CONTENTS

Introduction .............................................................................. 1
Document conventions ......................................................... 2

Part 3 General reference

Chapter 7 Data types and structures .................................. 7
Data types ............................................................................. 7
Data structures .................................................................... 9
BITMAP ............................................................................... 10
Bitmap data structure ....................................................... 10
BITMAPCOREHEADER ....................................................... 11
Device-independent bitmap format information .................. 11
BITMAPCOREINFO .......................................................... 12
Device-independent bitmap information ......................... 12
BITMAPFILEHEADER ....................................................... 14
Bitmap file information .................................................... 14
BITMAPINFO .................................................................. 14
Device-independent bitmap information ......................... 14
BITMAPINFOHEADER ...................................................... 16
Device-independent bitmap format information ................. 16
CLIENTCREATESTRUCT .................................................. 20
MDI client window creation structure ............................... 20
COLORREF .................................................................... 20
Color specification ............................................................ 20
Palette-relative rgb ............................................................ 21
COMPAREITEMSTRUCT .................................................. 22
Owner-draw item-sorting information ............................... 22
COMSTAT ....................................................................... 23
Communication device status ......................................... 23
CREATESTRUCT ............................................................ 24
Window-creation structure .............................................. 24

DCB ................................................................................ 25
Communications device control block ............................. 25
DELETEITEMSTRUCT .................................................. 28
Deleted owner-draw list-box item .................................... 28
DEVMODE ....................................................................... 29
Printer driver initialization information ............................ 29
DLGTEMPLATE .............................................................. 32
Dialog template ............................................................... 32
HEADER ......................................................................... 32
Device-independent bitmap information ......................... 32
BITMAPFILEHEADER ...................................................... 14
Bitmap file information .................................................... 14
BITMAPINFO .................................................................. 14
Device-independent bitmap information ......................... 14
BITMAPINFOHEADER ...................................................... 16
Device-independent bitmap format information ................. 16
CLIENTCREATESTRUCT .................................................. 20
MDI client window creation structure ............................... 20
COLORREF .................................................................... 20
Color specification ............................................................ 20
Palette-relative rgb ............................................................ 21
COMPAREITEMSTRUCT .................................................. 22
Owner-draw item-sorting information ............................... 22
COMSTAT ....................................................................... 23
Communication device status ......................................... 23
CREATESTRUCT ............................................................ 24
Window-creation structure .............................................. 24

HANDLETABLE .............................................................. 38
Window-handle table ........................................................ 38
LOGBRUSH .................................................................. 38
Logical-brush attribute information ................................. 38
LOGFONT ....................................................................... 40
Logical-font descriptor .................................................... 40
LOGPALETTE ................................................................. 43
Logical color palette information ..................................... 43
LOGPEN ........................................................................ 44
Logical-pen attribute information ..................................... 44
MDICREATESTRUCT ......................................................... 45
Mdi child window creation structure ............................... 45
MEASUREITEMSTRUCT .................................................. 46
Owner-draw control dimensions ..................................... 46
MENUITEMTEMPLATE .................................................. 47
Menu-item template ........................................................ 47
METAFILEPICT ............................................................... 49
Metafile picture structure ............................................... 49
MSG .............................................................................. 50
Message data structure .................................................. 50
MULTIKEYHELP ............................................................. 50
Windows help key word table
structure ........................................ 50
OFSTRUCT ........................................ 51
Open-file structure .............................. 51
PAINTSTRUCT .................................... 52
WINDOWS paint information .................... 52
PALETTEENTRY ................................... 52
Logical palette color entry ..................... 52
POINT ............................................. 54
Point data structure ............................. 54
RECT ............................................... 54
Rectangle data structure ....................... 54
RGBQUAD ........................................... 55
Rgb color structure ............................. 55
RGBTRIPLE .......................................... 55
Rgb color structure ............................. 55
TEXTMETRIC ....................................... 56
Basic font metrics ............................... 56
WNDCLASS ......................................... 58
Window class data structure .................... 58

Chapter 8 Resource script statements
Single-line statements ......................... 61
User-defined resources ......................... 63
Rcdata statement ................................ 64
Stringtable statement ......................... 65
Accelerators statement ......................... 67
Menu statement .................................. 68
Item-definition statements ..................... 70
MENUITEM ......................................... 70
POPUP .............................................. 71
MENUITEM SEPARATOR ......................... 73
DIALOG statement .................................. 73
Dialog option statements ....................... 75
STYLE ............................................. 75
CAPTION ........................................... 77
MENU ............................................... 78
CLASS ............................................. 78
FONT .............................................. 79
Dialog control statements ..................... 79
LTEXT ............................................. 80
RTEXT ............................................. 81

Chapter 9 File formats
Bitmap file formats .................. 109
Icon resource file format ........... 110
Cursor resource file format .......... 111
Clipboard file format ............... 113
Metafile format .......................... 113
Metafile header .......................... 114
Metafile records ......................... 115
Typical metafile record ............ 116
Function-specific records .......... 116
AnimatePalette record 3.0 ........... 117
BitBlt record (prior to 3.0) .......... 117
BitBlt record 3.0 ......................... 118
CreateBrushIndirect record .......... 118
CreateFontIndirect record .......... 119
CreatePalette record 3.0 .............. 119
CreatePatternBrush record (prior to 3.0) ................. 119
CreatePatternBrush record 3.0 ...... 120
CreatePenIndirect record .......... 121
Create region record ........................... 121
DeleteObject 3.0 .................................. 122
DrawText record ..................................... 122
Escape record ....................................... 122
ExtTextOut record .................................. 123
Polygon record ....................................... 124
PolyPolygon record .................................. 124
Polyline record ....................................... 125
SelectClipRegion .................................... 125
SelectObject ......................................... 125
SelectPalette record 3.0 ......................... 126
SetDIBitsToDevice record 3.0 .................... 126
SetPaletteEntries record 3.0 .................... 126
StretchBlt record (prior to 3.0) ............... 127
StretchBlt record 3.0 ................................ 128
StretchDIBits record 3.0 ............................ 129
TextOut record ....................................... 130
Sample metafile program output ................. 130
Summary .............................................. 132

Chapter 10 Module-definition statements 133
CODE .................................................. 134
DATA .................................................. 134
DESCRIPTION ....................................... 135
EXETYPE ............................................. 136
EXPORTS ............................................. 136
HEAPSIZE ............................................ 137
IMPORTS ............................................. 138
LIBRARY .............................................. 139
NAME .................................................. 139
SEGMENTS ............................................ 140
STACKSIZE .......................................... 141
STUB ................................................... 141

Chapter 11 Binary and ternary raster-operation codes 143
Binary raster operations .......................... 143
Ternary raster operations ......................... 146

Chapter 12 Printer escapes 153
ABORTDOC ........................................... 153
BANDINFO .......................................... 154
BEGIN_PATH ......................................... 156

CLIP_TO_PATH ....................................... 157
DEVICEDATA ....................................... 158
DRAFTMODE ......................................... 158
DRAWPATTERNRECT ................................ 158
ENABLEDUPLEX ..................................... 160
ENABLEPAIRKERNING ................................ 160
ENABLERELATIVEWIDTHS ......................... 161
ENDDOC ............................................. 162
END_PATH ............................................ 162
ENUMPAPERBINS ..................................... 164
ENUMPAPERMETRICS ................................ 165
EPSPRINTING ....................................... 166
EXT_DEVICE_CAPS .................................. 166
EXTTEXTOUT ........................................ 168
FLUSHOUTPUT ....................................... 169
GETCOLORTABLE .................................... 169
GETEXTENDEDTEXTMETRICS ...................... 170
GETEXTENTTABLE ................................... 173
GETFACENAME ....................................... 174
GETPAIRKERNTABLE ................................ 174
GETPHYSICALSIZEx ................................. 176
GETPRINTINGOFFSET ............................... 176
GETSCALINGFACTOR ................................ 176
GETSETPAPERBINS .................................. 177
GETSETPAPERMETRICS .............................. 178
GETSETPORTORIENT ................................ 179
GETSETSCREENPARAMS .............................. 180
GETTECHNOLOGY .................................... 181
GETTRACKKERNTABLE ................................ 181
GETVECTORBRUSHSIZE ............................... 182
GETVECTORPENSIZE ................................ 183
MFCOMMENT .......................................... 183
NEWFRAME .......................................... 184
NEXTBAND .......................................... 184
PASSTHROUGH ....................................... 185
QUERYESCSUPPORT .................................. 186
RESTORE_CTM ....................................... 186
SAVE_CTM .......................................... 187
SELECTPAPERSOURCE ............................... 188
SETABORTPROC ..................................... 188
SETALLJUSTVALUES ................................ 189
SET_ARC_DIRECTION ............................... 190
SET_BACKGROUND_COLOR ......................... 191
8.1: Window styles .................... 75
8.2: Control classes .................... 94
8.3: Control styles ..................... 96
9.1: Bit mask results .................. 112
9.2: GDI functions and values .......... 115
11.1: Operation indexes for DPo and
       DPan .................................. 144
11.2: Operation Indexes for PSo and
       DPSoo .................................. 146
11.3: Raster-operation codes ........... 147
12.1: Meaning of BANDINFOSTRUCT
       fields .................................. 155
13.1: DDE messages .................... 206
This manual gives the Windows-application developer general as well as detailed information about Windows functions, messages, data types, Resource Compiler statements, assembly language macros, and file formats. This manual provides detailed descriptions of each component of the Windows application program interface (API) for readers who already have a basic understanding of Windows programming.

This manual is divided into two volumes. Volume 1 contains reference information describing the Windows functions and messages.

Volume 2 contains reference material for other components of the Windows API. It contains the following nine chapters and five appendixes:

Chapter 7, "Data types and structures," contains a table of data types and an alphabetical list of structures found in Windows.

Chapter 8, "Resource script statements," describes the statements that define resources which the Resource Compiler adds to an application's executable file. The statements are arranged according to functional groups.

Chapter 9, "File formats," describes the formats of five types of files: bitmap files, icon resource files, cursor resource files, clipboard files, and metafiles. Each description gives the general file structure and information about specific parts of the file.

Chapter 10, "Module-definition statements," describes the statements contained in the module-definition file that defines the application's contents and system requirements for the LINK program.

Chapter 11, "Binary and ternary raster-operation codes," describes the raster operations used for line output and those used for bitmap output.

Chapter 12, "Printer escapes," lists the printer escapes that are available in Windows.
Chapter 13, "Windows DDE protocol definition," contains an alphabetical listing and description of the Windows messages that comprise the Windows Dynamic Data Exchange protocol.

Appendix A, "Virtual-key codes," lists the symbolic names and hexadecimal values of Windows virtual-key codes and includes a brief description of each key.

Appendix B, "RC Diagnostic messages," contains a listing of Resource Compiler error messages and provides a brief description of each message.

Document conventions

Throughout this manual, the term "DOS" refers to both MS-DOS® and PC-DOS, except when noting features that are unique to one or the other.

The following document conventions are used throughout this manual:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold text</strong></td>
<td>Bold letters indicate a specific term or punctuation mark intended to be used literally: language key words or functions (such as EXETYPE or CreateWindow), DOS commands, and command-line options (such as /ZI). You must type these terms and punctuation marks exactly as shown. However, the use of uppercase or lowercase letters is not always significant. For instance, you can invoke the linker by typing either LINK, link, or Link at the DOS prompt.</td>
</tr>
<tr>
<td>( )</td>
<td>In syntax statements, parentheses enclose one or more parameters that you pass to a function.</td>
</tr>
<tr>
<td><em>Italic text</em></td>
<td>Words in italics indicate a placeholder; you are expected to provide the actual value. For example, the following syntax for the SetCursorPos function indicates that you must substitute values for the X and Y coordinates, separated by a comma: SetCursorPos(X, Y)</td>
</tr>
<tr>
<td>Monospaced type</td>
<td>Code examples are displayed in a nonproportional typeface.</td>
</tr>
<tr>
<td>:</td>
<td>Vertical ellipses in program examples indicate that a portion of the program is omitted.</td>
</tr>
<tr>
<td>...</td>
<td>Ellipses following an item indicate that more items having the same form may appear. In the following example, the</td>
</tr>
</tbody>
</table>
horizontal ellipses indicate that you can specify more than one breakaddress for the g command:

\[ g [[[=startaddress]] [[breakaddress]]]... \]

Double brackets enclose optional fields or parameters in command lines and syntax statements. In the following example, option and executable-file are optional parameters of the RC command:

\[ RC [[[option]] filename [[executable-file]]] \]

A vertical bar indicates that you may enter one of the entries shown on either side of the bar. The following command-line syntax illustrates the use of a vertical bar:

\[ DB [[[address \| range]]] \]

The bar indicates that following the DB (dump bytes) command, you can specify either an address or a range.

Curly braces indicate that you must specify one of the enclosed items.

Small capital letters indicate the names of keys and key sequences, such as:

\[ ALT + SPACEBAR \]

The Microsoft Windows version number indicates that a function, message, or data structure is compatible only with the specified version and later versions.
Part 3 provides general reference information on components of the Windows application programming interface that are in addition to the functions and messages described in the preceding parts.
Data types and structures

This chapter describes the data types and structures used by Microsoft Windows functions and messages. It contains two parts: a table of data types and a list of Windows data structures, each arranged alphabetically.

Data types

The data types in the following list are key words that define the size and meaning of parameters and return values associated with Windows functions. This list contains character, integer, and Boolean types, pointer types, and handles. The character, integer, and Boolean types are common to most C compilers. Most of the pointer-type names begin with either a P prefix (for short pointers) or an LP prefix (for long pointers). A short pointer accesses data within the current data segment; a long pointer contains a 32-bit segment/offset value. A Windows application uses a handle to refer to a resource that has been loaded into memory. Windows provides access to these resources through internally maintained tables that contain individual entries for each handle. Each entry in the handle table contains the address of the resource and a means of identifying the resource type. The Windows data types are defined in the following list:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>16-bit Boolean value.</td>
</tr>
<tr>
<td>BYTE</td>
<td>Unsigned 8-bit integer.</td>
</tr>
<tr>
<td>char</td>
<td>ASCII character or a signed 8-bit integer.</td>
</tr>
<tr>
<td><strong>DWORD</strong></td>
<td>Unsigned 32-bit integer or a segment/offset address.</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td><strong>FAR</strong></td>
<td>Data-type attribute that can be used to create a long pointer.</td>
</tr>
<tr>
<td><strong>FARPROC</strong></td>
<td>Long pointer to a function obtained by calling the MakeProcInstance function.</td>
</tr>
<tr>
<td><strong>GLOBALHANDLE</strong></td>
<td>Handle to global memory. It is a 16-bit index to a block of memory allocated from the system's global heap.</td>
</tr>
<tr>
<td><strong>HANDLE</strong></td>
<td>General handle. It represents a 16-bit index to a table entry that identifies program data.</td>
</tr>
<tr>
<td><strong>HBITMAP</strong></td>
<td>Handle to a physical bitmap. It is a 16-bit index to GDI's physical drawing objects.</td>
</tr>
<tr>
<td><strong>HBRUSH</strong></td>
<td>Handle to a physical brush. It is a 16-bit index to GDI's physical drawing objects.</td>
</tr>
<tr>
<td><strong>HCURSOR</strong></td>
<td>Handle to a cursor resource. It is a 16-bit index to a resource-table entry.</td>
</tr>
<tr>
<td><strong>HDC</strong></td>
<td>Handle to a display context. It is a 16-bit index to GDI's device-context tables.</td>
</tr>
<tr>
<td><strong>HFONT</strong></td>
<td>Handle to a physical font. It is a 16-bit index to GDI's physical drawing objects.</td>
</tr>
<tr>
<td><strong>HICON</strong></td>
<td>Handle to an icon resource. It is a 16-bit index to a resource-table entry.</td>
</tr>
<tr>
<td><strong>HMENU</strong></td>
<td>Handle to a menu resource. It is a 16-bit index to a resource-table entry.</td>
</tr>
<tr>
<td><strong>HPALETTE</strong></td>
<td>Handle to a logical palette. It is a 16-bit index to GDI's physical drawing objects.</td>
</tr>
<tr>
<td><strong>HPEN</strong></td>
<td>Handle to a physical pen. It is a 16-bit index to GDI's physical drawing objects.</td>
</tr>
<tr>
<td><strong>HRGN</strong></td>
<td>Handle to a physical region. It is a 16-bit index to GDI's physical drawing objects.</td>
</tr>
<tr>
<td><strong>HSTR</strong></td>
<td>Handle to a string resource. It is a 16-bit index to a resource-table entry.</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>Signed 16-bit integer.</td>
</tr>
<tr>
<td><strong>LOCALHANDLE</strong></td>
<td>Handle to local memory. It is a 16-bit index to a block of memory allocated from the application's local heap.</td>
</tr>
<tr>
<td><strong>long</strong></td>
<td>Signed 32-bit integer.</td>
</tr>
<tr>
<td><strong>LONG</strong></td>
<td>Signed 32-bit integer.</td>
</tr>
<tr>
<td><strong>LPBITMAP</strong></td>
<td>Long pointer to a BITMAP data structure.</td>
</tr>
<tr>
<td><strong>LPBITMAPCOREHEADER</strong></td>
<td>Long pointer to a BITMAPCOREHEADER data structure.</td>
</tr>
<tr>
<td><strong>LPBITMAPCOREINFO</strong></td>
<td>Long pointer to a BITMAPCOREINFO data structure.</td>
</tr>
<tr>
<td><strong>LPBITMAPFILEHEADER</strong></td>
<td>Long pointer to a BITMAPFILEHEADER data structure.</td>
</tr>
<tr>
<td><strong>LPBITMAPINFO</strong></td>
<td>Long pointer to a BITMAPINFO data structure.</td>
</tr>
<tr>
<td><strong>LPBITMAPINFOHEADER</strong></td>
<td>Long pointer to a BITMAPINFOHEADER data structure.</td>
</tr>
<tr>
<td><strong>LPCOMPAREITEMSTRUCT</strong></td>
<td>Long pointer to a COMAREITEMSTRUCT data structure.</td>
</tr>
<tr>
<td><strong>LPCREATESTRUCT</strong></td>
<td>Long pointer to a CREATESTRUCT data structure.</td>
</tr>
</tbody>
</table>
### Data structures

This section lists data structures that are used by Windows. The data structures are presented in alphabetical order. The structure definition is given, followed by a description of each field.
The **BITMAP** structure defines the height, width, color format, and bit values of a logical bitmap.

```c
typedef struct tagBITMAP { short bmType; short bmWidth; short bmHeight; short bmWidthBytes; BYTE bmPlanes; BYTE bmBitsPixel; LPSTR bmBits; } BITMAP;
```

The **BITMAP** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmType</td>
<td>Specifies the bitmap type. For logical bitmaps, the <code>bmType</code> field must be zero.</td>
</tr>
<tr>
<td>bmWidth</td>
<td>Specifies the width of the bitmap (in pixels). The width must be greater than zero.</td>
</tr>
<tr>
<td>bmHeight</td>
<td>Specifies the height of the bitmap (in raster lines). The height must be greater than zero.</td>
</tr>
<tr>
<td>bmWidthBytes</td>
<td>Specifies the number of bytes in each raster line. This value must be an even number since the graphics device interface (GDI) assumes that the bit values of a bitmap form an array of integer (two-byte) values. In other words, <code>bmWidthBytes</code> must be the next multiple of 16 greater than or equal to the <code>bmWidth</code> field.</td>
</tr>
<tr>
<td>bmPlanes</td>
<td>Points to the number of color planes in the bitmap.</td>
</tr>
<tr>
<td>bmBitsPixel</td>
<td>Points to the number of adjacent color bits on each plane needed to define a pixel.</td>
</tr>
<tr>
<td>bmBits</td>
<td>Points to the location of the bit values for the bitmap. The <code>bmBits</code> field must be a long pointer to an array of character (one-byte) values.</td>
</tr>
</tbody>
</table>

**Comments**

The currently used bitmap formats are monochrome and color. The monochrome bitmap uses a one-bit, one-plane format. Each scan is a multiple of 16 bits.

Scans are organized as follows for a monochrome bitmap of height $n$:

```
Scan 0
Scan 1
```

Software development kit
The pixels on a monochrome device are either black or white. If the corresponding bit in the bitmap is 1, the pixel is turned on (white); if the corresponding bit in the bitmap is zero, the pixel is turned off (black).

All devices that have the RC_BITBLT bit set in the device capabilities support bitmaps.

Each device has its own unique color format. In order to transfer a bitmap from one device to another, use `GetDIBits` and `SetDIBits`.

See also The `CreateBitmapIndirect` and `GetObject` functions in Chapter 4, "Functions directory," in Reference, Volume 1.

**BITMAPCOREHEADER**

Device-independent bitmap format information

The `BITMAPCOREHEADER` structure contains information about the dimensions and color format of a device-independent bitmap that is compatible with Microsoft OS/2 Presentation Manager versions 1.1 and 1.2 bitmaps.

```c
typedef struct tagBITMAPCOREHEADER {
    DWORD  bcSize;
    WORD   bcWidth;
    WORD   bcHeight;
    WORD   bcPlanes;
    WORD   bcBitCount;
} BITMAPCOREHEADER;
```

The `BITMAPCOREHEADER` structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcSize</td>
<td>Specifies the number of bytes required by the <code>BITMAPCOREHEADER</code> structure.</td>
</tr>
<tr>
<td>bcWidth</td>
<td>Specifies the width of the bitmap in pixels.</td>
</tr>
<tr>
<td>bcHeight</td>
<td>Specifies the height of the bitmap in pixels.</td>
</tr>
<tr>
<td>bcPlanes</td>
<td>Specifies the number of planes for the target device and must be set to 1.</td>
</tr>
<tr>
<td>bcBitCount</td>
<td>Specifies the number of bits per pixel. This value must be 1, 4, 8, or 24.</td>
</tr>
</tbody>
</table>
The BITMAPCOREINFO data structure combines the BITMAPCOREHEADER structure and a color table to provide a complete definition of the dimensions and colors of a device-independent bitmap. See the description of the BITMAPCOREINFO data structure for more information about specifying a device-independent bitmap.

An application should use the information stored in the bcSize field to locate the color table in a BITMAPCOREINFO data structure with a method such as the following:

```c
pColor = ((LPSTR) pBitmapCoreInfo + (WORD) (pBitmapCoreInfo -&gt; bcSize))
```

The BITMAPCOREINFO structure fully defines the dimensions and color information for a device-independent bitmap that is compatible with Microsoft OS/2 Presentation Manager versions 1.1 and 1.2 bitmaps.

```c
typedef struct _BITMAPCOREINFO {
   BITMAPCOREHEADER bmciHeader;
   RGBTRIPLE bmciColors[];
} BITMAPCOREINFO;
```

The BITMAPCOREINFO structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmciHeader</td>
<td>Specifies a BITMAPCOREHEADER data structure that contains information about the dimensions and color format of a device-independent bitmap.</td>
</tr>
<tr>
<td>bmciColors</td>
<td>Specifies an array of RGBTRIPLE data structures that define the colors in the bitmap.</td>
</tr>
</tbody>
</table>

Comments

An OS/2 Presentation Manager device-independent bitmap consists of two distinct parts: a BITMAPCOREINFO data structure that describes the dimensions and colors of the bitmap, and an array of bytes which define the pixels of the bitmap. The bits in the array are packed together, but each scan line must be zero-padded to end on a LONG boundary. Segment boundaries can appear anywhere in the bitmap, however. The origin of the bitmap is the lower-left corner.
The `bcBitCount` field of the `BITMAPCOREHEADER` structure determines the number of bits which define each pixel and the maximum number of colors in the bitmap. This field may be set to any of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The bitmap is monochrome, and the <code>bmciColors</code> field must contain two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the <code>bmciColors</code> table; if the bit is set, the pixel has the color of the second entry in the table.</td>
</tr>
<tr>
<td>4</td>
<td>The bitmap has a maximum of 16 colors, and the <code>bmciColors</code> field contains 16 entries. Each pixel in the bitmap is represented by a four-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, then the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the 16th table entry.</td>
</tr>
<tr>
<td>8</td>
<td>The bitmap has a maximum of 256 colors, and the <code>bmciColors</code> field contains 256 entries. In this case, each byte in the array represents a single pixel.</td>
</tr>
<tr>
<td>24</td>
<td>The bitmap has a maximum of $2^{24}$ colors. The <code>bmciColors</code> field is NULL, and each three bytes in the bitmap array represents the relative intensities of red, green, and blue, respectively, of a pixel.</td>
</tr>
</tbody>
</table>

The colors in the `bmciColors` table should appear in order of importance.

Alternatively, for functions that use device-independent bitmaps, the `bmciColors` field can be an array of 16-bit unsigned integers that specify an index into the currently realized logical palette instead of explicit RGB values. In this case, an application using the bitmap must call device-independent bitmap functions with the `wUsage` parameter set to `DIB_PAL_COLORS`.

The `bmciColors` field should not contain palette indexes if the bitmap is to be stored in a file or transferred to another application. Unless the application uses the bitmap exclusively and under its complete control, the bitmap color table should contain explicit RGB values.
**Bitmap file information**

The **BITMAPFILEHEADER** data structure contains information about the type, size, and layout of a device-independent bitmap (DIB) file.

```c
typedef struct tagBITMAPFILEHEADER {
    WORD bfType;
    DWORD bfSize;
    WORD bfReserved1;
    WORD bfReserved2;
    DWORD bfOffBits;
} BITMAPFILEHEADER;
```

The **BITMAPFILEHEADER** data structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfType</td>
<td>Specifies the type of file. It must be BM.</td>
</tr>
<tr>
<td>bfSize</td>
<td>Specifies the size in DWORDs of the file.</td>
</tr>
<tr>
<td>bfReserved1</td>
<td>Is reserved and must be set to zero.</td>
</tr>
<tr>
<td>bfReserved2</td>
<td>Is reserved and must be set to zero.</td>
</tr>
<tr>
<td>bfOffBits</td>
<td>Specifies in bytes the offset from the BITMAPFILEHEADER of the actual bitmap in the file.</td>
</tr>
</tbody>
</table>

**Comments**

A **BITMAPINFO** or **BITMAPCOREINFO** data structure immediately follows the **BITMAPFILEHEADER** structure in the DIB file.

---

**Device-independent bitmap information**

The **BITMAPINFO** structure fully defines the dimensions and color information for a Windows 3.0 device-independent bitmap.

```c
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD bmiColors[1];
} BITMAPINFO;
```

The **BITMAPINFO** structure contains the following fields:
A Windows 3.0 device-independent bitmap consists of two distinct parts: a BITMAPINFO data structure that describes the dimensions and colors of the bitmap, and an array of bytes that define the pixels of the bitmap. The bits in the array are packed together, but each scan line must be zero-padded to end on a LONG boundary. Segment boundaries can appear anywhere in the bitmap, however. The origin of the bitmap is the lower-left corner.

The biBitCount field of the BITMAPINFOHEADER structure determines the number of bits which define each pixel and the maximum number of colors in the bitmap. This field may be set to any of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The bitmap is monochrome, and the bmiColors field must contain two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the bmiColors table; if the bit is set, the pixel has the color of the second entry in the table.</td>
</tr>
<tr>
<td>4</td>
<td>The bitmap has a maximum of 16 colors, and the bmiColors field contains up to 16 entries. Each pixel in the bitmap is represented by a four-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, then the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the 16th table entry.</td>
</tr>
<tr>
<td>8</td>
<td>The bitmap has a maximum of 256 colors, and the bmiColors field contains up to 256 entries. In this case, each byte in the array represents a single pixel.</td>
</tr>
<tr>
<td>24</td>
<td>The bitmap has a maximum of 2^{24} colors. The bmiColors field is NULL, and each three bytes in the bitmap array represents the relative intensities of red, green, and blue, respectively, of a pixel.</td>
</tr>
</tbody>
</table>

The biClrUsed field of the BITMAPINFOHEADER structure specifies the number of color indexes in the color table actually used by the bitmap. If the biClrUsed field is set to 0, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount field.

The colors in the bmiColors table should appear in order of importance.

Alternatively, for functions that use device-independent bitmaps, the bmiColors field can be an array of 16-bit unsigned integers that specify an...
index into the currently realized logical palette instead of explicit RGB values. In this case, an application using the bitmap must call device-independent bitmap functions with the wUsage parameter set to DIB_PAL_COLORS.

The bmiColors field should not contain palette indices if the bitmap is to be stored in a file or transferred to another application. Unless the application uses the bitmap exclusively and under its complete control, the bitmap color table should contain explicit RGB values.

The BITMAPINFOHEADER structure contains information about the dimensions and color format of a Windows 3.0 device-independent bitmap.

```c
typedef struct tagBITMAPINFOHEADER{
    DWORD biSize;
    DWORD biWidth;
    DWORD biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    DWORD biXPelsPerMeter;
    DWORD biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;
```

The BITMAPINFOHEADER structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>biSize</td>
<td>Specifies the number of bytes required by the BITMAPINFOHEADER structure.</td>
</tr>
<tr>
<td>biWidth</td>
<td>Specifies the width of the bitmap in pixels.</td>
</tr>
<tr>
<td>biHeight</td>
<td>Specifies the height of the bitmap in pixels.</td>
</tr>
<tr>
<td>biPlanes</td>
<td>Specifies the number of planes for the target device and must be set to 1.</td>
</tr>
<tr>
<td>biBitCount</td>
<td>Specifies the number of bits per pixel. This value must be 1, 4, 8, or 24.</td>
</tr>
</tbody>
</table>
biCompression

Specifies the type of compression for a compressed bitmap. It can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI_RGB</td>
<td>Specifies that the bitmap is not compressed.</td>
</tr>
<tr>
<td>BI_RLE8</td>
<td>Specifies a run-length encoded format for bitmaps with 8 bits per pixel.</td>
</tr>
<tr>
<td></td>
<td>The compression format is a two-byte format consisting of a count byte</td>
</tr>
<tr>
<td></td>
<td>followed by a byte containing a color index. See the following &quot;Comments&quot;</td>
</tr>
<tr>
<td></td>
<td>section for more information.</td>
</tr>
<tr>
<td>BI_RLE4</td>
<td>Specifies a run-length encoded format for bitmaps with 4 bits per pixel.</td>
</tr>
<tr>
<td></td>
<td>The compression format is a two-byte format consisting of a count byte</td>
</tr>
<tr>
<td></td>
<td>followed by two word-length color indexes. See the following &quot;Comments&quot;</td>
</tr>
<tr>
<td></td>
<td>section for more information.</td>
</tr>
</tbody>
</table>

biSizeImage

Specifies the size in bytes of the image.

biXpelsPerMeter

Specifies the horizontal resolution in pixels per meter of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.

biYpelsPerMeter

Specifies the vertical resolution in pixels per meter of the target device for the bitmap.

biClrUsed

Specifies the number of color indexes in the color table actually used by the bitmap. If this value is 0, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount field. See the description of the BITMAPINFO data structure earlier in this chapter for more information on the maximum sizes of the color table. If biClrUsed is nonzero, then the biClrUsed field specifies the actual number of colors which the graphics engine or device driver will access if the biBitCount field is less than 24. If the biBitCount field is set to 24, the biClrUsed field specifies the size of the reference color table used to optimize performance of Windows color palettes. If the bitmap is a "packed" bitmap (that is, a bitmap in which the bitmap array immediately follows the BITMAPINFO header which is referenced by a single pointer), the biClrUsed field must be set to 0 or to the actual size of the color table.

biClrImportant

Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is 0, then all colors are important.

Comments

The BITMAPINFO data structure combines the BITMAPINFOHEADER structure and a color table to provide a complete definition of the dimensions and colors of a Windows 3.0 device-independent bitmap. See the description of the BITMAPINFO data structure for more information about specifying a Windows 3.0 device-independent bitmap.
An application should use the information stored in the `biSize` field to locate the color table in a `BITMAPINFO` data structure with a method such as the following:

```c
pColor = ((LPSTR) pBitmapInfo + (WORD) (pBitmapInfo -> biSize))
```

**Bitmap compression formats**

Windows supports formats for compressing bitmaps that define their colors with 8 bits per pixel and with 4 bits per pixel. Compression reduces the disk and memory storage required for the bitmap. The following paragraphs describe these formats.

When the `biCompression` field is set to `BI_RLE8`, the bitmap is compressed using a run-length encoding format for an 8-bit bitmap. This format may be compressed in either of two modes:

- Encoded
- Absolute

Both modes can occur anywhere throughout a single bitmap.

Encoded mode consists of two bytes: the first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte. In addition, the first byte of the pair can be set to zero to indicate an escape that denotes an end of line, end of bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair. The following list shows the meaning of the second byte:

<table>
<thead>
<tr>
<th>Second Byte Of Escape</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End of line.</td>
</tr>
<tr>
<td>1</td>
<td>End of bitmap.</td>
</tr>
<tr>
<td>2</td>
<td>Delta. The two bytes following the escape contain unsigned values indicating the horizontal and vertical offset of the next pixel from the current position.</td>
</tr>
</tbody>
</table>

Absolute mode is signalled by the first byte set to zero and the second byte set to a value between 03H and FFH. In absolute mode, the second byte represents the number of bytes which follow, each of which contains the color index of a single pixel. When the second byte is set to 2 or less, the escape has the same meaning as in encoded mode. In absolute mode, each run must be aligned on a word boundary.

The following example shows the hexadecimal values of an 8-bit compressed bitmap:

```
03 04 05 06 00 03 45 56 67 00 02 78 00 02 05 01
```
This bitmap would expand as follows (two-digit values represent a color index for a single pixel):

04 04 04
06 06 06 06 06
45 56 67
78 78
move current position 5 right and 1 down
78 78
end of line
1E 1E 1E 1E 1E 1E 1E 1E

end of RLE bitmap

When the biCompression field is set to BI_RLE4, the bitmap is compressed using a run-length encoding format for a 4-bit bitmap, which also uses encoded and absolute modes. In encoded mode, the first byte of the pair contains the number of pixels to be drawn using the color indexes in the second byte. The second byte contains two color indexes, one in its high-order nibble (that is, its low-order four bits) and one in its low-order nibble. The first of the pixels is drawn using the color specified by the high-order nibble, the second is drawn using the color in the low-order nibble, the third is drawn with the color in the high-order nibble, and so on, until all the pixels specified by the first byte have been drawn.

In absolute mode, the first byte contains zero, the second byte contains the number of color indexes that follow, and subsequent bytes contain color indexes in their high- and low-order nibbles, one color index for each pixel. In absolute mode, each run must be aligned on a word boundary. The end-of-line, end-of-bitmap, and delta escapes also apply to BI_RLE4.

The following example shows the hexadecimal values of a 4-bit compressed bitmap:

03 04 05 06 00 06 45 56 67 00 04 78 00 02 05 01
04 78 00 00 09 1E 00 01

This bitmap would expand as follows (single-digit values represent a color index for a single pixel):

0 4 0
0 6 0 6 0
4 5 5 6 6 7
7 8 7 8
move current position 5 right and 1 down
7 8 7 8
end of line
1 E 1 E 1 E 1 E 1 E
end of RLE bitmap

Chapter 7, Data types and structures
The **CLIENTCREATESTRUCT** data structure contains information about the menu and first multiple document interface (MDI) child window of an MDI client window. An application passes a long pointer to this structure as the `lpParam` parameter of the `CreateWindow` function when creating an MDI client window.

```c
typedef struct tagCLIENTCREATESTRUCT
{
    HMENU hWindowMenu;
    WORD idFirstChild;
} CLIENTCREATESTRUCT;
```

The **CLIENTCREATESTRUCT** structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hWindowMenu</td>
<td>Is the menu handle of the application's Window menu. An application can retrieve this handle from the MDI frame window's menu using the <code>GetSubMenu</code> function.</td>
</tr>
<tr>
<td>idFirstChild</td>
<td>Is the child window ID of the first MDI child window created. Windows increments the ID for each additional MDI child window that the application creates, and reassigns identifiers when the application destroys a window to keep the range of identifiers continuous. These identifiers are used in WM_COMMAND messages to the application's MDI frame window when a child window is selected from the Window menu, and should not conflict with any other command identifiers.</td>
</tr>
</tbody>
</table>

**COLORREF**

A **COLORREF** color value is a long integer that specifies a color. GDI functions that require a color (such as `CreatePen` and `FloodFill`) accept a **COLORREF** value as a parameter. Depending on how an application uses the **COLORREF** value, the value has three distinct forms. It may specify any of the following:

- Explicit values for red, green, and blue (RGB)
- An index into a logical color palette
- A palette-relative RGB value
TColorRef = Longint;

**Explicit RGB**

When specifying an explicit RGB value, the **COLORREF** value has the following hexadecimal form:

```
0x00bbggrr
```

The low-order byte contains a value for the relative intensity of red; the second byte contains a value for green, and the third byte contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is FF (hexadecimal). The following list illustrates the hexadecimal values that produce the indicated colors.

<table>
<thead>
<tr>
<th>Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000ff</td>
<td>Pure red</td>
</tr>
<tr>
<td>0x0000ff00</td>
<td>Pure green</td>
</tr>
<tr>
<td>0x00ff0000</td>
<td>Pure blue</td>
</tr>
<tr>
<td>0x00000000</td>
<td>Black</td>
</tr>
<tr>
<td>0x00ffffff</td>
<td>White</td>
</tr>
<tr>
<td>0x08080808</td>
<td>Medium gray</td>
</tr>
</tbody>
</table>

The **RGB** macro accepts values for red, green, and blue, and returns an explicit RGB **COLORREF** value.

**Palette index**

When specifying an index into a logical color palette, the **COLORREF** value has the following hexadecimal form:

```
0x0100iiii
```

The two low-order bytes consist of a 16-bit integer specifying an index into a logical palette. The third byte is not used and must be zero. The fourth (high-order) byte must be set to 1.

For example, the hexadecimal value 0x01000000 specifies the color in the palette entry of index 0; 0x0100000C specifies the color in the entry of index 12, and so on.

The **PALETTEINDEX** macro accepts an integer representing an index into a logical palette and returns a palette-index **COLORREF** value.

**Palette-relative rgb**

When specifying a palette-relative RGB value, the **COLORREF** value has the following hexadecimal form:

```
0x02bbggrr
```

As with an explicit RGB, the three low-order bytes contain values for red, green, and blue; the high-order byte must be set to 2.
For output devices that support logical palettes, Windows matches a palette-relative RGB value to the nearest color in the logical palette of the device context, as though the application had specified an index to that palette entry. If an output device does not support a system palette, then Windows uses the palette-relative RGB as though it were an explicit RGB COLORREF value.

The PALETTERGB macro accepts values for red, green, and blue, and returns a palette-relative RGB COLORREF value.

Before passing a palette-index or palette-relative RGB COLORREF value to a function that also requires a device-context parameter, an application that uses its own palette must select its palette into the device context (by calling the SelectPalette function) and realize the palette (by calling RealizePalette). This ensures that the function will use the correct palette-entry color. For functions that create an object (such as CreatePen), the application must select and realize the palette before selecting the object for the device context.

The COMPAREITEMSTRUCT structure supplies the identifiers and application-supplied data for two items in a sorted owner-draw combo box or list box.

Whenever an application adds a new item to an owner-draw combo or list box created with the CBS_SORT or LBS_SORT style, Windows sends the owner a WM_COMPAREITEM message. The lParam parameter of the message contains a long pointer to a COMPAREITEMSTRUCT data structure. When the owner receives the message, the owner compares the two items and returns a value indicating which item sorts before the other. For more information, see the description of the WM_COMPAREITEM message in Chapter 6, "Messages directory," in Reference, Volume 1.

```c
typedef struct tagCOMPAREITEMSTRUCT {
    WORD CtlType;
    WORD CtlID;
    HWND hwndItem;
    WORD itemID1;
    DWORD itemData1;
    WORD itemID2;
    DWORD itemData2;
} COMPAREITEMSTRUCT;
```

TCompareItemStruct = record
    CtlType: Word;
    CtlID: Word;
    hwndItem: HWnd;
    itemID1: Word;
    itemData1: Longint;
    itemID2: Word;
    itemData2: Longint;
end;

Owner-draw item-sorting information

Software development kit
The **COMPAREITEMSTRUCT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtlType</td>
<td>Is ODT_LISTBOX (which specifies an owner-draw list box) or ODT_COMBOBOX</td>
</tr>
<tr>
<td></td>
<td>(which specifies an owner-draw combo box).</td>
</tr>
<tr>
<td>CtlID</td>
<td>Is the control ID for the list box or combo box.</td>
</tr>
<tr>
<td>hwndltem</td>
<td>Is the window handle of the control.</td>
</tr>
<tr>
<td>ItemCount1</td>
<td>Is the index of the first item in the list box or combo box being</td>
</tr>
<tr>
<td></td>
<td>compared.</td>
</tr>
<tr>
<td>ItemData1</td>
<td>Is application-supplied data for the first item being compared. This value</td>
</tr>
<tr>
<td></td>
<td>was passed as the IParam parameter of the message that added the item to</td>
</tr>
<tr>
<td></td>
<td>the combo or list box.</td>
</tr>
<tr>
<td>ItemCount2</td>
<td>Is the index of the second item in the list box or combo box being</td>
</tr>
<tr>
<td></td>
<td>compared.</td>
</tr>
<tr>
<td>ItemData2</td>
<td>Is application-supplied data for the second item being compared. This</td>
</tr>
<tr>
<td></td>
<td>value was passed as the IParam parameter of the message that added the</td>
</tr>
<tr>
<td></td>
<td>item to the combo or list box.</td>
</tr>
</tbody>
</table>

The **COMSTAT** structure contains information about a communications device.

```c
typedef struct tagCOMSTAT {
    BYTE fCtsHold: 1;
    BYTE fDsrHold: 1;
    BYTE fRlsdHold: 1;
    BYTE fXoffHold: 1;
    BYTE fXoffSent: 1;
    BYTE fEof: 1;
    BYTE fTxim: 1;
    WORD cbInQue;
    WORD cbOutQue;
} COMSTAT;
```

The **COMSTAT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fCtsHold: 1</td>
<td>Specifies whether transmission is waiting for the clear-to-send (CTS)</td>
</tr>
<tr>
<td></td>
<td>signal to be sent.</td>
</tr>
<tr>
<td>fDsrHold: 1</td>
<td>Specifies whether transmission is waiting for the data-set-ready (DSR)</td>
</tr>
<tr>
<td></td>
<td>signal to be sent.</td>
</tr>
<tr>
<td>fRlsdHold:1</td>
<td>Specifies whether transmission is waiting for the receive-line-signal-detect</td>
</tr>
<tr>
<td></td>
<td>(RLSD) signal to be sent.</td>
</tr>
</tbody>
</table>
COMSTAT

fXoffHold: 1
Specifies whether transmission is waiting as a result of the XoffChar character being received.

fXoffSent: 1
Specifies whether transmission is waiting as a result of the XoffChar character being transmitted. Transmission halts when the XoffChar character is transmitted and used by systems that take the next character as XON, regardless of the actual character.

fEof: 1
Specifies whether the EofChar character has been received.

fTxim: 1
Specifies whether a character is waiting to be transmitted.

cbInQue
Specifies the number of characters in the receive queue.

cbOutQue
Specifies the number of characters in the transmit queue.

See also
The GetCommError function in Chapter 4, "Functions directory," in Reference, Volume 1.

CREATESTRUCT

Window-creation structure

The CREATESTRUCT structure defines the initialization parameters passed to an application's window function.

typedef struct tagCREATESTRUCT { TCreateStruct = record
  LPSTR lpCreateParams;
  HANDLE hInstance;
  HANDLE hMenu;
  HWND hwndParent;
  int cy;
  int cx;
  int y;
  int x;
  long style;
  LPSTR lpszName;
  LPSTR lpszClassName;
  long dwExStyle;
} CREATESTRUCT;

The CREATESTRUCT structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpCreateParams</td>
<td>Points to data to be used for creating the window.</td>
</tr>
<tr>
<td>hInstance</td>
<td>Identifies the module-instance handle of the module that owns the new window.</td>
</tr>
<tr>
<td>hMenu</td>
<td>Identifies the menu to be used by the new window.</td>
</tr>
<tr>
<td>hwndParent</td>
<td>Identifies the window that owns the new window. This field is NULL if the new window is a top-level window.</td>
</tr>
<tr>
<td>cy</td>
<td>Specifies the height of the new window.</td>
</tr>
</tbody>
</table>
**CREATESTRUCT**

- **cx**
  Specifies the width of the new window.

- **y**
  Specifies the y-coordinate of the upper-left corner of the new window. Coordinates are relative to the parent window if the new window is a child window. Otherwise, the coordinates are relative to the screen origin.

- **x**
  Specifies the x-coordinate of the upper-left corner of the new window. Coordinates are relative to the parent window if the new window is a child window. Otherwise, the coordinates are relative to the screen origin.

- **style**
  Specifies the new window's style.

- **lpszName**
  Points to a null-terminated character string that specifies the new window's name.

- **lpszClass**
  Points to a null-terminated character string that specifies the new window's class name.

- **ExStyle**
  Specifies extended style for the new window.

