - 807	WM_NCRBUTTONUP - 837	WM_SIZECLIPBOARD - 863
WM_MDIDESTROY - 809	WM_NEXTDLGCTRL - 838	WM_SPOOLERSTATUS - 863
WM_MDIGETACTIVE - 809	WM_NOTIFY - 838	WM_STYLECHANGED - 864
WM_MDIICONARRANGE - 810	WM_NOTIFYFORMAT - 840	WM_STYLECHANGING - 865
WM_MDIMAXIMIZE - 810	WM_PAINT - 841	WM_SYSCHAR - 866
WM_MDINEXT - 811	WM_PAINTCLIPBOARD - 842	WM_SYSCOLORCHANGE - 868
WM_MDIREFRESHMENU - 812	WM_PALETTECHANGED - 843	WM_SYSCOMMAND - 868
WM_MDIRESTORE - 812	WM_PALETTEISCHANGING - 844	WM_SYSDEADCHAR - 870
WM_MDISETMENU - 813	WM_PARENTNOTIFY - 844	WM_SYSKEYDOWN - 872
WM_MDITILE - 814	WM_PASTE - 846	WM_SYSKEYUP - 873
WM_MEASUREITEM - 815	WM_QUERYDRAGICON - 847	WM_TIMECHANGE - 874
WM_MENUCHAR - 816	WM_QUERYENDSESSION - 847	WM_TIMER - 875
WM_MENUSELECT - 817	WM_QUERYNEWPALETTE - 848	WM_UNDO - 876
WM_MOUSEACTIVATE - 818	WM_QUERYOPEN - 849	WM_VKEYTOITEM - 876
WM_MOUSEMOVE - 819	WM_QUIT - 849	WM_VSCROLL - 877
WM_MOVE - 820	WM_RBUTTONDBLCLK - 850	WM_VSCROLLCLIPBOARD - 878
WM_NCACTIVATE - 821	WM_RBUTTONDOWN - 851	WM_WINDOWPOSCHANGED - 879
WM_NCCALCSIZE - 822		
WM_NCCREATE - 827		

**908 ■**

WM\_WINDOWPOSCHANGING - 880