---

**DCB**

The **DCB** structure defines the control setting for a serial communications device.

```c
typedef struct tagDCB {
    BYTE Id;
    WORD BaudRate;
    BYTE ByteSize;
    BYTE Parity;
    BYTE StopBits;
    WORD RlsTimeout;
    WORD CtsTimeout;
    WORD DsrTimeout;
    BYTE fBinary: 1;
    BYTE fRtsDisable: 1;
    BYTE fParity: 1;
    BYTE fOutxCtsFlow: 1;
    BYTE fOutxDsrFlow: 1;
    BYTE fDummy: 2;
    BYTE fDtrDisable: 1;
    BYTE fOutX: 1;
    BYTE fInX: 1;
    BYTE fPeChar: 1;
    BYTE fNull: 1;
    BYTE fChEvt: 1;
    BYTE fDtrflow: 1;
    BYTE fRtsflow: 1;
} TDCB = record
    Id: Byte;
    BaudRate: Word;
    ByteSize: Byte;
    Parity: Byte;
    StopBits: Byte;
    RlsTimeout: Word;
    CtsTimeout: Word;
    DsrTimeout: Word;
    Flags: Word;
    XonChar: Char;
    XoffChar: Char;
    XonLim: Word;
    XoffLim: Word;
    PeChar: Char;
    EofChar: Char;
    EvtChar: Char;
    TxDelay: Word;
end;
```
BYTE fDummy2: 1;
char XonChar;
char XoffChar;
WORD XonLim;
WORD XoffLim;
char PeChar;
char EofChar;
char EvtChar;
WORD TxDelay;
} DCB;

The **DCB** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Specifies the communication device. This value is set by the device driver. If the most significant bit is set, then the <strong>DCB</strong> structure is for a parallel device.</td>
</tr>
<tr>
<td>BaudRate</td>
<td>Specifies the baud rate at which the communications device operates.</td>
</tr>
<tr>
<td>ByteSize</td>
<td>Specifies the number of bits in the characters transmitted and received. The <strong>ByteSize</strong> field can be any number from 4 to 8.</td>
</tr>
<tr>
<td>Parity</td>
<td>Specifies the parity scheme to be used. The <strong>Parity</strong> field can be any one of the following values:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>EVENPARITY</td>
</tr>
<tr>
<td></td>
<td>MARKPARITY</td>
</tr>
<tr>
<td></td>
<td>NOPARITY</td>
</tr>
<tr>
<td></td>
<td>ODDPARITY</td>
</tr>
<tr>
<td></td>
<td>SPACEPARITY</td>
</tr>
<tr>
<td>StopBits</td>
<td>Specifies the number of stop bits to be used. The <strong>StopBits</strong> field can be any one of the following values:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>ONESTOPBIT</td>
</tr>
<tr>
<td></td>
<td>ONE5STOPBITS</td>
</tr>
<tr>
<td></td>
<td>TWOSTOPBITS</td>
</tr>
<tr>
<td>RlsTimeout</td>
<td>Specifies the maximum amount of time (in milliseconds) the device should wait for the receive-line-signal-detect (RLSD) signal. (RLSD is also known as the carrier detect (CD) signal.)</td>
</tr>
<tr>
<td>CtsTimeout</td>
<td>Specifies the maximum amount of time (in milliseconds) the device should wait for the clear-to-send (CTS) signal.</td>
</tr>
<tr>
<td>DsrTimeout</td>
<td>Specifies the maximum amount of time (in milliseconds) the device should wait for the data-set-ready (DSR) signal.</td>
</tr>
<tr>
<td>fBinary: 1</td>
<td>Specifies binary mode. In nonbinary mode, the <strong>EofChar</strong> character is recognized on input and remembered as the end of data.</td>
</tr>
<tr>
<td>fRtsDisable: 1</td>
<td>Specifies whether or not the request-to-send (RTS) signal is disabled. If the <strong>fRtsDisable</strong> field is set, RTS is not used and</td>
</tr>
</tbody>
</table>
remains low. If fRtsDisable is clear, RTS is sent when the device is opened and turned off when the device is closed. Specifies whether parity checking is enabled. If the fParity field is set, parity checking is performed and errors are reported.

fOutxCtsFlow: 1 Specifies that clear-to-send (CTS) signal is to be monitored for output flow control. If the fOutxCtsFlow field is set and CTS is turned off, output is suspended until CTS is again sent.

fOutxDsrFlow: 1 Specifies that the data-set-ready (DSR) signal is to be monitored for output flow control. If the fOutxDsrFlow field is set and DSR is turned off, output is suspended until DSR is again sent.

fDummy: 2 Reserved.

fDtrDisable: 1 Specifies whether the data-terminal-ready (DTR) signal is disabled. If the fDtrDisable field is set, DTR is not used and remains low. If fDtrDisable is clear, DTR is sent when the device is opened and turned off when the device is closed.

fOutX: 1 Specifies that XON/XOFF flow control is used during transmission. If the fOutX field is set, transmission stops when the XoffChar character is received, and starts again when the XonChar character is received.

fInX: 1 Specifies that XON/XOFF flow control is used during reception. If the fInX field is set, the XonChar character is sent when the receive queue comes within XoffLim characters of being full, and the XonChar character is sent when the receive queue comes within XonLim characters of being empty.

fPeChar: 1 Specifies that characters received with parity errors are to be replaced with the character specified by the fPeChar field. The fParity field must be set for the replacement to occur.

fNull: 1 Specifies that received null characters are to be discarded.

fChEvt: 1 Specifies that reception of the EvtChar character is to be flagged as an event.

fDtrflow: 1 Specifies that the data-terminal-ready (DTR) signal is to be used for receive flow control. If the fDtrflow field is set, DTR is turned off when the receive queue comes within XoffLim characters of being full, and sent when the receive queue comes within XonLim characters of being empty.

fRtsflow: 1 Specifies that the ready-to-send (RTS) signal is to be used for receive flow control. If the fRtsflow field is set, RTS is turned off when the receive queue comes within XoffLim characters of being full, and sent when the receive queue comes within XonLim characters of being empty.

fdummy2: 1 Reserved.

XonChar Specifies the value of the XON character for both transmission and reception.

XoffChar Specifies the value of the XOFF character for both transmission and reception.

XonLim Specifies the minimum number of characters allowed in the receive queue before the XON character is sent.

XoffLim Specifies the maximum number of characters allowed in the receive queue before the XOFF character is sent. The XoffLim
value is subtracted from the size of the receive queue (in bytes) to calculate the maximum number of characters allowed.

**PeChar**
Specifies the value of the character used to replace characters received with a parity error.

**EofChar**
Specifies the value of the character used to signal the end of data.

**EvtChar**
Specifies the value of the character used to signal an event.

**TxDelay**
Not currently used.

**See also**
The `BuildCommDCB`, `GetCommState`, and `SetCommState` functions in Chapter 4, "Functions directory," in Reference, Volume 1.

### DELETEITEMSTRUCT

The **DELETEITEMSTRUCT** structure describes a deleted owner-draw list-box or combo-box item. When an item is removed from the list box or combo box, or when the list box or combo box is destroyed, Windows sends the WM_DELETEITEM message to the owner for each deleted item; the `IParam` parameter of the message contains a pointer to this structure.

```c
typedef struct tagDELETEITEMSTRUCT
{
    WORD    CtlType;
    WORD    CtlID;
    WORD    itemID;
    HWND    hwndItem;
    DWORD   itemData;
} DELETEITEMSTRUCT;
```

The **DELETEITEMSTRUCT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtlType</td>
<td>Is ODT_LISTBOX (which specifies an owner-draw list box) or ODT_COMBOBOX (which specifies an owner-draw combo box).</td>
</tr>
<tr>
<td>CtlID</td>
<td>Is the control ID for the list box or combo box.</td>
</tr>
<tr>
<td>itemID</td>
<td>Is the index of the item in the list box or combo box being removed.</td>
</tr>
<tr>
<td>hwndItem</td>
<td>Is the window handle of the control.</td>
</tr>
<tr>
<td>itemData</td>
<td>Contains the value passed to the control in the <code>IParam</code> parameter of the LB_INSERTSTRING, LB_ADDSTRING, CB_INSERTSTRING, or CB_ADDSTRING message when the item was added to the list box.</td>
</tr>
</tbody>
</table>
The **DEVMODE** data structure contains information about the device initialization and environment of a printer driver. An application passes this structure to the **DeviceCapabilities** and **ExtDeviceMode** functions.

```c
typedef struct _devicemode {
    char    dmDeviceName[32];
    WORD    dmSpecVersion;
    WORD    dmDriverVersion;
    WORD    dmSize;
    WORD    dmDriverExtra;
    DWORD   dmFields;
    short   dmOrientation;
    short   dmPaperSize;
    short   dmPaperLength;
    short   dmPaperWidth;
    short   dmScale;
    short   dmCopies;
    short   dmDefaultSource;
    short   dmPrintQuality;
    short   dmColor;
    short   dmDuplex;
    BYTE   .dmDriverData[dmDriverExtra];
} DEVMODE;
```

The **DEVMODE** structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmDeviceName</td>
<td>Specifies the name of the device the driver supports; for example, &quot;PCL/HP LaserJet&quot; in the case of PCL/HP® LaserJet®. This string is unique among device drivers.</td>
</tr>
<tr>
<td>dmSpecVersion</td>
<td>Specifies the version number of the initialization data specification upon which the structure is based. The version number follows the Windows version number and is currently 0x300.</td>
</tr>
<tr>
<td>dmDriverVersion</td>
<td>Specifies the printer driver version number assigned by the printer driver developer.</td>
</tr>
<tr>
<td>dmSize</td>
<td>Specifies the size in bytes of the <strong>DEVMODE</strong> structure except the <strong>dmDriverData</strong> (device-specific) field. If an application manipulates only the driver-independent portion of the data, it can use this field to determine the length of the structure without having to account for different versions.</td>
</tr>
</tbody>
</table>
DEVMODE

**dmDriverExtra**
Contains the size of the **dmDriverData** field and is the length of the device-specific data in the **DEVMODE** structure. If an application does not use device-specific information, it should set this field to zero.

**dmFields**
Is a bitfield that specifies which of the remaining fields in the **DEVMODE** structure have been initialized. Bit 0 (defined as DM_ORIENTATION) corresponds to **dmOrientation**; bit 1 (defined as DM_PAPERSIZE) specifies **dmPaperSize**, and so on. A printer driver supports only those fields that are appropriate for the printer technology.

**dmOrientation**
Selects the orientation of the paper. It can be either DMORIENT_PORTrait (1) or DMORIENT_LANDScape (2).

**dmPaperSize**
Selects the size of the paper to print on. This field may be set to zero if the length and width of the paper are both set by the **dmPaperLength** and **dmPaperWidth** fields. Otherwise, the **dmPaperSize** field can be set to one of the following predefined values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMPAPER_LETTER</td>
<td>8/2-by-11-inch paper</td>
</tr>
<tr>
<td>DMPAPER_LEGAL</td>
<td>8/2-by-14-inch paper</td>
</tr>
<tr>
<td>DMPAPER_A4</td>
<td>210-by-297-millimeter paper</td>
</tr>
<tr>
<td>DMPAPER_CSheet</td>
<td>17-by-22-inch paper</td>
</tr>
<tr>
<td>DMPAPER_DSheet</td>
<td>22-by-34-inch paper</td>
</tr>
<tr>
<td>DMPAPER_ESheet</td>
<td>34-by-44-inch paper</td>
</tr>
<tr>
<td>DMPAPER_ENV_9</td>
<td>3/8-by-8/8-inch #9 envelope</td>
</tr>
<tr>
<td>DMPAPER_ENV_10</td>
<td>4/8-by-9/5-inch #10 envelope</td>
</tr>
<tr>
<td>DMPAPER_ENV_11</td>
<td>4/2-by-10/8-inch #11 envelope</td>
</tr>
<tr>
<td>DMPAPER_ENV_12</td>
<td>4/4-by-11-inch #12 envelope</td>
</tr>
<tr>
<td>DMPAPER_ENV_14</td>
<td>5-by-11/2-inch #14 envelope</td>
</tr>
</tbody>
</table>

**dmPaperLength**
Overrides the length of the paper specified by the **dmPaperSize** field, either for custom paper sizes or for devices such as dot-matrix printers which can print on a page of arbitrary length. These values, along with all other values which specify a physical length, are in tenths of a millimeter.

**dmPaperWidth**
Overrides the width of the paper specified by the **dmPaperSize** field.

**dmScale**
Scales the printed output. The apparent page size is scaled by a factor of **dmScale**/100 from the physical page size. A letter-size paper with a **dmScale** value of 50 would appear to be 17 by 22 inches, and output text and graphics would be correspondingly half their normal height and width.

**dmCopies**
Selects the number of copies printed if the device supports multiple-page copies.

**dmDefaultSource**
Specifies the paper bin from which the paper is fed by default. The application can override this selection by using the GETSETPAPERBINS escape. Possible bins include the following:
dmPrintQuality

Specifies the printer resolution. There are four predefined device-independent values:

- DMRRES_HIGH (-4)
- DMRRES_MEDIUM (-3)
- DMRRES_LOW (-2)
- DMRRES_DRAFT (-1)

If a positive value is given, it specifies the number of dots per inch (DPI) and is therefore device dependent.

dmColor

Switches between color and monochrome on color printers. Possible values are:

- DMCOLOR_COLOR (1)
- DMCOLOR_MONOCHROME (2).

dmDuplex

Selects duplex or double-sided printing for printers capable of duplex printing. Values for this field include:

- DMDUP_SIMPLEX (1)
- DMDUP_HORIZONTAL (2)
- DMDUP_VERTICAL (3).

dmDriverData[ ]

Contains device-specific data defined by the device driver.

Comments

Only drivers fully updated for Windows version 3.0 and which export the ExtDeviceMode function use the DEVMODE data structure.
The **DLGTEMPLATE** defines the contents of a dialog box. This structure is divided into three distinct parts:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header Data Structure</td>
<td>Contains a general description of the dialog box.</td>
</tr>
<tr>
<td>Font-Information Data Structure</td>
<td>Defines the font with which text is drawn in the dialog box. This part is optional.</td>
</tr>
<tr>
<td>List of Items</td>
<td>Describes the parts that compose the dialog box.</td>
</tr>
</tbody>
</table>

The `CreateDialogIndirect`, `CreateDialogIndirectParam`, `DialogBoxIndirect`, and `DialogBoxIndirectParam` functions use this structure.

The **DLGTEMPLATE** header is shown here:

```c
typedef struct {
    long dtStyle;
    BYTE dtItemCount;
    int  dtX;
    int  dtY;
    int  dtCX;
    int  dtCY;
    char dtMenuName[];
    char dtClassName[];
    char dtCaptionText[];
} DLGTEMPLATE;
```

The **DLGTEMPLATE** header has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtStyle</td>
<td>Specifies the style of the dialog box. This field may be any or all of these values:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS_LOCALEEDIT</td>
<td>Specifies that text storage for edit controls will be allocated in the application's local data segment. This allows the use of the EM_GETHANDLE and EM_SETHANDLE messages. If</td>
</tr>
</tbody>
</table>
this style is not specified, edit-control data is located in a separate global data block.

**DS_SYSMODAL**
Specifies a system-modal dialog box.

**DS_MODALFRAME**
 Specifies a dialog box with a modal dialog-box border. This style can be combined with the WS_CAPTION and WS_SYSMENU style flags to create a dialog box with a title bar and System menu.

**DS_ABSALIGN**
Indicates that dtX and dtY are relative to the screen origin, not to the owner of the dialog box.

**DS_SETFONT**
Specifies that a font other than the system font is to be used to draw text in the dialog box. If this flag is set, the FONTINFO data structure described in the following paragraphs must immediately follow the DLGTEMPLATE header. When Windows creates a dialog box with this attribute, Windows sends the WM_SETFONT message to the dialog-box window prior to creating the controls.

**DS_NOIDLEMSG**
Specifies that Windows will not send the WM_ENTERIDLE message to the owner of the dialog box while the dialog box is displayed.

dtItemCount:
Specifies the number of items in the dialog box. A dialog box can contain up to 255 controls.

dtX:
Specifies the x-coordinate of the upper-left corner of the dialog box in units of /4 of the current dialog base width unit. The dialog base units are computed from the height and width of the current system font; the GetDialogBaseUnits function returns the current dialog base units in pixels. Unless DS_ABSALIGN is set in the dtStyle field, this value is relative to the origin of the parent window's client area.

dtY:
Specifies the y-coordinate of the upper-left corner of the dialog box in units of /8 of the current dialog base height unit. Unless DS_ABSALIGN is set in the dtStyle field, this value is relative to the origin of the parent window's client area.

dtCX:
Specifies the width of the dialog box in units of /4 of the dialog base width unit.

dtCY:
Specifies the height of the dialog box in units of /8 of the dialog base height unit.

Chapter 7, Data types and structures
The **FONTINFO** data structure contains information about the point size and face name of the font which Windows is to use to draw text in the dialog box.

```c
typedef struct {
    short int PointSize;
    char szTypeFace[]; /* A null-terminated string */
} FONTINFO;
```

The **FONTINFO** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointSize</td>
<td>Specifies the size of the typeface in points.</td>
</tr>
<tr>
<td>szTypeFace</td>
<td>Specifies the name of the typeface; for example, &quot;Courier&quot;.</td>
</tr>
</tbody>
</table>

**Comments**

The font specified must have been previously loaded, either from WIN.INI or explicitly by calling the `LoadFont` function.

**Item list**

The item list consists of one or more **DLGITEMTEMPLATE** data structures, one for each control in the dialog box. The first such structure immediately follows the **FONTINFO** structure or the header at the first byte after the terminating null character in the `szTypeFace` field or the `dtCaptionText[]` field. The following shows the format of the **DLGITEMTEMPLATE** structure.

```c
typedef struct {
    int dti1X;
    int dti1Y;
    int dti1CX;
    int dti1CY;
    int dti1ID;
} DLGITEMTEMPLATE;
```
The **DLGITEMTEMPLATE** data structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtiIX</td>
<td>Specifies the x-coordinate of the upper-left corner of the dialog-box item in units of /4 of the current dialog base width unit, relative to the origin of the dialog box. The dialog base units are computed from the height and width of the current system font. The <strong>GetDialogBaseUnits</strong> function returns the current dialog base units in pixels.</td>
</tr>
<tr>
<td>dtiY</td>
<td>Specifies the y-coordinate of the upper-left corner of the dialog-box item in units of /8 of the current dialog base height unit. This value is relative to the origin of the dialog box.</td>
</tr>
<tr>
<td>dtiCX</td>
<td>Specifies the width-extent of the dialog-box item in units of /4 of the current dialog base width unit. Dialog base units are computed from the height and width of the current system font. The <strong>GetDialogBaseUnits</strong> function returns the current dialog base units.</td>
</tr>
<tr>
<td>dtiCY</td>
<td>Specifies the height of the dialog-box item in units of /8 of the dialog base height unit.</td>
</tr>
<tr>
<td>dtillD</td>
<td>Specifies the dialog-box item identification number.</td>
</tr>
<tr>
<td>dtiStyle</td>
<td>Specifies the style of the dialog-box item.</td>
</tr>
<tr>
<td>dtiClass[ ]</td>
<td>A null-terminated string that specifies the control's class. It may be one of the following class names:</td>
</tr>
<tr>
<td></td>
<td>□ BUTTON</td>
</tr>
<tr>
<td></td>
<td>□ EDIT</td>
</tr>
<tr>
<td></td>
<td>□ STATIC</td>
</tr>
<tr>
<td></td>
<td>□ LISTBOX</td>
</tr>
<tr>
<td></td>
<td>□ SCROLLBAR</td>
</tr>
<tr>
<td></td>
<td>□ COMBOBOX</td>
</tr>
<tr>
<td>dtiText[ ]</td>
<td>Specifies the text for the item; it is a null-terminated string.</td>
</tr>
<tr>
<td>dtillInfo</td>
<td>Specifies the number of bytes of additional data that follows this item description and precedes the next item description.</td>
</tr>
<tr>
<td>dtillData</td>
<td>Specifies additional data which the <strong>CreateWindow</strong> function receives through the IpCreateParams field of the <strong>CREATESTRUCT</strong> data structure. This field is zero length if dtillInfo is zero.</td>
</tr>
</tbody>
</table>
The DRAWITEMSTRUCT structure provides information the owner needs to determine how to paint an owner-draw control. The owner of the owner-draw control receives a pointer to this structure as the IParam parameter of the WM_DRAWITEM message.

```c
typedef struct tagDRAWITEMSTRUCT {
    WORD CtlType;
    WORD CtlID;
    WORD itemID;
    WORD itemAction;
    HWND hwndItem;
    HDC hDC;
    RECT rcItem;
    DWORD itemData;
} DRAWITEMSTRUCT;
```

The DRAWITEMSTRUCT structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtlType</td>
<td>Is the control type. The values for control types are as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td>ODT_BUTTON</td>
<td>Owner-draw button.</td>
</tr>
<tr>
<td>ODT_COMBOBOX</td>
<td>Owner-draw combo box.</td>
</tr>
<tr>
<td>ODT_LISTBOX</td>
<td>Owner-draw list box.</td>
</tr>
<tr>
<td>ODT_MENU</td>
<td>Owner-draw menu.</td>
</tr>
<tr>
<td>CtlID</td>
<td>Is the control ID for a combo box, list box or button. This field is not used for a menu.</td>
</tr>
<tr>
<td>itemID</td>
<td>Is the menu-item ID for a menu or the index of the item in a list box or combo box. For an empty list box or combo box, this field can be -1. This allows the application to draw only the focus rectangle at the coordinates specified by the rcItem field even though there are no items in the control. This indicates to the user whether the list box or combo box has input focus. The setting of the bits in the itemAction field determines whether the rectangle is to be drawn as though the list box or combo box has input focus.</td>
</tr>
<tr>
<td>itemAction</td>
<td>Defines the drawing action required. This will be one or more of the following bits:</td>
</tr>
</tbody>
</table>
### Value | Description
--- | ---
ODA_DRAWENTIRE | This bit is set when the entire control needs to be drawn.
ODA_FOCUS | This bit is set when the control gains or loses input focus. The **itemState** field should be checked to determine whether the control has focus.
ODA_SELECT | This bit is set when only the selection status has changed. The **itemState** field should be checked to determine the new selection state.

**itemState**

Specifies the visual state of the item after the current drawing action takes place. That is, if a menu item is to be grayed, the state flag ODS_GRAYED will be set. The state flags are:

| Value | Description |
--- | ---|
ODS_CHECKED | This bit is set if the menu item is to be checked. This bit is used only in a menu.
ODS_DISABLED | This bit is set if the item is to be drawn as disabled.
ODS_FOCUS | This bit is set if the item has input focus.
ODS_GRAYED | This bit is set if the item is to be grayed. This bit is used only in a menu.
ODS_SELECTED | This bit is set if the item's status is selected.

**hwndItem**

For combo boxes, list boxes and buttons, this field specifies the window handle of the control; for menus, it contains the handle of the menu (**HMENU**) containing the item.

**hDC**

Identifies a device context; this device context must be used when performing drawing operations on the control.

**rcltem**

Is a rectangle in the device context specified by the **hDC** field that defines the boundaries of the control to be drawn. Windows automatically clips anything the owner draws in the device context for combo boxes, list boxes, and buttons, but does not clip menu items. When drawing menu items, the owner must ensure that the owner does not draw outside the boundaries of the rectangle defined by the **rcltem** field.

**itemData**

For a combo box or list box, this field contains the value that was passed to the list box in the **IParam** parameter of one of the the following messages:

- CB_ADDSTRING
- CB_INSERTSTRING
- LB_ADDSTRING
- LB_INSERTSTRING

For a menu, this field contains the **DWORD** value passed as the **lpNewItem** parameter of the **InsertMenu** which inserted the menu item. Its contents are undefined for buttons.
The **HANDLETABLE** structure is an array of handles, each of which identifies a GDI object.

```pascal
HANDLE objectHandle[1] THandleTable = record
  objectHandle: array[0..0] of THandle;
end;
```

The **HANDLETABLE** structure has the following field:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectHandle[1]</td>
<td>Identifies an array of handles.</td>
</tr>
</tbody>
</table>

The **LOGBRUSH** structure defines the style, color, and pattern of a physical brush to be created by using the `CreateBrushIndirect` function.

```pascal
typedef struct tagLOGBRUSH {
  WORD  IbStyle;
  COLORREF IbColor;
  short int IbHatch;
} LOGBRUSH;
TLogBrush = record
  IbStyle: Word;
  IbColor: Longint;
  IbHatch: Integer;
end;
```

The **LOGBRUSH** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IbStyle</td>
<td>Specifies the brush style. The <strong>IbStyle</strong> field can be any one of the following styles:</td>
</tr>
<tr>
<td></td>
<td><strong>Style</strong></td>
</tr>
<tr>
<td></td>
<td>BS_DIBPATTERN</td>
</tr>
<tr>
<td></td>
<td>BS_HATCHED</td>
</tr>
<tr>
<td></td>
<td>BS_HOLLOW</td>
</tr>
</tbody>
</table>
LOGBRUSH

BS_PATTERN        Specifies a pattern brush defined by a memory bitmap.
BS_SOLID          Specifies a solid brush.

IbColor

Specifies the color in which the brush is to be drawn. If IbStyle is BS_HOLLOW or BS_PATTERN, IbColor is ignored. If IpStyle is BS_DIBPATTERN, the low-order word of IbColor specifies whether the bmiColors fields of the BITMAPINFO data structure contain explicit RGB values or indexes into the currently realized logical palette. The IbColor field must be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIB_PAL_COLORS</td>
<td>The color table consists of an array of 16-bit indexes into the currently realized logical palette.</td>
</tr>
<tr>
<td>DIB_RGB_COLORS</td>
<td>The color table contains literal RGB values.</td>
</tr>
</tbody>
</table>

IbHatch

Specifies a hatch style. The meaning depends on the brush style. If IbStyle is BS_DIBPATTERN, the IbHatch field contains a handle to a packed DIB. To obtain this handle, an application calls the GlobalAlloc function to allocate a block of global memory and then fills the memory with the packed DIB. A packed DIB consists of a BITMAPINFO data structure immediately followed by the array of bytes which define the pixels of the bitmap. If IbStyle is BS_HATCHED, the IbHatch field specifies the orientation of the lines used to create the hatch. It can be any one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS_BDIAGONAL</td>
<td>45-degree upward hatch (left to right)</td>
</tr>
<tr>
<td>HS_CROSS</td>
<td>Horizontal and vertical crosshatch</td>
</tr>
<tr>
<td>HS_DIAGCROSS</td>
<td>45-degree crosshatch</td>
</tr>
<tr>
<td>HS_FDIAGONAL</td>
<td>45-degree downward hatch (left to right)</td>
</tr>
<tr>
<td>HS_HORIZONTAL</td>
<td>Horizontal hatch</td>
</tr>
<tr>
<td>HS_VERTICAL</td>
<td>Vertical hatch</td>
</tr>
</tbody>
</table>

If IbStyle is BS_PATTERN, IbHatch must be a handle to the bitmap that defines the pattern. If IbStyle is BS_SOLID or BS_HOLLOW, IbHatch is ignored.

See also

The CreateBrushIndirect function in Chapter 4, "Functions directory," in Reference, Volume 1.
The **LOGFONT** structure defines the attributes of a font, a drawing object used to write text on a display surface.

```c
typedef struct tagLOGFONT {
    short int  lfHeight;
    short int  lfWidth;
    short int  lfEscapement;
    short int  lfOrientation;
    short int  lfWeight;
    BYTE      lfItalic;
    BYTE      lfUnderline;
    BYTE      lfStrikeOut;
    BYTE      lfCharSet;
    BYTE      lfOutPrecision;
    BYTE      lfClipPrecision;
    BYTE      lfQuality;
    BYTE      lfPitchAndFamily;
    BYTE      lfFaceName[LF_FACESIZE];
} LOGFONT;
```

The **LOGFONT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lfHeight</td>
<td>Specifies the average height of the font (in user units). The height of a font can be specified in the following three ways. If the <code>lfHeight</code> field is greater than zero, it is transformed into device units and matched against the cell height of the available fonts. If <code>lfHeight</code> is zero, a reasonable default size is used. If <code>lfHeight</code> is less than zero, it is transformed into device units and the absolute value is matched against the character height of the available fonts. To ensure compatibility with the font-scaling engine of future versions of Windows, <code>lfHeight</code> should be less than zero. Setting the high-order bit indicates that the font height does not take internal leading into consideration. This corresponds to the standard typographical EM height.</td>
</tr>
<tr>
<td>lfWidth</td>
<td>Specifies the average width of characters in the font (in device units). If the <code>lfWidth</code> field is zero, the aspect ratio of the device is matched against the digitization aspect ratio of the available fonts for the closest match by absolute value of the difference.</td>
</tr>
<tr>
<td>lfEscapement</td>
<td>Specifies the angle (in tenths of degrees) between the escapement vector and the x-axis of the display surface. The escapement vector is the line through the origins of the first</td>
</tr>
</tbody>
</table>
and last characters on a line. The angle is measured counterclockwise from the x-axis.

**IfOrientation**  
Specifies the angle (in tenths of degrees) between the baseline of a character and the x-axis. The angle is measured counterclockwise from the x-axis.

**IfWeight**  
Specifies the font weight (in inked pixels per 1000). Although the **IfWeight** field can be any integer value from 0 to 1000, the common values are as follows:

- 400 Normal
- 700 Bold

These values are approximate; the actual appearance depends on the font face. If **IfWeight** is zero, a default weight is used.

**IfItalic**  
Specifies an italic font if set to nonzero.

**IfUnderline**  
Specifies an underlined font if set to nonzero.

**IfStrikeOut**  
Specifies a strikeout font if set to nonzero.

**IfCharSet**  
Specifies the font's character set. The three values are predefined:

- ANSI_CHARSET
- OEM_CHARSET
- SYMBOL_CHARSET

The OEM character set is system-dependent. Fonts with other character sets may exist in the system. If an application uses a font with an unknown character set, it should not attempt to translate or interpret strings that are to be rendered with that font. Instead, the strings should be passed directly to the output device driver.

**IfOutPrecision**  
Specifies the font's output precision, which defines how closely the output must match the requested font's height, width, character orientation, escapement, and pitch. The default setting is OUT_DEFAULT_PRECIS.

**IfClipPrecision**  
Specifies the font's clipping precision, which defines how to clip characters that are partially outside the clipping region. The default setting is CLIP_DEFAULT_PRECIS.

**IfQuality**  
Specifies the font's output quality, which defines how carefully GDI must attempt to match the logical-font attributes to those of an actual physical font. It can be any one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_QUALITY</td>
<td>Appearance of the font does not matter.</td>
</tr>
<tr>
<td>DRAFT_QUALITY</td>
<td>Appearance of the font is less important than when PROOF_QUALITY is used. For GDI fonts, scaling is enabled, which means that more font sizes are available, but the quality may be lower. Bold, italic, underline, and strikeout fonts are synthesized if necessary.</td>
</tr>
</tbody>
</table>
PROOF_QUALITY

Character quality of the font is more important than exact matching of the logical-font attributes. For GDI fonts, scaling is disabled and the font closest in size is chosen. Although the chosen font size may not be mapped exactly when PROOF_QUALITY is used, the quality of the font is high and there is no distortion of appearance. Bold, italic, underline, and strikeout fonts are synthesized if necessary.

IfPitchAndFamily

Specifies the font pitch and family. The two low-order bits specify the pitch of the font and can be any one of the following values:

- DEFAULT_PITCH
- FIXED_PITCH
- VARIABLE_PITCH

The four high-order bits of the field specify the font family and can be any one of the following values:

- FF_DECORATIVE
- FF_DONTCARE
- FF_MODERN
- FF_ROMAN
- FF_SCRIPT
- FF_SWISS

The proper value can be obtained by using the Boolean OR operator to join one pitch constant with one family constant. Font families describe the look of a font in a general way. They are intended for specifying fonts when the exact typeface desired is not available. The values for font families are as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_DECORATIVE</td>
<td>Novelty fonts. Old English, for example.</td>
</tr>
<tr>
<td>FF_DONTCARE</td>
<td>Don’t care or don’t know.</td>
</tr>
<tr>
<td>FF_MODERN</td>
<td>Fonts with constant stroke width (fixed-pitch), with or without serifs. Fixed-pitch fonts are usually modern. Pica, Elite, and Courier, for example.</td>
</tr>
<tr>
<td>FF_ROMAN</td>
<td>Fonts with variable stroke width (proportionally spaced) and with serifs. Times Roman, Palatino, and Century Schoolbook, for example.</td>
</tr>
<tr>
<td>FF_SCRIPT</td>
<td>Fonts designed to look like handwriting. Script and Cursive, for example.</td>
</tr>
<tr>
<td>FF_SWISS</td>
<td>Fonts with variable stroke width (proportionally spaced) and</td>
</tr>
</tbody>
</table>

Software development kit
IfFaceName specifies the font's typeface. It must be a null-terminated character string. If **IfFaceName** is NULL, GDI uses a default typeface.

See also The **CreateFontIndirect** function in Chapter 4, "Functions directory," in Reference, Volume 1.

---

**LOGPALETTE**

Logical color palette information

The **LOGPALETTE** data structure defines a logical color palette.

```c
typedef struct
{
    WORD palVersion;
    WORD palNumEntries;
    PALETTEENTRY palPalEntry[];
} LOGPALETTE;
```

The **LOGPALETTE** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>palVersion</td>
<td>Specifies the Windows version number for the structure (currently 0x300).</td>
</tr>
<tr>
<td>palNumEntries</td>
<td>Specifies the number of palette color entries.</td>
</tr>
<tr>
<td>palPalEntry[]</td>
<td>Specifies an array of <strong>PALETTEENTRY</strong> data structures that define the color and usage of each entry in the logical palette.</td>
</tr>
</tbody>
</table>

Comments

The colors in the palette entry table should appear in order of importance. This is because entries earlier in the logical palette are most likely to be placed in the system palette.

This data structure is passed as a parameter to the **CreatePalette** function.
The **LOGPEN** structure defines the style, width, and color of a pen, a drawing object used to draw lines and borders. The **CreatePenIndirect** function uses the **LOGPEN** structure.

```c
typedef struct tagLOGPEN {
    WORD lopnStyle;
    POINT lopnWidth;
    COLORREF lopnColor;
} LOGPEN;
```

The **LOGPEN** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lopnStyle</td>
<td>Specifies the pen type, which can be any one of the following values:</td>
</tr>
<tr>
<td></td>
<td><strong>Constant Name</strong></td>
</tr>
<tr>
<td></td>
<td>PS_SOLID</td>
</tr>
<tr>
<td></td>
<td>PS_DASH</td>
</tr>
<tr>
<td></td>
<td>PS_DOT</td>
</tr>
<tr>
<td></td>
<td>PS_DASHDOT</td>
</tr>
<tr>
<td></td>
<td>PS_DASHDOTDOT</td>
</tr>
<tr>
<td></td>
<td>PS_NULL</td>
</tr>
<tr>
<td></td>
<td>PS_INSIDEFRAME</td>
</tr>
</tbody>
</table>

If the width of the pen is greater than 1 and the pen style is PS_INSIDEFRAME, the line is drawn inside the frame of all primitives except polygons and polyline; the pen is drawn with a logical (dithered) color if the pen color does not match an available RGB value. The PS_INSIDEFRAME style is identical to PS_SOLID if the pen width is less than or equal to 1.

| lopnWidth  | Specifies the pen width (in logical units). If the lopnWidth field is zero, the pen is one pixel wide on raster devices. |

| lopnColor  | Specifies the pen color. |

**Comments**

The **y** value in the **POINT** structure for **lopnWidth** is not used.

**See also**

The **CreatePenIndirect** function in Chapter 4, "Functions directory," in **Reference, Volume 1**.
The `MDICREATESTRUCT` data structure contains information about the class, title, owner, location, and size of a multiple document interface (MDI) child window.

```c
typedef struct tagMDICREATESTRUCT {
    LPSTR szClass;
    LPSTR szTitle;
    HANDLE hOwner;
    int x;
    int y;
    int cx;
    int cy;
    LONG style;
    LONG lParam;
} MDICREATESTRUCT;
```

The `MDICREATESTRUCT` structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szClass</td>
<td>Contains a long pointer to the application-defined class of the MDI child window.</td>
</tr>
<tr>
<td>szTitle</td>
<td>Contains a long pointer to the window title of the MDI child window.</td>
</tr>
<tr>
<td>hOwner</td>
<td>Is the instance handle of the application creating the MDI child window.</td>
</tr>
<tr>
<td>x</td>
<td>Specifies the initial position of the left side of the MDI child window. If set to CW_USEDEFAULT, the MDI child window is assigned a default horizontal position.</td>
</tr>
<tr>
<td>y</td>
<td>Specifies the initial position of the top edge of the MDI child window. If set to CW_USEDEFAULT, the MDI child window is assigned a default vertical position.</td>
</tr>
<tr>
<td>cx</td>
<td>Specifies the initial width of the MDI child window. If set to CW_USEDEFAULT, the MDI child window is assigned a default width.</td>
</tr>
<tr>
<td>cy</td>
<td>Specifies the initial height of the MDI child window. If set to CW_USEDEFAULT, the MDI child window is assigned a default height.</td>
</tr>
<tr>
<td>style</td>
<td>Specifies additional styles for the MDI child window. The <code>style</code> field may be set to one or more of the following values:</td>
</tr>
</tbody>
</table>

Chapter 7, Data types and structures
MDICREATESTRUCT

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_MINIMIZE</td>
<td>The MDI child window is created in a minimized state.</td>
</tr>
<tr>
<td>WS_MAXIMIZE</td>
<td>The MDI child window is created in a maximized state.</td>
</tr>
<tr>
<td>WS_HSCROLL</td>
<td>The MDI child window is created with a horizontal scroll bar.</td>
</tr>
<tr>
<td>WS_VSCROLL</td>
<td>The MDI child window is created with a vertical scroll bar.</td>
</tr>
</tbody>
</table>

IParam          Is an application-defined 32-bit value.

Comments

When the MDI child window is created, Windows sends the WM_CREATE message to the window. The IParam parameter of the WM_CREATE message contains a pointer to a CREATESTRUCT data structure. The IpCreateParams field of the CREATESTRUCT structure contains a pointer to the MDICREATESTRUCT data structure passed with the WM_MDICREATE message that created the MDI child window.

MEASUREITEMSTRUCT

Owner-draw control dimensions

The MEASUREITEMSTRUCT data structure informs Windows of the dimensions of an owner-draw control. This allows Windows to process user interaction with the control correctly. The owner of an owner-draw control receives a pointer to this structure as the IParam parameter of an WM_MEASUREITEM message. The owner-draw control sends this message to its owner window when the control is created; the owner then fills in the appropriate fields in the structure for the control and returns. This structure is common to all owner-draw controls.

The MEASUREITEMSTRUCT structure has the following format:

```c
typedef struct tagMEASUREITEMSTRUCT
{
    WORD CtlType;
    WORD CtlID;
    WORD itemID;
    WORD itemWidth;
    WORD itemHeight;
    DWORD itemData
} MEASUREITEMSTRUCT;
```

The MEASUREITEMSTRUCT structure contains the following fields:
### MEASUREITEMSTRUCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CtlType</td>
<td>Is the control type. The values for control types are as follows:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>ODT_BUTTON</td>
</tr>
<tr>
<td></td>
<td>ODT_COMBOBOX</td>
</tr>
<tr>
<td></td>
<td>ODT_LISTBOX</td>
</tr>
<tr>
<td></td>
<td>ODT_MENU</td>
</tr>
<tr>
<td>CtlID</td>
<td>Is the control ID for a combo box, list box, or button. This field is not used for a menu.</td>
</tr>
<tr>
<td>itemID</td>
<td>Is the menu-item ID for a menu or the list-box item ID for a variable-height combo box or list box. This field is not used for a fixed-height combo box or list box, or for a button.</td>
</tr>
<tr>
<td>itemWidth</td>
<td>Specifies the width of a menu item. The owner of the owner-draw menu item must fill this field before returning from the message.</td>
</tr>
<tr>
<td>itemHeight</td>
<td>Specifies the height of an individual item in a list box or a menu. Before returning from the message, the owner of the owner-draw combo box, list box, or menu item must fill out this field.</td>
</tr>
<tr>
<td>itemData</td>
<td>Contains the value that was passed to the combo box or list box in the IParam parameter of one of the following messages:</td>
</tr>
<tr>
<td></td>
<td>CB_ADDSTRING</td>
</tr>
<tr>
<td></td>
<td>CB_INSERTSTRING</td>
</tr>
<tr>
<td></td>
<td>LB_ADDSTRING</td>
</tr>
<tr>
<td></td>
<td>LB_INSERTSTRING</td>
</tr>
<tr>
<td></td>
<td>Contains the DWORD value passed as the lpNewItem parameter of the AppendMenu, InsertMenu, or ModifyMenu function that added or modified the menu item. Its contents are undefined for buttons.</td>
</tr>
</tbody>
</table>

**Comments**

Failure to fill out the proper fields in the MEASUREITEM structure will cause improper operation of the control.

### MENUITEMTEMPLATE

**Menu-itemtemplate**

A complete menu template consists of a header and one or more menu-item lists. The following shows the structure of the menu-template header:

```c
typedef struct {
    WORD versionNumber;
    WORD offset;
} MENUITEMTEMPLATEHEADER;
```

The menu-template header contains the following fields:
**MENUITEMTEMPLATE**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionNumber</td>
<td>Specifies the version number. Should be zero.</td>
</tr>
<tr>
<td>offset</td>
<td>Specifies the offset from the header in bytes where the menu-item list begins.</td>
</tr>
</tbody>
</table>

One or more **MENUITEMTEMPLATE** structures are combined to form the menu-item list.

```c
typedef struct {
    WORD mtOption;
    WORD mtID;
    char mtString;
} MENUITEMTEMPLATE;
```

The **MENUITEMTEMPLATE** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtOption</td>
<td>Specifies a mask of one or more predefined menu options that specify the appearance of the menu item. The menu options are as follows:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>MF_CHECKED</td>
<td>Item has a checkmark next to it.</td>
</tr>
<tr>
<td>MF_END</td>
<td>Item must be specified for the last item in a pop-up menu or a static menu.</td>
</tr>
<tr>
<td>MF_GRAYED</td>
<td>Item is initially inactive and drawn with a gray effect.</td>
</tr>
<tr>
<td>MF_HELP</td>
<td>Item has a vertical separator to its left.</td>
</tr>
<tr>
<td>MF_MENUBARBREAK</td>
<td>Item is placed in a new column. The old and new columns are separated by a bar.</td>
</tr>
<tr>
<td>MF_MENUBREAK</td>
<td>Item is placed in a new column.</td>
</tr>
<tr>
<td>MF_OWNERDRAW</td>
<td>The owner of the menu is responsible for drawing all visual aspects of the menu item, including highlighted, checked and inactive states. This option is not valid for a top-level menu item.</td>
</tr>
<tr>
<td>MF_POPUP</td>
<td>Item displays a sublist of menu items when selected.</td>
</tr>
<tr>
<td>mtID</td>
<td>Specifies an identification code for a nonpop-up menu item. The <strong>MENUITEMTEMPLATE</strong> data structure for a pop-up menu item does not contain the mtID field.</td>
</tr>
<tr>
<td>mtString</td>
<td>Specifies a null-terminated character string that contains the name of the menu item.</td>
</tr>
</tbody>
</table>

See also The **LoadMenuIndirect** function in Chapter 4, "Functions directory," in *Reference, Volume 1*.
The **METAFILEPICT** structure defines the metafile picture format used for exchanging metafile data through the clipboard.

```c
typedef struct tagMETAFILEPICT
{
    int mm;
    int xExt, yExt;
    HANDLE hMF;
} METAFILEPICT;
```

The **METAFILEPICT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mm</td>
<td>Specifies the mapping mode in which the picture is drawn. Specifies the size of the metafile picture for all modes except the MM_ISOTROPIC and MM_ANISOTROPIC modes. The x-extent specifies the width of the rectangle within which the picture is drawn. The coordinates are in units that correspond to the mapping mode.</td>
</tr>
<tr>
<td>xExt</td>
<td>Specifies the size of the metafile picture for all modes except the MM_ISOTROPIC and MM_ANISOTROPIC modes. The x-extent specifies the width of the rectangle within which the picture is drawn. The coordinates are in units that correspond to the mapping mode. For MM_ISOTROPIC and MM_ANISOTROPIC modes, which can be scaled, the xExt and yExt fields contain an optional suggested size in MM_HIMETRIC units. For MM_ANISOTROPIC pictures, xExt and yExt can be zero when no suggested size is supplied. For MM_ISOTROPIC pictures, an aspect ratio must be supplied even when no suggested size is given. (If a suggested size is given, the aspect ratio is implied by the size.) To give an aspect ratio without implying a suggested size, set xExt and yExt to negative values whose ratio is the appropriate aspect ratio. The magnitude of the negative xExt and yExt values will be ignored; only the ratio will be used.</td>
</tr>
<tr>
<td>yExt</td>
<td>Specifies the size of the metafile picture for all modes except the MM_ISOTROPIC and MM_ANISOTROPIC modes. The y-extent specifies the height of the rectangle within which the picture is drawn. The coordinates are in units that correspond to the mapping mode.</td>
</tr>
<tr>
<td>hMF</td>
<td>Identifies a memory metafile.</td>
</tr>
</tbody>
</table>
The MSG structure contains information from the Windows application queue.

```c
typedef struct tagMSG {
    HWND hwnd;
    WORD message;
    WORD wParam;
    LONG lParam;
    DWORD time;
    POINT pt;
} MSG;
```

The MSG structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwnd</td>
<td>Identifies the window that receives the message.</td>
</tr>
<tr>
<td>message</td>
<td>Specifies the message number.</td>
</tr>
<tr>
<td>wParam</td>
<td>Specifies additional information about the message. The exact meaning depends on the message value.</td>
</tr>
<tr>
<td>lParam</td>
<td>Specifies additional information about the message. The exact meaning depends on the message value.</td>
</tr>
<tr>
<td>time</td>
<td>Specifies the time at which the message was posted.</td>
</tr>
<tr>
<td>pt</td>
<td>Specifies the position of the cursor (in screen coordinates) when the message was posted.</td>
</tr>
</tbody>
</table>

The MULTIKEYHELP structure specifies a key-word table and an associated key word to be used by the Windows Help application.

```c
typedef struct tagMULTIKEYHELP {
    WORD mkSize;
    BYTE mkKeyList;
    BYTE szKeyPhrase[];
} MULTIKEYHELP;
```

The MULTIKEYHELP data structure contains the following fields:
OFSTRUCT

Open-file structure

The OFSTRUCT structure contains file information which results from opening that file.

typedef struct tagOFSTRUCT {
    BYTE cBytes;
    BYTE fFixedDisk;
    WORD nErrCode;
    BYTE reserved[4];
    BYTE szPathName[120];
} OFSTRUCT;

The OFSTRUCT structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cBytes</td>
<td>Specifies the length of the OFSTRUCT structure (in bytes).</td>
</tr>
<tr>
<td>fFixedDisk</td>
<td>Specifies whether the file is on a fixed disk. The fFixedDisk field is nonzero if the file is on a fixed disk.</td>
</tr>
<tr>
<td>nErrCode</td>
<td>Specifies the DOS error code if the OpenFile function returns -1 (that is, OpenFile failed).</td>
</tr>
<tr>
<td>szPathName[120]</td>
<td>Specifies 120 bytes that contain the pathname of the file. This string consists of characters from the OEM character set.</td>
</tr>
</tbody>
</table>
The **PAINTSTRUCT** structure contains information for an application. This information can be used to paint the client area of a window owned by that application.

```c
typedef struct _PAINTSTRUCT {
    HDC hdc;
    BOOL fErase;
    RECT rcPaint;
    BOOL fRestore;
    BOOL fIncUpdate;
    BYTE rgbReserved[16];
} PAINTSTRUCT;
```

The **PAINTSTRUCT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdc</td>
<td>Identifies the display context to be used for painting.</td>
</tr>
<tr>
<td>fErase</td>
<td>Specifies whether the background has been redrawn. It has been redrawn if nonzero.</td>
</tr>
<tr>
<td>rcPaint</td>
<td>Specifies the upper-left and lower-right corners of the rectangle in which the painting is requested.</td>
</tr>
<tr>
<td>fRestore</td>
<td>Reserved field. It is used internally by Windows.</td>
</tr>
<tr>
<td>fIncUpdate</td>
<td>Reserved field. It is used internally by Windows.</td>
</tr>
<tr>
<td>rgbReserved</td>
<td>Reserved field. A reserved block of memory used internally by Windows.</td>
</tr>
</tbody>
</table>

The **PALETTEENTRY** data structure specifies the color and usage of an entry in a logical color palette. A logical palette is defined by a **LOGPALETTE** data structure.

```c
typedef struct {
    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

```c
TPaletteEntry = record
    peRed: Byte;
    peGreen: Byte;
    peBlue: Byte;
    peFlags: Byte;
end;
```
The **PALETTEENTRY** structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peRed</td>
<td>Specifies the intensity of red for the palette entry color.</td>
</tr>
<tr>
<td>peGreen</td>
<td>Specifies the intensity of green for the palette entry color.</td>
</tr>
<tr>
<td>peBlue</td>
<td>Specifies the intensity of blue for the palette entry color.</td>
</tr>
<tr>
<td>peFlags</td>
<td>Specifies how the palette entry is to be used. The <strong>peFlags</strong> field may be set to NULL or one of these values:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC_EXPLICIT</td>
<td>Specifies that the low-order word of the logical palette entry designates a hardware palette index. This flag allows the application to show the contents of the display-device palette.</td>
</tr>
<tr>
<td>PC_NOCOLLAPSE</td>
<td>Specifies that the color will be placed in an unused entry in the system palette instead of being matched to an existing color in the system palette. If there are no unused entries in the system palette, the color is matched normally. Once this color is in the system palette, colors in other logical palettes can be matched to this color.</td>
</tr>
<tr>
<td>PC_RESERVED</td>
<td>Specifies that the logical palette entry will be used for palette animation; this prevents other windows from matching colors to this palette entry since the color will frequently change. If an unused system-palette entry is available, this color is placed in that entry. Otherwise, the color will not be available for animation.</td>
</tr>
</tbody>
</table>
Point data structure

The **POINT** structure defines the *x* - and *y*-coordinates of a point.

```c
typedef struct tagPOINT {
    int x;
    int y;
} POINT;
```

The **POINT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Specifies the <em>x</em>-coordinate of a point.</td>
</tr>
<tr>
<td>y</td>
<td>Specifies the <em>y</em>-coordinate of a point.</td>
</tr>
</tbody>
</table>

See also

The **ChildWindowFromPoint**, **PtInRect**, and **WindowFromPoint** functions in Chapter 4, "Functions directory," in Reference, Volume 1.

RECT

Rectangle data structure

The **RECT** structure defines the coordinates of the upper-left and lower-right corners of a rectangle.

```c
typedef struct tagRECT {
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

The **RECT** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Specifies the <em>x</em>-coordinate of the upper-left corner of a rectangle.</td>
</tr>
<tr>
<td>top</td>
<td>Specifies the <em>y</em>-coordinate of the upper-left corner of a rectangle.</td>
</tr>
<tr>
<td>right</td>
<td>Specifies the <em>x</em>-coordinate of the lower-right corner of a rectangle.</td>
</tr>
<tr>
<td>bottom</td>
<td>Specifies the <em>y</em>-coordinate of the lower-right corner of a rectangle.</td>
</tr>
</tbody>
</table>

Comments

The width of the rectangle defined by the **RECT** structure must not exceed 32,768 units.
The **RGBQUAD** data structure describes a color consisting of relative intensities of red, green, and blue. The `bmciColors` field of the **BITMAPINFO** data structure consists of an array of RGBQUAD data structures.

```c
typedef struct tagRGBQUAD {
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;
```

The **RGBQUAD** structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgbBlue</td>
<td>Specifies the intensity of blue in the color.</td>
</tr>
<tr>
<td>rgbGreen</td>
<td>Specifies the intensity of green in the color.</td>
</tr>
<tr>
<td>rgbRed</td>
<td>Specifies the intensity of red in the color.</td>
</tr>
<tr>
<td>rgbReserved</td>
<td>Is not used and must be set to zero.</td>
</tr>
</tbody>
</table>

The **RGBTRIPLE** data structure describes a color consisting of relative intensities of red, green, and blue. The `bmciColors` field of the **BITMAPCOREINFO** data structure consists of an array of RGBTRIPLE data structures.

```c
typedef struct tagRGBTRIPLE {
    BYTE rgbtBlue;
    BYTE rgbtGreen;
    BYTE rgbtRed;
} RGBTRIPLE;
```

The **RGBTRIPLE** structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgbtBlue</td>
<td>Specifies the intensity of blue in the color.</td>
</tr>
<tr>
<td>rgbtGreen</td>
<td>Specifies the intensity of green in the color.</td>
</tr>
<tr>
<td>rgbtRed</td>
<td>Specifies the intensity of red in the color.</td>
</tr>
</tbody>
</table>
The **TEXTMETRIC** structure contains basic information about a physical font. All sizes are given in logical units; that is, they depend on the current mapping mode of the display context.

```c
typedef struct tagTEXTMETRIC {
    short int tmHeight;
    short int tmAscent;
    short int tmDescent;
    short int tmInternalLeading;
    short int tmExternalLeading;
    short int tmAveCharWidth;
    short int tmMaxCharWidth;
    short int tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    short int tmOverhang;
    short int tmDigitizedAspectX;
    short int tmDigitizedAspectY;
} TEXTMETRIC;
```

The **TEXTMETRIC** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmHeight</td>
<td>Specifies the height (ascent + descent) of characters.</td>
</tr>
<tr>
<td>tmAscent</td>
<td>Specifies the ascent (units above the baseline) of characters.</td>
</tr>
<tr>
<td>tmDescent</td>
<td>Specifies the descent (units below the baseline) of characters.</td>
</tr>
<tr>
<td>tmInternalLeading</td>
<td>Specifies the amount of leading (space) inside the bounds set by the tmHeight field. Accent marks and other foreign characters may occur in this area. The designer may set this field to zero.</td>
</tr>
<tr>
<td>tmExternalLeading</td>
<td>Specifies the amount of extra leading (space) that the application adds between rows. Since this area is outside the font, it contains no marks and will not be altered by text output calls in either OPAQUE or TRANSPARENT mode. The designer may set this field to zero.</td>
</tr>
</tbody>
</table>
**tmAveCharWidth**
Specifies the average width of characters in the font (loosely defined as the width of the letter x). This value does not include overhang required for bold or italic characters.

**tmMaxCharWidth**
Specifies the width of the widest character in the font.

**tmWeight**
Specifies the weight of the font.

**tmItalic**
Specifies an italic font if it is nonzero.

**tmUnderlined**
Specifies an underlined font if it is nonzero.

**tmStruckOut**
Specifies a struckout font if it is nonzero.

**tmFirstChar**
Specifies the value of the first character defined in the font.

**tmLastChar**
Specifies the value of the last character defined in the font.

**tmDefaultChar**
Specifies the value of the character that will be substituted for characters that are not in the font.

**tmBreakChar**
Specifies the value of the character that will be used to define word breaks for text justification.

**tmPitchAndFamily**
Specifies the pitch and family of the selected font. The low-order bit specifies the pitch of the font. If it is 1, the font is variable pitch. If it is 0, the font is fixed pitch. The four high-order bits designate the font family. The **tmPitchAndFamily** field can be combined with the hexadecimal value 0xF0 by using the bitwise AND operator, and then be compared with the font family names for an identical match. For a description of the font families, see the **LOGFONT** structure, earlier in this chapter.

**tmCharSet**
Specifies the character set of the font.

**tmOverhang**
Specifies the per-string extra width that may be added to some synthesized fonts. When synthesizing some attributes, such as bold or italic, GDI or a device may have to add width to a string on both a per-character and per-string basis. For example, GDI makes a string bold by expanding the intracharacter spacing and overstriking by an offset value; it italicizes a font by skewing the string. In either case, there is an overhang past the basic string. For bold strings, the overhang is the distance by which the overstrike is offset. For italic strings, the overhang is the amount the top of the font is skewed past the bottom of the font. The **tmOverhang** field allows the application to determine how much of the character width returned by a **GetTextExtent** function call on a single character is the actual character width and how much is the per-string extra width. The actual width is the extent minus the overhang.

**tmDigitizedAspectX**
Specifies the horizontal aspect of the device for which the font was designed.

**tmDigitizedAspectY**
Specifies the vertical aspect of the device for which the font was designed. The ratio of the **tmDigitizedAspectX** and **tmDigitizedAspectY** fields is the aspect ratio of the device for which the font was designed.

**See also**
The **GetDeviceCaps** and **GetTextMetrics** functions in Chapter 4, "Functions directory," in Reference, Volume 1.
THE **WNDCLASS** structure contains the class attributes that are registered by the **RegisterClass** function.

```c
typedef struct tagWNDCLASS {
    WORD    style;
    long    lpfnWndProc();
    int     cbClsExtra;
    int     cbWndExtra;
    HANDLE  hWndInstance;
    HICON   hIcon;
    HCURSOR hCursor;
    HBRUSH  hbrBackground;
    LPSTR   lpszMenuName;
    LPSTR   lpszClassName;
} WNDCLASS;
```

The **WNDCLASS** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>Specifies the class style. These styles can be combined by using the bitwise OR operator. The <em>style</em> field can be any combination of the following values:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS_BYTEALIGNCLIENT</td>
<td>Aligns a window's client area on the byte boundary (in the x direction).</td>
</tr>
<tr>
<td>CS_BYTEALIGNWINDOW</td>
<td>Aligns a window on the byte boundary (in the x direction).</td>
</tr>
<tr>
<td>CS_CLASSDC</td>
<td>Gives the window class its own display context (shared by instances).</td>
</tr>
<tr>
<td>CS_DBLCLKS</td>
<td>Sends double-click messages to a window.</td>
</tr>
<tr>
<td>CS_GLOBALCLASS</td>
<td>Specifies that the window class is an application global class. An application global class is created by an application or library and is available to all applications. The class is destroyed when the application or library that created the class terminates; it is essential,</td>
</tr>
</tbody>
</table>
therefore, that all windows created with the application global class be closed before this occurs.

**CS_HREDRAW** Redraws the entire window if the horizontal size changes.

**CS_NOCLOSE** Inhibits the close option on the System menu.

**CS_OWNDC** Gives each window instance its own display context. Note that although the CS_OWNDC style is convenient, it must be used with discretion because each display context occupies approximately 800 bytes of memory.

**CS_PARENTDC** Gives the parent window's display context to the window class.

**CS_SAVEBITS** Saves the portion of the screen image that is obscured by a window; Windows uses the saved bitmap to re-create a screen image when the window is removed. Windows displays the bitmap at its original location and does not send WM_PAINT messages to windows which had been obscured by the window if the memory used by the bitmap has not been discarded and if other screen actions have not invalidated the stored image. An application should set this bit only for small windows that are displayed briefly and then removed before much other screen activity takes place. Setting this bit for a window increases the amount of time required to display the window due to the time required to allocate memory to store the bitmap.

**CS_VREDRAW** Redraws the entire window if the vertical size changes.

**IpfnWndProc** Points to the window function.

**cbClsExtra** Specifies the number of bytes to allocate following the window-class structure.

**cbWndExtra** Specifies the number of bytes to allocate following the window instance. If an application is using the WNDCLASS structure to
**WNDCLASS**

register a dialog box created with the `CLASS` directive in the .RC script file, it must set this field to DLGWINDOWNEXTRA.

<table>
<thead>
<tr>
<th>hlnstance</th>
<th>Identifies the class module. The <code>hlnstance</code> field must be an instance handle and must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hlcon</td>
<td>Identifies the class icon. The <code>hlcon</code> field must be a handle to an icon resource. If <code>hlcon</code> is NULL, the application must draw an icon whenever the user minimizes the application's window.</td>
</tr>
<tr>
<td>hCursor</td>
<td>Identifies the class cursor. The <code>hCursor</code> field must be a handle to a cursor resource. If <code>hCursor</code> is NULL, the application must explicitly set the cursor shape whenever the mouse moves into the application's window.</td>
</tr>
<tr>
<td>hbrBackground</td>
<td>Identifies the class background brush. The <code>hbrBackground</code> field can be either a handle to the physical brush that is to be used for painting the background, or it can be a color value. If a color value is given, it must be one of the standard system colors listed below, and the value 1 must be added to the chosen color (for example, COLOR_BACKGROUND + 1 specifies the system background color). If a color value is given, it must be converted to one of these HBRUSH types:</td>
</tr>
<tr>
<td></td>
<td>• COLOR_ACTIVEBORDER</td>
</tr>
<tr>
<td></td>
<td>• COLOR_ACTIVECAPTION</td>
</tr>
<tr>
<td></td>
<td>• COLOR_APPWORKSPACE</td>
</tr>
<tr>
<td></td>
<td>• COLOR_BACKGROUND</td>
</tr>
<tr>
<td></td>
<td>• COLOR_BTNFACE</td>
</tr>
<tr>
<td></td>
<td>• COLOR_BTNSHADOW</td>
</tr>
<tr>
<td></td>
<td>• COLOR_BTNTEXT</td>
</tr>
<tr>
<td></td>
<td>• COLOR_CAPTIONTEXT</td>
</tr>
<tr>
<td></td>
<td>• COLOR_GRAYTEXT</td>
</tr>
<tr>
<td></td>
<td>• COLOR_HIGHLIGHT</td>
</tr>
<tr>
<td></td>
<td>• COLOR_HIGHLIGHTTEXT</td>
</tr>
<tr>
<td></td>
<td>• COLOR_INACTIVEBORDER</td>
</tr>
<tr>
<td></td>
<td>• COLOR_INACTIVECAPTION</td>
</tr>
<tr>
<td></td>
<td>• COLOR_MENU</td>
</tr>
<tr>
<td></td>
<td>• COLOR_MENUTEXT</td>
</tr>
<tr>
<td></td>
<td>• COLOR_SCROLLBAR</td>
</tr>
<tr>
<td></td>
<td>• COLOR_WINDOW</td>
</tr>
<tr>
<td></td>
<td>• COLOR_WINDOWFRAME</td>
</tr>
<tr>
<td></td>
<td>• COLOR_WINDOWTEXT</td>
</tr>
<tr>
<td></td>
<td>When <code>hbrBackground</code> is NULL, the application must paint its own background whenever it is requested to paint in its client area. The application can determine when the background needs painting by processing the WM_ERASEBKGND message or by testing the <code>fErase</code> field of the PAINTSTRUCT structure filled by the <code>BeginPaint</code> function.</td>
</tr>
<tr>
<td>ipszMenuName</td>
<td>Points to a null-terminated character string that specifies the resource name of the class menu (as the name appears in the resource file). If an integer is used to identify the menu, the MAKEINTRESOURCE macro can be used. If the <code>ipszMenuName</code> field is NULL, windows belonging to this class have no default menu.</td>
</tr>
<tr>
<td>ipszClassName</td>
<td>Points to a null-terminated character string that specifies the name of the window class.</td>
</tr>
</tbody>
</table>
This chapter describes the statements that define resources that the Microsoft Windows Resource Compiler (RC) adds to an application's executable file. See *Tools* for information on running the Resource Compiler.

This chapter describes resource script statements in the following categories:

- Single-line statements
- User-defined resources
- RCDATA statement
- STRINGTABLE statement
- ACCELERATORS statement
- Menu statements
- Dialog statements
- Directives

**Single-line statements**

The single-line statements define resources that are contained in a single file, such as cursors, icons, and fonts. The statements associate the filename of the resource with an identifying name or number. The resource is added to the executable file when the application is created, and can be extracted during execution by referring to the name or number.
The following is the general form for all single-line statements:

   nameID resource-type [[load-option]] [[mem-option]] filename

The nameID field specifies either a unique name or an integer value identifying the resource. For a font resource, nameID must be a number; it cannot be a name.

The resource-type field specifies one of the following key words, which identify the type of resource to be loaded:

<table>
<thead>
<tr>
<th>Key word</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURSOR</td>
<td>Specifies a bitmap that defines the shape of the cursor on the display screen.</td>
</tr>
<tr>
<td>ICON</td>
<td>Specifies a bitmap that defines the shape of the icon to be used for a given application.</td>
</tr>
<tr>
<td>BITMAP</td>
<td>Specifies a custom bitmap that an application is going to use in its screen display or as an item in a menu.</td>
</tr>
<tr>
<td>FONT</td>
<td>Specifies a file that contains a font.</td>
</tr>
</tbody>
</table>

The optional load-option field takes a key word that specifies when the resource is to be loaded. The key word must be one of the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRELOAD</td>
<td>Resource is loaded immediately.</td>
</tr>
<tr>
<td>LOADONCALL</td>
<td>Resource is loaded when called. This is the default option.</td>
</tr>
</tbody>
</table>

Icon and cursor resources can contain more than one image. If the resource is marked as PRELOAD, Windows loads all images in the resource when the application executes.

The optional mem-option field takes the following key word or key words, which specify whether the resource is fixed or moveable and whether it is discardable:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED</td>
<td>Resource remains at a fixed memory location.</td>
</tr>
<tr>
<td>MOVEABLE</td>
<td>Resource can be moved if necessary in order to compact memory.</td>
</tr>
<tr>
<td>DISCARDABLE</td>
<td>Resource can be discarded if no longer needed.</td>
</tr>
</tbody>
</table>

The default is MOVEABLE and DISCARDABLE for cursor, icon, and font resources. The default for bitmap resources is MOVEABLE.

The filename field is an ASCII string that specifies the DOS filename of the file that contains the resource. A full pathname must be given if the file is not in the current working directory.
The following example demonstrates the correct usage for a single-line statement:

```
cursor CURSOR point.cur
cursor CURSOR DISCARDABLE point.cur
10 CURSOR custom.cur
desk ICON desk.ico
desk ICON DISCARDABLE desk.ico
11 ICON custom.ico
disk BITMAP disk.bmp
disk BITMAP DISCARDABLE disk.bmp
12 BITMAP custom.bmp
5 FONT CMROMAN.FNT
```

User-defined resources

An application can also define its own resource. The resource can be any data that the application intends to use. A user-defined resource statement has the following form:

```
nam01D typeID [[load-option]] [[mem-option]] [[filename]] |
[BEGIN
 raw-data
 END]]
```

The `nameID` field specifies either a unique name or an integer value that identifies the resource.

The `typeID` field specifies either a unique name or an integer value that identifies the resource type. If a number is given, it must be greater than 255. The numbers 1 through 255 are reserved for existing and future predefined resource types.

The optional `load-option` field takes a key word that specifies when the resource is to be loaded. The key word must be one of the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRELOAD</td>
<td>Resource is loaded immediately.</td>
</tr>
<tr>
<td>LOADONCALL</td>
<td>Resource is loaded when called. This is the default option.</td>
</tr>
</tbody>
</table>

The optional `mem-option` field takes the following key word or key words, which specify whether the resource is fixed or moveable and whether it is discardable:
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED</td>
<td>Resource remains at a fixed memory location.</td>
</tr>
<tr>
<td>MOVEABLE</td>
<td>Resource can be moved if necessary in order to compact memory. This is the default option.</td>
</tr>
<tr>
<td>DISCARDABLE</td>
<td>Resource can be discarded if it is no longer needed.</td>
</tr>
</tbody>
</table>

The optional *filename* field is an ASCII string that specifies the DOS filename of the file that contains the resource. A full pathname must be given if the file is not in the current working directory. Do not use the *filename* field if you supply raw data between the optional BEGIN and END statements.

The *raw-data* field specifies one or more integers and strings. Integers can be in decimal, octal, or hexadecimal format. Do not use *raw-data* field and the BEGIN and END statements if you specify a filename.

The following example demonstrates the correct usage for user-defined statements:

```plaintext
array  MYRES  data.res
14    300    custom.res
18    MYRES2
BEGIN
    "Here is a data string\0", /* A string. Note: explicitly null-terminated */
    1024, /* int */
    0x029a, /* hex int */
    0733, /* octal int */
    "\07" /* octal byte */
END
```

### RCDATA statement

#### Syntax

```
nameID  RCDATA  [[load-option]]  [[mem-option]]
BEGIN
    raw-data
END
```

The **RCDATA** statement defines a raw data resource for an application. Raw data resources permit the inclusion of binary data directly in the executable file.

The `nameID` field specifies either a unique name or an integer value that identifies the resource.
The optional load-option field takes a key word that specifies when the resource is to be loaded. It must be one of the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRELOAD</td>
<td>Resource is loaded immediately.</td>
</tr>
<tr>
<td>LOADONCALL</td>
<td>Resource is loaded when called. This is the default option.</td>
</tr>
</tbody>
</table>

The optional mem-option field takes the following key word or key words, which specify whether the resource is fixed or moveable and whether it is discardable:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED</td>
<td>Resource remains at a fixed memory location.</td>
</tr>
<tr>
<td>MOVEABLE</td>
<td>Resource can be moved if necessary in order to compact memory.</td>
</tr>
<tr>
<td>DISCARDABLE</td>
<td>Resource can be discarded if no longer needed.</td>
</tr>
</tbody>
</table>

The default is MOVEABLE and DISCARDABLE.

The raw-data field specifies one or more integers and strings. Integers can be in decimal, octal, or hexadecimal format.

The following example demonstrates the correct usage for the RCDATA statement:

```c
resname RCDATA
BEGIN
   "Here is a data string\0", /* A string. Note: explicitly null-terminated */
      1024, /* int */
      0x029a, /* hex int */
      0o1733, /* octal int */
      "\07", /* octal byte */
END
```

### Stringtable statement

**Syntax**

```
stringtable [[load-option]] [[mem-option]]
BEGIN
  stringID string
END
```

The STRINGTABLE statement defines one or more string resources for an application. String resources are simply null-terminated ASCII strings that
can be loaded when needed from the executable file, using the `LoadString` function.

The optional *load-option* field takes a key word that specifies when the resource is to be loaded. It must be one of the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRELOAD</td>
<td>Resource is loaded immediately.</td>
</tr>
<tr>
<td>LOADONCALL</td>
<td>Resource is loaded when called. This is the default option.</td>
</tr>
</tbody>
</table>

The optional *mem-option* field takes the following key word or key words, which specify whether the resource is fixed or moveable and whether or not it is discardable:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED</td>
<td>Resource remains at a fixed memory location.</td>
</tr>
<tr>
<td>MOVEABLE</td>
<td>Resource can be moved if necessary in order to compact memory.</td>
</tr>
<tr>
<td>DISCARDABLE</td>
<td>Resource can be discarded if no longer needed.</td>
</tr>
</tbody>
</table>

The default is **MOVEABLE** and **DISCARDABLE**.

The *stringID* field specifies an integer value that identifies the resource.

The *string* field specifies one or more ASCII strings, enclosed in double quotation marks. The string must be no longer than 255 characters and must occupy a single line in the source file. To add a carriage return to the string, use this character sequence: `\012`. For example, "Line one\012Line two" would define a string that would be displayed as follows:

```
    Line one
    Line two
```

Grouping strings in separate segments allows all related strings to be read in at one time and discarded together. When possible, an application should make the table moveable and discardable. The Resource Compiler allocates 16 strings per segment and uses the identifier value to determine which segment is to contain the string. Strings with the same upper 12 bits in their identifiers are placed in the same segment.

The following example demonstrates the correct usage of the STRINGTABLE statement:

```
#define IDS_HELLO   1
#define IDS_GOODBYE 2

STRINGTABLE
BEGIN
```
Accelerators statement

Syntax

acctablename ACCELERATORS
BEGIN
  event, idvalue, [[type]] [[NOINVERT]] [[ALT]] [[SHIFT]] [[CONTROL]]
:;
END

The ACCELERATORS statement defines one or more accelerators for an application. An accelerator is a key stroke defined by the application to give the user a quick way to perform a task. The TranslateAccelerator function is used to translate accelerator messages from the application queue into WM_COMMAND or WM_SYSCOMMAND messages.

The acctablename field specifies either a unique name or an integer value that identifies the resource.

The event field specifies the key stroke to be used as an accelerator. It can be any one of the following:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;char&quot;</td>
<td>A single ASCII character enclosed in double quotes. The character can be preceded by a caret (^), meaning that the character is a control character.</td>
</tr>
<tr>
<td>ASCII character</td>
<td>An integer value representing an ASCII character. The type field must be ASCII.</td>
</tr>
<tr>
<td>Virtual key character</td>
<td>An integer value representing a virtual key. The virtual key for alphanumeric keys can be specified by placing the uppercase letter or number in double quotation marks (for example, &quot;9&quot; or &quot;C&quot;). The type field must be VIRTKEY.</td>
</tr>
</tbody>
</table>

The idvalue field specifies an integer value that identifies the accelerator.

The type field is required only when event is an ASCII character or a virtual key character. The type field specifies either ASCII or VIRTKEY; the integer value of event is interpreted accordingly. When VIRTKEY is specified and the event field contains a string, the event field must be uppercase.
The **NOINVERT** option, if given, means that no top-level menu item is highlighted when the accelerator is used. This is useful when defining accelerators for actions such as scrolling that do not correspond to a menu item. If **NOINVERT** is omitted, a top-level menu item will be highlighted (if possible) when the accelerator is used.

The **ALT** option, if given, causes the accelerator to be activated only if the ALT key is down.

The **SHIFT** option, if given, causes the accelerator to be activated only if the SHIFT key is down.

The **CONTROL** option, if given, defines the character as a control character (the accelerator is only activated if the CONTROL key is down). This has the same effect as using a caret (^) before the accelerator character in the event field.

The **ALT**, **SHIFT**, and **CONTROL** options apply only to virtual keys.

The following example demonstrates the correct usage of accelerator keys:

```plaintext
1 ACCELERATORS
BEGIN
"^C", IDDCLEAR        ; control C
"K", IDDCLEAR         ; shift K
"k", IDDELLIPSE, ALT  ; alt K
98, IDDRECT, ASCII    ; b
66, IDDSTAR, ASCII    ; B (shift b)
"g", IDDRECT          ; g
"G", IDDSTAR          ; G (shift G)
VK_F1, IDDCLEAR, VIRTKEY ; F1
VK_F1, IDDSTAR, CONTROL, VIRTKEY ; control F1
VK_F1, IDDELLIPSE, SHIFT, VIRTKEY ; shift F1
VK_F1, IDDRECT, ALT, VIRTKEY  ; alt F1
VK_F2, IDDCLEAR, ALT, SHIFT, VIRTKEY ; alt shift F2
VK_F2, IDDSTAR, CONTROL, SHIFT, VIRTKEY ; ctrl shift F2
VK_F2, IDDRECT, ALT, CONTROL, VIRTKEY ; alt control F2
END
```

**Menu statement**

**Syntax**

```
menuID MENU [[load-option]] [[mem-option]]
BEGIN
item-definitions
END
```
The **MENU** statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user select commands from a list of command names.

The **menuID** field specifies a name or number used to identify the menu resource.

The optional **load-option** field takes a key word that specifies when the resource is to be loaded. It must be one of the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRELOAD</td>
<td>Resource is loaded immediately.</td>
</tr>
<tr>
<td>LOADONCALL</td>
<td>Resource is loaded when called. This is the default option.</td>
</tr>
</tbody>
</table>

The optional **mem-option** field takes the following key word or key words, which specify whether the resource is fixed or moveable and whether it is discardable:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED</td>
<td>Resource remains at a fixed memory location.</td>
</tr>
<tr>
<td>MOVEABLE</td>
<td>Resource can be moved to compact memory.</td>
</tr>
<tr>
<td>DISCARDABLE</td>
<td>Resource can be discarded if no longer needed.</td>
</tr>
</tbody>
</table>

The default is **MOVEABLE** and **DISCARDABLE**.

The **item-definition** field specifies special resource statements that define the items in the menu. These statements are defined in the following sections. The following is an example of a complete **MENU** statement:

```sample
sample MENU
BEGIN
MENUITEM "&Soup", 100
MENUITEM "&Salad", 101
POPUP "&Entree"
BEGIN
MENUITEM "&Fish", 200
MENUITEM "&Chicken", 201, CHECKED
POPUP "&Beef"
BEGIN
MENUITEM "&Steak", 301
MENUITEM "&Prime Rib", 302
END
END
MENUITEM "&Dessert", 103
END
```
The **MENUITEM** and **POPUP** statements are used in the *item-definition* section of a **MENU** statement to define the names and attributes of the actual menu items. Any number of statements can be given; each defines a unique item. The order of the statements defines the order of the menu items.

The **MENUITEM** and **POPUP** statements can be used only within an *item-definition* section of a **MENU** statement.

**MENUITEM**

**Syntax**

```
MENUITEM text, result, [[optionlist]]
```

This optional statement defines a menu item.

The *text* field takes an ASCII string, enclosed in double quotation marks, that specifies the name of the menu item.

The string can contain the escape characters `\t` and `\a`. The `\t` character inserts a tab in the string and is used to align text in columns. Tab characters should be used only in pop-up menus, not in menu bars. (See the following section for information on pop-up menus.) The `\a` character aligns all text that follows it flush right to the menu bar or pop-up menu.

To insert a double quotation mark (""") in the string, use two double quotation marks (""").

To add a mnemonic to the text string, place the ampersand (&) ahead of the letter that will be the mnemonic. This will cause the letter to appear underlined in the control and to function as the mnemonic. To use the ampersand as a character in a string, insert two ampersands (&&).

The *result* field takes an integer value that specifies the result generated when the user selects the menu item. Menu-item results are always integers; when the user clicks the menu-item name, the result is sent to the window that owns the menu.

The optional *optionlist* field takes one or more predefined menu options, separated by commas or spaces, that specify the appearance of the menu item. The menu options are as follows:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKED</td>
<td>Item has a checkmark next to it. This option is not valid for a top-level pop-up menu.</td>
</tr>
<tr>
<td>GRAYED</td>
<td>Item name is initially inactive and appears on the menu in gray or a lightened shade of the menu-text color.</td>
</tr>
<tr>
<td>HELP</td>
<td>Item has a vertical separator to its left.</td>
</tr>
<tr>
<td>INACTIVE</td>
<td>Item name is displayed, but it cannot be selected.</td>
</tr>
<tr>
<td>MENUBARBREAK</td>
<td>Same as MF_MENUBREAK except that for pop-up menus, it separates the new column from the old column with a vertical line.</td>
</tr>
<tr>
<td>MENUBREAK</td>
<td>Places the menu item on a new line for static menu-bar items. For pop-up menus, places the menu item in a new column, with no dividing line between the columns.</td>
</tr>
</tbody>
</table>

The **INACTIVE** and **GRAYED** options cannot be used together.

The following example demonstrates the correct usage of the MENUITEM statement:

```plaintext
MENUITEM "&Alpha", 1, CHECKED, GRAYED
MENUITEM "&Beta", 2
```

## Syntax

POPUP  

**Syntax**  

POPUP text, [[optionlist]]
BEGIN  
item-definitions  
END  

This statement marks the beginning of the definition of a pop-up menu. A pop-up menu (which is also known as a drop-down menu) is a special menu item that displays a sublist of menu items when it is selected.

The **text** field takes an ASCII string, enclosed in double quotation marks, that specifies the name of the pop-up menu.

The optional **optionlist** field takes one or more predefined menu options that specify the appearance of the menu item. The menu options are as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKED</td>
<td>Item has a checkmark next to it. This option is not valid for a top-level pop-up menu.</td>
</tr>
<tr>
<td>GRAYED</td>
<td>Item name is initially inactive and appears on the menu in gray or a lightened shade of the menu-text color.</td>
</tr>
<tr>
<td>INACTIVE</td>
<td>Item name is displayed, but it cannot be selected.</td>
</tr>
</tbody>
</table>
MENUBARBREAK  Same as MF_MENUBREAK except that for pop-up menus, it separates the new column from the old column with a vertical line.

MENUBREAK  Places the menu item on a new line for static menu bar items. For pop-up menus, places the menu item in a new column, with no dividing line between the columns.

The options can be combined using the bitwise OR operator. The INACTIVE and GRAYED options cannot be used together.

The item-definitions field can specify any number of MENUITEM or POPUP statements. As a result, any pop-up menu item can display another pop-up menu.

The following example demonstrates the correct usage of the POPUP statement:

```
chem MENU
BEGIN
POPUP "&Elements"
BEGIN
    MENUITEM "&Oxygen", 200
    MENUITEM "&Carbon", 201, CHECKED
    MENUITEM "&Hydrogen", 202
    MENUITEM "&Sulfur", 203
    MENUITEM "&Chlorine", 204
END
POPUP "&Compounds"
BEGIN
    POPUP "&Sugars"
    BEGIN
        MENUITEM "&Glucose", 301
        MENUITEM "&Sucrose", 302, CHECKED
        MENUITEM "&Lactose", 303, MENUBREAK
        MENUITEM "&Fructose", 304
    END
    POPUP "&Acids"
    BEGIN
        "&Hydrochloric", 401
        "&Sulfuric", 402
    END
END
END
```
MENUITEM SEPARATOR

Syntax

This special form of the MENUITEM statement creates an inactive menu item that serves as a dividing bar between two active menu items in a pop-up menu.

The following demonstrates the correct usage of the MENUITEM SEPARATOR statement:

```plaintext
MENUITEM "&Roman", 206
MENUITEM SEPARATOR
MENUITEM "&20 Point", 301
```

DIALOG statement

The DIALOG statement defines a template that can be used by an application to create dialog boxes.

Syntax

```plaintext
nameID DIALOG [[load-option]] [[mem-option]] x, y, width, height
[[option-statements]]
BEGIN
control-statements
END
```

This statement marks the beginning of a DIALOG template. It defines the name of the dialog box, the memory and load options, the box’s starting location on the display screen, and the box’s width and height.

The nameID field specifies either a unique name or an integer value that identifies the resource.

The optional load-option field takes a key word that specifies when the resource is to be loaded. It must be one of the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRELOAD</td>
<td>Resource is loaded immediately.</td>
</tr>
<tr>
<td>LOADONCALL</td>
<td>Resource is loaded when called. This is the default option.</td>
</tr>
</tbody>
</table>

The optional mem-option field takes the following key word or key words, which specify whether the resource is fixed or moveable and whether it is discardable:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED</td>
<td>Resource remains at a fixed memory location.</td>
</tr>
</tbody>
</table>

Chapter 8, Resource script statements 73
MOVEABLE
Resource can be moved if necessary in order to compact memory. This is the default option.

DISCARDABLE
Resource can be discarded if no longer needed.

The $x$ and $y$ fields take integer values that specify the $x$ and $y$ coordinates of the upper-left corner of the dialog box. The horizontal units are $1/4$ of the dialog base width unit; the vertical units are $1/8$ of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The **GetDialogBaseUnits** function returns the dialog base units in pixels. The exact meaning of the coordinates depends on the style defined by the **STYLE** option statement. For child-style dialog boxes, the coordinates are relative to the origin of the parent window, unless the dialog box has the style **DS_ABSALIGN**; in that case, the coordinates are relative to the origin of the display screen.

The **width** and **height** fields take integer values that specify the width and height of the box. The width units are $1/4$ of the dialog base width unit; the height units are $1/8$ of the dialog base height unit.

The option and control statements are described in the following sections.

The following demonstrates the correct usage of the **DIALOG** statement:

```c
#include "WINDOWS.H"
errmess DIALOG 10, 10, 300, 110
STYLE WS_POPUP|WS_BORDER
CAPTION "Error!"
BEGIN
    CTEXT "Select One:", 1, 10, 10, 280, 12
    RADIOBUTTON "&Retry", 2, 75, 30, 60, 12
    RADIOBUTTON "&Abort", 3, 75, 50, 60, 12
    RADIOBUTTON "&Ignore", 4, 75, 80, 60, 12
END
```

**Comments**
Do not use the **WS_CHILD** style with a modal dialog box. The **DialogBox** function always disables the parent/owner of the newly-created dialog box. When a parent window is disabled, its child windows are implicitly disabled. Since the parent window of the child-style dialog box is disabled, the child-style dialog box is too.

If a dialog box has the **DS_ABSALIGN** style, the dialog coordinates for its upper-left corner are relative to the screen origin instead of to the upper-left corner of the parent window. You would typically use this style when you wanted the dialog box to start in a specific part of the display no matter where the parent window may be on the screen.
The name **DIALOG** can also be used as the class-name parameter to the **CreateWindow** function in order to create a window with dialog-box attributes.

**Dialog option statements**

The dialog option statements, given in the *option-statements* section of the **DIALOG** statement, define special attributes of the dialog box, such as its style, caption, and menu. The option statements are optional. If the application does not supply a particular option statement, the dialog box is given default attributes for that option. Dialog option statements include the following:

- STYLE
- CAPTION
- MENU
- CLASS
- FONT

The option statements are discussed individually in the following sections.

**STYLE**

**Syntax**

```
STYLE style
```

This optional statement defines the window style of the dialog box. The window style specifies whether the box is a pop-up or a child window. The default style has the following attributes:

- WS_POPUP
- WS_BORDER
- WS_SYSMENU

The *style* field takes an integer value or predefined name that specifies the window style. It can be any of the window styles defined in Table 8.1, "Window styles."

**Comments**

If the predefined names are used, the **#include** directive must be used so that the **WINDOWS.H** file will be included in the resource script.

<table>
<thead>
<tr>
<th>Style</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS_LOCALEDIT</td>
<td>Specifies that edit controls in the dialog box will use memory in the application’s data segment. By default, all edit controls in dialog boxes use memory outside the</td>
</tr>
</tbody>
</table>
Table 8.1: Window styles (continued)

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS_MODALFRAME</td>
<td>Creates a dialog box with a modal dialog-box frame that can be combined with a title bar and system menu by specifying the WS_CAPTION and WS_SYSMENU styles.</td>
</tr>
<tr>
<td>DS_NOIDLEMESG</td>
<td>Suppresses WM_ENTERIDLE messages that Windows would otherwise send to the owner of the dialog box while the dialog box is displayed.</td>
</tr>
<tr>
<td>DS_SYSMODAL</td>
<td>Creates a system-modal dialog box.</td>
</tr>
<tr>
<td>WS_BORDER</td>
<td>Creates a window that has a border.</td>
</tr>
<tr>
<td>WS_CAPTION</td>
<td>Creates a window that has a title bar (implies WS_BORDER).</td>
</tr>
<tr>
<td>WS_CHILD</td>
<td>Creates a child window. It cannot be used with WS_POPUP.</td>
</tr>
<tr>
<td>WS_CHILDWINDOW</td>
<td>Creates a child window that has the style WS_CHILD.</td>
</tr>
<tr>
<td>WS_CLIPSIBLINGS</td>
<td>Clips child windows relative to each other; that is, when a particular child window receives a WP_PAINT message, this style clips all other top-level child windows out of the region of the child window to be updated. (If WS_CLIPSIBLINGS is not given and child windows overlap, it is possible, when drawing in the client area of a child window, to draw in the client area of a neighboring child window.) For use with WS_CHILD only.</td>
</tr>
<tr>
<td>WS_DISABLED</td>
<td>Creates a window that is initially disabled.</td>
</tr>
<tr>
<td>WS_DLGFRAME</td>
<td>Creates a window with a modal dialog-box frame but no title.</td>
</tr>
<tr>
<td>WS_GROUP</td>
<td>Specifies the first control of a group of controls in which the user can move from one control to the next by using the arrow keys. All controls defined with the WS_GROUP style after the first control belong to the same group. The next control with the WS_GROUP style ends the style group and starts the next group (i.e., one group ends where the next begins). This style is valid only for controls.</td>
</tr>
<tr>
<td>WS_HSCROLL</td>
<td>Creates a window that has a horizontal scroll bar.</td>
</tr>
<tr>
<td>WS_ICONIC</td>
<td>Creates a window that is initially iconic. For use with WS_OVERLAPPED only.</td>
</tr>
<tr>
<td>WS_MAXIMIZE</td>
<td>Creates a window of maximum size.</td>
</tr>
<tr>
<td>WS_MAXIMIZEBOX</td>
<td>Creates a window that has a Maximize box.</td>
</tr>
<tr>
<td>WS_MINIMIZE</td>
<td>Creates a window of minimum size.</td>
</tr>
<tr>
<td>WS_MINIMIZEBOX</td>
<td>Creates a window that has a Minimize box.</td>
</tr>
</tbody>
</table>

Application's data segment. This feature can be suppressed by adding the DS_LOCALEDIT flag to the STYLE command for the dialog box. If this flag is not used, EM_GETHANDLE and EM_SETHANDLE messages must not be used since the storage for the control is not in the application's data segment. This feature does not affect edit controls created outside of dialog boxes.
Table 8.1: Window styles (continued)

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_OVERLAPPED</td>
<td>Creates an overlapped window. An overlapped window has a caption and a border.</td>
</tr>
<tr>
<td>WS_OVERLAPPEDWINDOW</td>
<td>Creates an overlapped window having the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles.</td>
</tr>
<tr>
<td>WS_POPUP</td>
<td>Creates a pop-up window. It cannot be used with WS_CHILD.</td>
</tr>
<tr>
<td>WS_POPUPWINDOW</td>
<td>Creates a pop-up window that has the styles WS_POPUP, WS_BORDER, and WS_SYSMENU. The WS_CAPTION style must be combined with the WS_POPUPWINDOW style to make the system menu visible.</td>
</tr>
<tr>
<td>WS_SIZEBOX</td>
<td>Creates a window that has a size box. Used only for windows with a title bar or with vertical and horizontal scroll bars.</td>
</tr>
<tr>
<td>WS_SYSMENU</td>
<td>Creates a window that has a System-menu box in its title bar. Used only for windows with title bars. If used with a child window, this style creates a Close box instead of a System-menu box.</td>
</tr>
<tr>
<td>WS_TABSTOP</td>
<td>Specifies one of any number of controls through which the user can move by using the TAB key. The TAB key moves the user to the next control specified by the WS_TABSTOP style. This style is valid only for controls.</td>
</tr>
<tr>
<td>WS_THICKFRAME</td>
<td>Creates a window with a thick frame that can be used to size the window.</td>
</tr>
<tr>
<td>WS_VISIBLE</td>
<td>Creates a window that is initially visible. This applies to overlapping and pop-up windows. For overlapping windows, the y parameter is used as a ShowWindow function parameter.</td>
</tr>
<tr>
<td>WS_VSCROLL</td>
<td>Creates a window that has a vertical scroll bar.</td>
</tr>
</tbody>
</table>

CAPTION

Syntax

CAPTION captiontext

This optional statement defines the dialog box's title. The title appears in the box's caption bar (if it has one).

The default caption is empty.

The captiontext field specifies an ASCII character string enclosed in double quotation marks.

The following example demonstrates the correct usage of the CAPTION statement:

CAPTION "Error!"
**MENU**

**Syntax**

`:MENU menuname`

This optional statement defines the dialog box's menu. If no statement is given, the dialog box has no menu.

The *menuname* field specifies the resource name or number of the menu to be used.

The following example demonstrates the correct usage of the **MENU** statement:

```plaintext```
MENU errmenu
```

**CLASS**

**Syntax**

`:CLASS class`

This optional statement defines the class of the dialog box. If no statement is given, the Windows standard dialog class will be used as the default.

The *class* field specifies an integer or a string, enclosed in double quotation marks, that identifies the class of the dialog box. If the window procedure for the class does not process a message sent to it, it must call the `DefDlgProc` function to ensure that all messages are handled properly for the dialog box. A private class can use `DefDlgProc` as the default window procedure. The class must be registered with the *cbWndExtra* field of the `WNDCLASS` data structure set to DLGWINDOWEXTRA.

The following example demonstrates the correct usage of the **CLASS** statement:

```plaintext```
CLASS "myclass"
```

**Comments**

The **CLASS** statement should be used with special cases, since it overrides the normal processing of a dialog box. The **CLASS** statement converts a dialog box to a window of the specified class; depending on the class, this could give undesirable results. Do not use the predefined control class names with this statement.
Syntax  FONT *pointsize, typeface*

This optional statement defines the font with which Windows will draw text in the dialog box. The font must have been previously loaded, either from WIN.INI or by calling *LoadFont*.

The *pointsize* field is an integer that specifies the size in points of the font.

The *typeface* field specifies an ASCII character string enclosed in double quotation marks that specifies the name of the typeface. This name must be identical to the name defined in the [fonts] section of WIN.INI.

The following example demonstrates the correct usage of the FONT statement:

```
FONT 12, "Helv"
```

Dialog control statements

The dialog control statements, given in the control-statements section of the DIALOG statement, define the attributes of the control windows that appear in the dialog box. A dialog box is empty unless one or more control statements are given. Control statements include the following:

- LTEXT
- RTEXT
- CTEXT
- CHECKBOX
- PUSHBUTTON
- LISTBOX
- GROUPBOX
- DEFPUSHBUTTON
- RADIOBUTTON
- EDITTEXT
- COMBOBOX
- ICON
- SCROLLBAR
- CONTROL

The control statements are discussed individually in the following sections. For more information on control classes and styles, see Tables 8.2, "Control classes," and 8.3, "Control styles."
### Syntax

LTEXT text, id, x, y, width, height, [[style]]

This statement defines a flush-left text control. It creates a simple rectangle that displays the given text flush-left in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

The text field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (&) ahead of the letter that will be the mnemonic. To use the ampersand as a character in a string, insert two ampersands (&&).

The id field takes a unique integer value that identifies the control.

The x and y fields take integer values that specify the x and y coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The GetDialogBaseUnits function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The width and height fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional style field can contain any combination (or none) of the following styles:
- WS_TABSTOP
- WS_GROUP

These styles are described in Table 8.1, "Window styles." Styles can be combined using the bitwise OR operator.

### Comments

The x, y, width, and height fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the x field.

The default style for LTEXT is SS_LEFT and WS_GROUP.

The following example demonstrates the correct usage of the LTEXT statement:
LTEXT "Enter Name:", 3, 10, 10, 40, 10

Syntax

RTEXT text, id, x, y, width, height, [[style]]

This statement defines a flush-right text control. It creates a simple rectangle that displays the given text flush-right in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

The text field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (&) ahead of the letter that will be the mnemonic. To use the ampersand as a character in a string, insert two ampersands (&&).

The id field takes a unique integer value that identifies the control.

The x and y fields take integer values that specify the x and y coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The GetDialogBaseUnits function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The width and height fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional style field can contain any combination (or none) of the following styles:

- WS_TABSTOP
- WS_GROUP

These styles are described in Table 8.1, "Window styles." Styles can be combined using the bitwise OR operator.

Comments

The x, y, width, and height fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the x field.

The default style for RTEXT is SS_RIGHT and WS_GROUP.

The following example demonstrates the correct usage of the RTEXT statement:
Syntax

CTEXT text, id, x, y, width, height, [[style]]

This statement defines a centered text control. It creates a simple rectangle that displays the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

The text field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (&) ahead of the letter that will be the mnemonic. To use the ampersand as a character in a string, insert two ampersands (&&).

The id field takes a unique integer value that identifies the control.

The x and y fields take integer values that specify the x and y coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The GetDialogBaseUnits function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The width and height fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional style field can contain any combination (or none) of the following styles:

- WS_TABSTOP
- WS_GROUP

These styles are described in Table 8.1, "Window styles." Styles can be combined using the bitwise OR operator.

Comments

The x, y, width, and height fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the x field.

The default style for CTEXT is SS_CENTER and WS_GROUP.

The following example demonstrates the correct usage of the CTEXT statement:
CHECKBOX

**Syntax**

CHECKBOX text, id, x, y, width, height, [[style]]

This statement defines a check-box control belonging to the BUTTON class. It creates a small rectangle (check box) that is highlighted when clicked. The given text is displayed just to the right of the check box. The control highlights the rectangle when the user clicks the mouse in it, and removes the highlight on the next click.

The `text` field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (`&`) ahead of the letter that will be the mnemonic. To use the ampersand as a character in a string, insert two ampersands (`&&`).

The `id` field takes a unique integer value that identifies the control.

The `x` and `y` fields take integer values that specify the `x` and `y` coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The `GetDialogBaseUnits` function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The `width` and `height` fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional `style` field can contain any combination (or none) of the following styles:

- `WS_TABSTOP`
- `WS_GROUP`

These styles are described in Table 8.1, "Window styles."

In addition to these styles, the `style` field may contain any combination (or none) of the BUTTON-class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

**Comments**

The `x`, `y`, `width`, and `height` fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the `x` field.
The default style for CHECKBOX is BS_CHECKBOX and WS_TABSTOP.
The following example demonstrates the correct usage of the CHECKBOX statement:

```
CHECKBOX "Arabic", 3, 10, 10, 40, 10
```

### Syntax

**PUSHBUTTON**  
```
PUSHBUTTON text, id, x, y, width, height, [style]
```

This statement defines a push-button control belonging to the BUTTON class. It creates a rectangle containing the given text. The control sends a message to its parent whenever the user clicks the mouse inside the rectangle.

The *text* field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (&) ahead of the letter that will be the mnemonic. To use the ampersand as a character in a string, insert two ampersands (&&).

The *id* field takes a unique integer value that identifies the control.

The *x* and *y* fields take integer values that specify the *x* and *y* coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The `GetDialogBaseUnits` function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The *width* and *height* fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional *style* field can contain any combination (or none) of the following styles:

- WS_TABSTOP
- WS_DISABLED
- WS_GROUP

These styles are described in Table 8.1, "Window styles."
In addition to these styles, the *style* field may contain any combination (or none) of the BUTTON-class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

**Comments**
The *x*, *y*, *width*, and *height* fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the *x* field.

The default style for **PUSHBUTTON** is BS_PUSHBUTTON and WS_TABSTOP.

The following example demonstrates the correct usage of the **PUSHBUTTON** statement:

```
PUSHBUTTON "ON", 7, 10, 10, 20, 10
```

---

**LISTBOX**

**Syntax**

```
LISTBOX id, x, y, width, height, [style]
```

This statement defines a list box belonging to the LISTBOX class. It creates a rectangle that contains a list of strings (such as filenames) from which the user can make selections.

The *id* field takes a unique integer value that identifies the control.

The *x* and *y* fields take integer values that specify the *x* and *y* coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The **GetDialogBaseUnits** function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The *width* and *height* fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional *style* field can contain any combination (or none) of the following styles:

- WS_BORDER
- WS_VSCROLL

These styles are described in Table 8.1, "Window styles."
In addition to these styles, the *style* field may contain any combination (or none) of the LISTBOX-class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

**Comments**
The *x*, *y*, *width*, and *height* fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the *x* field.

The default style for LISTBOX is LBS_NOTIFY, WS_VSCROLL, and WS_BORDER.

For information on the recommended keys for use in list-box controls, see the *System Application Architecture, Common User Access: Advanced Interface Design Guide*.

The following example demonstrates the correct usage of the LISTBOX statement:

```
LISTBOX 666, 10, 10, 50, 54
```

**GROUPBOX**

**Syntax**

```
GROUPBOX text, id, x, y, width, height, [[style]]
```

This statement defines a group box belonging to the BUTTON class. It creates a rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper-left corner.

The *text* field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (&) ahead of the letter that will be the mnemonic. Selecting the mnemonic moves the input focus to the next control in the group, in the order set in the resource file. To use the ampersand as a character in a string, insert two ampersands (&&).

The *id* field takes a unique integer value that identifies the control.

The *x* and *y* fields take integer values that specify the *x* and *y* coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The *GetDialogBaseUnits* function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.
The *width* and *height* fields take integer values that specify the width and height of the control. The width units are \(1/4\) of the dialog base width unit; the height units are \(1/8\) of the dialog base height unit.

The optional *style* field can contain any combination (or none) of the following styles:

- WS_TABSTOP
- WS_DISABLED

These styles are described in Table 8.1, "Window styles."

In addition to these styles, the *style* field may contain any combination (or none) of the BUTTON-class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

**Comments**

The *x*, *y*, *width*, and *height* fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the *x* field.

The default style for GROUPBOX is BS_GROUPBOX and WS_TABSTOP.

The following example demonstrates the correct usage of the GROUPBOX statement:

```
GROUPBOX "Output", 42, 10, 10, 30, 50
```

---

**DEFPUSHBUTTON**

**Syntax**

```
DEFPUSHBUTTON text, id, x, y, width, height, [[style]]
```

This statement defines a default push-button control that belongs to the BUTTON class. It creates a small rectangle with a bold outline that represents the default response for the user. The given text is displayed inside the button. The control highlights the button in the usual way when the user clicks the mouse in it and sends a message to its parent window.

The *text* field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (&) ahead of the letter that will be the mnemonic. To use the ampersand as a character in a string, insert two ampersands (&&).

The *id* field takes a unique integer value that identifies the control.

The *x* and *y* fields take integer values that specify the *x* and *y* coordinates of the upper-left corner of the control. The horizontal units are \(1/4\) of the dialog base width unit; the vertical units are \(1/8\) of the dialog base height unit. The current dialog base units are computed from the height and...
width of the current system font. The \texttt{GetDialogBaseUnits} function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The \texttt{width} and \texttt{height} fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional \textit{style} field can contain any combination (or none) of the following styles:

- \texttt{WS\_TABSTOP}
- \texttt{WS\_GROUP}
- \texttt{WS\_DISABLED}

These styles are described in Table 8.1, "Window styles."

In addition to these styles, the \textit{style} field may contain any combination (or none) of the \texttt{BUTTON} class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

\textbf{Comments}  
The \texttt{x}, \texttt{y}, \texttt{width}, and \texttt{height} fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the \texttt{x} field.

The default style for \texttt{DEFPUSHBUTTON} is \texttt{BS\_DEFPUSHBUTTON} and \texttt{WS\_TABSTOP}.

The following example demonstrates the correct usage of the \texttt{DEFPUSHBUTTON} statement:

\begin{verbatim}
DEFPUSHBUTTON "ON", 7, 10, 10, 20, 10
\end{verbatim}

\textbf{RADIOBUTTON}  

\textbf{Syntax}  \texttt{RADIOBUTTON text, id, x, y, width, height, [[style]]}

This statement defines a radio-button control belonging to the \texttt{BUTTON} class. It creates a small circle that has the given text displayed just to its right. The control highlights the button when the user clicks the mouse in it and sends a message to its parent window. The control removes the highlight and sends a message on the next click.

The \texttt{text} field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks. To add a mnemonic to the text string, place the ampersand (\&) ahead of the letter that will be the mnemonic. To use the ampersand as a character in a string, insert two ampersands (&&).
The \textit{id} field takes a unique integer value that identifies the control.

The $x$ and $y$ fields take integer values that specify the $x$ and $y$ coordinates of the upper-left corner of the control. The horizontal units are $1/4$ of the dialog base width unit; the vertical units are $1/8$ of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The \texttt{GetDialogBaseUnits} function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The \textit{width} and \textit{height} fields take integer values that specify the width and height of the control. The width units are $1/4$ of the dialog base width unit; the height units are $1/8$ of the dialog base height unit.

The optional \textit{style} field can contain any combination (or none) of the following styles:

- \texttt{WS\_TABSTOP}
- \texttt{WS\_GROUP}
- \texttt{WS\_DISABLED}

These styles are described in Table 8.1, "Window styles."

In addition to these styles, the \textit{style} field may contain any combination (or none) of the BUTTON-class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

\textbf{Comments}

The $x$, $y$, \textit{width}, and \textit{height} fields can use the addition operator (+) for relative positioning. For example, "$15 + 6$" can be used for the $x$ field.

The default style for \texttt{RADIOBUTTON} is \texttt{BS\_RADIOBUTTON} and \texttt{WS\_TABSTOP}.

The following example demonstrates the correct usage of the \texttt{RADIOBUTTON} statement:

\begin{verbatim}
RADIOBUTTON "AM 101", 10, 10, 10, 40, 10
\end{verbatim}

\textbf{EDITTEXT}

\texttt{EDITTEXT} \textit{id}, \textit{x}, \textit{y}, \textit{width}, \textit{height}, [[\textit{style}]]

This statement defines an EDIT control belonging to the EDIT class. It creates a rectangular region in which the user can enter and edit text. The control displays a cursor when the user clicks the mouse in it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the \texttt{BACKSPACE} and \texttt{DELETE} keys. The user can also use the
mouse to select characters to be deleted, or to select the place to insert new characters.

The \textit{id} field takes a unique integer value that identifies the control.

The \textit{x} and \textit{y} fields take integer values that specify the \textit{x} and \textit{y} coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The \texttt{GetDialogBaseUnits} function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The \textit{width} and \textit{height} fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional \textit{style} field can contain any combination (or none) of the following styles:

- \texttt{WS_TABSTOP}
- \texttt{WS_GROUP}
- \texttt{WS_VSCROLL}
- \texttt{WS_HSCROLL}
- \texttt{WS_DISABLED}

These styles are described in Table 8.1, "Window styles."

In addition to these styles, the \textit{style} field may contain any combination (or none) of the EDIT-class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator. The EDIT-class styles must not conflict with each other.

\textbf{Comments}

The \textit{x}, \textit{y}, \textit{width}, and \textit{height} fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the \textit{x} field.

The default style for \texttt{EDITTEXT} is \texttt{WS_TABSTOP}, \texttt{ES_LEFT}, and \texttt{WS_BORDER}.


The following example demonstrates the correct usage of the \texttt{EDITTEXT} statement:

\begin{verbatim}
EDITTEXT 3, 10, 10, 100, 10
\end{verbatim}
COMBOBOX

Syntax

COMBOBOX id, x, y, width, height, [[style]]

This statement defines a combo box belonging to the COMBOBOX class. A combo box consists of either a static text field or edit field combined with a list box. The list box can be displayed at all times or pulled down by the user. If the combo box contains a static text field, the text field always displays the selection (if any) in the list-box portion of the combo box. If it uses an edit field, the user can type in the desired selection; the list box highlights the first item (if any) which matches what the user has entered in the edit field. The user can then select the item highlighted in the list box to complete the choice. In addition, the combo box can be owner-draw and of fixed or variable height.

The id field takes a unique integer value that identifies the control.

The x and y fields take integer values that specify the x and y coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The GetDialogBaseUnits function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The width and height fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional style field can contain any combination (or none) of the following styles:

- WS_TABSTOP
- WS_GROUP
- WS_VSCROLL
- WS_DISABLED

These styles are described in Table 8.1, "Window styles."

In addition to these styles, the style field may contain any combination (or none) of the combo-box styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

- Comments

The x, y, width, and height fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the x field.
The default style for COMBOBOX is WS_TABSTOP and CBS_SIMPLE.

The following example demonstrates the correct usage of the COMBOBOX statement:

```
COMBOBOX 777, 10, 10, 50, 54, CBS_SIMPLE | WS_VSCROLL | WS_TABSTOP
```

**ICON**

**Syntax**

```
ICON text, id, x, y, width, height, [[style]]
```

This statement defines an icon control belonging to the STATIC class. It creates an icon displayed in the dialog box.

The *text* field specifies the name of an icon (not a filename) defined elsewhere in the resource file.

The *id* field takes a unique integer value that identifies the control.

The *x* and *y* fields take integer values that specify the *x* and *y* coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The `GetDialogBaseUnits` function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

For the ICON statement, the *width* and *height* fields are ignored; the icon automatically sizes itself.

The optional *style* field allows only the SS_ICON style.

**Comments**

The *x*, *y*, *width*, and *height* fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the *x* field.

The default style for ICON is SS_ICON.

The following example demonstrates the correct usage of the ICON statement:

```
ICON "myicon" 901, 30, 30
```
Syntax

SCROLLBAR id, x, y, width, height, [[style]]

This statement defines a scroll-bar control belonging to the SCROLLBAR class. It is a rectangle that contains a scroll thumb and has direction arrows at both ends. The scroll-bar control sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the thumb position. Scroll-bar controls can be positioned anywhere in a window and used whenever needed to provide scrolling input.

The id field takes a unique integer value that identifies the control.

The x and y fields take integer values that specify the location of the upper-left corner of the control in dialog units relative to the origin of the dialog box. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The GetDialogBaseUnits function returns the dialog base units in pixels.

The width and height fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

The optional style field can contain any combination (or none) of the following styles:

- WS_TABSTOP
- WS_GROUP
- WS_DISABLED

These styles are described in Table 8.1, "Window styles."

In addition to these styles, the style field may contain any combination (or none) of the SCROLLBAR-class styles described in Table 8.3, "Control styles." Styles can be combined using the bitwise OR operator.

Comments

The x, y, width, and height fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the x field.

The default style for SCROLLBAR is SBS_HORZ.

The following example demonstrates the correct usage of the SCROLLBAR statement:
CONTROL

Syntax

CONTROL text, id, class, style, x, y, width, height

This statement defines a user-defined control window.

The text field takes an ASCII string that specifies the text to be displayed. The string must be enclosed in double quotation marks.

The id field takes a unique integer value that identifies the control.

The class field takes a predefined name, character string, or integer value that defines the class. This can be any one of the control classes; for a list of the control classes, see Table 8.2, "Control classes." If the value is a predefined name supplied by the application, it must be an ASCII string enclosed in double quotation marks.

The style field takes a predefined name or integer value that specifies the style of the given control. The exact meaning of style depends on the class value. Tables 8.2, "Control classes," and 8.3, "Control styles," list the control classes and corresponding styles.

The x and y fields take integer values that specify the x and y coordinates of the upper-left corner of the control. The horizontal units are 1/4 of the dialog base width unit; the vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The GetDialogBaseUnits function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

The width and height fields take integer values that specify the width and height of the control. The width units are 1/4 of the dialog base width unit; the height units are 1/8 of the dialog base height unit.

Comments

The x, y, width, and height fields can use the addition operator (+) for relative positioning. For example, "15 + 6" can be used for the x field.

Table 8.2 describes the six control classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUTTON</td>
<td>A button control is a small rectangular child window that represents a &quot;button&quot; that the user can turn on or off by clicking it with the mouse. Button controls can be used alone or in groups, and can either be labeled or appear without text.</td>
</tr>
<tr>
<td><strong>Table 8.2: Control classes (continued)</strong></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Button</strong> controls typically change appearance when the user</td>
<td></td>
</tr>
<tr>
<td>clicks them.</td>
<td></td>
</tr>
<tr>
<td><strong>COMBOBOX</strong></td>
<td></td>
</tr>
<tr>
<td>Combo-box controls consist of a selection field similar to an</td>
<td></td>
</tr>
<tr>
<td>edit control plus a list box. The list box may be displayed at</td>
<td></td>
</tr>
<tr>
<td>all times or may be dropped down when the user selects a &quot;pop</td>
<td></td>
</tr>
<tr>
<td>box&quot; next to the selection field.</td>
<td></td>
</tr>
<tr>
<td>Depending on the style of the combo box, the user can or</td>
<td></td>
</tr>
<tr>
<td>cannot edit the contents of the selection field. If the list</td>
<td></td>
</tr>
<tr>
<td>box is visible, typing characters into the selection box will</td>
<td></td>
</tr>
<tr>
<td>cause the first list box entry which matches the characters</td>
<td></td>
</tr>
<tr>
<td>typed to be highlighted. Conversely, selecting an item in the</td>
<td></td>
</tr>
<tr>
<td>list box displays the selected text in the selection field.</td>
<td></td>
</tr>
<tr>
<td><strong>EDIT</strong></td>
<td></td>
</tr>
<tr>
<td>An edit control is a rectangular child window in which the</td>
<td></td>
</tr>
<tr>
<td>user can enter text from the keyboard. The user selects the</td>
<td></td>
</tr>
<tr>
<td>control, and gives it the input focus, by clicking the mouse</td>
<td></td>
</tr>
<tr>
<td>inside it or pressing the TAB key. The user can enter text</td>
<td></td>
</tr>
<tr>
<td>when the control displays a flashing caret. The mouse can be</td>
<td></td>
</tr>
<tr>
<td>used to move the cursor and select characters to be replaced,</td>
<td></td>
</tr>
<tr>
<td>or to position the cursor for inserting characters. The BACKSPACE key can be used to delete characters. Edit controls use the fixed-pitch font and display ANSI characters. They expand tab characters into as many space characters as are required to move the cursor to the next tab stop. Tab stops are assumed to be at every eighth character position.</td>
<td></td>
</tr>
<tr>
<td><strong>LISTBOX</strong></td>
<td></td>
</tr>
<tr>
<td>List-box controls consist of a list of character strings. The</td>
<td></td>
</tr>
<tr>
<td>control is used whenever an application needs to present a list</td>
<td></td>
</tr>
<tr>
<td>of names, such as filenames, that the user can view and select.</td>
<td></td>
</tr>
<tr>
<td>The user can select a string by pointing to the string with the</td>
<td></td>
</tr>
<tr>
<td>mouse and clicking a mouse button. When a string is selected,</td>
<td></td>
</tr>
<tr>
<td>it is highlighted, and a notification message is passed to the</td>
<td></td>
</tr>
<tr>
<td>parent window. A scroll bar can be used with a list-box control</td>
<td></td>
</tr>
<tr>
<td>to scroll lists that are too long or too wide for the control</td>
<td></td>
</tr>
<tr>
<td>window.</td>
<td></td>
</tr>
<tr>
<td><strong>SCROLLBAR</strong></td>
<td></td>
</tr>
<tr>
<td>A scroll-bar control is a rectangle that contains a scroll</td>
<td></td>
</tr>
<tr>
<td>thumb and has direction arrows at both ends. The scroll bar</td>
<td></td>
</tr>
<tr>
<td>sends a notification message to its parent whenever the user</td>
<td></td>
</tr>
<tr>
<td>clicks the mouse in the control. The parent is responsible for</td>
<td></td>
</tr>
<tr>
<td>updating the thumb position, if necessary. Scroll-bar controls</td>
<td></td>
</tr>
<tr>
<td>have the same appearance and function as the scroll bars used</td>
<td></td>
</tr>
<tr>
<td>in ordinary windows. But unlike scroll bars, scroll-bar controls</td>
<td></td>
</tr>
<tr>
<td>can be positioned anywhere within a window and used whenever</td>
<td></td>
</tr>
<tr>
<td>needed to provide scrolling input for a window. The scroll-bar</td>
<td></td>
</tr>
<tr>
<td>class also includes size-box controls. A size-box control is a</td>
<td></td>
</tr>
<tr>
<td>small rectangle that the user can expand to change the size of</td>
<td></td>
</tr>
<tr>
<td>the window.</td>
<td></td>
</tr>
<tr>
<td><strong>STATIC</strong></td>
<td></td>
</tr>
<tr>
<td>Static controls are simple text fields, boxes, and rectangles</td>
<td></td>
</tr>
<tr>
<td>that can be used to label, box, or separate other controls.</td>
<td></td>
</tr>
<tr>
<td>Static controls take no input and provide no output.</td>
<td></td>
</tr>
</tbody>
</table>
Table 8.3 describes the control styles for each of the control classes:

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUTTON class</td>
<td></td>
</tr>
<tr>
<td>BS_PUSHBUTTON</td>
<td>A small elliptical button containing the given text. The control sends a message to its parent whenever the user clicks the mouse inside the rectangle.</td>
</tr>
<tr>
<td>BS_DEFPUSHBUTTON</td>
<td>A small elliptical button with a bold border. This button represents the default user response. Any text is displayed within the button. Windows sends a message to the parent window when the user clicks the mouse in this button.</td>
</tr>
<tr>
<td>BS_CHECKBOX</td>
<td>A small rectangular button that can be checked; its border becomes bold when the user clicks the mouse in it. Any text appears to the right of the button.</td>
</tr>
<tr>
<td>BS_AUTOCHECKBOX</td>
<td>Identical to BS_CHECKBOX except that the button automatically toggles its state whenever the user clicks it.</td>
</tr>
<tr>
<td>BS_RADIobutton</td>
<td>A small circular button whose border becomes bold when the user clicks the mouse in it. In addition, to make the border bold, Windows sends a message to the button's parent notifying it that a click occurred. On the next click, Windows makes the border normal again and sends another message.</td>
</tr>
<tr>
<td>BS_AUTORADIOBUTTON</td>
<td>Identical to BS_RADIobutton except that when the button is checked, the application is notified with BN_CLICKED, and all other radio buttons in the group are unchecked.</td>
</tr>
<tr>
<td>BS_LEFTTEXT</td>
<td>Text appears on the left side of the radio button or check-box button. Use this style with BS_CHECKBOX, BS_3STATE, or BS_RADIobutton styles.</td>
</tr>
<tr>
<td>BS_3STATE</td>
<td>Identical to BS_CHECKBOX except that a button can be grayed as well as checked or unchecked. The grayed state is typically used to show that a check box has been disabled.</td>
</tr>
<tr>
<td>BS_AUTO3STATE</td>
<td>Identical to BS_3STATE except that the button automatically toggles its state when the user clicks it.</td>
</tr>
<tr>
<td>BS_GROUPBOX</td>
<td>A rectangle into which other buttons are grouped. Any text is displayed in the rectangle's upper-left corner.</td>
</tr>
<tr>
<td>BS_OWNERDRAW</td>
<td>An owner-draw button. The parent window is notified when the button is clicked. Notification includes a request to paint, invert, and disable the button.</td>
</tr>
</tbody>
</table>
Table 8.3: Control styles (continued)

<table>
<thead>
<tr>
<th>COMBOBOX class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBS_SIMPLE</td>
<td>Displays the list box at all times. The current selection in the list box is displayed in the edit control.</td>
</tr>
<tr>
<td>CBS_DROPDOWN</td>
<td>Is similar to CBS_SIMPLE, except that the list box is not displayed unless the user selects an icon next to the selection field.</td>
</tr>
<tr>
<td>CBS_DROPDOWNLIST</td>
<td>Is similar to CBS_DROPDOWN, except that the edit control is replaced by a static text item which displays the current selection in the list box.</td>
</tr>
<tr>
<td>CBS_OWNERDRAWFIXED</td>
<td>Specifies a fixed-height owner-draw combo box. The owner of the list box is responsible for drawing its contents; the items in the list box are all the same height.</td>
</tr>
<tr>
<td>CBS_OWNERDRAWVARIABLE</td>
<td>Specifies a variable-height owner-draw combo box. The owner of the list box is responsible for drawing its contents; the items in the list box can have different heights.</td>
</tr>
<tr>
<td>CBS_AUTOHSCROLL</td>
<td>Scrolls the text in the edit control to the right when the user types a character at the end of the line. If this style is not set, only text which fits within the rectangular boundary is allowed.</td>
</tr>
<tr>
<td>CBS_SORT</td>
<td>Sorts strings entered into the list box.</td>
</tr>
<tr>
<td>CBS_HASSTRINGS</td>
<td>Specifies an owner-draw combo box that contains items consisting of strings. The combo box maintains the memory and pointers for the strings so that the application can use the LB_GETTEXT message to retrieve the text for a particular item.</td>
</tr>
<tr>
<td>CBS_OEMCONVERT</td>
<td>Text entered in the combo box edit control is converted from the ANSI character set to the OEM character set and then back to ANSI. This ensures proper character conversion when the application calls the ANSIToOem function to convert an ANSI string in the combo box to OEM characters. This style is most useful for combo boxes that contain filenames and applies only to combo boxes created with the CBS_SIMPLE or CBS_DROPDOWN styles.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EDIT class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES_LEFT</td>
<td>Flush-left text.</td>
</tr>
<tr>
<td>ES_CENTER</td>
<td>Centered text. This style is valid in multiline edit controls only.</td>
</tr>
<tr>
<td>ES_RIGHT</td>
<td>Flush-right text. This style is valid in multiline edit controls only.</td>
</tr>
<tr>
<td>Control Style</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ES_Lowercase</td>
<td>Lowercase edit control. An edit control with this style converts all characters to lowercase as they are typed into the edit control.</td>
</tr>
<tr>
<td>ES_Uppercase</td>
<td>Uppercase edit control. An edit control with this style converts all characters to uppercase as they are typed into the edit control.</td>
</tr>
<tr>
<td>ES_Password</td>
<td>Password edit control. An edit control with this style displays all characters as an asterisk (*) as they are typed into the edit control. An application can use the EM_SETPASSWORDCHAR message to change the character that is displayed.</td>
</tr>
<tr>
<td>ES_Multiline</td>
<td>Multiple-line edit control. (The default is single-line.) If the ES_AUTOVSCROLL style is specified, the edit control shows as many lines as possible and scrolls vertically when the user presses the ENTER key. (This is actually the carriage-return character, which the edit control expands to a carriagereturn/line-feed combination. A line feed is not treated the same as a carriage return.) If ES_AUTOVSCROLL is not given, the edit control shows as many lines as possible and beeps if the user presses ENTER when no more lines can be displayed. If the ES_AUTOHSCROLL style is specified, the multiple-line edit control automatically scrolls horizontally when the caret goes past the right edge of the control. To start a new line, the user must press the ENTER key. If ES_AUTOHSCROLL is not given, the control automatically wraps words to the beginning of the next line when necessary; a new line is also started if the user presses ENTER. The position of the wordwrap is determined by the window size. If the window size changes, the wordwrap position changes, and the text is redisplayed. Multiple-line edit controls can have scroll bars. An edit control with scroll bars processes its own scroll-bar messages. Edit controls without scroll bars scroll as described above, and process any scroll messages sent by the parent window.</td>
</tr>
<tr>
<td>ES_AutoVScroll</td>
<td>Text is automatically scrolled up one page when the user presses the ENTER key on the last line.</td>
</tr>
<tr>
<td>ES_AutoHScroll</td>
<td>Text is automatically scrolled to the right by 10 characters when the user types a character at the end of the line. When the user presses</td>
</tr>
</tbody>
</table>
Table 8.3: Control styles (continued)

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES_NOHIDESEL</td>
<td>Normally, an edit control hides the selection when the control loses the input focus, and inverts the selection when the control receives the input focus. Specifying ES_NOHIDESEL overrides this default action. Text entered in the edit control is converted from the ANSI character set to the OEM character set and then back to ANSI. This ensures proper character conversion when the application calls the <code>AnsiToOem</code> function to convert an ANSI string in the edit control to OEM characters. This style is most useful for edit controls that contain filenames.</td>
</tr>
<tr>
<td>ES_OEMCONVERT</td>
<td>Text entered in the edit control is converted from the ANSI character set to the OEM character set and then back to ANSI. This ensures proper character conversion when the application calls the <code>AnsiToOem</code> function to convert an ANSI string in the edit control to OEM characters. This style is most useful for edit controls that contain filenames.</td>
</tr>
</tbody>
</table>

LISTBOX class

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBS_STANDARD</td>
<td>Strings in the list box are sorted alphabetically and the parent window receives an input message whenever the user clicks or double-clicks a string. The list box contains borders on all sides.</td>
</tr>
<tr>
<td>LBS_EXTENDEDSEL</td>
<td>The user can select multiple items using the mouse with the SHIFT and/or the CONTROL key or special key combinations.</td>
</tr>
<tr>
<td>LBS_HASSTRINGS</td>
<td>An owner-draw list box contains items consisting of strings. The list box maintains the memory and pointers for the strings so the application can use the <code>LB_GETTEXT</code> message to retrieve the text for a particular item.</td>
</tr>
<tr>
<td>LBS_NOTIFY</td>
<td>The parent receives an input message whenever the user clicks or double-clicks a string.</td>
</tr>
<tr>
<td>LBS_MULTIPLESEL</td>
<td>The string selection is toggled each time the user clicks or double-clicks the string. Any number of strings can be selected.</td>
</tr>
<tr>
<td>LBS_MULTICOLUMN</td>
<td>The list box contains multiple columns. The list box can be scrolled horizontally. The <code>LB_SETCOLUMNWIDTH</code> message sets the width of the columns.</td>
</tr>
<tr>
<td>LBS_NOINTEGRALHEIGHT</td>
<td>The size of the list box is exactly the size specified by the application when it created the list box. Normally, Windows sizes a list box so that the list box does not display partial items.</td>
</tr>
<tr>
<td>LBS_SORT</td>
<td>The strings in the list box are sorted alphabetically.</td>
</tr>
<tr>
<td>LBS_NOREDRAW</td>
<td>The list-box display is not updated when changes are made. This style can be changed</td>
</tr>
</tbody>
</table>
Table 8.3: Control styles (continued)

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBS_OWNERDRAWFIXED</td>
<td>The owner of the list box is responsible for drawing its contents; the items in the list box are all the same height.</td>
</tr>
<tr>
<td>LBS_OWNERDRAWVARIABLE</td>
<td>The owner of the list box is responsible for drawing its contents; the items in the list box are variable in height.</td>
</tr>
<tr>
<td>LBS_USETABSTOPS</td>
<td>The list box is able to recognize and expand tab characters when drawing its strings. The default tab positions are set at every 32 dialog units. (A dialog unit is a horizontal or vertical distance. One horizontal dialog unit is equal to 1/4 of the current dialog base width unit. The dialog base units are computed from the height and width of the current system font. The GetDialogBaseUnits function returns the size of the dialog base units in pixels.)</td>
</tr>
<tr>
<td>LBS_WANTKEYBOARDINPUT</td>
<td>The owner of the list box receives WM_VKEYTOITEM or WM_CHARTOITEM messages whenever the user presses a key when the list box has input focus. This allows an application to perform special processing on the keyboard input.</td>
</tr>
</tbody>
</table>

SCROLLBAR class

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBS_VERT</td>
<td>Vertical scroll bar. If neither SBS_RIGHTALIGN nor SBS_LEFTALIGN is specified, the scroll bar has the height, width, and position given in the CreateWindow function.</td>
</tr>
<tr>
<td>SBS_RIGHTALIGN</td>
<td>Used with SBS_VERT. The right edge of the scroll bar is aligned with the right edge of the rectangle specified by the x, y, width, and height values given in the CreateWindow function. The scroll bar has the default width for system scroll bars.</td>
</tr>
<tr>
<td>SBS_LEFTALIGN</td>
<td>Used with SBS_VERT. The left edge of the scroll bar is aligned with the left edge of the rectangle specified by the x, y, width, and height values given in the CreateWindow function. The scroll bar has the default width for system scroll bars.</td>
</tr>
<tr>
<td>SBS_HORZ</td>
<td>Horizontal scroll bar. If neither SBS_BOTTOMALIGN nor SBS_TOPALIGN is specified, the scroll bar has the height, width, and position given in the CreateWindow function.</td>
</tr>
<tr>
<td>SBS_TOPALIGN</td>
<td>Used with SBS_HORZ. The top edge of the scroll bar is aligned with the top edge of the rectangle specified by the x, y, width, and height values given in the CreateWindow function.</td>
</tr>
<tr>
<td>Control Style</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>SBS_BOTTOMALIGN</strong></td>
<td>Used with SBS_HORIZ. The bottom edge of the scroll bar is aligned with the bottom edge of the rectangle specified by the (x, y, width,) and (height) values given in the <code>CreateWindow</code> function. The scroll bar has the default height for system scroll bars.</td>
</tr>
<tr>
<td><strong>SBS_SIZEBOX</strong></td>
<td>Size box. If neither <code>SBS_SIZEBOXBOTTOMRIGHTALIGN</code> nor <code>SBS_SIZEBOXTOPLEFTALIGN</code> is specified, the size box has the height, width, and position given in the <code>CreateWindow</code> function.</td>
</tr>
<tr>
<td><strong>SBS_SIZEBOXTOPLEFTALIGN</strong></td>
<td>Used with SBS_SIZEBOX. The top-left corner of the size box is aligned with the top-left corner of the rectangle specified by the (x, y, width,) and (height) values given in the <code>CreateWindow</code> function. The size box has the default size for system size boxes.</td>
</tr>
<tr>
<td><strong>SBS_SIZEBOXBOTTOMRIGHTALIGN</strong></td>
<td>Used with SBS_SIZEBOX. The bottom-right corner of the size box is aligned with the bottom-right corner of the rectangle specified by the (x, y, width,) and (height) values given in the <code>CreateWindow</code> function. The size box has the default size for system size boxes.</td>
</tr>
<tr>
<td><strong>STATIC class</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SS_LEFT</strong></td>
<td>A simple rectangle displaying the given text flush left in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.</td>
</tr>
<tr>
<td><strong>SS_CENTER</strong></td>
<td>A simple rectangle displaying the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.</td>
</tr>
<tr>
<td><strong>SS_RIGHT</strong></td>
<td>A simple rectangle displaying the given text flush right in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.</td>
</tr>
<tr>
<td><strong>SS_LEFTNOWORDWRAP</strong></td>
<td>A simple rectangle displaying the given text flush left in the rectangle. Tabs are expanded, but words are not wrapped. Text that extends past the end of a line is clipped.</td>
</tr>
</tbody>
</table>
Table 8.3: Control styles (continued)

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS_SIMPLE</td>
<td>Designates a simple rectangle and displays a single line of text flush-left in the rectangle. The line of text cannot be shortened or altered in any way. (The control's parent window or dialog box must not process the WM_CTLCOLOR message.)</td>
</tr>
<tr>
<td>SS_NOPREFIX</td>
<td>Unless this style is specified, windows will interpret any &quot;&amp;&quot; characters in the control's text to be accelerator prefix characters. In this case, the &quot;&amp;&quot; is removed and the next character in the string is underlined. If a static control is to contain text where this feature is not wanted, SS_NOPREFIX may be added. This static-control style may be included with any of the defined static controls. You can combine SS_NOPREFIX with other styles by using the bitwise OR operator. This is most often used when filenames or other strings that may contain an &quot;&amp;&quot; need to be displayed in a static control in a dialog box.</td>
</tr>
<tr>
<td>SS_ICON</td>
<td>An icon displayed in the dialog box. The given text is the name of an icon (not a filename) defined elsewhere in the resource file. For the ICON statement, the width and height parameters in the CreateWindow function are ignored; the icon automatically sizes itself.</td>
</tr>
<tr>
<td>SS_BLACKRECT</td>
<td>A rectangle filled with the color used to draw window frames. This color is black in the default Windows color scheme.</td>
</tr>
<tr>
<td>SS_GRAYRECT</td>
<td>A rectangle filled with the color used to fill the screen background. This color is gray in the default Windows color scheme.</td>
</tr>
<tr>
<td>SS_WHITERECT</td>
<td>A rectangle filled with the color used to fill window backgrounds. This color is white in the default Windows color scheme.</td>
</tr>
<tr>
<td>SS_BLACKFRAME</td>
<td>Box with a frame drawn with the same color as window frames. This color is black in the default Windows color scheme.</td>
</tr>
<tr>
<td>SS_GRAYFRAME</td>
<td>Box with a frame drawn with the same color as the screen background (desktop). This color is gray in the default Windows color scheme.</td>
</tr>
<tr>
<td>SS_WHITEFRAME</td>
<td>Box with a frame drawn with the same color as window backgrounds. This color is white in the default Windows color scheme.</td>
</tr>
<tr>
<td>SS_USERITEM</td>
<td>User-defined item.</td>
</tr>
</tbody>
</table>
Directives

The resource directives are special statements that define actions to be performed on the script file before it is compiled. The directives can assign values to names, include the contents of files, and control compilation of the script file.

The resource directives are identical to the directives used in the C programming language.

#include statement

Syntax  
#include filename

This directive copies the contents of the file specified by filename into your resource script before the Resource Compiler processes the script. It replaces the rcinclude directive of versions prior to Windows 3.0.

The filename field is an ASCII string that specifies the DOS filename of the file to be included, using the same syntax as the C-language preprocessor #include directive. A forward slash (/) can be used instead of a backslash (for example, "root/sub"). If the filename has the .H or .C extension, only the preprocessor directives in the file are processed. Otherwise, this directive processes the entire contents of the file.

The following example demonstrates the correct usage of the #include statement:

```
#include "WINDOWS.H"
PenSelect MENU
BEGIN
    MenuItem "$Black pen", BLACK_PEN
END
```

#define statement

Syntax  
#define name value

This directive assigns the given value to name. All subsequent occurrences of name are replaced by value.

The value field takes any integer value, character string, or line of text.

Chapter 8, Resource script statements 103
The following example demonstrates the correct usage of the `#define` statement:

```
#define nonzero 1
#define USERCLASS "MyControlClass"
```

---

### `#undef` statement

**Syntax**

```
#undef name
```

This directive removes the current definition of `name`. All subsequent occurrences of `name` are processed without replacement.

The following example demonstrates the correct usage of the `#undef` statement:

```
#undef nonzero
#undef USERCLASS
```

---

### `#ifdef` statement

**Syntax**

```
#ifdef name
```

This directive carries out conditional compilation of the resource file by checking the specified `name`. If `name` has been defined using a `#define` directive, `#ifdef` directs the Resource Compiler to continue with the statement immediately after `#ifdef`. If `name` has not been defined, `#ifdef` directs the compiler to skip all statements up to the next `#endif` directive.

The following example demonstrates the correct usage of the `#ifdef` statement:

```
#ifdef Debug
  errbox BITMAP errbox.bmp
#endif
```
#ifndef statement

Syntax  
#ifndef name

This directive carries out conditional compilation of the resource file by checking the specified name. If name has not been defined or if its definition has been removed using the #undef directive, ifndef directs the Resource Compiler to continue processing statements up to the next #endif, #else, or #elif directive, and then to skip to the statement after #endif. If name is defined, ifndef directs the compiler to skip to the next #endif, #else, or #elif directive.

The following example demonstrates the correct usage of the ifndef statement:

```
#ifndef Optimize
  errbox BITMAP errbox.bmp
#endif
```

# if statement

Syntax  
#if constant-expression

This directive carries out conditional compilation of the resource file by checking the specified constant-expression. If constant-expression is nonzero, #if directs the Resource Compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after #endif. If constant-expression is zero, #if directs the compiler to skip to the next #endif, #else, or #elif directive.

The constant-expression field specifies a defined name, an integer constant, or an expression consisting of names, integers, and arithmetical and relational operators.

The following example demonstrates the correct usage of the #if statement:

```
#if Version<3
  errbox BITMAP errbox.bmp
#endif
```
#elif statement

**Syntax**

```c
#elif constant-expression
```

This directive marks an optional clause of a conditional compilation block defined by an `#ifdef`, `#ifndef`, or `#if` directive. The `#elif` directive carries out conditional compilation of the resource file by checking the specified `constant-expression`. If `constant-expression` is nonzero, `#elif` directs the Resource Compiler to continue processing statements up to the next `#endif`, `#else`, or `#elif` directive, then skip to the statement after `#endif`. If `constant-expression` is zero, `#elif` directs the compiler to skip to the next `#endif`, `#else`, or `#elif` directive. Any number of `#elif` directives can be used in a conditional block.

The `constant-expression` field specifies a defined name, an integer constant, or an expression consisting of names, integers, and arithmetical and relational operators.

The following demonstrates the correct usage of the `#elif` statement:

```c
#if Version<3
 errbox BITMAP errbox.bmp
#elif Version<7
 errbox BITMAP userbox.bmp
#endif
```

---

#else statement

**Syntax**

```c
#else
```

This directive marks an optional clause of a conditional compilation block defined by an `#ifdef`, `#ifndef`, or `#if` directive. The `#else` directive must be the last directive before `#endif`.

The following example demonstrates the correct usage of the `#else` statement:

```c
#ifdef Debug
 errbox BITMAP errbox.bmp
#else
 errbox BITMAP userbox.bmp
#endif
```
#ifdef statement

Syntax  #endif

This directive marks the end of a conditional compilation block defined by an #if or #ifdef directive. One #if or #endif is required for each #ifdef directive.
This chapter describes the file formats used to create, execute, and supply data to Microsoft Windows applications. These files include the following:

- Bitmap files
- Icon resource files
- Cursor resource files
- Clipboard files
- Metafiles

**Bitmap file formats**

Windows version 3.0 bitmap files store a bitmap in a device-independent format which allows Windows to display the bitmap on any device. In this case, the term "device independent" means that the bitmap specifies pixel color in a form independent of the method used by any particular device to represent color. The assumed file extension of a Windows device-independent bitmap file is .BMP.

Each bitmap file contains a **BITMAPFILEHEADER** data structure immediately followed by a single, device-independent bitmap (DIB) consisting of a **BITMAPINFO** data structure and an array of bytes that defines the bitmap bits.

Windows version 3.0 also reads bitmap files in the format read by Microsoft OS/2 Presentation Manager version 1.2. These files consist of a **BITMAPFILEHEADER** data structure immediately followed by a
BITMAPCOREINFO data structure. Following this data structure is an array of bytes that defines the bitmap bits.

See Chapter 7, "Data types and structures," for information on the BITMAPFILEHEADER, BITMAPCOREINFO and BITMAPINFO data structures.

Icon resource file format

An icon resource file (with the .ICO file extension) can be device independent both for color and resolution.

Icon resource files can contain multiple device-independent bitmaps defining the icon image, one for each targeted display-device resolution. Windows detects the resolution of the current display and matches it against the x and y pixel-size values specified for each version of the image. If Windows determines that there is an exact match between an icon image and the current device, then it uses the matching image; otherwise, it selects the closest match and stretches the image to the proper size.

If an icon resource file contains more than one image for a particular resolution, Windows uses the icon image that most closely matches the color capabilities of the current display device. If no image exists which exactly matches the device capabilities, Windows selects the image which has the greatest number of colors without exceeding the number of display-device colors. If all images exceed the color capabilities of the current display device, then Windows uses the icon image with the least number of colors.

The icon resource file contains a header structure at the beginning of the file which identifies the type and number of icon images contained in the file. The following shows the format of this header:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icoReserved</td>
<td>WORD Is reserved and must be set to 0.</td>
</tr>
<tr>
<td>icoResourceType</td>
<td>WORD Specifies the type of resource contained in the file. For an icon resource, this field must be 1.</td>
</tr>
<tr>
<td>icoResourceCount</td>
<td>WORD Specifies the number of images contained in the file.</td>
</tr>
</tbody>
</table>

The resource directory follows this header. The resource directory consists of one or more arrays of resource descriptors. The icoResourceCount specifies the number of arrays. This list shows the format of the array:
Icons can be in color. To achieve transparency, the DIB for each icon will consist of two parts:

1. A color bitmap which supplies the XOR mask for the icon.
2. A monochrome bitmap which provides the AND mask that defines the transparent portion of the icon.

The monochrome bitmap does not contain a DIB header, but instead immediately follows the color bitmap. It must have the same pixel height as the color bitmap.

**Cursor resource file format**

Like icon resource files, cursor resource files (with the .CUR file extension) may contain multiple images to match targeted display-device resolutions. In the case of cursors, Windows determines the best match for a particular display-device driver by examining the width and height of the cursor images.

The cursor resource file contains a header structure at the beginning of the file which identifies the type and number of resources in the file. The following shows the format of this header:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curReserved</td>
<td>WORD Is reserved and must be set to 0.</td>
</tr>
<tr>
<td>curResourceType</td>
<td>WORD Specifies the type of resource contained in the file. For a cursor resource, this field must be 2.</td>
</tr>
<tr>
<td>curResourceCount</td>
<td>WORD Specifies the number of resources contained in the file.</td>
</tr>
</tbody>
</table>
The resource directory follows this header. The resource directory consists of one or more arrays of resource descriptors. The `curResourceCount` specifies the number of arrays. The following shows the format of the array:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>curWidth</code></td>
<td>BYTE Specifies the width in pixels of this form of the cursor image.</td>
</tr>
<tr>
<td><code>curHeight</code></td>
<td>BYTE Specifies the height in pixels of this form of the cursor image.</td>
</tr>
<tr>
<td><code>ColorCount</code></td>
<td>BYTE Specifies the number of colors in this form of the icon image. Acceptable values are 2, 8, or 16.</td>
</tr>
<tr>
<td><code>Reserved</code></td>
<td>BYTE Is reserved and must be set to 0.</td>
</tr>
<tr>
<td><code>curXHotspot</code></td>
<td>WORD Specifies in pixels the horizontal position of the hotspot.</td>
</tr>
<tr>
<td><code>curYHotspot</code></td>
<td>WORD Specifies in pixels the vertical position of the hotspot.</td>
</tr>
<tr>
<td><code>curDIBSize</code></td>
<td>DWORD Specifies in bytes the size of the pixel array for this form of the cursor image.</td>
</tr>
<tr>
<td><code>curDIBOffset</code></td>
<td>DWORD Specifies in bytes the offset to the device-independent bitmap for this form. The offset is from the beginning of the file.</td>
</tr>
</tbody>
</table>

Cursors are monochrome. The bitmap for a cursor consists of two parts; the first half is the XOR mask specifying the visible image, and the second half is the AND mask specifying the transparent portion of the cursor image. The two parts must be of equal width and height. By combining the values in corresponding mask bits, Windows determines whether a pixel is black, white, inverted, or transparent.

Table 9.1 shows what values are necessary to produce the corresponding colors, inversions, or transparencies:

<table>
<thead>
<tr>
<th>AND mask</th>
<th>XOR mask</th>
<th>Resultant Pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Value</td>
<td>Bit Value</td>
<td>Bit Value</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Software development kit
Clipboard file format

The Windows clipboard saves and reads clipboard data in files with the .CLP extension. A clipboard-data file contains a value that identifies it as a clipboard-data file, one or more data structures defining the format, size, and location of the clipboard data, and one or more blocks of the actual data.

The clipboard-data file begins with a header consisting of two fields. The following describes these fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileIdentifier</td>
<td>WORD Identifies the file as a clipboard-data file. This field must be set to CLP_ID.</td>
</tr>
<tr>
<td>FormatCount</td>
<td>WORD Specifies the number of clipboard formats contained in the file.</td>
</tr>
</tbody>
</table>

This header is followed by one or more data structures, each of which identifies the format, size, and offset of a block of clipboard data. The following shows the fields of this data structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormatID</td>
<td>WORD Specifies the clipboard-format ID of the clipboard data. See the description of the SetClipboardData function in Chapter 4, &quot;Functions directory,&quot; in Reference, Volume 1, for information on clipboard formats.</td>
</tr>
<tr>
<td>LenData</td>
<td>DWORD Specifies in bytes the length of the clipboard data.</td>
</tr>
<tr>
<td>OffData</td>
<td>DWORD Specifies in bytes the offset of the clipboard-data block. The offset is from the beginning of the file.</td>
</tr>
<tr>
<td>Name</td>
<td>Is a 79-character array that specifies the format name for a private clipboard format.</td>
</tr>
</tbody>
</table>

The first block of clipboard data follows the last of these structures. For bitmaps and metafiles, the bits follow immediately after the bitmap header and the METAFILEPICT data structures.

Metafile format

A metafile consists of a collection of graphics device interface (GDI) function calls that create specific images on a device. Metafiles provide convenient storage for images that appear repeatedly in applications, and also allow you to use the clipboard to cut and paste images from one application to another.
Metafiles store images as a series of GDI function calls. After storing the function calls, applications play a metafile to generate an image on a device.

When an object is created during playback, GDI adds the handle of the object to the first available entry in the metafile handle table. GDI clears the table entry corresponding to the object when it is deleted during playback, allowing the table entry to be reused when another object is created.

Functions described in this section are discussed in greater detail in Chapter 4, "Functions directory," in Reference, Volume 1.

The metafile itself consists of two parts: a header and a list of records. The header contains a description of the size (in words) of the metafile and the number of drawing objects it uses. The list of records contains the GDI functions. The drawing objects can be pens, brushes, or bitmaps.

---

**Metafile header**

The following structured list shows the format of the metafile header:

```c
struct{
    WORD  mtType;
    WORD  mtHeaderSize;
    WORD  mtVersion;
    DWORD mtSize;
    WORD  mtNoObjects;
    DWORD mtMaxRecord;
    WORD  mtNoParameters;
}
```

The metafile header contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtType</td>
<td>Specifies whether the metafile is in memory or recorded in a disk file. It is one of these two values:</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong>  <strong>Meaning</strong></td>
</tr>
<tr>
<td></td>
<td>1            Metafile is in memory</td>
</tr>
<tr>
<td></td>
<td>2            Metafile is in a disk file</td>
</tr>
<tr>
<td>mtHeaderSize</td>
<td>Specifies the size in words of the metafile header.</td>
</tr>
<tr>
<td>mtVersion</td>
<td>Specifies the Windows version number. The version number for Windows version 3.0 is 0x300.</td>
</tr>
<tr>
<td>mtSize</td>
<td>Specifies the size in words of the file.</td>
</tr>
<tr>
<td>mtNoObjects</td>
<td>Specifies the maximum number of objects that exist in the metafile at the same time.</td>
</tr>
</tbody>
</table>
mtMaxRecord  Specifies the size in words of the largest record in the metafile.
mtNoParameters  Is not used.

Metafile records

A series of records follows the metafile header. Metafile records describe GDI functions. GDI stores most of the GDI functions that an application can use to create metafiles in similar, typical records. "Typical metafile record," later in this section, shows the format of the typical metafile record. Table 9.2, "GDI functions and values," lists the functions which GDI records in typical records, along with their respective function numbers.

The remainder of the functions contain more complex structures in their records. "Function-specific records," later in this section, describes the records for these functions.

In some cases, there are two versions of a metafile record. One version represents the record created by versions of Windows prior to version 3.0, while the second version represents the record created by Windows versions 3.0 and later. Windows 3.0 plays all metafile versions, but stores only 3.0 versions. Windows versions prior to 3.0 will not play metafiles recorded by Windows 3.0.

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc</td>
<td>0x0817</td>
</tr>
<tr>
<td>Chord</td>
<td>0x0830</td>
</tr>
<tr>
<td>Ellipse</td>
<td>0x0418</td>
</tr>
<tr>
<td>ExcludeClipRect</td>
<td>0x0415</td>
</tr>
<tr>
<td>FloodFill</td>
<td>0x0419</td>
</tr>
<tr>
<td>IntersectClipRect</td>
<td>0x0416</td>
</tr>
<tr>
<td>LineTo</td>
<td>0x0213</td>
</tr>
<tr>
<td>MoveTo</td>
<td>0x0214</td>
</tr>
<tr>
<td>OffsetClipRgn</td>
<td>0x0220</td>
</tr>
<tr>
<td>OffsetViewportOrg</td>
<td>0x0211</td>
</tr>
<tr>
<td>OffsetWindowOrg</td>
<td>0x020F</td>
</tr>
<tr>
<td>PatBlt</td>
<td>0x061D</td>
</tr>
<tr>
<td>Pie</td>
<td>0x081A</td>
</tr>
<tr>
<td><strong>RealizePalette (3.0 and later)</strong></td>
<td><strong>0x0035</strong></td>
</tr>
<tr>
<td><strong>Rectangle</strong></td>
<td><strong>0x041B</strong></td>
</tr>
<tr>
<td><strong>ResizePalette (3.0 and later)</strong></td>
<td><strong>0x0139</strong></td>
</tr>
<tr>
<td>RestoreDC</td>
<td>0x0127</td>
</tr>
<tr>
<td>RoundRect</td>
<td>0x061C</td>
</tr>
<tr>
<td>SaveDC</td>
<td>0x001E</td>
</tr>
<tr>
<td>ScaleViewportExt</td>
<td>0x0412</td>
</tr>
</tbody>
</table>
### Table 9.2: GDI functions and values (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScaleWindowExt</td>
<td>0x0400</td>
</tr>
<tr>
<td>SetBkColor</td>
<td>0x0201</td>
</tr>
<tr>
<td>SetBkMode</td>
<td>0x0102</td>
</tr>
<tr>
<td>SetMapMode</td>
<td>0x0103</td>
</tr>
<tr>
<td>SetMapperFlags</td>
<td>0x0231</td>
</tr>
<tr>
<td>SetPixel</td>
<td>0x041F</td>
</tr>
<tr>
<td>SetPolyFillMode</td>
<td>0x0106</td>
</tr>
<tr>
<td>SetROP2</td>
<td>0x0104</td>
</tr>
<tr>
<td>SetStretchBltMode</td>
<td>0x0107</td>
</tr>
<tr>
<td>SetTextAlign</td>
<td>0x0108</td>
</tr>
<tr>
<td>SetTextCharExtra</td>
<td>0x0209</td>
</tr>
<tr>
<td>SetTextColor</td>
<td>0x020A</td>
</tr>
<tr>
<td>SetTextJustification</td>
<td>0x020B</td>
</tr>
<tr>
<td>SetViewportExt</td>
<td>0x020C</td>
</tr>
<tr>
<td>SetViewportOrg</td>
<td>0x020D</td>
</tr>
<tr>
<td>SetWindowExt</td>
<td>0x020E</td>
</tr>
<tr>
<td>SetWindowOrg</td>
<td>0x020F</td>
</tr>
</tbody>
</table>

### Typical metafile record

The following structured list shows the format of a typical metafile record:

```c
struct{
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

A typical metafile record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the size in words of the record.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Is an array of words containing the function parameters, in the reverse order in which they are passed to the function.</td>
</tr>
</tbody>
</table>

### Function-specific records

Some metafile records contain data structures in the parameter field. This section contains definitions for these records.
AnimatePalette record 3.0

The AnimatePalette record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0436.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>First entry to be animated.</td>
</tr>
<tr>
<td>numentries</td>
<td>Number of entries to be animated.</td>
</tr>
<tr>
<td>entries</td>
<td>PALETTEENTRY blocks.</td>
</tr>
</tbody>
</table>

BitBlt record (prior to 3.0)

The BitBlt record stored by Windows versions prior to 3.0 contains a device-dependent bitmap which may not be suitable for playback on all devices. The following is the format of this record:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0922.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>raster op</td>
<td>High word of the raster operation.</td>
</tr>
<tr>
<td>SY</td>
<td>The y-coordinate of the source origin.</td>
</tr>
<tr>
<td>SX</td>
<td>The x-coordinate of the source origin.</td>
</tr>
<tr>
<td>DYE</td>
<td>Destination y-extent.</td>
</tr>
<tr>
<td>DXE</td>
<td>Destination x-extent.</td>
</tr>
<tr>
<td>DY</td>
<td>The y-coordinate of destination origin.</td>
</tr>
<tr>
<td>DX</td>
<td>The x-coordinate of destination origin.</td>
</tr>
</tbody>
</table>
### BitBlt record 3.0

The **BitBlt** record stored by Windows versions 3.0 and later contains a device-independent bitmap suitable for playback on any device. The following is the format of this record:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0940.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td></td>
<td><strong>Element</strong></td>
</tr>
<tr>
<td></td>
<td>raster op</td>
</tr>
<tr>
<td></td>
<td>SY</td>
</tr>
<tr>
<td></td>
<td>SX</td>
</tr>
<tr>
<td></td>
<td>DYE</td>
</tr>
<tr>
<td></td>
<td>DXE</td>
</tr>
<tr>
<td></td>
<td>DY</td>
</tr>
<tr>
<td></td>
<td>DX</td>
</tr>
<tr>
<td></td>
<td>BitmapInfo</td>
</tr>
<tr>
<td></td>
<td>bits</td>
</tr>
</tbody>
</table>

### CreateBrushIndirect record

The **CreateBrushIndirect** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    LOGBRUSH rdParm;
}
```

This record contains the following fields:
CreateFontIndirect record

The **CreateFontIndirect** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    LOGFONT rdParm;
};
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x02FC.</td>
</tr>
<tr>
<td>rdParm</td>
<td>Specifies the logical brush.</td>
</tr>
</tbody>
</table>

CreatePalette record 3.0

The **CreatePalette** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    LOGPALETTE rdParm;
};
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x00F7.</td>
</tr>
<tr>
<td>rdParm</td>
<td>Specifies the logical palette.</td>
</tr>
</tbody>
</table>

CreatePatternBrush record (prior to 3.0)

The **CreatePatternBrush** record stored by Windows versions prior to 3.0 contains a device-dependent bitmap which may not be suitable for playback on all devices. The following is the format of this record:
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x01F9.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>bmWidth</td>
<td>Bitmap width.</td>
</tr>
<tr>
<td>bmHeight</td>
<td>Bitmap height.</td>
</tr>
<tr>
<td>bmWidthBytes</td>
<td>Bytes per raster line.</td>
</tr>
<tr>
<td>bmPlanes</td>
<td>Number of color planes.</td>
</tr>
<tr>
<td>bmBitsPixel</td>
<td>Number of adjacent color bits that define a pixel.</td>
</tr>
<tr>
<td>bmBits</td>
<td>Pointer to bit values.</td>
</tr>
<tr>
<td>bits</td>
<td>Actual bits of pattern.</td>
</tr>
</tbody>
</table>

CreatePatternBrush record 3.0

The **CreatePatternBrush** record stored by Windows versions 3.0 and later contains a device-independent bitmap suitable for playback on all devices. The following is the format of this record:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0142.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>type</td>
<td>Bitmap type. This field may be either of these two values:</td>
</tr>
<tr>
<td></td>
<td>▶ BS_PATTERN—Brush is defined by a device-dependent bitmap through a call to the <strong>CreatePatternBrush</strong> function.</td>
</tr>
</tbody>
</table>
BS_DIBPATTERN—Brush is defined by a device-independent bitmap through a call to the CreateDIBPatternBrush function.

Specifies whether the bmiColors[] field of the BITMAPINFO data structure contains explicit RGB values or indexes into the currently realized logical palette. This field must be one of the following values:

- DIB_RGB_COLORS—The color table contains literal RGB values.
- DIB_PAL_COLORS—The color table consists of an array of indexes into the currently realized logical palette.

BITMAPINFO data structure.

Actual device-independent bitmap bits.

CreatePenIndirect record

The format and field descriptions of the CreatePenIndirect record follow:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    LOGPEN rdParm;
}
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x02FA.</td>
</tr>
<tr>
<td>rdParm</td>
<td>Specifies the logical pen.</td>
</tr>
</tbody>
</table>

Create region record

The format and field descriptions of the Create Region record follow:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x06FF.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Specifies the region to be created.</td>
</tr>
</tbody>
</table>
DeleteObject 3.0

The **DeleteObject** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm;
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x01F0.</td>
</tr>
<tr>
<td>rdParm</td>
<td>Specifies the handle-table index of the object to be deleted.</td>
</tr>
</tbody>
</table>

DrawText record

The **DrawText** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x062F.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>format</td>
<td>Method of formatting.</td>
</tr>
<tr>
<td>count</td>
<td>Number of bytes in the string.</td>
</tr>
<tr>
<td>rectangle</td>
<td>Rectangular structure defining area where text is to be defined.</td>
</tr>
<tr>
<td>string</td>
<td>Byte array containing the string. The array is ((count + 1) &gt;&gt;&gt; 1) words long.</td>
</tr>
</tbody>
</table>

Escape record

The format and field descriptions of the **Escape** record follow:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
```

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Method of formatting.</td>
</tr>
<tr>
<td>count</td>
<td>Number of bytes in the string.</td>
</tr>
<tr>
<td>rectangle</td>
<td>Rectangular structure defining area where text is to be defined.</td>
</tr>
<tr>
<td>string</td>
<td>Byte array containing the string. The array is ((count + 1) &gt;&gt;&gt; 1) words long.</td>
</tr>
</tbody>
</table>
WORD rdFunction;
WORD rdParm[];
}

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0626.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>escape number</td>
<td>Number identifying individual escape.</td>
</tr>
<tr>
<td>count</td>
<td>Number of bytes of information.</td>
</tr>
<tr>
<td>input data</td>
<td>Variable length field. The field is ((count+1) &gt;&gt;&gt;&gt; 1) words long.</td>
</tr>
</tbody>
</table>

ExtTextOut record

The ExtTextOut record has the following format:

```c
struct{
  DWORD rdSize;
  WORD rdFunction;
  WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0A32.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Logical y-value of string's starting point.</td>
</tr>
<tr>
<td>x</td>
<td>Logical x-value of string's starting point.</td>
</tr>
<tr>
<td>count</td>
<td>Length of the string.</td>
</tr>
<tr>
<td>options</td>
<td>Rectangle type.</td>
</tr>
<tr>
<td>rectangle</td>
<td>RECT structure defining the ExtTextOut rectangle if options element is nonzero; nonexistent if options element equals zero</td>
</tr>
<tr>
<td>string</td>
<td>Byte array containing the string. The array is ((count + 1) &gt;&gt;&gt;&gt; 1) words long.</td>
</tr>
<tr>
<td>dxarray</td>
<td>Optional word array of intercharacter distances.</td>
</tr>
</tbody>
</table>
Polygon record

The **Polygon** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0324.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>count</td>
<td>Number of points.</td>
</tr>
<tr>
<td>list of points</td>
<td>List of individual points.</td>
</tr>
</tbody>
</table>

PolyPolygon record

The **PolyPolygon** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0538.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>count</td>
<td>Total number of points.</td>
</tr>
<tr>
<td>list of polygon counts</td>
<td>List of number of points for each polygon.</td>
</tr>
<tr>
<td>list of points</td>
<td>List of individual points.</td>
</tr>
</tbody>
</table>
Polyline record

The **Polyline** record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0325.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>element</td>
<td>Description</td>
</tr>
<tr>
<td>count</td>
<td>Number of points.</td>
</tr>
<tr>
<td>list of points</td>
<td>List of individual points.</td>
</tr>
</tbody>
</table>

SelectClipRegion

The **SelectClipRegion** record has the following format:

```c
struct{
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm;
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x012C.</td>
</tr>
<tr>
<td>rdParm</td>
<td>Specifies the handle-table index of the region being selected.</td>
</tr>
</tbody>
</table>
This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x012D.</td>
</tr>
<tr>
<td>rdParm</td>
<td>Specifies the handle-table index of the object being selected.</td>
</tr>
</tbody>
</table>

**SelectPalette record 3.0**

The SelectPalette record has the following format:

```c
struct{
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm;
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0234.</td>
</tr>
<tr>
<td>rdParm</td>
<td>Specifies the handle-table index of the logical palette being selected.</td>
</tr>
</tbody>
</table>

**SetDIBitsToDevice record 3.0**

The SetDIBitsToDevice record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0D33.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>wUsage</td>
<td>Flag indicating whether the bitmap color table contains RGB values or indexes into the currently realized logical palette</td>
</tr>
<tr>
<td>numscans</td>
<td>Number of scan lines in the bitmap.</td>
</tr>
</tbody>
</table>
startscan  First scan line in the bitmap.
srcY     The y-coordinate of the origin of the source
in the bitmap.
srcX     The x-coordinate of the origin of the source
in the bitmap.
extY     Height of the source in the bitmap.
extX     Width of the source in the bitmap.
destY    The y-coordinate of the origin of the
         destination rectangle.
destX    The x-coordinate of the origin of the
         destination rectangle.
BitmapInfo BITMAPINFO data structure.
bits     Actual device-independent bitmap bits.

SetPaletteEntries record 3.0

The SetPaletteEntries record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0037.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>start</td>
<td>First entry to be set in the palette.</td>
</tr>
<tr>
<td>numentries</td>
<td>Number of entries to be set in the palette.</td>
</tr>
<tr>
<td>entries</td>
<td>PALETTEENTRY blocks.</td>
</tr>
</tbody>
</table>

StretchBit record (prior to 3.0)

The StretchBit record stored by Windows versions prior to 3.0 contains a
device-dependent bitmap which may not be suitable for playback on all
devices. The following is the format of this record:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:
The StretchBit record stored by Windows versions 3.0 and later contains a device-independent bitmap suitable for playback on all devices. The following is the format of this record:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0B41.</td>
</tr>
<tr>
<td>rdParm[ ]</td>
<td>Contains the following elements:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>raster op</td>
<td>Low word of the raster operation.</td>
</tr>
<tr>
<td>raster op</td>
<td>High word of the raster operation.</td>
</tr>
<tr>
<td>SYE</td>
<td>The y-extent of the source.</td>
</tr>
<tr>
<td>SXE</td>
<td>The x-extent of the source.</td>
</tr>
<tr>
<td>SY</td>
<td>The y-coordinate of the source origin.</td>
</tr>
<tr>
<td>SX</td>
<td>The x-coordinate of the source origin.</td>
</tr>
<tr>
<td>DYE</td>
<td>The y-extent of the destination.</td>
</tr>
<tr>
<td>DXE</td>
<td>The x-extent of the destination.</td>
</tr>
<tr>
<td>DY</td>
<td>The y-coordinate of destination origin.</td>
</tr>
<tr>
<td>DX</td>
<td>The x-coordinate of destination origin.</td>
</tr>
<tr>
<td>bmWidth</td>
<td>Width of the bitmap in pixels.</td>
</tr>
<tr>
<td>bmHeight</td>
<td>Height of the bitmap in raster lines.</td>
</tr>
<tr>
<td>bmWidthBytes</td>
<td>Number of bytes in each raster line.</td>
</tr>
<tr>
<td>bmPlanes</td>
<td>Number of color planes in the bitmap.</td>
</tr>
<tr>
<td>bmBitsPixel</td>
<td>Number of adjacent color bits.</td>
</tr>
<tr>
<td>bits</td>
<td>Actual bitmap bits.</td>
</tr>
</tbody>
</table>

Software development kit
DYE The y-extent of the destination.
DXE The x-extent of the destination.
DY The y-coordinate of destination origin.
DX The x-coordinate of destination origin.

BitmapInfo BITMAPINFO data structure.
bits Actual device-independent bitmap bits.

StretchDIBits record 3.0

The StretchDIBits record has the following format:

```
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0F43.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td></td>
<td><strong>Element</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>dwRop</td>
<td>Raster operation to be performed.</td>
</tr>
<tr>
<td>wUsage</td>
<td>Flag indicating whether the bitmap color table contains RGB values or indexes into the currently realized logical palette.</td>
</tr>
<tr>
<td>srcYExt</td>
<td>Height of the source in the bitmap.</td>
</tr>
<tr>
<td>srcXExt</td>
<td>Width of the source in the bitmap.</td>
</tr>
<tr>
<td>srcY</td>
<td>The y-coordinate of the origin of the source in the bitmap.</td>
</tr>
<tr>
<td>srcX</td>
<td>The x-coordinate of the origin of the source in the bitmap.</td>
</tr>
<tr>
<td>dstYExt</td>
<td>Height of the destination rectangle.</td>
</tr>
<tr>
<td>dstXExt</td>
<td>Width of the destination rectangle.</td>
</tr>
<tr>
<td>dstY</td>
<td>The y-coordinate of the origin of the destination rectangle.</td>
</tr>
<tr>
<td>dstX</td>
<td>The x-coordinate of the origin of the destination rectangle.</td>
</tr>
<tr>
<td>BitmapInfo</td>
<td>BITMAPINFO data structure.</td>
</tr>
<tr>
<td>bits</td>
<td>Actual device-independent bitmap bits.</td>
</tr>
</tbody>
</table>
TextOut record

The TextOut record has the following format:

```c
struct {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm[];
}
```

This record contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdSize</td>
<td>Specifies the record size in words.</td>
</tr>
<tr>
<td>rdFunction</td>
<td>Specifies the function number 0x0521.</td>
</tr>
<tr>
<td>rdParm[]</td>
<td>Contains the following elements:</td>
</tr>
<tr>
<td>count</td>
<td>The string's length.</td>
</tr>
<tr>
<td>string</td>
<td>The actual string.</td>
</tr>
<tr>
<td>y-value</td>
<td>Logical y-coordinate of string's starting point.</td>
</tr>
<tr>
<td>x-value</td>
<td>Logical x-coordinate of string's starting point.</td>
</tr>
</tbody>
</table>

Sample metafile program output

This section shows the metafile created by a sample program.

The following sample program creates a small metafile in which a purple rectangle with a green border is drawn, and the words "Hello People" are written in the rectangle.

```c
MakeAMetaFile(hDC)
HDC hDC;
{
    HPEN hMetaGreenPen;
    HBRUSH hMetaVioletBrush;
    HDC hDCMeta;
    HANDLE hMeta;

    /* create the metafile with output going to the disk */
    hDCMeta = CreateMetaFile((LPSTR) "sample.met");
    hMetaGreenPen = CreatePen(0, 0, (DWORD) 0x0000FF00);
    SelectObject(hDCMeta, hMetaGreenPen);
    hMetaVioletBrush = CreateSolidBrush((DWORD) 0x00FF00FF);
    SelectObject(hDCMeta, hMetaVioletBrush);
    Rectangle(hDCMeta, 0, 0, 150, 70);
```
TextOut(hDCMeta, 10, 10, (LPSTR) "Hello People", 12);

/* we are done with the metafile */
hMeta = CloseMetaFile(hDCMeta);

/* play the metafile that we just created */
PlayMetaFile(hDC, hMeta);
}

The resulting binary file SAMPLE.MET looks like this:

```
0001      mtType... disk metafile
0009      mtSize...
0100      mtVersion
0000 0036  mtSize
0002      mtNoObjects
0000 000C  mtMaxrecord
0000      mtNoParameters
0000 0008  rdSize
02FA      rdFunction (CreatePen function call)
0000 0000 0000 0000 FFOO rdParm (LOGPEN structure defining pen)
0000 0004  rdSize
012D      rdFunction (SelectObject)
0000      rdParm (index to object #0... the above pen)
0000 0007  rdSize
02FC      rdFunction (CreateBrush)
0000 00FF 00FF 0000 rdParm (LOGBRUSH structure defining the brush)
0000 0004  rdSize
012D      rdFunction (SelectObject)
0001      rdParm (index to object #1... the brush)
0000 0007  rdSize
041B      rdFunction (Rectangle)
0046 0096 0000 0000 rdParm (parameters sent to Rectangle...in reverse order)
0000 000C  rdSize
0521      rdFunction (TextOut)
rdParm
000C      count
string
48 65 6C 6C 6F 20 50 65 6F 6C 65 "Hello People"
000A      y-value
000A      x-value
```
Windows files store information required to create Windows applications as well as data needed by the Windows system and Windows applications during execution. For more information on topics related to Windows files, see the following:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metafile functions</td>
<td>Reference, Volume 1: Chapter 1, &quot;Window manager interface functions,&quot; and Chapter 4, &quot;Functions directory&quot;</td>
</tr>
</tbody>
</table>
This chapter describes the statements contained in the module-definition file that defines the application's contents and system requirements for the LINK program. LINK links compiled source files with Microsoft Windows and other libraries to create an executable Windows application. For information on running LINK, see Tools.

The module-definition file contains one or more of the following module statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>Code-segment attributes</td>
</tr>
<tr>
<td>DATA</td>
<td>Data-segment attributes</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>One-line description of the module</td>
</tr>
<tr>
<td>EXETYPE</td>
<td>.EXE header type (Windows or OS/2)</td>
</tr>
<tr>
<td>EXPORTS</td>
<td>Exported functions</td>
</tr>
<tr>
<td>HEAPSIZE</td>
<td>Size of local heap in bytes</td>
</tr>
<tr>
<td>IMPORTS</td>
<td>Imported functions</td>
</tr>
<tr>
<td>LIBRARY</td>
<td>Dynamic-link library name</td>
</tr>
<tr>
<td>NAME</td>
<td>Module name</td>
</tr>
<tr>
<td>SEGMENTS</td>
<td>Additional code segment</td>
</tr>
<tr>
<td>STACKSIZE</td>
<td>Size of local stack in bytes</td>
</tr>
<tr>
<td>STUB</td>
<td>Old-style executable</td>
</tr>
</tbody>
</table>

This chapter describes these statements, their syntax, required and optional parameters, and usage.
CODE

Syntax

CODE [[FIXED | MOVEABLE]] [[DISCARDABLE]] [[\PRELOAD | LOADONCALL]]

This statement defines the attributes of the standard code segment. The standard code segment is the application segment having the name _TEXT and belonging to the class CODE. In C applications, the standard data segment is created automatically if no specific segment name is included in the C-Compiler command line.

The **FIXED** option, if included, means that the segment remains at a fixed memory location; the **MOVEABLE** option means that the segment can be moved, if necessary, in order to compact memory.

The **DISCARDABLE** option, if included, means that the segment can be discarded if it is no longer needed.

The **PRELOAD** option, if included, means that the segment is loaded when the module is first loaded; the **LOADONCALL** option means that the segment is loaded when it is called. The Resource Compiler may override this option. See **Tools** for more information.

Comments

There are no default attributes for code segments. The .DEF file should always explicitly define code-segment attributes.

If conflicting options are included in the same statement, **LINK** uses the overriding option to determine the segment attributes. The following list shows which options override which:

- MOVEABLE overrides **FIXED**.
- PRELOAD overrides **LOADONCALL**.

Example

CODE MOVEABLE LOADONCALL

In this example, the loader forces all fixed and moveable (but not discardable) code segments to be loaded. Libraries cannot have code that is moveable but not discardable.

DATA

Syntax

Data [[NONE | SINGLE | MULTIPLE]] [[FIXED | MOVEABLE]]

This statement defines the attributes of the standard data segment. The standard data segment is all application segments belonging to the group
DGROUP and the class DATA. In C applications, the standard data segment is created automatically. The data is always preloaded.

The **NONE** option, if included, means that there is no data segment. To be effective, this option should be the only attribute of the segment. This option is available only for libraries.

The **SINGLE** option, if included, means that a single segment is shared by all instances of the module, and is valid only for libraries. The **MULTIPLE** option means that one segment exists for each instance, and is only valid for applications.

**NONE, SINGLE, and MULTIPLE** are mutually exclusive.

The **FIXED** option, if included, means that the segment remains at a fixed memory location. The **MOVEABLE** option means that the segment can be moved if necessary, in order to compact memory.

**Comments**

There are no default attributes for data segments. The .DEF file should always explicitly define data-segment attributes.

Data segments are always preloaded.

If conflicting options are included in the same statement, **LINK** uses the overriding option to determine the segment attributes. The following list shows which options override which:

- **MULTIPLE** overrides **NONE**.
- **SINGLE** overrides **NONE**.
- **MOVEABLE** overrides **FIXED**.

**Example**

```
DATA MOVEABLE SINGLE
```

This example tells **LINK** that this module has a single, moveable data segment.

**DESCRIPTION**

**Syntax**

```
DESCRIPTION ‘text’
```

This statement inserts text into the application’s module. It is useful for embedding source-control or copyright information.

**Parameters**

- `text` Specifies one or more ASCII characters. The string must be enclosed in single quotation marks.
Example  

**DESCRIPTION**

This example embeds the text “Microsoft Windows Template Application” in the application module.

**EXETYPE**

**Syntax**

```plaintext
EXETYPE headertype
```

This statement specifies the default executable-file (.EXE) header type (Windows or OS/2). It is required for every Windows application.

**Parameters**

- `headertype` Determines the header type. When linking an application intended for the Windows environment, you must set this parameter to the value "WINDOWS". For an MS OS/2 application, set this parameter to the value "OS/2".

**Example**

```plaintext
EXETYPE WINDOWS
```

**EXPORTS**

**Syntax**

```plaintext
EXPORTS exportname [[ordinal-option]] [[\res-option]] [[data-option]] [[parameter-option]]
```

This statement defines the names and attributes of the functions to be exported to other applications. The **EXPORTS** key word marks the beginning of the definitions. It can be followed by any number of export definitions, each on a separate line.

**Parameters**

- `exportname` Specifies one or more ASCII characters that define the function name. It has the following form:
  ```plaintext
  <entryname>=[internalname]
  ```

  where the `entryname` parameter specifies the name to be used by other applications to access the exported function, and `internalname` is an optional parameter that defines the actual name of the function if `entryname` is not the actual name.

- `ordinal-option` Defines the function's ordinal value. It has the following form:
  ```plaintext
  @ordinal
  ```
where *ordinal* takes an integer value that specifies the function’s ordinal value. The ordinal value defines the location of the function’s name in the application’s string table. (When exporting functions from libraries, it is better to use an ordinal rather than a name; using ordinals conserves space.)

*res-option* Is the optional key word RESIDENTNAME, which specifies that the function’s name must be resident at all times.

*data-option* Is the optional key word NODATA, which specifies that the function is not bound to a specific data segment. When invoked, the function uses the current data segment.

*parameter-option* Is an optional integer value that specifies the number of words the function expects to be passed as parameters.

**Example**

```plaintext
EXPORTS

SampleRead=read2bin @l 8
StringIn=str1 @2 4
CharTest NODATA
```

This example exports the functions SampleRead, StringIn and CharTest so that other applications, or Windows itself, can call them.

**HEAPSIZE**

**Syntax**

HEAPSIZE bytes

This statement defines the number of bytes needed by the application for its local heap. An application uses the local heap whenever it allocates local memory.

The default heap size is zero. The minimum size is 256 bytes. For an application, the size of the local heap must be at least large enough to hold the current environment.

**Parameters**

*bytes* Is an integer value that specifies the heap *size* in bytes. It must not exceed 65,536 (the size of a single physical segment).

**Example**

HEAPSIZE 4096

This example sets the size of the application’s local heap to 4096 bytes.
IMPORTS

Syntax

IMPORTS [[internal-option]] modulename [[entry-option]]

This statement defines the names and attributes of the functions to be imported from dynamic-link libraries. The IMPORTS key word marks the beginning of the definitions. It can be followed by any number of import definitions, each on a separate line.

Parameters

internal-option

Specifies the name that the application will use to call the function. It has the following form:

internal-name=

where internal-name is one or more ASCII characters. This name must be unique.

modulename

Specifies one or more uppercase ASCII characters that define the name of the executable module that contains the function. The module name must match the name of the executable file. For example, an application with the executable file SAMPLE.DLL has the module name "SAMPLE". The executable file must be named with the .DLL extension.

entry-option

Specifies the function to be imported. It can be one of the following:

.entryname

.entryordinal

where entryname is the actual name of the function, and entryordinal is the ordinal value of the function.

Example

IMPORTS
Sample.SampleRead
write2hex=Sample.SampleWrite
Read.1

Instead of listing imported DLL functions in the IMPORTS statement, you can specify an "import library" for the DLL in your application's LINK command line. It also saves space to import by ordinal.
LIBRARY

Syntax
LIBRARY libraryname

This statement defines the name of a library module. Library modules are resource modules that contain code, data, and other resources but are not intended to be executed as an independent program. Like an application's module name, a library's module name must match the name of the executable file. For example, the library USER.EXE has the module name "USER".

Parameters
libraryname Specifies one or more ASCII characters that define the name of the library module.

Comments
The start address of the library module is determined by the library's object files; it is an internally defined function.

The libraryname parameter is optional. If the parameter is not included, LINK uses the filename part of the executable file (that is, the name with the extension removed).

If the .DEF file includes neither a NAME nor a LIBRARY statement, LINK assumes a NAME statement without a modulename parameter is desired.

Example
LIBRARY Utilities

This example gives a library the module name "Utilities."

NAME

Syntax
NAME modulename

This statement defines the name of the application's executable module. The module name identifies the module when exporting functions.

Parameters
modulename Specifies one or more uppercase ASCII characters that define the name of the executable module. The module name must match the name of the executable file. For example, an application with the executable file SAMPLE.EXE has the module name "SAMPLE". Do not use OS/2 system library names. Examples of these names are DOSCALLS, VIOCALLS, and MOUCALLS.

Comments
The modulename parameter is optional. If the parameter is not included, LINK assumes that the module name matches the the filename of the executable file. For example, if you do not specify a module name and the
NAME

executable file is named MYAPP.EXE, LINK assumes that the module name is "MYAPP".

If the .DEF file includes neither a NAME nor a LIBRARY statement, LINK assumes a NAME statement without a modulename parameter is desired.

Example

NAME Calendar

This example gives an application the module name "Calendar".

SEGMENTS

Syntax

SEGMENTS segmentname [[CLASS 'class-name']] [[minalloc]] \\
[[FIXED | MOVEABLE]]
[[DISCARDABLE]] [[SHARED | NONSHARED]] [[PRELOAD | LOADONCALL]]

This statement defines the segment attributes of additional code and data segments.

The FIXED option, if included, means that the segment remains at a fixed memory location. The MOVEABLE option means that the segment can be moved if necessary, in order to compact memory.

The DISCARDABLE option, if included, means that the segment can be discarded if it is no longer needed.

The PRELOAD option, if included, means that the segment is loaded immediately. The LOADONCALL option means that the segment is loaded when it is accessed or called. The Resource Compiler may override this option. See Tools for more information.

Parameters

segmentname Specifies a character string that names the new segment. It can be any name, including the standard segment names _TEXT and _DATA, which represent the standard code and data segments.

class-name Is an optional key word that specifies the class name of the specified segment. If no class name is specified, LINK uses the class name CODE by default.

minalloc Is an optional integer value that specifies the minimum allocation size for the segment.

Comments

There are no default attributes for additional segments. The .DEF file should always explicitly define the attributes of additional segments.
If conflicting options are included in the same statement, **LINK** uses the overriding option to determine the segment attributes. The following list shows which options override which:

- **MOVEABLE** overrides **FIXED**.
- **PRELOAD** overrides **LOADONCALL**.

**Example**

```plaintext
SEGMENTS
  _TEXT FIXED
  _INIT PRELOAD DISCARDABLE
  _RES CLASS 'DATA' PRELOAD DISCARDABLE
```

### STACKSIZE

**Syntax**

STACKSIZE bytes

This statement defines the number of bytes needed by the application for its local stack. An application uses the local stack whenever it makes function calls.

The default stack size is zero if the application makes no function calls. If your application does make function calls and you specify a stack size smaller than 5K bytes, Windows automatically sets the stack size to 5K bytes.

**Parameters**

- **bytes** Is an integer value that specifies the stack size in bytes.

**Comments**

Do not use the **STACKSIZE** statement for dynamic-link libraries.

**Example**

STACKSIZE 6144

This example sets the size of an application's stack to 6144 bytes.

### STUB

**Syntax**

STUB "filename"

This statement appends the old-style executable file specified by filename to the beginning of the module. The executable stub should display a warning message and terminate if the user attempts to execute the module without having loaded Windows. The default file WINSTUB.EXE can be used if no other actions are required.
Parameters  filename  Specifies the name of the old-style executable file that will be appended to the module. The name must have the DOS filename format.

Comments  If the file named by filename is not in the current directory, LINK searches for the file in the directories specified by the user’s PATH environment variable.

Example  STUB ‘WINSTUB.EXE’

This example specifies the executable file WINSTUB.EXE as the application’s stub. If a user tries to run this application in the DOS environment, rather than with Windows, the program WINSTUB.EXE starts instead.
Binary and ternary raster-operation codes

This chapter lists and describes the binary and ternary raster operations used by the graphics device interface (GDI). A binary raster operation uses two operands: a pen and a destination bitmap. A ternary raster operation uses three operands: a source bitmap, a brush, and a destination bitmap. Both binary and ternary raster operations use Boolean operators.

Binary raster operations

This section lists the binary raster-operation codes used by the `GetROP2` and `SetROP2` functions. Raster-operation codes define how GDI combines the bits from the selected pen with the bits in the destination bitmap.

Each raster-operation code represents a Boolean operation in which the selected pen and the destination bitmap are combined. There are two operands used in these operations:

- D Destination bitmap
- P Selected pen

The Boolean operators used in these operations are as follows:

- a Bitwise AND
- n Bitwise NOT (inverse)
- o Bitwise OR
- x Bitwise Exclusive OR (XOR)
All Boolean operations are presented in reverse Polish notation. For example, the following operation replaces the destination with a combination of the pen and the selected brush:

\[ \text{DPo} \]

Each raster-operation code is a 32-bit integer value whose high-order word is a Boolean operation index and whose low-order word is the operation code. The 16-bit operation index is a zero-extended 8-bit value that represents the result of the Boolean operation on predefined pen and destination values. For example, the operation indexes for the DPo and DPan operations are shown in Table 11.1:

<table>
<thead>
<tr>
<th>P</th>
<th>D</th>
<th>PSo</th>
<th>DPSoo</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The following list outlines the drawing modes and the Boolean operations that they represent:

<table>
<thead>
<tr>
<th>Raster operation</th>
<th>Boolean operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2_BLACK</td>
<td>0</td>
</tr>
<tr>
<td>R2_COPYPEN</td>
<td>P</td>
</tr>
<tr>
<td>R2_MASKNOTPEN</td>
<td>DPna</td>
</tr>
<tr>
<td>R2_MASKPEN</td>
<td>DPa</td>
</tr>
<tr>
<td>R2_MASKPENNOT</td>
<td>PDna</td>
</tr>
<tr>
<td>R2_MERGEINOTPEN</td>
<td>DPno</td>
</tr>
<tr>
<td>R2_MERGEPEN</td>
<td>DPo</td>
</tr>
<tr>
<td>R2_MERGEPENNOT</td>
<td>PDno</td>
</tr>
<tr>
<td>R2_NOP</td>
<td>D</td>
</tr>
<tr>
<td>R2_NOT</td>
<td>Dn</td>
</tr>
<tr>
<td>R2_NOTCOPYPEN</td>
<td>Pn</td>
</tr>
<tr>
<td>R2_NOTMASKPEN</td>
<td>DPan</td>
</tr>
<tr>
<td>R2_NOTMERGEPEN</td>
<td>DPon</td>
</tr>
<tr>
<td>R2_NOTXORPEN</td>
<td>DPxn</td>
</tr>
<tr>
<td>R2_WHITE</td>
<td>1</td>
</tr>
<tr>
<td>R2_XORPEN</td>
<td>DPx</td>
</tr>
</tbody>
</table>

When a monochrome device is used, GDI maps the value 0 to black and the value 1 to white. Given an application that attempts to draw with a black pen on a white destination by using the available binary raster operations, the following results will occur:
When a color device is used, GDI uses RGB values to represent the colors of the pen and the destination. An RGB color value is a long integer that contains a red, a green, and a blue color field, each specifying the intensity of the given color. Intensities range from 0 to 255. The values are packed in the three low-order bytes of the long integer. The color of a pen is always a solid color, but the color of the destination may be a mixture of any two or three colors. Given an application that attempts to draw with a white pen on a blue destination by using the available binary raster operations, the following results will occur:

<table>
<thead>
<tr>
<th>Raster Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2_BLACK</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_COPYPEN</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_MASKNOTPEN</td>
<td>No visible line</td>
</tr>
<tr>
<td>R2_MASKPEN</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_MASKPENNOT</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_MERGENOTPEN</td>
<td>No visible line</td>
</tr>
<tr>
<td>R2_MERGEPEN</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_MERGEPENNOT</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_NOP</td>
<td>No visible line</td>
</tr>
<tr>
<td>R2_NOT</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_NOTCOPYPEN</td>
<td>No visible line</td>
</tr>
<tr>
<td>R2_NOTMASKPEN</td>
<td>No visible line</td>
</tr>
<tr>
<td>R2_NOTMERGEPEN</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_NOTXORPEN</td>
<td>Visible black line</td>
</tr>
<tr>
<td>R2_WHITE</td>
<td>No visible line</td>
</tr>
<tr>
<td>R2_XORPEN</td>
<td>No visible line</td>
</tr>
</tbody>
</table>
Ternary raster operations

This section lists the ternary raster-operation codes used by the BitBlt, PatBlt, and StretchBlt functions. Ternary raster-operation codes define how GDI combines the bits in a source bitmap with the bits in the destination bitmap.

Each raster-operation code represents a Boolean operation in which the source, the selected brush, and the destination bitmap are combined. There are three operands used in these operations:

- D Destination bitmap
- P Selected brush (also called pattern)
- S Source bitmap

The Boolean operators used in these operations are as follows:

- a Bitwise AND
- n Bitwise NOT (inverse)
- o Bitwise OR
- x Bitwise Exclusive OR (XOR)

All Boolean operations are presented in reverse Polish notation. For example, the following operation replaces the destination with a combination of the source and brush:

\[
\text{PSo}
\]

The following operation combines the source and brush with the destination (there are alternate spellings of the same function, so although a particular spelling may not be in the list, an equivalent form will be):

\[
\text{DPSoo}
\]

Each raster-operation code is a 32-bit integer value whose high-order word is a Boolean operation index and whose low-order word is the operation code. The 16-bit operation index is a zero-extended, 8-bit value that represents the result of the Boolean operation on predefined brush, source, and destination values. For example, the operation indexes for the PSo and DPSoo operations are shown in Table 11.2:

<table>
<thead>
<tr>
<th>P</th>
<th>S</th>
<th>D</th>
<th>PSo</th>
<th>DPSoo</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
For more information about RGB values, see the RGB structure in Chapter 7, "Data types and structures."

Table 11.2: Operation Indexes for PSo and DPSoo (continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Operation index:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00FC</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>00FE</td>
<td></td>
</tr>
</tbody>
</table>

In this case, PSo has the operation index 00FC (read from the bottom up); DPSoo has the operation index 00FE. These values define the location of the corresponding raster-operation codes, as shown in Table 11.1, "Operation indexes for DPs and DPa." The PSo operation is in line 252 (FCh) of the table; DPSoo is in line 254 (FEh).

The most commonly used raster operations have been given special names in the Windows include file, windows.h. You should use these names whenever possible in your applications.

When the source and destination are monochrome, a bit value of zero represents a black pixel and a bit value of 1 represents a white pixel. When the source and the destination are color, those colors are represented with RGB values.

Table 11.3 lists the raster-operation codes:

<table>
<thead>
<tr>
<th>Boolean Function in Hex, Hex</th>
<th>Hex ROP</th>
<th>Boolean Function in Reverse Polish</th>
<th>Common Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00000042</td>
<td>0</td>
<td>BLACKNESS</td>
</tr>
<tr>
<td>01</td>
<td>00010289</td>
<td>DPSoo</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>00020C89</td>
<td>DPSona</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>000300AA</td>
<td>Pson</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>00040C88</td>
<td>SDPona</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>000500A9</td>
<td>DPon</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>00060865</td>
<td>PDSxnon</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>000702C5</td>
<td>PDSaon</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>00080F08</td>
<td>SDPnaa</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>00090245</td>
<td>PDSxnon</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>00A0329</td>
<td>DPna</td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>00B0B2A</td>
<td>PSDnaon</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>00C0324</td>
<td>SPna</td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>00D0B25</td>
<td>PDSnaon</td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>00E08A5</td>
<td>PDSonon</td>
<td></td>
</tr>
<tr>
<td>0F</td>
<td>00F0001</td>
<td>Pn</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00100C85</td>
<td>PDSona</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>001100A6</td>
<td>DSon</td>
<td>NOTSRCERASE</td>
</tr>
<tr>
<td>12</td>
<td>00120868</td>
<td>SDPxnnon</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>001302C8</td>
<td>SDPxoan</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>00140869</td>
<td>DPSxnon</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>001502C9</td>
<td>DPSaon</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Value</td>
<td>Description</td>
<td>Status</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>16</td>
<td>00165CCA</td>
<td>PSDPSanaxx</td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>00171D54</td>
<td>SSPxDSxaaxn</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>00180D59</td>
<td>SPDPSaox</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>00191CC8</td>
<td>SDPSanaxn</td>
<td>-</td>
</tr>
<tr>
<td>1A</td>
<td>001A06C5</td>
<td>PDSaox</td>
<td>-</td>
</tr>
<tr>
<td>1B</td>
<td>001B0768</td>
<td>SDPSxaaxn</td>
<td>-</td>
</tr>
<tr>
<td>1C</td>
<td>001C06CA</td>
<td>PSDPaox</td>
<td>-</td>
</tr>
<tr>
<td>1D</td>
<td>001D0766</td>
<td>DSDPSanaxn</td>
<td>-</td>
</tr>
<tr>
<td>1E</td>
<td>001E01A5</td>
<td>PDSox</td>
<td>-</td>
</tr>
<tr>
<td>1F</td>
<td>001F0385</td>
<td>PDSoan</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>00200F09</td>
<td>DDPNaan</td>
<td>-</td>
</tr>
<tr>
<td>21</td>
<td>00210248</td>
<td>SDPox</td>
<td>-</td>
</tr>
<tr>
<td>22</td>
<td>00220326</td>
<td>DSNaox</td>
<td>-</td>
</tr>
<tr>
<td>23</td>
<td>00230B24</td>
<td>SPDSnaan</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td>00240D55</td>
<td>DSDSaox</td>
<td>-</td>
</tr>
<tr>
<td>25</td>
<td>00251CC5</td>
<td>PSDPsnaxna</td>
<td>-</td>
</tr>
<tr>
<td>26</td>
<td>002606C8</td>
<td>SPDSaax</td>
<td>-</td>
</tr>
<tr>
<td>27</td>
<td>00271868</td>
<td>SPDSoax</td>
<td>-</td>
</tr>
<tr>
<td>28</td>
<td>00280369</td>
<td>DPDsox</td>
<td>-</td>
</tr>
<tr>
<td>29</td>
<td>002916CA</td>
<td>PSDPSaaxna</td>
<td>-</td>
</tr>
<tr>
<td>2A</td>
<td>002A0CC9</td>
<td>DPDsox</td>
<td>-</td>
</tr>
<tr>
<td>2B</td>
<td>002B1D58</td>
<td>SPDPDxaaxn</td>
<td>-</td>
</tr>
<tr>
<td>2C</td>
<td>002C0784</td>
<td>SPDSaox</td>
<td>-</td>
</tr>
<tr>
<td>2D</td>
<td>002D060A</td>
<td>PSDPsox</td>
<td>-</td>
</tr>
<tr>
<td>2E</td>
<td>002E064A</td>
<td>PSDPsox</td>
<td>-</td>
</tr>
<tr>
<td>2F</td>
<td>002F0E2A</td>
<td>PSDPsoax</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td>0030032A</td>
<td>PDSaax</td>
<td>-</td>
</tr>
<tr>
<td>31</td>
<td>00310B28</td>
<td>SPDSoan</td>
<td>-</td>
</tr>
<tr>
<td>32</td>
<td>00320688</td>
<td>SDPSaox</td>
<td>-</td>
</tr>
<tr>
<td>33</td>
<td>00330008</td>
<td>Sn</td>
<td>NOTSRCOPY</td>
</tr>
<tr>
<td>34</td>
<td>003406C4</td>
<td>SPDSaax</td>
<td>-</td>
</tr>
<tr>
<td>35</td>
<td>00351864</td>
<td>SPDSoax</td>
<td>-</td>
</tr>
<tr>
<td>36</td>
<td>003601A8</td>
<td>SPDSxaox</td>
<td>-</td>
</tr>
<tr>
<td>37</td>
<td>00370388</td>
<td>SDPox</td>
<td>-</td>
</tr>
<tr>
<td>38</td>
<td>0038078A</td>
<td>SPDSoan</td>
<td>-</td>
</tr>
<tr>
<td>39</td>
<td>00390604</td>
<td>SPDPsaax</td>
<td>-</td>
</tr>
<tr>
<td>3A</td>
<td>003A0644</td>
<td>SPDPsaax</td>
<td>-</td>
</tr>
<tr>
<td>3B</td>
<td>003B0E24</td>
<td>SPDPsaan</td>
<td>-</td>
</tr>
<tr>
<td>3C</td>
<td>003C004A</td>
<td>PSn</td>
<td>-</td>
</tr>
<tr>
<td>3D</td>
<td>003D18A4</td>
<td>SPDPsonox</td>
<td>-</td>
</tr>
<tr>
<td>3E</td>
<td>003E1B24</td>
<td>SPDSnaxo</td>
<td>-</td>
</tr>
<tr>
<td>3F</td>
<td>003F00EA</td>
<td>PSSan</td>
<td>-</td>
</tr>
<tr>
<td>40</td>
<td>00400F0A</td>
<td>PSDPsaax</td>
<td>-</td>
</tr>
<tr>
<td>41</td>
<td>00410249</td>
<td>DPDSoan</td>
<td>-</td>
</tr>
<tr>
<td>42</td>
<td>00420D5D</td>
<td>SDPDsxaox</td>
<td>-</td>
</tr>
<tr>
<td>43</td>
<td>00431CC4</td>
<td>SPDPSanaxn</td>
<td>-</td>
</tr>
<tr>
<td>44</td>
<td>00440328</td>
<td>SDPsox</td>
<td>SRCERASE</td>
</tr>
<tr>
<td>45</td>
<td>00450B29</td>
<td>SPDSoax</td>
<td>-</td>
</tr>
<tr>
<td>46</td>
<td>004606C6</td>
<td>SPDPDsaax</td>
<td>-</td>
</tr>
<tr>
<td>47</td>
<td>0047076A</td>
<td>PSDPxaaxn</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td>00480368</td>
<td>SDPxa</td>
<td>-</td>
</tr>
<tr>
<td>Code</td>
<td>Value</td>
<td>Operation</td>
<td>Notes</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>49</td>
<td>004916C5</td>
<td>PDSPDaoxxn</td>
<td>-</td>
</tr>
<tr>
<td>4A</td>
<td>004A0789</td>
<td>DPSDoax</td>
<td>-</td>
</tr>
<tr>
<td>4B</td>
<td>004B0605</td>
<td>PDSnox</td>
<td>-</td>
</tr>
<tr>
<td>4C</td>
<td>004C0CC8</td>
<td>SPDpana</td>
<td>-</td>
</tr>
<tr>
<td>4D</td>
<td>004D1954</td>
<td>SSPxDSxoxn</td>
<td>-</td>
</tr>
<tr>
<td>4E</td>
<td>004E0645</td>
<td>PDSPxox</td>
<td>-</td>
</tr>
<tr>
<td>4F</td>
<td>004F0E25</td>
<td>PDSnoan</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>00500325</td>
<td>PDna</td>
<td>-</td>
</tr>
<tr>
<td>51</td>
<td>00510B26</td>
<td>DSPnaon</td>
<td>-</td>
</tr>
<tr>
<td>52</td>
<td>005206C9</td>
<td>DPSDax</td>
<td>-</td>
</tr>
<tr>
<td>53</td>
<td>00530764</td>
<td>SPDxaxn</td>
<td>-</td>
</tr>
<tr>
<td>54</td>
<td>005408A9</td>
<td>DPSnoan</td>
<td>-</td>
</tr>
<tr>
<td>55</td>
<td>00550009</td>
<td>Dn</td>
<td>DSTINVERT</td>
</tr>
<tr>
<td>56</td>
<td>005601A9</td>
<td>DPSox</td>
<td>-</td>
</tr>
<tr>
<td>57</td>
<td>00570389</td>
<td>DPSoan</td>
<td>-</td>
</tr>
<tr>
<td>58</td>
<td>00580785</td>
<td>PDSPoax</td>
<td>-</td>
</tr>
<tr>
<td>59</td>
<td>00590609</td>
<td>DPSnox</td>
<td>-</td>
</tr>
<tr>
<td>5A</td>
<td>005A0049</td>
<td>DPx</td>
<td>PATINVERT</td>
</tr>
<tr>
<td>5B</td>
<td>005B18A9</td>
<td>DPSDno ox</td>
<td>-</td>
</tr>
<tr>
<td>5C</td>
<td>005C0649</td>
<td>DPSDxox</td>
<td>-</td>
</tr>
<tr>
<td>5D</td>
<td>005D0E29</td>
<td>DPSnoan</td>
<td>-</td>
</tr>
<tr>
<td>5E</td>
<td>005E1B29</td>
<td>DPSDnoox</td>
<td>-</td>
</tr>
<tr>
<td>5F</td>
<td>005F00E9</td>
<td>DPan</td>
<td>-</td>
</tr>
<tr>
<td>60</td>
<td>00600365</td>
<td>PDSxa</td>
<td>-</td>
</tr>
<tr>
<td>61</td>
<td>006116C6</td>
<td>DSPDsaoxn</td>
<td>-</td>
</tr>
<tr>
<td>62</td>
<td>00620786</td>
<td>DSPDox</td>
<td>-</td>
</tr>
<tr>
<td>63</td>
<td>00630608</td>
<td>SPDnox</td>
<td>-</td>
</tr>
<tr>
<td>64</td>
<td>00640788</td>
<td>SPDsoax</td>
<td>-</td>
</tr>
<tr>
<td>65</td>
<td>00650606</td>
<td>DSPnox</td>
<td>-</td>
</tr>
<tr>
<td>66</td>
<td>00660046</td>
<td>DSx</td>
<td>SRCINVERT</td>
</tr>
<tr>
<td>67</td>
<td>006718A8</td>
<td>SPDso nox</td>
<td>-</td>
</tr>
<tr>
<td>68</td>
<td>006858A6</td>
<td>DSPDsooxn</td>
<td>-</td>
</tr>
<tr>
<td>69</td>
<td>00690145</td>
<td>PDSxxn</td>
<td>-</td>
</tr>
<tr>
<td>6A</td>
<td>006A01E9</td>
<td>DPSax</td>
<td>-</td>
</tr>
<tr>
<td>6B</td>
<td>006B178A</td>
<td>PSDPs oaxn</td>
<td>-</td>
</tr>
<tr>
<td>6C</td>
<td>006C01E8</td>
<td>SPDax</td>
<td>-</td>
</tr>
<tr>
<td>6D</td>
<td>006D1785</td>
<td>PDSPD oaxn</td>
<td>-</td>
</tr>
<tr>
<td>6E</td>
<td>006E1E28</td>
<td>SPD snoax</td>
<td>-</td>
</tr>
<tr>
<td>6F</td>
<td>006F0C65</td>
<td>PDSxnan</td>
<td>-</td>
</tr>
<tr>
<td>70</td>
<td>00700CC5</td>
<td>PD sana</td>
<td>-</td>
</tr>
<tr>
<td>71</td>
<td>00711D5C</td>
<td>SSDxPDxaxn</td>
<td>-</td>
</tr>
<tr>
<td>72</td>
<td>00720648</td>
<td>SPDsox</td>
<td>-</td>
</tr>
<tr>
<td>73</td>
<td>00730E28</td>
<td>SPDnoan</td>
<td>-</td>
</tr>
<tr>
<td>74</td>
<td>00740646</td>
<td>DSPDxox</td>
<td>-</td>
</tr>
<tr>
<td>75</td>
<td>00750E26</td>
<td>DSPnoan</td>
<td>-</td>
</tr>
<tr>
<td>76</td>
<td>00761B28</td>
<td>SPDsnoax</td>
<td>-</td>
</tr>
<tr>
<td>77</td>
<td>007700E6</td>
<td>DSan</td>
<td>-</td>
</tr>
<tr>
<td>78</td>
<td>007801E5</td>
<td>PD sax</td>
<td>-</td>
</tr>
<tr>
<td>79</td>
<td>00791786</td>
<td>DSPDso oax</td>
<td>-</td>
</tr>
<tr>
<td>7A</td>
<td>007A1E29</td>
<td>DPSDnoax</td>
<td>-</td>
</tr>
<tr>
<td>7B</td>
<td>007B0C68</td>
<td>SPDxnan</td>
<td>-</td>
</tr>
<tr>
<td>Raster-operation codes (continued)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7C 007C1E24 007C1E24 SPDSnoax -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7D 007D00C9 007D00C9 DPSxnan -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7E 007E0095 007E0095 SPxDSxo -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7F 007F03C9 007F03C9 DPSaan -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 008003E9 008003E9 DPSaa -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81 00810975 00810975 SPxDSxox -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82 00820C49 00820C49 DPSxna -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>83 00831E04 00831E04 SPDSnoaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84 00840C48 00840C48 SDPxna -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85 00851E05 00851E05 PSxPnoaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86 00861A6 00861A6 DSPSaoaxx -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87 008701C5 008701C5 PDSaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88 008800C6 008800C6 DSa SRCAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89 00891B08 00891B08 SDPSnaoxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8A 008A0E06 008A0E06 DSPnoa -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8B 008B0666 008B0666 DSPDxoxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8C 008C0E08 008C0E08 SDPxoa -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8D 008D00E8 008D00E8 SDPSxoxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8E 008E17C 008E17C SSDxDax -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8F 008F0E0E 008F0E0E PDSaan -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90 00900C45 00900C45 PDSxna -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 00910E08 00910E08 SDPSnoaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92 00921A6 00921A6 DSPDspoaxx -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93 009301C4 009301C4 SPDaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94 009401A4 009401A4 PSDPSaoaxx -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95 009501C9 009501C9 DPSaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96 00960169 00960169 DPSxx -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97 0097088A 0097088A PSDPSnoaxx -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98 00981188 00981188 SDPSnoaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99 00909066 00909066 DSx -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9A 009A0709 009A0709 DPsax -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9B 009B07A8 009B07A8 SDPSaoaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9C 009C0704 009C0704 SPDax -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9D 009D07A6 009D07A6 DSPDxoaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9E 009E1E6E 009E1E6E DSPSdoaxx -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9F 009F0345 009F0345 PSxan -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 00A000C9 00A000C9 DPa -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1 00A11B05 00A11B05 PSDPnoaaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2 00A20E09 00A20E09 DSPnoa -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3 00A30669 00A30669 DSPDxoxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4 00A41885 00A41885 PSDPonoxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5 00A50065 00A50065 PDxan -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6 00A60706 00A60706 DSPNax -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7 00A707A5 00A707A5 PSDPoaaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8 00A803A9 00A803A9 DSA -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9 00A90189 00A90189 DPSoxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9 00A9185 00A9185 DPPDoxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA 00AA0029 00AA0029 D -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB 00AB0889 00AB0889 DPPSono -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC 00AC0744 00AC0744 SPDSxax -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD 00AD06E9 00AD06E9 DSPDaxn -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AE 00AE0B06 00AE0B06 DSPano -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>00AF0229</td>
<td>DPno</td>
<td>-</td>
</tr>
<tr>
<td>B0</td>
<td>00B00E05</td>
<td>PDSnoa</td>
<td>-</td>
</tr>
<tr>
<td>B1</td>
<td>00B10665</td>
<td>PDSPxoxn</td>
<td>-</td>
</tr>
<tr>
<td>B2</td>
<td>00B21974</td>
<td>SSPxDSxox</td>
<td>-</td>
</tr>
<tr>
<td>B3</td>
<td>00B30CE8</td>
<td>SDPanan</td>
<td>-</td>
</tr>
<tr>
<td>B4</td>
<td>00B4070A</td>
<td>PSDnax</td>
<td>-</td>
</tr>
<tr>
<td>B5</td>
<td>00B507A9</td>
<td>DPSDoaxn</td>
<td>-</td>
</tr>
<tr>
<td>B6</td>
<td>00B616E9</td>
<td>DPSDPaoxx</td>
<td>-</td>
</tr>
<tr>
<td>B7</td>
<td>00B70348</td>
<td>SDPxan</td>
<td>-</td>
</tr>
<tr>
<td>B8</td>
<td>00B8074A</td>
<td>PSDPxax</td>
<td>-</td>
</tr>
<tr>
<td>B9</td>
<td>00B906E6</td>
<td>DPSDaoxn</td>
<td>-</td>
</tr>
<tr>
<td>BA</td>
<td>00BA0609</td>
<td>DPSnao</td>
<td>-</td>
</tr>
<tr>
<td>BB</td>
<td>00BB0226</td>
<td>DSno</td>
<td>MERGEPAINT</td>
</tr>
<tr>
<td>BC</td>
<td>00BC1CE4</td>
<td>SPDSanax</td>
<td>-</td>
</tr>
<tr>
<td>BD</td>
<td>00BD0D7D</td>
<td>SDxPDxan</td>
<td>-</td>
</tr>
<tr>
<td>BE</td>
<td>00BE0269</td>
<td>DPSxo</td>
<td>-</td>
</tr>
<tr>
<td>BF</td>
<td>00BF08C9</td>
<td>DPSano</td>
<td>-</td>
</tr>
<tr>
<td>C0</td>
<td>00C000CA</td>
<td>PsA</td>
<td>MERGECOPY</td>
</tr>
<tr>
<td>C1</td>
<td>00C11B04</td>
<td>SPDSnaoxn</td>
<td>-</td>
</tr>
<tr>
<td>C2</td>
<td>00C21884</td>
<td>SPDSnooxn</td>
<td>-</td>
</tr>
<tr>
<td>C3</td>
<td>00C3006A</td>
<td>PSxn</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>00C40E04</td>
<td>SPDnoa</td>
<td>-</td>
</tr>
<tr>
<td>C5</td>
<td>00C50664</td>
<td>SPDSxoxn</td>
<td>-</td>
</tr>
<tr>
<td>C6</td>
<td>00C60708</td>
<td>SDPnax</td>
<td>-</td>
</tr>
<tr>
<td>C7</td>
<td>00C707AA</td>
<td>SPDPoaxn</td>
<td>-</td>
</tr>
<tr>
<td>C8</td>
<td>00C803A8</td>
<td>SDPoa</td>
<td>-</td>
</tr>
<tr>
<td>C9</td>
<td>00C90184</td>
<td>SPDOxn</td>
<td>-</td>
</tr>
<tr>
<td>CA</td>
<td>00CA0749</td>
<td>DPSDxax</td>
<td>-</td>
</tr>
<tr>
<td>CB</td>
<td>00CB06E4</td>
<td>SPDSaoxn</td>
<td>-</td>
</tr>
<tr>
<td>CC</td>
<td>00CC0020</td>
<td>S</td>
<td>SRCCOPY</td>
</tr>
<tr>
<td>CD</td>
<td>00CD0888</td>
<td>SPDono</td>
<td>-</td>
</tr>
<tr>
<td>CE</td>
<td>00CE0B08</td>
<td>SDPnao</td>
<td>-</td>
</tr>
<tr>
<td>CF</td>
<td>00CF0224</td>
<td>SPno</td>
<td>-</td>
</tr>
<tr>
<td>D0</td>
<td>00D00E0A</td>
<td>PSDnoa</td>
<td>-</td>
</tr>
<tr>
<td>D1</td>
<td>00D1066A</td>
<td>SPDPxoxn</td>
<td>-</td>
</tr>
<tr>
<td>D2</td>
<td>00D20705</td>
<td>PDSnax</td>
<td>-</td>
</tr>
<tr>
<td>D3</td>
<td>00D307A4</td>
<td>SPDSaoxn</td>
<td>-</td>
</tr>
<tr>
<td>D4</td>
<td>00D41D78</td>
<td>SPSPxPDxax</td>
<td>-</td>
</tr>
<tr>
<td>D5</td>
<td>00D50CE9</td>
<td>DPSPanan</td>
<td>-</td>
</tr>
<tr>
<td>D6</td>
<td>00D616EA</td>
<td>PSDP Saoxx</td>
<td>-</td>
</tr>
<tr>
<td>D7</td>
<td>00D70349</td>
<td>DPSxan</td>
<td>-</td>
</tr>
<tr>
<td>D8</td>
<td>00D80745</td>
<td>PDSPxax</td>
<td>-</td>
</tr>
<tr>
<td>D9</td>
<td>00D906E8</td>
<td>SPDSaoxn</td>
<td>-</td>
</tr>
<tr>
<td>DA</td>
<td>00DA1CE9</td>
<td>DPSDanax</td>
<td>-</td>
</tr>
<tr>
<td>DB</td>
<td>00DB0D75</td>
<td>SPxDSxan</td>
<td>-</td>
</tr>
<tr>
<td>DC</td>
<td>00DC0B04</td>
<td>SPDNSao</td>
<td>-</td>
</tr>
<tr>
<td>DD</td>
<td>00DD0228</td>
<td>SDno</td>
<td>-</td>
</tr>
<tr>
<td>DE</td>
<td>00DE0268</td>
<td>SPDxao</td>
<td>-</td>
</tr>
<tr>
<td>DF</td>
<td>00DF08C8</td>
<td>SPDano</td>
<td>-</td>
</tr>
<tr>
<td>E0</td>
<td>00E003A5</td>
<td>SPDSoa</td>
<td>-</td>
</tr>
<tr>
<td>E1</td>
<td>00E10185</td>
<td>PDSoxn</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 11.3: Raster-operation codes (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2</td>
<td>00E20746</td>
<td>DSPDxax</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>00E306EA</td>
<td>PSDPaoxn</td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>00E40748</td>
<td>SDPSxax</td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>00E506E5</td>
<td>PDSPaoxn</td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td>00E61CE8</td>
<td>SDPSanax</td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>00E70D79</td>
<td>SPxPDxan</td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>00E81D74</td>
<td>SSPxDSxax</td>
<td></td>
</tr>
<tr>
<td>E9</td>
<td>00E95CE6</td>
<td>DSPDSanaxx</td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td>00EA02E9</td>
<td>DSPao</td>
<td></td>
</tr>
<tr>
<td>EB</td>
<td>00EB0849</td>
<td>DSPxno</td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td>00EC02E8</td>
<td>SDPao</td>
<td></td>
</tr>
<tr>
<td>ED</td>
<td>00ED0848</td>
<td>SDPxno</td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>00EE0086</td>
<td>DSoo</td>
<td>SRCPAINT</td>
</tr>
<tr>
<td>EF</td>
<td>00EF0A08</td>
<td>SDPnoo</td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>00F00021</td>
<td>P</td>
<td>PATCOPY</td>
</tr>
<tr>
<td>F1</td>
<td>00F10885</td>
<td>PDSono</td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>00F20B05</td>
<td>PDSnao</td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>00F3022A</td>
<td>PSno</td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>00F40B0A</td>
<td>PSDnao</td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>00F50225</td>
<td>PDo</td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>00F60265</td>
<td>PDSxo</td>
<td></td>
</tr>
<tr>
<td>F7</td>
<td>00F708C5</td>
<td>PDSano</td>
<td></td>
</tr>
<tr>
<td>F8</td>
<td>00F802E5</td>
<td>PDSao</td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>00F90845</td>
<td>PDSxno</td>
<td></td>
</tr>
<tr>
<td>FA</td>
<td>00FA0089</td>
<td>DPo</td>
<td></td>
</tr>
<tr>
<td>FB</td>
<td>00FB0A09</td>
<td>DPSnoo</td>
<td>PATPAINT</td>
</tr>
<tr>
<td>FC</td>
<td>00FC008A</td>
<td>PSoo</td>
<td></td>
</tr>
<tr>
<td>FD</td>
<td>00FD0A0A</td>
<td>PSDnoo</td>
<td></td>
</tr>
<tr>
<td>FE</td>
<td>00FE02A9</td>
<td>DPSoo</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>00FF0062</td>
<td>1</td>
<td>WHITENESS</td>
</tr>
</tbody>
</table>

For more information on topics related to raster-operation codes, see the following:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using raster-operation codes with GDI functions</td>
<td>Reference, Volume 1: Chapter 2, &quot;Graphics device interface functions,&quot; and Chapter 4, &quot;Functions directory&quot;</td>
</tr>
<tr>
<td>Setting the current drawing mode with <strong>SetROP2</strong></td>
<td>Reference, Volume 1: Chapter 4, &quot;Functions directory&quot;</td>
</tr>
</tbody>
</table>
This chapter contains an alphabetical list of the individual Microsoft Windows printer escapes. The printer escapes allow applications to access facilities of a particular output device that are not available directly through the graphics device interface (GDI). The escape calls are made by an application, translated by Windows, and then sent to the printer device driver.

**ABORTDOC**

**Syntax**

short Escape(hdc, ABORTDOC, NULL, NULL, NULL)

This escape terminates the current job, erasing everything the application has written to the device since the last ENDDOC escape.

The **ABORTDOC** escape should be used to terminate:

- Printing operations that do not specify an abort function using the SETABORTPROC escape
- Printing operations that have not yet reached their first NEWFRAME or NEXTBAND escape call

**Parameters**

- **hDC**
  - HDC Identifies the device context.

**Return value**

The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.
Comments

If an application encounters a printing error or a canceled print operation, it must not attempt to terminate the operation by using the `Escape` function with either the `ENDDOC` or `ABORTDOC` escape. GDI automatically terminates the operation before returning the error value.

If the application displays a dialog box to allow the user to cancel the print operation, it must send the `ABORTDOC` escape before destroying the dialog box.

The application must send the `ABORTDOC` escape before freeing the procedure-instance address of the abort function, if any.

BANDINFO

Syntax

`short Escape(hDC, BANDINFO, sizeof(BANDINFOSTRUCT), lpInData, lpOutData)`

This escape copies information about a device with banding capabilities to a structure pointed to by the `lpOutData` parameter. It is implemented only for devices that use banding.

Banding is a property of an output device that allows a page of output to be stored in a metafile and divided into bands, each of which is sent to the device to create a complete page.

The information copied to the structure pointed to by `lpOutData` includes:

- A value that indicates whether there are graphics in the next band
- A value that indicates whether there is text on the page
- A `RECT` data structure that contains a bounding rectangle for all graphics on the page

The `lpOutData` parameter is NULL if no data are returned.

The `lpInData` parameter specifies information sent by the application to the device driver. This information is read by the device driver only on the first `BANDINFO` escape call on a page.

Parameters

- `hDC` `HDC` Identifies the device context.
- `lpInData` `BANDINFOSTRUCT FAR *` Points to a `BANDINFOSTRUCT` data structure that contains information to be passed to the driver. See the following "Comments" section for more information on the `BANDINFOSTRUCT` data structure.
- `lpOutData` `BANDINFOSTRUCT FAR *` Points to a `BANDINFOSTRUCT` data structure that contains information returned by the
driver. See the following “Comments” section for more information on the BANDINFOSTRUCT data structure.

Return value

The return value specifies the outcome of the escape. It is 1 if the escape is successful. It is zero if the function fails or is not implemented by the driver.

Comments

The BANDINFOSTRUCT data structure contains information about the contents of a page and supplies a bounding rectangle for graphics on the page. The following shows the format of BANDINFOSTRUCT:

```c
typedef struct {
    BOOL fGraphicsFlag;
    BOOL fTextFlag;
    RECT GraphicsRect;
} BANDINFOSTRUCT;
```

The BANDINFOSTRUCT structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fGraphicsFlag</td>
<td>Is TRUE if graphics are or are expected to be on the page or in the band; otherwise, it is FALSE.</td>
</tr>
<tr>
<td>fTextFlag</td>
<td>Is TRUE if text is or is expected to be on the page or in the band; otherwise, it is FALSE.</td>
</tr>
<tr>
<td>GraphicsRect</td>
<td>Contains a RECT data structure that supplies a bounding region for all graphics on the page.</td>
</tr>
</tbody>
</table>

Table 12.1 shows the meaning of these fields, depending on which parameter contains the structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>When used in IpInData</th>
<th>When used in IpOutData</th>
</tr>
</thead>
<tbody>
<tr>
<td>fGraphicsFlag</td>
<td>TRUE if the application is informing the driver that graphics are on the page.</td>
<td>TRUE if the driver is informing the application that it expects graphics in this band.</td>
</tr>
<tr>
<td>fTextFlag</td>
<td>TRUE if the application is informing the driver that text is on the page.</td>
<td>TRUE if the driver is informing the application that it expects text in this band.</td>
</tr>
<tr>
<td>GraphicsRect</td>
<td>Supplies the bounding rectangle for all graphics on the page.</td>
<td>No valid return data.</td>
</tr>
</tbody>
</table>

An application should call this escape immediately after each call to the NEXTBAND escape. It is in reference to the band the driver returned to that escape.

An application should use this escape in the following manner:
On the first band, the driver may give the application a full-page band and ask for text only (\texttt{fGraphicsFlag} is set to FALSE and \texttt{fTextFlag} is set to TRUE). The application sends only text to the driver.

If in the first band the application indicated that it had graphics (\texttt{fGraphicsFlag} is set to TRUE), or that the driver encountered vector fonts, then the driver will band the rest of the page. If there are no graphics or vector fonts, then the next \texttt{NEXTBAND} will return an empty rectangle to indicate that the application should move on to the next page.

If there are graphics but no vector fonts (the application set \texttt{fGraphicsFlag} to TRUE, but there were no graphics in the first full-page text band), then for subsequent bands the driver may optionally band only into the rectangle the application passed. This rectangle bounds all graphics on the page. If there are vector fonts, then the driver will band the entire width and depth of the page with \texttt{fTextFlag} set to TRUE. It will also set \texttt{fGraphicsFlag} to true if the application set it.

The driver assumes that an application using \texttt{BANDINFO} will only send text in the first full-page text band since that is all the driver requested. Therefore, if the driver encounters a vector font or graphics in the band, it assumes they were generated by a text primitive and sets \texttt{fTextFlag} to TRUE for all subsequent graphics bands so they can be output as graphics. If the application does not satisfy this expectation, the image will still be generated properly, but the driver will spend time sending spurious text primitives to graphics bands.

Older drivers written before the \texttt{BANDINFO} escape was designed used full-page banding for text. If a particular driver does not support the \texttt{BANDINFO} escape but sets RC_BANDING, the application can detect full-page banding for text by determining if the first band on the page covers the entire page.

---

**BEGIN_PATH**

**Syntax**

| short Escape(hDC, BEGIN_PATH, NULL, NULL, NULL) |

This escape opens a path. A path is a connected sequence of primitives drawn in succession to form a single polyline or polygon. Paths enable applications to draw complex borders, filled shapes, and clipping areas by supplying a collection of other primitives that define the desired shape.

Printer escapes supporting paths enable applications to render images on sophisticated devices such as PostScript printers without generating huge polygons to simulate the images.
To draw a path, an application first issues the `BEGIN_PATH` escape. It then draws the primitives defining the border of the desired shape and issues an `END_PATH` escape. The `END_PATH` escape includes a parameter specifying how the path is to be rendered.

**Parameters**

- `hDC`  
  Identifies the device context.

**Return value**

The return value specifies the current path nesting level. If the escape is successful, the return value is the number of `BEGIN_PATH` escape calls without a corresponding `END_PATH` escape call. Otherwise, the return value is zero.

**Comments**

An application may begin a subpath within another path. If the subpath is closed, it is treated exactly like a polygon. If it is open, it is treated exactly like a polyline.

An application may use the `CLIP_TO_PATH` escape to define a clipping area corresponding to the interior or exterior of the currently open path.

---

**CLIP_TO_PATH**

**Syntax**

```c
short Escape(hDC, CLIP_TO_PATH, sizeof(int), lpClipMode, NULL)
```

This escape defines a clipping area bounded by the currently open path. It enables the application to save and restore the current clipping area and to set up an inclusive or exclusive clipping area bounded by the currently open path. If the path defines an inclusive clipping area, portions of primitives falling outside the interior bounded by the path are clipped. If the path defines an exclusive clipping area, portions of primitives falling inside the interior are clipped.

**Parameters**

- `hDC`  
  Identifies the device context.

- `lpClipMode`  
  Points to a short integer specifying the clipping mode. It can be one of the following values:
  - CLIP_SAVE (0) Saves the current clipping area.
  - CLIP_RESTORE (1) Restores the previous clipping area.
  - CLIP_INCLUSIVE (2) Sets an inclusive clipping area.
  - CLIP_EXCLUSIVE (3) Sets an exclusive clipping area.

**Return value**

The return value specifies the outcome of the escape. It is nonzero if the escape was successful. Otherwise, it is zero.

**Comments**

To clip a set of primitives against a path, an application should follow these steps:

1. Save the current clipping area using the `CLIP_TO_PATH` escape.
2. Begin a path using the BEGIN_PATH escape.
3. Draw the primitives bounding the clipping area.
4. Close the path using the END_PATH escape.
5. Set the clipping area using the CLIP_TO_PATH escape.
6. Draw the primitives to be clipped.
7. Restore the original clipping area using the CLIP_TO_PATH escape.

**DEVICEDATA**

**Syntax**

short Escape(hDC, DEVICEDATA, nCount, lpInData, lpOutData)

This escape is identical to the PASSTHROUGH escape. See the description of PASSTHROUGH for further information.

**DRAFTMODE**

**Syntax**

short Escape(hDC, DRAFTMODE, sizeof(int), lpDraftMode, NULL)

This escape turns draft mode off or on. Turning draft mode on instructs the device driver to print faster and with lower quality (if necessary). The draft mode can be changed only at page boundaries, for example, after a NEWFRAME escape directing the driver to advance to a new page.

**Parameters**

- **hDC**  
  HDC Identifies the device context.
- **IpDraftMode**  
  LPINT Points to a short-integer value that specifies the draft mode. It may be one of the following values:
  - 0  
  Specifies draft mode off.
  - 1  
  Specifies draft mode on.

**Return value**

The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

**Comments**

The default draft mode is off.

**DRAWPATTERNRRECT**

**Syntax**

short Escape(hDC, DRAWPATTERNRECT, sizeof(PRECTSTRUCT), lpInData, NULL)

This escape creates a pattern, gray scale, or solid black rectangle by using the pattern/rule capabilities of Page Control Language (PCL) on
Hewlett-Packard® LaserJet® or LaserJet-compatible printers. A gray scale is a gray pattern that contains a specific mixture of black and white pixels.

**Parameters**

- **hDC**: HDC Identifies the device context.
- **lpInData**: PRECT_STRUCT FAR * Points to a PRECT_STRUCT data structure that describes the rectangle. See the following “Comments” section for more information on the PRECT_STRUCT data structure.

**Return value**

The return value specifies the outcome of the escape. It is 1 if the escape is successful. Otherwise, it is zero.

**Comments**

The *lpInData* parameter points to a PRECT_STRUCT data structure that defines the rectangle to be created. The PRECT_STRUCT structure has the following format:

```c
typedef struct {
    POINT prPosition;
    POINT prSize;
    WORD prStyle;
    WORD prPattern;
} PRECT_STRUCT;
```

This structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prPosition</td>
<td>Specifies the upper-left corner of the rectangle.</td>
</tr>
<tr>
<td>prSize</td>
<td>Specifies the lower-right corner of the rectangle.</td>
</tr>
<tr>
<td>prStyle</td>
<td>Specifies the type of pattern. It may be one of the following values:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td>Black rule</td>
</tr>
<tr>
<td>1</td>
<td>White rule that erases bitmap data previously written to same area; this pattern is available on the HP LaserJet II only.</td>
</tr>
<tr>
<td>2</td>
<td>Gray scale</td>
</tr>
<tr>
<td>3</td>
<td>HP-defined</td>
</tr>
<tr>
<td>prPattern</td>
<td>Specifies the pattern. It is ignored for a black rule. It specifies the percentage of gray for a gray-scale pattern. It represents one of six Hewlett-Packard-defined patterns.</td>
</tr>
</tbody>
</table>

An application should use the QUERYESCSUPPORT escape to determine whether a device is capable of drawing patterns and rules before using the DRAWPATTERNRECT escape. If an application uses the BANDINFO escape, all patterns and rectangles sent by using DRAWPATTERNRECT should be treated as text and sent on a text band.
Do not try to erase patterns and rules created with the **DRAWPATTERNRECT** escape by placing opaque objects over them. To erase such patterns and rules, use the function calls provided by GDI.

**ENABLEDUPLEX**

**Syntax**

short Escape(hDC, ENABLEDUPLEX, sizeof(WORD), lpInData, NULL)

This escape enables the duplex printing capabilities of a printer. A device that possesses duplex printing capabilities is able to print on both sides of the output medium.

**Parameters**

- **hDC**
  - HDC Identifies the device context.

- **lpInData**
  - WORD FAR * Points to an unsigned 16-bit integer that specifies whether duplex or simplex printing is used. It may be one of the following values:
    - 0 Simplex
    - 1 Duplex with vertical binding
    - 2 Duplex with horizontal binding

**Return value**

The return value specifies the outcome of the escape. It is 1 if the escape is successful. Otherwise, it is zero.

**Comments**

An application should use the **QUERYESCSUPPORT** escape to determine whether an output device is capable of creating duplex output. If **QUERYESCSUPPORT** returns a nonzero value, the application should send the **ENABLEDUPLEX** escape even if simplex printing is desired. This guarantees replacement of any values set in the driver-specific dialog box. If duplex printing is enabled and an uneven number of **NEXTFRAME** escapes are sent to the driver prior to the **ENDDOC** escape, the driver will eject an additional page before ending the print job.

**ENABLEPAIRKERNING**

**Syntax**

short Escape(hDC, ENABLEPAIRKERNING, sizeof(int), lpNewKernFlag, lpOldKernFlag)

This escape enables or disables the driver's ability to kern character pairs automatically. Kerning is the process of adding or subtracting space between characters in a string of text.

When pair kerning is enabled, the driver automatically kerns those pairs of characters that are listed in the font's character-pair kerning table. The
driver reflects this kerning both on the printer and in **GetTextExtent** function calls.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hDC</strong></td>
<td><strong>HDC</strong> Identifies the device context.</td>
</tr>
<tr>
<td><strong>lpNewKernFlag</strong></td>
<td><strong>LPINT</strong> Points to a short-integer value that specifies whether automatic pair kerning is to be enabled (1) or disabled (0).</td>
</tr>
<tr>
<td><strong>lpOldKernFlag</strong></td>
<td><strong>LPINT</strong> Points to a short-integer value that will receive the previous automatic pair-kerning value.</td>
</tr>
</tbody>
</table>

**Return value**

The return value specifies the outcome of the escape. It is 1 if the escape is successful; it is zero if the escape is not successful or not implemented.

**Comments**

The default state of this escape is zero; automatic character-pair kerning is disabled.

A driver does not have to support the **ENABLEPAIRKERNING** escape just because it supplies the character-pair kerning table to the application via the **GETPAIRKERNTABLE** escape. In the case where the **GETPAIRKERNTABLE** escape is supported but the **ENABLEPAIRKERNING** escape is not, the application must properly space the kerned characters on the output device using the **ExtTextOut** function.

---

**ENABLERELATIVEWIDTHS**

**Syntax**

`short Escape(hDC, ENABLERELATIVEWIDTHS, sizeof(int), lpNewWidthFlag, lpOldWidthFlag)`

This escape enables or disables relative character widths. When relative widths are disabled (the default), each character’s width can be expressed as a number of device units. This guarantees that the extent of a string will equal the sum of the extents of the characters in the string. This allows applications to build an extent table by using one-character **GetTextExtent** function calls.

When relative widths are enabled, the sum of a string may not equal the sum of the widths of the characters. Applications that enable this feature are expected to retrieve the font’s extent table and compute relatively scaled string widths.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hDC</strong></td>
<td><strong>HDC</strong> Identifies the device context.</td>
</tr>
<tr>
<td><strong>lpNewWidthFlag</strong></td>
<td><strong>LPINT</strong> Points to a short-integer value that specifies whether relative widths are to be enabled (1) or disabled (0).</td>
</tr>
</tbody>
</table>
ENABLERELATIVEWIDTHS

ennialWidthFlag LPIINT Points to a short-integer value that will receive the previous relative character width value.

Return value The return value specifies the outcome of the escape. It is 1 if the escape is successful; it is zero if the escape is not successful or not implemented.

Comments The default state of this escape is zero; relative character widths are disabled.

The values specified as font units and accepted and returned by the escapes described in this chapter are returned in the relative units of the font when the ENABLERELATIVEWIDTHS escape is enabled.

It is assumed that only linear-scaling devices will be dealt with in a relative mode. Nonlinear-scaling devices do not implement this escape.

ENDDOC

Syntax short Escape(hDC, ENDDOC, NULL, NULL, NULL)

This escape ends a print job started by a STARTDOC escape.

Parameters HDC Identifies the device context.

Return value The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

Comments If an application encounters a printing error or a canceled print operation, it must not attempt to terminate the operation by using the Escape function with either the ENDDOC or ABORTDOC escape. GDI automatically terminates the operation before returning the error value.

If the application displays a dialog box to allow the user to cancel the print operation, it must send the ENDDOC escape before destroying the dialog box.

The application must send the ENDDOC escape before freeing the procedure-instance address of the abort function, if any.

END_PATH

Syntax short Escape(hDC, END_PATH, sizeof(PATH_INFO), lpInData, NULL)

This escape ends a path. A path is a connected sequence of primitives drawn in succession to form a single polyline or polygon. Paths enable
applications to draw complex borders, filled shapes, and clipping areas by supplying a collection of other primitives defining the desired shape.

Printer escapes supporting paths enable applications to render images on sophisticated devices such as PostScript printers without generating huge polygons to simulate them.

To draw a path, an application first issues the **BEGIN_PATH** escape. It then draws the primitives defining the border of the desired shape and issues an **END_PATH** escape.

The **END_PATH** escape takes as a parameter a pointer to a structure specifying the manner in which the path is to be rendered. The structure specifies whether or not the path is to be drawn and whether it is open or closed. Open paths define polylines, and closed paths define fillable polygons.

### Parameters

<table>
<thead>
<tr>
<th><strong>hDC</strong></th>
<th>HDC</th>
<th>Identifies the device context.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>lpInData</strong></td>
<td>PATH_INFO FAR *</td>
<td>Points to a PATH_INFO data structure that defines how the path is to be rendered. See the following “Comments” section for more information on this data structure.</td>
</tr>
</tbody>
</table>

### Return value

The return value specifies the current path nesting level. If the escape is successful, the return value is the number of **BEGIN_PATH** escape calls without a corresponding **END_PATH** call. Otherwise, the return value is -1.

### Comments

An application may begin a subpath within another path. If the subpath is closed, it is treated exactly like a polygon. If it is open, it is treated exactly like a polyline.

An application may use the **CLIP_TO_PATH** escape to define a clipping area corresponding to the interior or exterior of the currently open path.

The **lpInData** parameter points to a **PATH_INFO** data structure that specifies how to render the path. This data structure has the following form:

```c
typedef struct {
    short    RenderMode;
    BYTE     FillMode;
    BYTE     BkMode;
    LOGPEN   Pen;
    LOGBRUSH Brush;
    DWORD    BkColor;
} PATH_INFO;
```
The PATH_INFO structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RenderMode</td>
<td>Specifies how the path is to be rendered. It may be one of the following values:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>NO_DISPLAY (0)</td>
<td>The path is not drawn.</td>
</tr>
<tr>
<td>OPEN (1)</td>
<td>The path is drawn as an open polygon.</td>
</tr>
<tr>
<td>CLOSED (2)</td>
<td>The path is drawn as a closed polygon.</td>
</tr>
<tr>
<td>FillMode</td>
<td>Specifies how the path is to be filled. It can be one of the following values:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>ALTERNATE (1)</td>
<td>The fill is done using the alternate fill algorithm.</td>
</tr>
<tr>
<td>WINDING (2)</td>
<td>The fill is done using the winding fill algorithm.</td>
</tr>
<tr>
<td>BkMode</td>
<td>Specifies the background mode for filling the path. It can be one of the following values:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>OPAQUE</td>
<td>The background is filled with the background color before the brush is drawn.</td>
</tr>
<tr>
<td>TRANSPARENT</td>
<td>The background is not changed.</td>
</tr>
<tr>
<td>Pen</td>
<td>Specifies the pen with which the path is to be drawn. If RenderMode is set to NO_DISPLAY, the pen is ignored.</td>
</tr>
<tr>
<td>Brush</td>
<td>Specifies the brush with which the path is to be filled. If RenderMode is set to NO_DISPLAY or OPEN, the brush is ignored.</td>
</tr>
<tr>
<td>BkColor</td>
<td>Specifies the color with which the path is filled if BkMode is set to OPAQUE.</td>
</tr>
</tbody>
</table>

**ENUMPAPERBINS**

**Syntax**

short Escape(hDC, ENUMPAPERBINS, sizeof(int), lpNumBins, lpOutData)

This escape retrieves attribute information about a specified number of paper bins. The GETSETPAPERBINS escape retrieves the number of bins available on a printer. This escape is provided only for backward compatibility. An application should call the ExtDeviceMode function instead.

**Parameters**

- **hDC** HDC Identifies the device context.
- **lpNumBins** LPINT Points to an integer that specifies the number of bins for which information is to be retrieved.
ENUMPAPERBINS

IpOutData LPSTR Points to a data structure to which information about the paper bins is copied. The size of the structure depends on the number of bins for which information was requested. See the following “Comments” section for a description of this data structure.

Return value The return value specifies the outcome of the escape. It is 1 if the escape is successful; it is zero if the escape is not successful or not implemented.

Comments The data structure to which the IpOutData parameter points consists of two arrays. The first is an array of short integers containing the paper-bin identifier numbers in the following format:

short BinList[cBinMax]

The number of integers in the array (cBinMax) is equal to the value pointed to by the IpNumBins parameter.

The second array in the data structure to which IpOutData points is an array of characters in the following format:

char PaperNames[cBinMax][cchBinName]

The cBinMax value is equal to the value pointed to by the IpNumBins parameter; the cchBinName value is the length of each string (currently 24).

ENUMPAPERMETRICS

Syntax short Escape(hDC, ENUMPAPERMETRICS, sizeof(int), lpMode, lpOutData)

This escape performs one of two functions according to the mode:

- It determines the number of paper types supported and returns this value, which can then be used to allocate an array of RECT data structures.
- It returns one or more RECT data structures that define the areas on the page that can receive an image.

This escape is provided only for backward compatibility. An application should call the ExtDeviceMode function instead.

Parameters hDC HDC Identifies the device context.
lpMode LPINT Points to an integer that specifies the mode for the escape. It can be one of the following values:
ENUMPAPERMETRICS

- 0 The return value indicates how many RECT data structures are required to contain the information about the available paper types.
- 1 The array of RECT structures to which lpOutData points is filled with the information.

lpOutData  LPRECT Points to an array of RECT data structures that return all the areas that can receive an image.

Return value  The return value is positive if successful, zero if the escape is not implemented, and negative if an error occurred.

EPSPRINTING

Syntax  short Escape(hDC, EPSPRINTING, sizeof(BOOL), lpBool, NULL)

This escape suppresses the output of the Windows PostScript header control section, which is about 7K. If an application uses this escape, no GDI calls are allowed.

Parameters  hDC  HDC Identifies the device context.
lpBool  BOOL FAR * Points to a Boolean value indicating that downloading should be enabled (TRUE) or disabled (FALSE).

Return value  The return value is positive if successful, zero if the escape is not implemented, and negative if an error occurred.

EXTDEVICECAPS

Syntax  short Escape(hDC, EXTDEVICECAPS, sizeof(int), lpIndex, lpCaps)

This escape retrieves information about device-specific capabilities. It supplements the GetDeviceCaps function.

Parameters  hDC  HDC Identifies the device context.
lpIndex  LPINT Points to a short integer specifying the index of the capability to be retrieved. It can be any one of the following values:
- R2_CAPS (1) The lpCaps parameter indicates which of the 16 binary raster operations the device driver supports. A bit will be set for each supported raster operation. For further information, see the description of the SetROP2.
function in Chapter 4, “Functions directory,” in Reference, Volume 1.

- **PATTERN_CAPS (2)** The *lpCaps* parameter returns the maximum dimensions of a pattern brush bitmap. The low-order word of the capability value contains the maximum width of a pattern brush bitmap, and the high-order word contains the maximum height.

- **PATH_CAPS (3)** The *lpCaps* parameter indicates whether the device is capable of creating paths using alternate and winding interiors, and whether the device can do exclusive or inclusive clipping to path interiors. The path capabilities are obtained using the logical OR operation on the following values:
  - PATH_ALTERNATE (1)
  - PATH_WINDING (2)
  - PATH_INCLUSIVE (4)
  - PATH_EXCLUSIVE (8)

- **POLYGON_CAPS (4)** The *lpCaps* parameter returns the maximum number of polygon points supported by the device. The capability value is an unsigned value specifying the maximum number of points.

- **PATTERN_COLOR_CAPS (5)** The *lpCaps* parameter indicates whether the device can convert monochrome pattern bitmaps to color. The capability value is 1 if the device can do pattern bitmap color conversions, and zero if it cannot.

- **R2_TEXT_CAPS (6)** The *lpCaps* parameter indicates whether the device is capable of performing binary raster operations on text. The low-order word of the capability value specifies which raster operations are supported for text. A bit is set for each supported raster operation, as in the R2_CAPS escape. The high-order word specifies the type of text to which the raster capabilities apply. It is obtained by applying the logical OR operation to the following values together:
  - RASTER_TEXT (1)
  - DEVICE_TEXT (2)
  - VECTOR_TEXT (4)

- **POLYMODE_CAPS (7)** Specifies which poly modes are supported by the printer driver. The capability value is obtained by using the bitwise OR operator to combine a bit in the corresponding position for each supported poly
EXT_DEVICE_CAPS

mode. For example, if the printer supports the
PM_POLYSCANLINE and PM_BEZIER poly modes, the
capability value would be:

(1 PM_POLYSCANLINE) (PM_BEZIER)

See the description of the SET_POLY_MODE escape for
information on the poly modes.

lpCaps    DWORD FAR * Points to a 32-bit integer to which the
capabilities will be copied.

Return value   The return value is nonzero if the specified extended capability is
supported, and zero if it is not.

EXTTEXTOUT

Syntax  short Escape(hDC, EXTTEXTOUT, sizeof(EXTTEXT_STRUCT), lpInData,
NULL)

This escape provides an efficient way for the application to call the GDI
TextOut function when justification, letter spacing, and/or kerning are
involved.

This function is provided only for backward compatibility. New
applications should use the GDI ExtTextOut function instead.

Parameters   hDC        HDC Identifies the device context.
lpInData     EXTTEXT_STRUCT FAR * Points to an EXTTEXT_STRUCT
data structure that specifies the initial position, characters,
and character widths of the string. See the following
“Comments” section for more information on the
EXTTEXT_STRUCT data structure.

Return value   The return value specifies the outcome of the escape. It is 1 if the escape is
successful; it is zero if the escape is not successful or not implemented.

Comments   The EXTTEXT_STRUCT data structure has the following format:

typedef struct {
    WORD     X;
    WORD     Y;
    WORD FAR *lpText;
    WORD FAR *lpWidths;
} EXTTEXT_STRUCT;

This structure has the following fields.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Specifies the x-coordinate of the upper-left corner of the string’s starting point.</td>
</tr>
<tr>
<td>Y</td>
<td>Specifies the y-coordinate of the upper-left corner of the string’s starting point.</td>
</tr>
<tr>
<td>IpText</td>
<td>Points to an array of cch character codes, where cch is the number of bytes in the string (cch is also the number of words in the width array).</td>
</tr>
<tr>
<td>IpWidths</td>
<td>Points to an array of cch character widths to use when printing the string. The first character appears at (X,Y), the second at (X + IpWidths[0],Y), the third at (X + IpWidths[0] + IpWidths[1],Y), and so on. These character widths are specified in the font units of the currently selected font. (The character widths will always be equal to device units unless the application has enabled relative character widths.) The units contained in the width array are specified as font units of the device.</td>
</tr>
</tbody>
</table>

**FLUSHOUTPUT**

**Syntax**

```c
short Escape(hDC, FLUSHOUTPUT, NULL, NULL, NULL)
```

This escape clears all output from the device’s buffer.

**Parameters**

- **hDC** HDC Identifies the device context.

**Return value**

The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

**GETCOLORTABLE**

**Syntax**

```c
short Escape(hDC, GETCOLORTABLE, sizeof(int), IpIndex, IpColor)
```

This escape retrieves an RGB color-table entry and copies it to the location specified by the IpColor parameter.

**Parameters**

- **hDC** HDC Identifies the device context.
- **lpIndex** LPINT Points to a short-integer value that specifies the index of a color-table entry. Color-table indexes start at zero for the first table entry.
- **lpColor** DWORD FAR * Points to the long-integer value that will receive the RGB color value for the given entry.
GETCOLORTABLE

Return value
The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

GETEXTENDEDTEXTMETRICS

Syntax
short Escape(hDC, GETEXTENDEDTEXTMETRICS, sizeof(WORD), lpInData, lpOutData)

This escape fills the buffer pointed to by the lpOutData parameter with the extended text metrics for the selected font.

Parameters
- **hDC**: HDC Identifies the device context.
- **lpInData**: WORD FAR * Points to an unsigned 16-bit integer that specifies the number of bytes pointed to by the lpOutData parameter.
- **lpOutData**: EXTTEXTMETRIC FAR * Points to an EXTTEXTMETRIC data structure. See the following “Comments” section for a description of this data structure.

Return value
The return value specifies the number of bytes copied to the buffer pointed to by the lpOutData parameter. This value will never exceed that specified in the nSize field pointed to by the lpInData parameter. The return value is zero if the escape fails or is not implemented.

Comments
The lpOutData parameter points to an EXTTEXTMETRIC data structure which has the following format:

```c
typedef struct{
    short etmSize;
    short etmPointSize;
    short etmOrientation;
    short etmMasterHeight;
    short etmMinScale;
    short etmMaxScale;
    short etmMasterUnits;
    short etmCapHeight;
    short etmXHeight;
    short etmLowerCaseAscent;
    short etmLowerCaseDescent;
    short etmSlant;
    short etmSuperScript;
    short etmSubScript;
    short etmSuperScriptSize;
    short etmSubScriptSize;
    short etmUnderlineOffset;
```

Software development kit
short etmUnderlineWidth;
short etmDoubleUpperUnderlineOffset;
short etmDoubleLowerUnderlineOffset;
short etmDoubleUpperUnderlineWidth;
short etmDoubleLowerUnderlineWidth;
short etmStrikeOutOffset;
short etmStrikeOutWidth;
WORD etmKernPairs;
WORD etmKernTracks;
}EXTEXTMETRIC;

The **EXTEXTMETRIC** data structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>etmSize</td>
<td>Specifies the size of the structure in bytes.</td>
</tr>
<tr>
<td>etmPointSize</td>
<td>Specifies the nominal point size of this font in twips (twentieths of a point, or 1/1440 inch). This is the intended size of the font; the actual size may differ slightly depending on the resolution of the device.</td>
</tr>
<tr>
<td>etmOrientation</td>
<td>Specifies the orientation of the font. The <strong>etmOrientation</strong> field may be any of the following values:</td>
</tr>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>etmMasterHeight</td>
<td>Specifies the font size in device units for which the values in this font's extent table are exact.</td>
</tr>
<tr>
<td>etmMinScale</td>
<td>Specifies the minimum valid size for this font. The following equation illustrates how the minimum point size is determined:</td>
</tr>
<tr>
<td></td>
<td>smallest point size = etmMinScale * 72/dVertRes</td>
</tr>
<tr>
<td></td>
<td>The value 72 represents the number of points per inch. The dVertRes value is the number of dots per inch.</td>
</tr>
<tr>
<td>etmMaxScale</td>
<td>Specifies the maximum valid size for this font. The following equation illustrates how the maximum point size is determined:</td>
</tr>
<tr>
<td></td>
<td>largest point size = etmMaxScale * 72/dVertRes</td>
</tr>
</tbody>
</table>
GETEXTENDEDTEXTMETRICS

The value 72 represents the number of points per inch. The *dfVertRes* value is the number of dots per inch.

**etmMasterUnits**

Specifies the integer number of units per em where an em equals *etmMasterHeight*. That is, *etmMasterUnits* is *etmMasterHeight* expressed in font units rather than device units.

**etmCapHeight**

Specifies the height in font units of uppercase characters in the font. Typically, this is the height of the capital H.

**etmXHeight**

Specifies the height in font units of lowercase characters in the font. Typically, this is the height of the lowercase x.

**etmLowerCaseAscent**

Specifies the distance in font units that the ascender of lowercase letters extends above the baseline. Typically, this is the height of the lowercase d.

**etmLowerCaseDescent**

Specifies the distance in font units that the descender of lowercase letters extends below the baseline. Typically, this is specified for the descender of the lowercase p.

**etmSlant**

Specifies for an italicized or slanted font the angle of the slant measured in tenths of a degree clockwise from the upright version of the font.

**etmSuperScript**

Specifies in font units the recommended amount to offset superscript characters from the baseline. This is typically a negative value.

**etmSubScript**

Specifies in font units the recommended amount to offset subscript characters from the baseline. This is typically a positive value.

**etmSuperScriptSize**

Specifies in font units the recommended size of superscript characters for this font.

**etmSubScriptSize**

Specifies in font units the recommended size of subscript characters for this font.

**etmUnderlineOffset**

Specifies in font units the offset downward from the baseline where the top of a single underline bar should appear.

**etmUnderlineWidth**

Specifies in font units the thickness of the underline bar.

**etmDoubleUpperUnderlineOffset**

Specifies the offset in font units downward from the baseline where the top of the upper double underline bar should appear.

**etmDoubleLowerUnderlineOffset**

Specifies the offset in font units downward from the baseline where the top of the lower double underline bar should appear.

**etmDoubleUpperUnderlineWidth**

Specifies in font units the thickness of the upper underline bar.
**GETEXTENDEDTEXTMETRICS**

- **etmDoubleLowerUnderlineWidth**: Specifies in font units the thickness of the lower underline bar.
- **etmStrikeOutOffset**: Specifies in font units the offset upward from the baseline where the top of a strike-out bar should appear.
- **etmStrikeOutWidth**: Specifies the thickness in font units of the strike-out bar.
- **etmKernPairs**: Specifies the number of character kerning pairs defined for this font. An application can use this value to calculate the size of the pair-kern table returned by the `GETPAIRKERNTABLE` escape. It will not be greater than 512 kern pairs.
- **etmKernTracks**: Specifies the number of kerning tracks defined for this font. An application can use this value to calculate the size of the track-kern table returned by the `GETTRACKKERNTABLE` escape. It will not be greater than 16 kern tracks.

The values returned in many of the fields of the `EXTTEXTMETRIC` structure are affected by whether relative character widths are enabled or disabled. For more information, see the description of `ENABLERELATIVEWIDTHS` escape earlier in this chapter.

---

**GETEXTENTTABLE**

**Syntax**

```
short Escape(hDC, GETEXTENTTABLE, sizeof(CHAR_RANGE_STRUCT), lpInData, lpOutData)
```

This escape retrieves the width (extent) of individual characters from a group of consecutive characters in the selected font's character set.

**Parameters**

- **hDC**: HDC Identifies the device context.
- **lpInData**: LPSTR Points to a `CHAR_RANGE_STRUCT` data structure that defines the range of characters for which the width is to be retrieved. See the following “Comments” section for more information on the `CHAR_RANGE_STRUCT` data structure.
- **lpOutData**: LPINT Points to an array of short integers that receives the character widths. The size of the array must be at least `(chLast - chFirst + 1)`.

**Return value**

The return value specifies the outcome of the escape. It is 1 if the escape is successful, and zero if the escape is not successful. If the escape is not implemented, the return value is zero.
GETEXTENTTABLE

Comments

The lpInData parameter points to a CHAR_RANGE_STRUCT data structure that defines the range of characters for which the width is to be retrieved. The CHAR_RANGE_STRUCT structure has the following format:

```c
typedef struct {
    BYTE chFirst;
    BYTE chLast;
} CHAR_RANGE_STRUCT
```

This structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chFirst</td>
<td>Specifies the character code of the first character whose width is to be retrieved.</td>
</tr>
<tr>
<td>chLast</td>
<td>Specifies the character code of the last character whose width is to be retrieved.</td>
</tr>
</tbody>
</table>

The values retrieved are affected by whether relative character widths are enabled or disabled. For more information, see the ENABLERELATIVEWIDTHS escape, earlier in this chapter.

GETFACENAME

Syntax

short Escape(hDC, GETFACENAME, NULL, NULL, lpFaceName)

This escape retrieves the face name of the current physical font.

Parameters

- **hDC** HDC Identifies the device context.
- **lpFaceName** LPSTR Points to a buffer of characters to receive the face name. This buffer must be at least 60 bytes in length.

Return value

The return value is positive if the escape was successful, zero if the escape is not implemented, or negative if an error occurred.

GETPAIRKERNTABLE

Syntax

short Escape(hDC, GETPAIRKERNTABLE, NULL, NULL, lpOutData)

This escape fills the buffer pointed to by the lpOutData parameter with the character-pair kerning table for the selected font.

Parameters

- **hDC** HDC Identifies the device context.
**IpOutData**  
*KERNPAIR FAR* * Points to an array of *KERNPAIR* data structures. This array must be large enough to accommodate the font's entire character-pair kerning table. The number of character-kerning pairs in the font can be obtained from the *EXTTEXTMETRIC* data structure returned by the *GETEXTENDEDTEXTMETRICS* escape. See the following “Comments” section for the format of the *KERNPAIR* data structure.

**Return value**  
The return value specifies the number of *KERNPAIR* structures copied to the buffer. This value is zero if the font does not have kerning pairs defined, the escape fails, or is not implemented.

**Comments**  
The *KERNPAIR* data structure has the following format:

```
typedef struct {
    union {
        BYTE each[2]; /* UNION: 'each' and 'both'
        share the same memory */
        WORD both;
    } kpPair;
    short kpKernAmount;
} KERNPAIR;
```

The *KERNPAIR* structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kpPair.each[0]</td>
<td>Specifies the character code for the first character in the kerning pair.</td>
</tr>
<tr>
<td>kpPair.each[1]</td>
<td>Specifies the character code for the second character in the kerning pair.</td>
</tr>
<tr>
<td>kpPair.both</td>
<td>Specifies a <em>WORD</em> in which the first character in the kerning pair is in the low-order byte and the second character is in the high-order byte.</td>
</tr>
<tr>
<td>kpKernAmount</td>
<td>Specifies the signed amount that this pair will be kerned if they appear side by side in the same font and size. This value is typically negative since pair-kerning usually results in two characters being set more tightly than normal.</td>
</tr>
</tbody>
</table>

The array of *KERNPAIR* structures is sorted in increasing order by the *kpPair.both* field.

The values returned in the *KERNPAIR* structures are affected by whether relative character widths are enabled or disabled. For more information, see the description of the *ENABLERELATIVEWIDTHS* escape earlier in this chapter.
GETPHYPAGESIZE

Syntax
short Escape(hDC, GETPHYPAGESIZE, NULL, NULL, lpDimensions)

This escape retrieves the physical page size and copies it to the location pointed to by the lpDimensions parameter.

Parameters
hDC HDC Identifies the device context.
lpDimensions LPPOINT Points to a POINT data structure that will receive the physical page dimensions. The x field of the POINT data structure receives the horizontal size in device units, and the y field receives the vertical size in device units.

Return value The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

GETPRINTINGOFFSET

Syntax
short Escape(hDC, GETPRINTINGOFFSET, NULL, NULL, lpOffset)

This escape retrieves the offset from the upper-left corner of the physical page where the actual printing or drawing begins. This escape is generally not useful for devices that allow the user to set the origin of the printable area directly.

Parameters
hDC HDC Identifies the device context.
lpOffset LPPOINT Points to a POINT structure that will receive the printing offset. The x field of the POINT structure receives the horizontal coordinate of the printing offset in device units, and the y field receives the vertical coordinate of the printing offset in device units.

Return value The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

GETSCALINGFACTOR

Syntax
short Escape(hDC, GETSCALINGFACTOR, NULL, NULL, lpFactors)

This escape retrieves the scaling factors for the x- and y-axes of a printing device. For each scaling factor, the escape copies an exponent of 2 to the location pointed to by the lpFactors parameter. For example, the value 3 is copied to lpFactors if the scaling factor is 8.
Scaling factors are used by printing devices that support graphics at a smaller resolution than text.

**Parameters**

- **hDC** (`HDC`) Identifies the device context.
- **lpFactors** (`LPPOINT`) Points to the `POINT` data structure that will receive the scaling factor. The `x` field of the `POINT` structure receives the scaling factor for the `x`-axis, and the `y` field receives the scaling factor for the `y`-axis.

**Return value**
The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

---

**GETSETPAPERBINS**

**Syntax**

```
short Escape(hDC, GETSETPAPERBINS, nCount, lpInData, lpOutData)
```

This escape retrieves the number of paper bins available on a printer and sets the current paper bin. See the following “Comments” section for more information on the actions performed by this escape.

**Parameters**

- **hDC** (`HDC`) Identifies the device context.
- **nCount** (`int`) Specifies the number of bytes pointed to by the `lpInData` parameter.
- **lpInData** (`BinInfo FAR *`) Points to a `BinInfo` data structure that specifies the new paper bin. It may be set to NULL.
- **lpOutData** (`BinInfo FAR *`) Points to a `BinInfo` data structure that contains information about the current or previous paper bin and the number of bins available.

**Comments**

There are three possible actions for this escape, depending on the values passed in the `lpInData` and `lpOutData` parameters:

<table>
<thead>
<tr>
<th>lpInData</th>
<th>lpOutData</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>BinInfo</td>
<td>Retrieves the number of bins and the number of the current bin.</td>
</tr>
<tr>
<td>BinInfo</td>
<td>BinInfo</td>
<td>Sets the current bin to the number specified in the <code>BinNumber</code> field of the data structure to which <code>lpInData</code> points and retrieves the number of the previous bin.</td>
</tr>
<tr>
<td>BinInfo</td>
<td>NULL</td>
<td>Sets the current bin to the number specified in the <code>BinNumber</code> field of the data structure to which <code>lpInData</code> points.</td>
</tr>
</tbody>
</table>

The `BinInfo` data structure has the following format:
typedef struct{
    DWORD BinNumber;
    DWORD NbrofBins;
    DWORD Reserved;
    DWORD Reserved;
    DWORD Reserved;
    DWORD Reserved;
} BinInfo;

The **BinInfo** structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinNumber</td>
<td>Identifies the current or previous paper bin.</td>
</tr>
<tr>
<td>NbrofBins</td>
<td>Specifies the number of paper bins available.</td>
</tr>
</tbody>
</table>

When setting a new bin, the setting does not take effect until a new device context is created (without initialization data). The setting will take immediate effect if the high bit of the bin number is set, so that the next page printed will come from the new bin. For example, 0x8001 uses the second bin immediately whenever 0x0001 sets the same bin as the default for later print jobs.

In general, only the immediate-selection form should be used by applications. Setting the bin for future print jobs is supported for backward compatibility to an earlier form of this escape which appeared in some versions of HP's Page Control Language (PCL) and PostScript.

**GETSETPAPERMETRICS**

**Syntax**
short Escape(hDC, GETSETPAPERMETRICS, sizeof(RECT), lpNewPaper, lpPrevPaper)

This escape sets the paper type according to the given paper metrics information. It also retrieves the current printer's paper metrics information. This escape is provided only for backward compatibility. An application should call the **ExtDeviceMode** function instead.

This escape expects a **RECT** data structure representing the imageable area of the physical page and assumes the origin is in the upper-left corner.

**Parameters**

- **hDC**: HDC Identifies the device context.
- **lpNewPaper**: LPRECT Points to a **RECT** data structure that defines the new imageable area.
IpPrevPaper  **LPRECT** Points to a **RECT** data structure that receives the previous imageable area.

**Return value**  The return value is positive if successful, zero if the escape is not implemented, and negative if an error occurs.

**Comments**  This escape is provided only for backward compatibility. New applications should use the GDI **DeviceCapabilities** and **ExtDeviceMode** functions instead.

---

**GETSETPAPERORIENT**

**Syntax**  
```c
short Escape(hDC, GETSETPAPERORIENT, nCount, lpInData, NULL)
```

This escape returns or sets the current paper orientation. This escape is provided only for backward compatibility. An application should call the **ExtDeviceMode** function instead.

**Parameters**

- **hDC**  **HDC** Identifies the device context.
- **nCount**  Specifies the number of bytes pointed to by the **lpInData** parameter.
- **lpInData**  **ORIENT FAR * Points to an ORIENT data structure that specifies the new paper orientation.** See the following “Comments” section for a description of this data structure. It may be set to NULL, in which case the **GETSETPAPERORIENT** escape returns the current paper orientation.

**Return value**  The return value specifies the current orientation if **lpInData** is NULL; otherwise, it is the previous orientation. The return value is -1 if the escape failed.

**Comments**  This escape is provided only for backward compatibility. New applications should use the GDI **DeviceCapabilities** and **ExtDeviceMode** functions instead.

The **ORIENT** data structure has the following format:

```c
typedef struct{
    DWORD Orientation;
    DWORD Reserved;
    DWORD Reserved;
    DWORD Reserved;
    DWORD Reserved;
} ORIENT;
```
The Orientation field can be either of these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The new orientation is portrait.</td>
</tr>
<tr>
<td>2</td>
<td>The new orientation is landscape.</td>
</tr>
</tbody>
</table>

This escape is also known as GETSETPAPERORIENTATION.

**GETSETSCREENPARAMS**

**Syntax**
short Escape(hDC, GETSETSCREENPARAMS, sizeof(SCRENNPARAMS), lpInData, lpOutData)

This escape retrieves or sets the current screen information for rendering halftones.

**Parameters**
- **hDC** HDC Identifies the device context.
- **lpInData** SCREENPARAMS FAR * Points to a SCREENPARAMS data structure that contains the new screen information. This parameter may be NULL.
- **lpOutData** SCREENPARAMS FAR * Points to a SCREENPARAMS data structure that retrieves the previous screen information. This parameter may be NULL.

**Return value**
The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

**Comments**
This escape affects how device-independent bitmaps (DIBs) are rendered and how color objects are filled.

The SCREENPARAMS data structure has the following format:

```
typedef struct {
  int    angle;
  int    frequency;
  DWORD  types;
} SCREENPARAMS;
```

The SCREENPARAMS structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>Specifies in degrees the angle of the halftone screen.</td>
</tr>
<tr>
<td>frequency</td>
<td>Specifies in dots per inch of the screen frequency.</td>
</tr>
<tr>
<td>types</td>
<td>Is a mask containing bits which indicate the type of screen cell. If a</td>
</tr>
<tr>
<td></td>
<td>pointer to this structure is passed as the lpInData parameter, only one</td>
</tr>
<tr>
<td></td>
<td>bit may be set. If the lpOutData parameter contains a pointer to</td>
</tr>
</tbody>
</table>

Software development kit
this structure, when the escape returns, the types field will have a bit set for each type supported by the printer driver. Acceptable bit values are:

- **DIAMOND**
- **DOT**
- **ELLIPSE**
- **LINE**

---

**GETTECHNOLOGY**

**Syntax**

```
short Escape(hDC, GETTECHNOLOGY, NULL, NULL, lpTechnology)
```

This escape retrieves the general technology type for a printer, thereby allowing an application to perform technology-specific actions.

**Parameters**

- **hDC**  
  HDC Identifies the device context.
- **lpTechnology**  
  LPSTR Points to a buffer to which the driver copies a null-terminated string containing the printer technology type, such as “PostScript.”

**Return value**

The return value specifies the outcome of the escape. It is 1 if the escape is successful, and is zero if the escape is not successful or is not implemented.

---

**GETTRACKKERNTABLE**

**Syntax**

```
short Escape(hDC, GETTRACKKERNTABLE, NULL, NULL, lpOutData)
```

This escape fills the buffer pointed to by the `lpOutData` parameter with the track-kerning table for the currently selected font.

**Parameters**

- **hDC**  
  HDC Identifies the device context.
- **lpOutdata**  
  KERNTRACK FAR * Points to an array of KERNTRACK structures. This array must be large enough to accommodate all the font’s kerning tracks. The number of tracks in the font can be obtained from the EXTTEXTMETRIC structure returned by the GETEXTENDEDTEXTMETRICS escape. See the following “Comments” section for the format of the KERNTRACK data structure.

**Return value**

The return value specifies the number of KERNTRACK structures copied to the buffer. This value is zero if the font does not have kerning tracks defined, or if the escape fails or is not implemented.
The **KERNTRACK** data structure has the following format:

```c
typedef struct {
    short ktDegree;
    short ktMinSize;
    short ktMinAmount;
    short ktMaxSize;
    short ktMaxAmount;
} KERNTRACK;
```

The **KERNTRACK** structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ktDegree</td>
<td>Specifies the amount of track kerning. Increasingly negative values represent tighter track kerning, and increasingly positive values represent looser track kerning.</td>
</tr>
<tr>
<td>ktMinSize</td>
<td>Specifies in device units the minimum font size for which linear track kerning applies.</td>
</tr>
<tr>
<td>ktMinAmount</td>
<td>Specifies in font units the amount of track kerning to apply to font sizes less than or equal to the size specified by the <code>ktMinSize</code> field.</td>
</tr>
<tr>
<td>ktMaxSize</td>
<td>Specifies in device units the maximum font size for which linear track kerning applies.</td>
</tr>
<tr>
<td>ktMaxAmount</td>
<td>Specifies in font units the amount of track kerning to apply to font sizes greater than or equal to the size specified by the <code>ktMaxSize</code> field.</td>
</tr>
</tbody>
</table>

Between the `ktMinSize` and `ktMaxSize` font sizes, track kerning is a linear function from `ktMinAmount` to `ktMaxAmount`. The values returned in the **KERNTRACK** structures are affected by whether relative character widths are enabled or disabled. For more information, see the description of the **ENABLERELATIVEWIDTHS** escape earlier in this chapter.

---

**GETVECTORBRUSHSIZE**

**Syntax**

```c
short Escape(hDC, GETVECTORBRUSHSIZE, sizeof(LOGBRUSH), lpInData, lpOutData)
```

This escape retrieves in device units the size of a plotter pen used to fill closed figures. GDI uses this information to prevent the plotter pen from writing over the borders of the figure when filling closed figures.

**Parameters**

- **hDC**
  - HDC Identifies the device context.

- **lpInData**
  - LOGBRUSH FAR * Points to a LOGBRUSH data structure that specifies the brush for which data are to be returned.
GETVECTORBRUSHSIZE

RETURN VALUE

IpOutData LPPOINT Points to a POINT data structure that contains in its second word the width of the pen in device units.

Return Value

The return value specifies the outcome of the escape. It is 1 if the escape is successful; it is zero if the escape is not successful or is not implemented.

GETVECTORPENSIZE

Syntax

short Escape(hDC, GETVECTORPENSIZE, sizeof(LOGPEN), IpInData, IpOutData)

This escape retrieves the size in device units of a plotter pen. GDI uses this information to prevent hatched brush patterns from overwriting the border of a closed figure.

Parameters

hDC HDC Identifies the device context.

IpInData LOGPEN FAR * Points to a LOGPEN data structure that specifies the pen for which the width is to be retrieved.

IpOutData LPPOINT Points to a POINT data structure that contains in its second word the width of the pen in device units.

Return Value

The return value specifies the outcome of the escape. It is 1 if the escape is successful; it is zero if the escape is not successful or if it is not implemented.

MFCOMMENT

Syntax

BOOL Escape(hDC, MFCOMMENT, nCount, IpComment, NULL)

This escape adds a comment to a metafile.

Parameters

hDC HDC Identifies the device context for the device on which the metafile appears.

nCount short Specifies the number of characters in the string pointed to by the IpComment parameter.

IpComment LPSTR Points to a string that contains the comment that will appear in the metafile.

Return Value

The return value specifies the outcome of the escape. It is -1 if an error such as insufficient memory or an invalid port specification occurs. Otherwise, it is positive.
NEWFRAME

**Syntax**

short Escape(hDC, NEWFRAME, NULL, NULL, NULL)

This escape informs the device that the application has finished writing to a page. This escape is typically used with a printer to direct the device driver to advance to a new page.

**Parameters**

* hDC HDC Identifies the device context.

**Return value**

The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP_APPABORT</td>
<td>Job was terminated because the application's abort function returned zero.</td>
</tr>
<tr>
<td>SP_ERROR</td>
<td>General error.</td>
</tr>
<tr>
<td>SP_OUTOFDISK</td>
<td>Not enough disk space is currently available for spooling, and no more space will become available.</td>
</tr>
<tr>
<td>SP_OUTOFMEMORY</td>
<td>Not enough memory is available for spooling.</td>
</tr>
<tr>
<td>SP_USERABORT</td>
<td>User terminated the job through the Print Manager.</td>
</tr>
</tbody>
</table>

**Comments**

Do not use the NEXTBAND escape with NEWFRAME. For banding drivers, GDI replays a metafile to the printer, simulating a sequence of NEXTBAND escapes.

The NEWFRAME escape restores the default values of the device context. Consequently, if a font other than the default font is selected when the application calls the NEWFRAME escape, the application must select the font again following the NEWFRAME escape.

NEXTBAND

**Syntax**

short Escape(hDC, NEXTBAND, NULL, NULL, lpBandRect)

This escape informs the device driver that the application has finished writing to a band, causing the device driver to send the band to the Print Manager and return the coordinates of the next band. Applications that process banding themselves use this escape.

**Parameters**

* hDC HDC Identifies the device context.

* lpBandRect LPRECT Points to the RECT data structure that will receive the next band coordinates. The device driver copies the device coordinates of the next band into this structure.
Nextband

Return value

The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP_APPABORT</td>
<td>Job was terminated because the application's abort function returned zero.</td>
</tr>
<tr>
<td>SP_ERROR</td>
<td>General error.</td>
</tr>
<tr>
<td>SP_OUTOFDISK</td>
<td>Not enough disk space is currently available for spooling, and no more space will become available.</td>
</tr>
<tr>
<td>SP_OUTOFMEMORY</td>
<td>Not enough memory is available for spooling.</td>
</tr>
<tr>
<td>SP_USERABORT</td>
<td>User terminated the job through the Print Manager.</td>
</tr>
</tbody>
</table>

Comments

The NEXTBAND escape sets the band rectangle to the empty rectangle when printing reaches the end of a page.

Do not use the NEWFRAME escape with NEXTBAND.

Passsthrough

Syntax

short Escape(hDC, PASSTHROUGH, nCount, lpInData, NULL)

This escape allows the application to send data directly to the printer, bypassing the standard print-driver code.

To use this escape, an application must have thorough knowledge of how the particular printer operates.

Parameters

- **hDC** HDC Identifies the device context.
- **nCount** short Specifies the number of bytes to which the lpInData parameter points.
- **lpInData** LPSTR Points to a structure whose first word (16 bits) contains the number of bytes of input data. The remaining bytes of the structure contain the data itself.

Return value

The return value specifies the number of bytes transferred to the printer if the escape is successful. It is less than zero if the escape is not implemented, and less than or equal to zero if the escape is not successful.

Comments

There may be restrictions on the kinds of device data an application can send to the device without interfering with the operation of the driver. In general, applications must avoid resetting the printer or causing the page to be printed.

It is strongly recommended that applications not perform functions that consume printer memory, such as downloading a font or a macro.
An application can avoid corrupting its data stream when issuing multiple, consecutive PASSTHROUGH escapes if it does not access the printer any other way during the sequence.

### Syntax
short Escape(hDC, QUERYESCSUPPORT, sizeof(int), lpEscNum, NULL)

This escape determines whether a particular escape is implemented by the device driver.

**Parameters**
- **hDC** HDC Identifies the device context.
- **lpEscNum** LPINT Points to a short-integer value that specifies the escape function to be checked.

**Return value**
The return value specifies whether a particular escape is implemented. It is nonzero for implemented escape functions. Otherwise, it is zero.

If the *lpEscNum* parameter is set to DRAWPATTERNRECT, the return value is one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DRAWPATTERNRECT is not implemented.</td>
</tr>
<tr>
<td>1</td>
<td>DRAWPATTERNRECT is implemented for a printer other than the HP LaserJet IIP; this printer supports white rules.</td>
</tr>
<tr>
<td>2</td>
<td>DRAWPATTERNRECT is implemented for the HP LaserJet IIP.</td>
</tr>
</tbody>
</table>

### RESTORE_CTM
short Escape(hDC, RESTORE_CTM, NULL, NULL, NULL)

This escape restores the previously saved current transformation matrix.

The current transformation matrix controls the manner in which coordinates are translated, rotated, and scaled by the device. By using matrices, an application can combine these operations in any order to produce the desired mapping for a particular picture.

**Parameters**
- **hDC** HDC Identifies the device context.

**Return value**
The return value specifies the number of SAVE_CTM escape calls without a corresponding RESTORE_CTM call. If the escape is unsuccessful, the return value is -1.
Applications should not make any assumptions about the initial contents of the current transformation matrix.

This escape uses a matrix specification based on the Microsoft OS/2 Presentation Manager graphics programming interface (GPI) model, which is an integer-coordinate system similar to the system which GDI uses.

**SAVE_CTM**

**Syntax**

```c
short Escape(hDC, SAVE_CTM, NULL, NULL, NULL)
```

This escape saves the current transformation matrix. The current transformation matrix controls the manner in which coordinates are translated, rotated, and scaled by the device. By using matrices, an application can combine these operations in any order to produce the desired mapping for a particular picture.

An application can restore the matrix by using the RESTORE_CTM escape.

An application typically saves the current transformation matrix before changing it. This allows the application to restore the previous state upon completion of a particular operation.

**Parameters**

- `hDC` HDC Identifies the device context.

**Return value**

The return value specifies the number of SAVE_CTM escape calls without a corresponding RESTORE_CTM call. The return value is zero if the escape was unsuccessful.

**Comments**

Applications should not make any assumptions about the initial contents of the current transformation matrix.

Applications are expected to restore the contents of the current transformation matrix.

This escape uses a matrix specification based on the OS/2 Presentation Manager graphics programming interface (GPI) model, which is an integer-coordinate system similar to the system that GDI uses.
This escape has been superseded by the **GETSETPAPERBINS** escape and is provided only for backward compatibility. New applications should use the **GETSETPAPERBINS** escape instead.

### SETABORTPROC

**Syntax**

```c
short Escape(hDC, SETABORTPROC, NULL, lpAbortFunc, NULL)
```

This escape sets the abort function for the print job. If an application is to allow the print job to be canceled during spooling, it must set the abort function before the print job is started with the **STARTDOC** escape. Print Manager calls the abort function during spooling to allow the application to cancel the print job or to process out-of-disk-space conditions. If no abort function is set, the print job will fail if there is not enough disk space for spooling.

**Parameters**

- **hDC** - HDC Identifies the device context.
- **lpAbortFunc** - FARPROC Points to the application-supplied abort function. See the following "Comments" section for details.

**Return value**

The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

**Comments**

The address passed as the **lpAbortFunc** parameter must be created by using the **MakeProcInstance** function.

The callback function must use the Pascal calling convention and must be declared **FAR**. The abort function must have the following form:

```pascal
short FAR PASCAL AbortFunc(hPr, code)
HDC hPr;
short code;

AbortFunc is a placeholder for the application-supplied function name. The actual name must be exported by including it in an **EXPORTS** statement in the application's module-definition file.
```

**Parameters**

- **hPr** - Identifies the device context.
- **code** - Specifies whether an error has occurred. It is zero if no error has occurred. It is **SP_OUTOFDISK** if Print Manager is currently out of disk space and more disk space will become available if the application waits.
If `code` is SP_OUTOFDISK, the application does not have to abort the print job. If it does not, it must yield to Print Manager by calling the `PeekMessage` or `GetMessage` function.

**Return value**
The return value should be nonzero if the print job is to continue, and zero if it is canceled.

---

**SETALLJUSTVALUES**

**Syntax**
```
short Escape(hDC, SETALLJUSTVALUES,
sizeof(JUST_VALUE_STRUCT), lpInData, NULL)
```

This escape sets all of the text-justification values that are used for text output.

Text justification is the process of inserting extra pixels among break characters in a line of text. The blank character is normally used as a break character.

**Parameters**
- `hDC`: HDC Identifies the device context.
- `lpInData`: JUST_VALUE_STRUCT FAR * Points to a JUST_VALUE_STRUCT data structure that defines the text-justification values. See the following “Comments” section for more information on the JUST_VALUE_STRUCT data structure.

**Return value**
The return value specifies the outcome of the escape. It is 1 if the escape is successful. Otherwise, it is zero.

**Comments**
The `lpInData` parameter points to a JUST_VALUE_STRUCT data structure that describes the text-justification values used for text output. The JUST_VALUE_STRUCT structure has the following format:

```c
typedef struct {
    short nCharExtra;
    WORD nCharCount;
    short nBreakExtra;
    WORD nBreakCount;
} JUST_VALUE_STRUCT;
```

This structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nCharExtra</td>
<td>Specifies the total extra space (in font units) that must be distributed over nCharCount characters.</td>
</tr>
</tbody>
</table>
SETALLJUSTVALUES

nCharCount  Specifies the number of characters over which nCharExtra is distributed.
nBreakExtra  Specifies the total extra space (in font units) that is distributed over nBreakCount characters.
nBreakCount  Specifies the number of break characters over which nBreakExtra is distributed.

The units used for nCharExtra and nBreakExtra are the font units of the device and are dependent on whether relative character widths were enabled with the ENABLERELATIVEWIDTHS escape.

The values set with this escape apply to subsequent calls to the TextOut function. The driver stops distributing the extra space specified in the nCharExtra field when it has output the number of characters specified in the nCharCount field. Likewise, it stops distributing the space specified by the nBreakExtra field when it has output the number of characters specified by the nBreakCount field. A call on the same string to the GetTextExtent function made immediately after the call to the TextOut function will be processed in the same manner.

To re-enable justification with the SetTextJustification and SetTextCharacterExtra functions, an application should call the SETALLJUSTVALUES escape and set the nCharExtra and nBreakExtra fields to zero.

SET_ARC_DIRECTION

Syntax  short Escape(hDC, SET_ARC_DIRECTION, sizeof(int), lpDirection, NULL)

This escape specifies the direction in which elliptical arcs are drawn using the GDI Arc function.

By convention, elliptical arcs are drawn counterclockwise by GDI. This escape lets an application draw paths containing arcs drawn clockwise.

Parameters  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDC</td>
<td>HDC</td>
<td>Identifies the device context.</td>
</tr>
<tr>
<td>lpDirection</td>
<td>LPINT</td>
<td>Points to a short integer specifying the arc direction. It can be either of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The return value is the previous arc direction.

Comments  This escape maps to PostScript language elements and is intended for PostScript line devices.
SET_BACKGROUND_COLOR

Syntax
short Escape(hDC, SET_BACKGROUND_COLOR, nCount, lpNewColor, lpOldColor)

This escape sets and retrieves the current background color for the device.

The background color is the color of the display surface before an application draws anything on the device. This escape is particularly useful for color printers and film recorders.

This escape should be sent before the application draws anything on the current page.

Parameters

- **hDC**
  HDC Identifies the device context.

- **nCount**
  int Specifies the number of bytes pointed to by the lpNewColor parameter.

- **lpNewColor**
  DWORD FAR * Points to a 32-bit integer specifying the desired background color. This parameter can be NULL if the application is merely retrieving the current background color.

- **lpOldColor**
  DWORD FAR * Points to a 32-bit integer which receives the previous background color. This parameter can be NULL if the application does not use the previous background color.

Return value
The return value is TRUE if the escape was successful and FALSE if it was unsuccessful.

Comments
The default background color is white.

The background color is reset to the default when the device driver receives an ENDDOC or ABORTDOC escape.

SET_BOUNDS

Syntax
short Escape(hDC, SET_BOUNDS, sizeof(RECT), lpInData, NULL)

This escape sets the bounding rectangle for the picture being produced by the device driver supporting the given device context. It is used when creating images in a file format such as Encapsulated PostScript (EPS) and Hewlett-Packard Graphics Language (HPGL) for which there is a device driver.

Parameters

- **hDC**
  HDC Identifies the device context.
**SET_BOUNDS**

*lpInData*  
**LPRECT** Points to a **RECT** data structure that specifies in device coordinates a rectangle that bounds the image to be output.

**Return value**  
The return value is **TRUE** if the escape was successful; otherwise, the return value is **FALSE**.

**Comments**  
An application should issue this escape before each page in the image. For single-page images, this escape should be issued immediately before the **STARTDOC** escape.

When an application uses coordinate-transformation escapes, device drivers may not perform bounding box calculations correctly. When an application uses the **SET_BOUNDS** escape, the driver does not have to calculate the bounding box.

Applications should always use this escape to ensure support for the Encapsulated PostScript (EPS) printing capabilities that will be built into future PostScript drivers.

---

**SET_CLIP_BOX**

**Syntax**  
short Escape(hDC, SET_CLIP_BOX, sizeof(RECT), lpInData, (LPSTR)NULL)

This escape sets the clipping rectangle or restores the previous clipping rectangle. This escape is implemented by printer drivers that implement the coordinate-transformation escapes **TRANSFORM_CTM, SAVE_CTM, and RESTORE_CTM**.

When an application calls a GDI output function, GDI calculates a clipping rectangle bounding the primitive and passes both the primitive and the clipping rectangle to the printer driver. The printer driver is expected to clip the primitive to the specified bounding rectangle. However, when an application uses the coordinate-transformation escapes, the clipping rectangle calculated by GDI is usually invalid. An application can use the **SET_CLIP_BOX** escape to specify the correct clipping rectangle when coordinate transformations are used.

**Parameters**  
*hDC*  
HDC Identifies the device context.

*lpClipBox*  
**LPRECT** Points to a **RECT** data structure containing the bounding rectangle of the clipping area. If *lpClipBox* is not NULL, the previous clipping rectangle is saved and the current clipping rectangle is set to the specified bounds. If
IpClipBox is NULL, the previous clipping rectangle is restored.

Return value  The return value is TRUE if the clipping rectangle was properly set. Otherwise, it is FALSE.

SETCOLORTABLE

Syntax  short Escape(hDC, SETCOLORTABLE, sizeof(COLORTABLE_STRUCT), lpInData, lpColor)

This escape sets an RGB color-table entry. If the device cannot supply the exact color, the function sets the entry to the closest possible approximation of the color.

Parameters  

hDC  HDC Identifies the device context.

lpInData  COLORTABLE_STRUCT FAR * Points to a COLORTABLE_STRUCT data structure that contains the index and RGB value of the color-table entry. See the following "Comments" section for more information on the COLORTABLE_STRUCT data structure.

lpColor  DWORD FAR * Points to the long-integer value that is to receive the RGB color value selected by the device driver to represent the requested color value.

Return value  The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

Comments  The COLORTABLE_STRUCT data structure has the following format:

```c
typedef struct {
    WORD   Index;
    DWORD  rgb;
} COLORTABLE_STRUCT;
```

This structure has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Specifies the color-table index. Color-table entries start at zero for the first entry.</td>
</tr>
<tr>
<td>rgb</td>
<td>Specifies the desired RGB color value.</td>
</tr>
</tbody>
</table>

A device's color table is a shared resource; changing the system display color for one window changes it for all windows. Only applications
developers who have a thorough knowledge of the display driver should use this escape.

The `SETCOLORTABLE` escape has no effect on devices with fixed color tables.

This escape is intended for use by both printer and display drivers. However, the EGA and VGA color drivers do not support it.

This escape changes the palette used by the display driver. However, since the driver’s color-mapping algorithms will probably no longer work with a different palette, an extension has been added to this function.

If the color index pointed to by the `lpInData` parameter is 0xFFFF, the driver is to leave all color-mapping functionality to the calling application. The application must use the proper color-mapping algorithm and take responsibility for passing the correctly mapped physical color to the driver (instead of the logical RGB color) in such device-driver functions as `RealizeObject` and `ColorInfo`.

For example, if the device supports 256 colors with palette indexes of 0 through 255, the application would determine which index contains the color that it wants to use in a certain brush. It would then pass this index in the low-order byte of the DWORD logical color passed to the `RealizeObject` device-driver function. The driver would then use this color exactly as passed instead of performing its usual color-mapping algorithm. If the application wants to reactivate the driver’s color-mapping algorithm (that is, if it restores the original palette when switching from its window context), then the color index pointed to by `lpInData` should be 0xFFFF.

---

**SETCOPYCOUNT**

**Syntax**

```
short Escape(hDC, SETCOPYCOUNT, sizeof(int), lpNumCopies, lpActualCopies)
```

This escape specifies the number of uncollated copies of each page that the printer is to print.

**Parameters**

- **hDC**
  - HDC Identifies the device context.

- **lpNumCopies**
  - LPINT Points to a short-integer value that contains the number of uncollated copies to be printed.

- **lpActualCopies**
  - LPINT Points to a short-integer value that will receive the number of copies to be printed. This may be less than the
number requested if the requested number is greater than the device's maximum copy count.

**Return value**
The return value specifies the outcome of the escape. It is 1 if the escape is successful; it is zero if the escape is not successful. If the escape is not implemented, the return value is zero.

### SETKERNTRACK

**Syntax**
short Escape(hDC, SETKERNTRACK, sizeof(int), lpNewTrack, lpOldTrack)

This escape specifies which kerning track to use for drivers that support automatic track kerning. A kerning track of zero disables automatic track kerning.

When track kerning is enabled, the driver will automatically kern all characters according to the specified track. The driver will reflect this kerning both on the printer and in `GetTextExtent` function calls.

**Parameters**
- **hDC**
  - HDC Identifies the device context.
- **lpNewTrack**
  - LPINT Points to a short-integer value that specifies the kerning track to use. A value of zero disables this feature. Values in the range 1 to `nKernTracks` correspond to positions in the track-kerning table (using 1 as the first item in the table). For more information, see the description of the `EXTTEXTMETRIC` structure provided under the description of the `GETEXTENDEDTEXTMETRICS` escape.
- **lpOldTrack**
  - LPINT Points to a short-integer value that will receive the previous kerning track.

**Return value**
The return value specifies the outcome of the escape. It is 1 if the escape is successful; it is zero if the escape is not successful or not implemented.

**Comments**
Automatic track kerning is disabled by default.

A driver does not have to support the `ENABLEPAIRKERNING` escape just because it supplies the track-kerning table to the application by using the `GETTRACKKERNTABLE` escape. In the case where `GETTRACKKERNTABLE` is supported but the `SETKERNTRACK` escape is not, the application must properly space the characters on the output device.
**SETLINECAP**

**Syntax**
short Escape(hDC, SETLINECAP, sizeof(int), lpNewCap, lpOldCap)

This escape sets the line end cap.

A line end cap is that portion of a line segment that appears on either end of the segment. The cap may be square or circular. It can extend past, or remain flush with the specified segment end points.

**Parameters**

- **hDC**  
  HDC Identifies the device context.

- **lpNewCap**  
  LPINT Points to a short-integer value that specifies the end-cap type. The possible values and their meanings are given in the following list:
  - -1 Line segments are drawn by using the default GDI end cap.
  - 0 Line segments are drawn with a squared end point that does not project past the specified segment length.
  - 1 Line segments are drawn with a rounded end point; the diameter of this semicircular arc is equal to the line width.
  - 2 Line segments are drawn with a squared end point that projects past the specified segment length. The projection is equal to half the line width.

- **lpOldCap**  
  LPINT Points to a short-integer value that specifies the previous end-cap setting.

**Return value**
The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

**Comments**
The interpretation of this escape varies with page-description languages (PDLs). Consult the PDL documentation for its exact meaning.

This escape is also known as **SETENDCAP**.

**SETLINEJOIN**

**Syntax**
short Escape(hDC, SETLINEJOIN, sizeof(int), lpNewJoin, lpOldJoin)

This escape specifies how a device driver will join two intersecting line segments. The intersection can form a rounded, squared, or mitered corner.

**Parameters**

- **hDC**  
  HDC Identifies the device context.
**SETLINEJOIN**

**IpNewJoin**  
**LPINT** Points to a short-integer value that specifies the type of intersection. The possible values and their meanings are given in the following list:

- **-1** Line segments are joined by using the default GDI setting.
- **0** Line segments are joined with a mitered corner; the outer edges of the lines extend until they meet at an angle. This is referred to as a miter join.
- **1** Line segments are joined with a rounded corner; a semicircular arc with a diameter equal to the line width is drawn around the point where the lines meet. This is referred to as a round join.
- **2** Line segments are joined with a squared end point; the outer edges of the lines are not extended. This is referred to as a bevel join.

**IpOldJoin**  
**LPINT** Points to a short-integer value that specifies the previous line join setting.

**Return value**  
The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

**Comments**  
The interpretation of this escape varies with page-description languages (PDLs). Consult the PDL documentation for its exact meaning.

If an application specifies a miter join but the angle of intersection is too small, the device driver ignores the miter setting and uses a bevel join instead.

---

**SET_MIRROR_MODE**

**Syntax**  
`short Escape(hDC, SET_MIRROR_MODE, sizeof(WORD), lpInData, (LPSTR)NULL)`

This escape sets the current mirror mode. The mirror mode produces mirror images along the horizontal axis, the vertical axis, or both axes.

To produce a mirror image of a given page, the application issues the **SET_MIRROR_MODE** escape before drawing the first primitive to be mirrored. When the last mirrored primitive has been drawn, the application issues a second **SET_MIRROR_MODE** escape to turn off mirroring.

**Parameters**  
- **hDC**  
  HDC Identifies the device context.
IpMirrorMode  **LPINT** Points to a short integer that specifies the mirror mode. It must be one of the following values:
- **MIRROR_NONE** (0) Disable mirroring.
- **MIRROR_HORIZONTAL** (1) Mirror along the horizontal axis.
- **MIRROR_VERTICAL** (2) Mirror along the vertical axis.
- **MIRROR_BOTH** (3) Mirror along both axes.

**Return value**
The return value is the previous mirror mode.

**Comments**
The default mirror mode is **MIRROR_NONE**.

Mirrored and unmirrored output can be mixed on a page. This allows the application to produce mirrored output with unmirrored page labels, crop marks, and so on.

SETMITERLIMIT

**Syntax**
short Escape(hDC, SETMITERLIMIT, sizeof(int), IpNewMiter, IpOldMiter)

This escape sets the miter limit for a device. The miter limit controls the angle at which a device driver replaces a miter join with a bevel join.

**Parameters**
- **hDC**  
  HDC Identifies the device context.
- **nCount**  
  **short** Specifies the number of bytes to which the IpNewMiter parameter points.
- **IpNewMiter**  
  **LPINT** Points to a short-integer value that specifies the desired miter limit. Only values greater than or equal to –1 are valid. If this value is –1, the driver will use the default GDI miter limit.
- **IpOldMiter**  
  **LPINT** Points to a short-integer value that specifies the previous miter-limit setting.

**Return value**
The return value specifies the outcome of the escape. It is positive if the escape is successful. Otherwise, it is negative.

**Comments**
The miter limit is defined as follows:

\[
\frac{\text{miter length}}{\text{line width}} = \frac{1}{\sin(x/2)}
\]

X is the angle of the line join in radians.

The interpretation of this escape varies with page-description languages (PDLs). Consult the PDL documentation for its exact meaning.
Syntax

short Escape(hDC, SET_POLY_MODE, sizeof(int), lpMode, NULL)

This escape sets the poly mode for the device driver. The poly mode is a state variable indicating how to interpret calls to the GDI **Polygon** and **Polyline** functions.

The **SET_POLY_MODE** escape enables a device driver to draw shapes (such as Bezier curves) not supported directly by GDI. This permits applications that draw complex curves to send the curve description directly to a device without having to simulate the curve as a polygon with a large number of points.

**Parameters**

- **hDC**
  - **HDC** Identifies the device context.

- **IpMode**
  - **LPINT** Points to a short integer specifying the desired poly mode. The poly mode is a state variable indicating how points in **Polygon** or **Polyline** function calls should be interpreted. All device drivers are not required to support all possible modes. A device driver returns zero if it does not support the specified mode. The **IpMode** parameter may be one of the following values:

  - **PM_POLYLINE** (1) The points define a conventional polygon or polyline.

  - **PM_BEZIER** (2) The points define a sequence of 4-point Bezier spline curves. The first curve passes through the first four points, with the first and fourth points as end points, and the second and third points as control points. Each subsequent curve in the sequence has the end point of the previous curve as its start point, the next two points as control points, and the third as its end point. The last curve in the sequence is permitted to have fewer than four points. If the curve has only one point, it is considered a point. If it has two points, it is a line segment. If it has three points, it is a parabola defined by drawing a Bezier curve with the first and third points as end points and the two control points equal to the second point.

  - **PM_POLYLINESEGMENT** (3) The points specify a list of coordinate pairs. Line segments are drawn connecting each successive pair of points.

  - **PM_POLYSCANLINE** (4) The points specify a list of coordinate pairs. Line segments are drawn connecting each successive pair of points. Each line segment is a
nominal-width line drawn using the current brush. Each line segment must be strictly vertical or horizontal, and scan lines must be passed in strictly increasing or decreasing order. This mode is only used for polygon calls.

**Return value**

The return value is the previous poly mode. If the return value is zero, the device driver did not handle the request.

**Comments**

An application should issue the **SET_POLY_MODE** escape before it draws a complex curve. It should then call the **Polyline** or **Polygon** function with the desired control points defining the curve. After drawing the curve, the application should reset the driver to its previous state by issuing the **SET_POLY_MODE** escape.

**Polyline** calls draw using the currently selected pen.

**Polygon** calls draw using the currently selected pen and brush. If the start and end points are not equal, a line is drawn from the start point to the end point before filling the polygon (or curve).

GDI treats **Polygon** calls using **PM_POLYLINESEGMENT** mode exactly the same as **Polyline** calls.

Four points define a Bezier curve. GDI generates the curve by connecting the first and second, second and third, and third and fourth points. GDI then connects the midpoints of these consecutive line segments. Finally, GDI connects the midpoints of the lines connecting the midpoints, and so forth.

The line segments drawn in this way converge to a curve defined by the following parametric equations, expressed as a function of the independent variable $t$.

\[
X(t) = (1-t)^3x_1 + 3(1-t)^2tx_2 + 3(1-t)t^2x_3 + t^3x_4
\]

\[
Y(t) = (1-t)^3y_1 + 3(1-t)^2ty_2 + 3(1-t)t^2y_3 + t^3y_4
\]

The points $(x_1,y_1)$, $(x_2,y_2)$, $(x_3,y_3)$ and $(x_4,y_4)$ are the control points defining the curve. The independent variable $t$ varies from 0 to 1.

The points $(Cx_1,Cy_1)$ and $(Cx_2,Cy_2)$ are third-degree control points of a second-degree Bezier curve specified by the points $(X_1,Y_1)$, $(X_2,Y_2)$, and $(X_3,Y_3)$.

Primitive types other than **PM_BEZIER** and **PM_POLYLINESEGMENT** may be added to this escape in the future. Applications should check the return value from this escape to determine whether or not the driver supports the specified poly mode.
SET_SCREEN_ANGLE

Syntax

short Escape(hDC, SET_SCREEN_ANGLE, sizeof(int), IpAngle, NULL)

This escape sets the current screen angle to the desired angle and enables an application to simulate the tilting of a photographic mask in producing a color separation for a particular primary.

Parameters

hDC HDC Identifies the device context.

lpAngle LPINT Points to a short-integer value specifying the desired screen angle in tenths of a degree. The angle is measured counterclockwise.

Return value

The return value is the previous screen angle.

Comments

Four-color process separation is the process of separating the colors comprising an image into four component primaries: cyan, magenta, yellow, and black. The image is then reproduced by overprinting each primary.

In traditional four-color process printing, half-tone images for each of the four primaries are photographed against a mask tilted to a particular angle. Tilting the mask in this manner minimizes unwanted moire patterns caused by overprinting two or more colors.

The device driver defines the default screen angle.

SET_SPREAD

Syntax

short Escape(hDC, SET_SPREAD, sizeof(int), IpSpread, NULL)

This function sets the amount that nonwhite primitives are expanded for a given device to provide a slight overlap between primitives to compensate for imperfections in the reproduction process.

Spot color separation is the process of separating an image into each distinct color used in the image. The image is reproduced by overprinting each successive color in the image.

When reproducing a spot-separated image, the printing equipment must be calibrated to align each page exactly on each pass. However, differences in temperature, humidity, and so forth, between passes often cause images to align imperfectly on subsequent passes. For this reason, lines in spot separations are often widened (spread) slightly to make up for problems in registering subsequent passes through the printer. This
process is called trapping. The **SET_SPREAD** escape implements this process.

**Parameters**

- **hDC**  
  *HDC* Identifies the device context.

- **lpSpread**  
  *LPINT* Points to a short-integer value that specifies the amount, in pixels, by which all nonwhite primitives are to be expanded.

**Return value**

The return value is the previous spread value.

**Comments**

The default spread is zero.

The current spread applies to all bordered primitives (whether or not the border is visible) and text.

---

**STARTDOC**

**Syntax**

```c
short Escape(hDC, STARTDOC, nCount, IpDocName, NULL)
```

This escape informs the device driver that a new print job is starting and that all subsequent **NEWFRAME** escape calls should be spooled under the same job until an **ENDDOC** escape call occurs. This ensures that documents longer than one page will not be interspersed with other jobs.

**Parameters**

- **hDC**  
  *HDC* Identifies the device context.

- **nCount**  
  *short* Specifies the number of characters in the string pointed to by the **IpDocName** parameter.

- **IpDocName**  
  *LPSTR* Points to a null-terminated string that specifies the name of the document. The document name is displayed in the Print Manager window. The maximum length of this string is 31 characters plus the terminating null character.

**Return value**

The return value specifies the outcome of the escape. It is -1 if an error such as insufficient memory or an invalid port specification occurs. Otherwise, it is positive.

**Comments**

The correct sequence of events in a printing operation is as follows:

1. Create the device context.
2. Set the abort function to keep out-of-disk-space errors from terminating a printing operation.
   
   An abort procedure that handles these errors must be set by using the **SETABORTPROC** escape.
3. Begin the printing operation with the **STARTDOC** escape.
4. Begin each new page with the **NEWFRAME** escape, or each new band with the **NEXTBAND** escape.

5. End the printing operation with the **ENDDOC** escape.

6. Destroy the cancel dialog box, if any.

7. Free the procedure-instance address of the abort function.

If an application encounters a printing error or a canceled print operation, it must not attempt to terminate the operation by using the **Escape** function with either the **ENDDOC** or **ABORTDOC** escape. GDI automatically terminates the operation before returning the error value.

---

**TRANSFORM\_CTM**

**Syntax**

short Escape(hDC, TRANSFORM\_CTM, 36, lpMatrix, NULL)

This escape modifies the current transformation matrix. The current transformation matrix controls the manner in which coordinates are translated, rotated, and scaled by the device. By using matrices, you can combine these operations in any order to produce the desired mapping for a particular picture.

The new current transformation matrix will contain the product of the matrix referenced by the lpMatrix parameter and the previous current transformation matrix (CTM = M \* CTM).

**Parameters**

- **hDC**
  - **HDC** Identifies the device context.

- **lpMatrix**
  - **LPSTR** Points to a 3-by-3 array of 32-bit integer values specifying the new transformation matrix. Entries in the matrix are scaled to represent fixed-point real numbers. Each matrix entry is scaled by 65,536. The high-order word of the entry contains the whole integer portion, and the low-order word contains the fractional portion.

**Return value**

The return value is **TRUE** if the escape was successful and **FALSE** if it was unsuccessful.

**Comments**

When an application modifies the current transformation matrix, it must specify the clipping rectangle by issuing the **SET\_CLIP\_BOX** escape.

Applications should not make any assumptions about the initial value of the current transformation matrix.
The matrix specification used for this escape is based on the Microsoft OS/2 Presentation Manager graphics programming interface (GPI) model, which is an integer-coordinate system similar to the one used by GDI.
Windows DDE protocol definition

The Microsoft Windows Dynamic Data Exchange (DDE) protocol defines the method for communicating among applications. This communication takes place as applications send messages to each other to initiate conversations, to request and share data, and to terminate conversations. This chapter describes these messages and the rules associated with their use. It also briefly describes several clipboard formats which a DDE application can register for use in a DDE conversation.

Guide to Programming provides an overview of DDE programming, including such concepts as client, server, application, topic and item. It also introduces the modes of DDE communication, including permanent data links, one-time transfers, and remote command execution, and it explains the flow of DDE messages.

Message-specific argument names bear prefixes indicating their type, as follows:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>An atom of word length (16 bits); for example, aName.</td>
</tr>
<tr>
<td>cf</td>
<td>A registered clipboard format number (word length); for example, cFormat.</td>
</tr>
<tr>
<td>f</td>
<td>A flag bit; for example, fName.</td>
</tr>
<tr>
<td>h</td>
<td>A handle (word length) to a global memory object; for example, hName.</td>
</tr>
<tr>
<td>w</td>
<td>Any other word-length argument; for example, wName.</td>
</tr>
</tbody>
</table>
Using the DDE message set

Each DDE message has two parameters. The first parameter, \textit{wParam} (word length), carries the handle of the sender's window; it is the same in all cases and so is not shown in Table 13.1. The second parameter, \textit{IParam} (a long word, 32 bits), is composed of a low-order word and a high-order word containing message-specific arguments, as follows:

<table>
<thead>
<tr>
<th>Message</th>
<th>\textbf{Arguments in IParam}</th>
<th>\textbf{Low-order word}</th>
<th>\textbf{High-order word}</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM_DDE_ACK</td>
<td></td>
<td>\textit{aApplication}</td>
<td>\textit{aTopic}</td>
</tr>
<tr>
<td>In reply to INITIATE</td>
<td></td>
<td>\textit{wStatus}</td>
<td>\textit{hCommands}</td>
</tr>
<tr>
<td>In reply to EXECUTE</td>
<td></td>
<td>\textit{wStatus}</td>
<td>\textit{alttem}</td>
</tr>
<tr>
<td>All other messages</td>
<td></td>
<td>\textit{wStatus}</td>
<td>\textit{alttem}</td>
</tr>
<tr>
<td>WM_DDE_ADVISE</td>
<td>\textit{hOptions}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WM_DDE_DATA</td>
<td>\textit{hData}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WM_DDE_EXECUTE</td>
<td>(Reserved)</td>
<td>\textit{hCommands}</td>
<td></td>
</tr>
<tr>
<td>WM_DDE_INITIATE</td>
<td>\textit{aApplication}</td>
<td>\textit{aTopic}</td>
<td></td>
</tr>
<tr>
<td>WM_DDE_POKE</td>
<td>\textit{hData}</td>
<td>\textit{alttem}</td>
<td></td>
</tr>
<tr>
<td>WM_DDE_REQUEST</td>
<td>\textit{cfFormat}</td>
<td></td>
<td>\textit{alttem}</td>
</tr>
<tr>
<td>WM_DDE_TERMINATE</td>
<td>(Reserved)</td>
<td>(Reserved)</td>
<td>\textit{alttem}</td>
</tr>
<tr>
<td>WM_DDE_UNADVISE</td>
<td>(Reserved)</td>
<td>(Reserved)</td>
<td>\textit{alttem}</td>
</tr>
</tbody>
</table>

An application calls the \texttt{SendMessage} function to issue the WM\_DDE\_INITIATE message or a WM\_DDE\_ACK message sent in response to WM\_DDE\_INITIATE. All other messages are sent using the \texttt{PostMessage} function. The window handle of the receiving window appears as the first parameter of these calls. The second parameter contains the message to be sent, the third parameter identifies the sending window, and the fourth parameter contains the message-specific arguments. For example:

\begin{verbatim}
PostMessage(hwndRecipient, WM_DDE_MESSAGE, hwndSender,
            MAKELONG(low_word, high_word))
\end{verbatim}

The \texttt{MAKELONG} macro combines low\_word and high\_word into a long word.

Synchronizing the DDE conversation

An application window that processes DDE requests from the window of a DDE partner must process them strictly in the order in which they are received from that partner. However, when handling messages from multiple DDE partners, the window does not have to follow this "first in,
first out" rule. In other words, only the conversations themselves must be synchronous; the window can shift from one conversation to another asynchronously.

For example, suppose the following messages are in a window's queue:

Message from window X
Message from window Y
Message from window X

The window must process message 1 before message 3, but it need not process message 2 before message 3. If window Y is a lower-priority DDE-conversation partner than window X, the window can defer processing the messages from window Y until it has finished dealing with the messages sent by window X. The following table shows acceptable processing orders for these messages and the relative priority implied by each order:

<table>
<thead>
<tr>
<th>Order</th>
<th>Relative Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>Window X = window Y</td>
</tr>
<tr>
<td>1 3 2</td>
<td>Window X &gt; window Y</td>
</tr>
<tr>
<td>2 1 3</td>
<td>Window X &lt; window Y</td>
</tr>
</tbody>
</table>

If an application is unable to process an incoming request because it is waiting for a DDE response, it must post a WM_DDE_ACK message with the fBusy flag set to 1 to prevent deadlock. An application can also send a busy WM_DDE_ACK message if for any reason the application cannot process an incoming request within a reasonable amount of time.

An application should be able to deal with the situation in which its DDE partner fails to respond with a message within a certain time-out interval. Since the length of this interval may vary depending on the nature of the application and the configuration of the user's system (including whether it is on a network), the application should provide a way for the user to specify the time-out interval.

Using atoms

Certain arguments of DDE messages (aItem, aTopic, and aApplication) are global atoms. Applications using these atoms must explicitly delete them to purge them from the atom list.

In all cases, the sender of a message must delete any atom which the intended receiver will not receive due to an error condition, such as failure of the PostMessage function.
Using shared memory objects

DDE uses shared memory objects for three purposes:

- To carry a data item value to be exchanged. This is an item referenced by the hData argument in the WM_DDE_DATA and WM_DDE_POKE messages.
- To carry options in a message. This is an item referenced by the hOptions argument in a WM_DDE_ADVISE message.
- To carry an execution-command string. This is an item referenced by the hCommands argument in the WM_DDE_EXECUTE message and its corresponding WM_DDE_ACK message.

Applications that receive a DDE shared memory object must treat it as read only. It must not be used as a mutual read/write area for the free exchange of data.

As with a DDE atom, a shared memory object should be freed properly to provide for effective memory management. Shared memory objects should be properly locked and unlocked.

In all cases, the sender of a message must delete any shared memory object which the intended receiver will not receive due to an error condition, such as failure of the PostMessage function.

Using clipboard formats

You can pass data by means of any of the standard clipboard formats or with a registered clipboard formats. See the description of the SetClipboardData function in Chapter 4, "Functions directory," in Reference, Volume 1, for more information on standard clipboards. See the description of the RegisterClipboardFormat function for information on registering clipboard formats.

A special, registered format named Link is used to identify an item in a DDE conversation. For more information, see Guide to Programming.

Using the System topic

Applications are encouraged to support at all times a special topic with the name System. This topic provides a context for items of information that may be of general interest to another application.
The following list contains suggested items for the System topic. This list is not exclusive. The data item values should be rendered in the CF_TEXT format. Individual elements of a System topic item value should be delimited by tab characters.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysItems</td>
<td>A list of the System-topic items supported by the application.</td>
</tr>
<tr>
<td>Topics</td>
<td>A list of the topics supported by the application at the current time; this list can vary from moment to moment.</td>
</tr>
<tr>
<td>ReturnMessage</td>
<td>Supporting detail for the most recently used WM_DDE_ACK message. This is useful when more than eight bits of application-specific return data are required.</td>
</tr>
<tr>
<td>Status</td>
<td>An indication of the current status of the application. When a server receives a WM_DDE_REQUEST message for this System-topic item, it should respond by posting a WM_DDE_DATA message with a string containing either &quot;Busy&quot; or &quot;Ready,&quot; as appropriate.</td>
</tr>
<tr>
<td>Formats</td>
<td>A list of clipboard format numbers that the application can render.</td>
</tr>
</tbody>
</table>

**DDE message directory**

This section describes the nine DDE messages. Included in each description is a list of the message-specific arguments and the rules for posting and receiving each message. The SDK contains the DDE.H header file, which defines the DDE messages and data structures described in this section.

**WM_DDE_ACK**

This message notifies an application of the receipt and processing of a WM_DDE_INITIATE, WM_DDE_EXECUTE, WM_DDE_DATA, WM_DDE_ADVISE, WM_DDE_UNADVISE, or WM_DDE_POKE message, and in some cases, of a WM_DDE_REQUEST message.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wParam</td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td>lParam</td>
<td>The meaning of the low-order and high-order words depends on the message to which the WM_DDE_ACK message is responding. When responding to WM_DDE_INITIATE:</td>
</tr>
</tbody>
</table>

Chapter 13, Windows DDE protocol definition
**Argument**  
*aApplication*  
**Description**  
Low-order word of wParam. An atom that contains the name of the replying application.

*aTopic*  
**Description**  
High-order word of wParam. An atom that contains the topic with which the replying server window is associated.

When responding to WM_DDE_EXECUTE:

**Argument**  
*wStatus*  
**Description**  
Low-order word of wParam. A series of flags that indicate the status of the response.

*hCommands*  
**Description**  
High-order word of wParam. A handle that identifies the data item containing the command string.

When replying to all other messages:

**Argument**  
*wStatus*  
**Description**  
Low-order word of wParam. A series of flags that indicate the status of the response.

*altem*  
**Description**  
High-order word of wParam. An atom that specifies the data item for which the response is sent.

---

**Comments**  
The *wStatus* word consists of a **DDEACK** data structure that contains the following information:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>fAck</td>
<td>1 = Request accepted. 0 = Request not accepted.</td>
</tr>
<tr>
<td>14</td>
<td>fBusy</td>
<td>1 = Busy. An application is expected to set fBusy if it is unable to respond to the request at the time it is received. The fBusy flag is defined only when fAck is zero. 0 = Not busy.</td>
</tr>
<tr>
<td>13–8</td>
<td></td>
<td>Reserved for Microsoft use.</td>
</tr>
<tr>
<td>7–0</td>
<td>bAppReturnCode</td>
<td>Reserved for application-specific return codes.</td>
</tr>
</tbody>
</table>

**Posting**  
Except in response to the WM_DDE_INITIATE message, post the WM_DDE_ACK message by calling the **PostMessage** function, not **SendMessage**. When responding to WM_DDE_INITIATE, send the WM_DDE_ACK message with **SendMessage**.

When acknowledging any message with an accompanying *altem* atom, the application that sends WM_DDE_ACK can reuse the *altem* atom that accompanied the original message, or it may delete it and create a new one.
When acknowledging WM_DDE_EXECUTE, the application that sends WM_DDE_ACK should reuse the `hCommands` object that accompanied the original WM_DDE_EXECUTE message.

If an application has initiated the termination of a conversation by sending WM_DDE_TERMINATE and is awaiting confirmation, the waiting application should not acknowledge (positively or negatively) any subsequent message sent by the other application. The waiting application should delete any atoms or shared memory objects received in these intervening messages.

**Receiving**

The application that receives WM_DDE_ACK should delete all atoms accompanying the message.

If the application receives WM_DDE_ACK in response to a message with an accompanying `hData` object, the application should delete the `hData` object.

If the application receives a negative WM_DDE_ACK message sent in reply to a WM_DDE_ADVISE message, the application should delete the `hOptions` object sent with the original WM_DDE_ADVISE message.

If the application receives a negative WM_DDE_ACK message sent in reply to a WM_DDE_EXECUTE message, the application should delete the `hCommands` object sent with the original WM_DDE_EXECUTE message.

**WM_DDE_ADVISE**

This message, posted by a client application, requests the receiving (server) application to supply an update for a data item whenever it changes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wParam</code></td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td><code>lParam</code></td>
<td>Identifies the requested data and specifies how the data is to be sent.</td>
</tr>
</tbody>
</table>

**Argument**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hOptions</code></td>
<td>Low-order word of <code>lParam</code>. A handle to a global memory object that specifies how the data is to be sent.</td>
</tr>
<tr>
<td><code>aItem</code></td>
<td>High-order word of <code>lParam</code>. An atom that specifies the data item being requested.</td>
</tr>
</tbody>
</table>

**Comments**

The global memory object identified by `hOptions` consists of a `DDEADVISE` data structure that contains the following:
<table>
<thead>
<tr>
<th>Word</th>
<th>Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fAckReq</td>
<td>If bit 15 is 1, the receiving (server) application is requested to send its WM_DDE_DATA messages with the ACK-requested bit (fAckReq) set. This offers a flow-control technique whereby the client application can avoid overload from incoming DATA messages.</td>
</tr>
<tr>
<td></td>
<td>fDeferUpd</td>
<td>If bit 14 is 1, the server is requested to send its WM_DDE_DATA messages with a null hData handle. These messages are alarms telling the client that the source data has changed. Upon receiving one of these alarms, the client can choose to call for the latest version of the data by issuing a WM_DDE_REQUEST message, or it can choose to ignore the alarm altogether. This would typically be used when there is a substantial resource cost associated with rendering and/or assimilating the data.</td>
</tr>
<tr>
<td></td>
<td>reserved</td>
<td>Bits 13–0 are reserved.</td>
</tr>
<tr>
<td>2</td>
<td>cfFormat</td>
<td>The client's preferred type of data. Must be a standard or registered clipboard data format number.</td>
</tr>
</tbody>
</table>

If an application supports more than one clipboard format for a single topic and item, it can post multiple WM_DDE_ADVISE messages for the topic and item, specifying a different clipboard format with each message.

**Posting**

Post the WM_DDE_ADVISE message by calling the `SendMessage` function, not `PostMessage`.

Allocate `hOptions` by calling the `GlobalAlloc` function with the GEMEM_DDE_SHARE option.

Allocate `altItem` by calling the `GlobalAddAtom` function.

If the receiving (server) application responds with a negative WM_DDE_ACK message, the sending (client) application must delete the `hOptions` object.

**Receiving**

Post the WM_DDE_ACK message to respond positively or negatively. When posting WM_DDE_ACK, reuse the `altItem` atom or delete it and create a new one. If the WM_DDE_ACK message is positive, delete the `hOptions` object; otherwise, do not delete the object.
This message, posted by a server application, sends a data item value to the receiving (client) application, or notifies it of the availability of data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wParam</td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td>lParam</td>
<td>Identifies the available data and specifies how it is sent.</td>
</tr>
</tbody>
</table>

**Argument**

<table>
<thead>
<tr>
<th>hData</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low-order word of lParam. A handle that identifies the global memory object containing the data and additional information. The handle should be set to NULL if the server is notifying the client that the data item value has changed during a &quot;warm link.&quot; A warm link is established by the client sending a WM_DDE_ADVISE message with the fDeferUpd bit set.</td>
</tr>
</tbody>
</table>

| atom   | High-order word of lParam. An atom that identifies the data item for which data or notification is sent. |

**Comments**

The global memory object identified by hData consists of a DDedata data structure that contains the following:

<table>
<thead>
<tr>
<th>Word</th>
<th>Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fAckReq</td>
<td>If bit 15 is 1, the receiving (client) application is expected to send a WM_DDE_ACK message after the WM_DDE_DATA message has been processed. If bit 15 is zero, the client application should not send a WM_DDE_ACK message. Bit 14 is reserved.</td>
</tr>
<tr>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fRelease</td>
<td>If bit 13 is 1, the client application is expected to free the hData memory object after processing it. If bit 13 is zero, the client application should not free the object. See the &quot;Posting&quot; and &quot;Receiving&quot; sections for exceptions.</td>
</tr>
<tr>
<td></td>
<td>fRequested</td>
<td>If bit 12 is 1, this data is offered in response to a WM_DDE_REQUEST message. If bit 12 is zero, this data is offered in response to a WM_DDE_ADVISE message.</td>
</tr>
<tr>
<td></td>
<td>reserved</td>
<td>Bits 11–0 are reserved.</td>
</tr>
<tr>
<td>2</td>
<td>cfFormat</td>
<td>This specifies the format in which the data are sent or offered to the client application. It must be a standard or registered clipboard data format.</td>
</tr>
<tr>
<td>3–n</td>
<td>Value[ ]</td>
<td>This is the data. It is in the format specified by cfFormat.</td>
</tr>
</tbody>
</table>
**Posting**

Post the WM_DDE_DATA message by calling the **PostMessage** function, not **SendMessage**.

Allocate `hData` by calling the **GlobalAlloc** function with the `GMEM_DDESHARE` option.

Allocate `altem` by calling the **GlobalAddAtom** function.

If the receiving (client) application responds with a negative WM_DDE_ACK message, the sending (server) application must delete the `hData` object.

If the sending (server) application sets the `fRelease` flag to zero, the sender is responsible for deleting `hData` upon receipt of either a positive or negative acknowledgement.

Do not set both the `fAckReq` and `fRelease` flags to zero. If both flags are set to zero, it is difficult for the sending (server) application to determine when to delete `hData`.

**Receiving**

If `fAckReq` is 1, post the WM_DDE_ACK message to respond positively or negatively. When posting WM_DDE_ACK, reuse the `altem` atom or delete it and create a new one.

If `fAckReq` is zero, delete the `altem` atom.

If the sending (server) application specified `hData` as NULL, the receiving (client) application can request the server to send the actual data by posting a WM_DDE_REQUEST message.

After processing the WM_DDE_DATA message in which `hData` is not NULL, delete `hData` unless either of the following conditions is true:

- The `fRelease` flag is zero.
- The `fRelease` flag is 1, but the receiving (client) application responds with a negative WM_DDE_ACK message.

---

**WM_DDE_EXECUTE**

This message, posted by a client application, sends a string to a server application to be processed as a series of commands. The server application is expected to post a WM_DDE_ACK message in response.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wParam</code></td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td><code>lParam</code></td>
<td>Specifies the commands to be executed.</td>
</tr>
</tbody>
</table>
**WM_DDE_EXECUTE**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>The low-order word of lParam is reserved.</td>
</tr>
<tr>
<td>hCommands</td>
<td>High-order word of lParam. A handle that identifies a global memory object containing the command(s) to be executed.</td>
</tr>
</tbody>
</table>

**Comments**

The command string is null-terminated. The command string should adhere to the syntax shown below. Optional syntax elements are enclosed in double brackets ([[]]); single brackets ([ ]) are a syntax element.

```
[opcodestring] [[opcodestring]] ...
```

The `opcodestring` uses the following syntax:

```
opcode[[ (parameter [[ ,parameter ]] ... ) ]] 
```

The `opcode` is any application-defined single token. It may not include spaces, commas, parentheses, or quotation marks.

The `parameter` is any application-defined value. Multiple parameters are separated by commas, and the entire parameter list is enclosed in parentheses. The parameter may not include commas or parentheses except inside a quoted string. If a bracket or parenthesis character is to appear in a quoted string, it must be doubled: `{`.

The following examples show valid command strings:

```
[connect] [download(query1, results.txt)] [disconnect]
[query("sales per employee for each district")] 
[open("sample.xlm")][run("rlc1")]
```

**Posting**

Post the WM_DDE_EXECUTE message by calling the `PostMessage` function, not `SendMessage`.

Allocate `hCommands` by calling the `GlobalAlloc` function with the GMEM_DDE_SHARE option.

When processing WM_DDE_ACK sent in reply to WM_DDE_EXECUTE, the sender of the original WM_DDE_EXECUTE message must delete the `hCommands` object sent back in the WM_DDE_ACK message.

**Receiving**

Post the WM_DDE_ACK message to respond positively or negatively, reusing the `hCommands` object.
WM_DDE_INITIATE

This message, sent by either a client or server application, initiates a conversation with applications responding to the specified application and topic names.

Upon receiving this message, all applications with names that match the \textit{aApplication} application and that support the \textit{aTopic} topic are expected to acknowledge it (see the WM_DDE_ACK message).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wParam</td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td>lParam</td>
<td>Specifies the target application and the topic.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>aApplication</td>
<td>Low-order word of lParam. An atom that specifies the name of the application with which a conversation is requested. The application name may not contain slashes or backslashes. These characters are reserved for future use in network implementations. If the application name is NULL, a conversation with all applications is requested.</td>
</tr>
<tr>
<td>aTopic</td>
<td>High-order word of lParam. An atom that specifies the topic for which a conversation is requested. If the topic is NULL, a conversation for all available topics is requested.</td>
</tr>
</tbody>
</table>

Comments

If the \textit{aApplication} argument is NULL, any application may respond. If the \textit{aTopic} argument is NULL, any topic is valid. Upon receiving a WM_DDE_INITIATE request with a null topic, an application is expected to send a WM_DDE_ACK message for each of the topics it supports.

Sending

Send the WM_DDE_INITIATE message by calling the \texttt{SendMessage} function, not the \texttt{PostMessage} function. Broadcast the message to all windows by setting the first parameter of \texttt{SendMessage} to -1, as shown:

\begin{verbatim}
SendMessage(-1,WM_DDE_INITIATE,hwndClient,MAKELONG(aApp,aTopic));
\end{verbatim}

If the application has already obtained the window handle of the desired server, it can send WM_DDE_INITIATE directly to the server window by passing the server's window handle as the first parameter of \texttt{SendMessage}.

Allocate \textit{aApplication} and \textit{aTopic} by calling \texttt{GlobalAddAtom}.

When \texttt{SendMessage} returns, delete the \textit{aApplication} and \textit{aTopic} atoms.
Receiving  To complete the initiation of a conversation, respond with one or more
WM_DDE_ACK messages, where each message is for a separate topic. When sending
WM_DDE_ACK message, create new aApplication and aTopic atoms; do not reuse the atoms sent with the WM_DDE_INITIATE message.

WM_DDE_POKE

This message, posted by a client application, requests the receiving
(server) application to accept an unsolicited data item value.

The receiving application is expected to reply with a positive
WM_DDE_ACK message if it accepts the data, or with a negative
WM_DDE_ACK message if it does not.

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wParam</td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td>lParam</td>
<td>Identifies the data and specifies how it is sent.</td>
</tr>
</tbody>
</table>

Argument Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hData</td>
<td>Low-order word of lParam. A handle that specifies the global memory object containing the data and other information.</td>
</tr>
<tr>
<td>aItem</td>
<td>High-order word of lParam. An atom that identifies the data item offered to the server application.</td>
</tr>
</tbody>
</table>

Comments

The global memory object identified by hData consists of a DDEPOKE data structure that contains the following:

<table>
<thead>
<tr>
<th>Word</th>
<th>Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved</td>
<td>Bits 15–14 are reserved. If bit 13 is 1, the receiving (server) application is expected to free the memory object after processing it. If bit 13 is zero, the receiving application should not free the object. See the following &quot;Posting&quot; and &quot;Receiving&quot; sections for exceptions. Bits 12–0 are reserved.</td>
</tr>
<tr>
<td>2</td>
<td>cfFormat</td>
<td>This specifies the client's preferred type of data. It must be a standard or registered clipboard data format.</td>
</tr>
<tr>
<td>3–n</td>
<td>Value[ ]</td>
<td>This is the data. It is in the format specified by cfFormat.</td>
</tr>
</tbody>
</table>
WM_DDE_POKE

Posting

Post the WM_DDE_POKE message by calling the **PostMessage** function, not **SendMessage**.

Allocate \( hData \) by calling the **GlobalAlloc** function with the \( \text{GMEM_DDESHARE} \) option.

Allocate \( altern \) by calling the **GlobalAddAtom** function.

If the receiving (server) application responds with a negative WM_DDE_ACK message, the sending (client) application must delete the \( hData \) object.

If the sending (client) application sets the \( fRelease \) flag to zero, the sending application must delete \( hData \) upon receiving either a positive or negative WM_DDE_ACK message.

Receiving

Post the WM_DDE_ACK message to respond positively or negatively. When posting WM_DDE_ACK, reuse the \( altern \) atom or delete it and create a new one.

After processing the WM_DDE_POKE message, delete \( hData \) unless either of the following conditions is true:

- The \( fRelease \) flag is zero.
- The \( fRelease \) flag is 1, but the receiving (server) application responds with a negative WM_DDE_ACK message.

WM_DDE_REQUEST

This message, posted by a client application, requests the receiving (server) application to provide the value of a data item.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( wParam )</td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td>( lParam )</td>
<td>Specifies the requested data and the clipboard format number for the data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( cfFormat )</td>
<td>Low-order word of ( lParam ). A standard or registered clipboard format number.</td>
</tr>
<tr>
<td>( altern )</td>
<td>High-order word of ( lParam ). An atom that specifies which data item is being requested from the server.</td>
</tr>
</tbody>
</table>

Posting

Post the WM_DDE_REQUEST message by calling the **PostMessage** function, not **SendMessage**.
Allocate `altItem` by calling the `GlobalAddAtom` function.

**Receiving**

If the receiving (server) application can satisfy the request, it responds with a `WM_DDE_DATA` message containing the requested data. Otherwise, it responds with a negative `WM_DDE_ACK` message.

When responding with either a `WM_DDE_DATA` or `WM_DDE_ACK` message, reuse the `altItem` atom or delete it and create a new one.

---

**WM_DDE_TERMINATE**

This message, posted by either a client or server application, terminates a conversation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wParam</code></td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td><code>lParam</code></td>
<td>Is reserved.</td>
</tr>
</tbody>
</table>

**Posting**

Post the `WM_DDE_TERMINATE` message by calling the `PostMessage` function, not `SendMessage`.

While waiting for confirmation of the termination, the sending application should not acknowledge any other messages sent by the receiving application. If the sending application receives messages (other than `WM_DDE_TERMINATE`) from the receiving application, it should delete any atoms or shared memory objects accompanying the messages.

**Receiving**

Respond by posting a `WM_DDE_TERMINATE` message.

---

**WM_DDE_UNADVISE**

This message, sent by a client application, informs a server application that the specified item, or a particular clipboard format for the item, should no longer be updated. This terminates the warm or hot link for the specified item.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wParam</code></td>
<td>Identifies the sending window.</td>
</tr>
<tr>
<td><code>lParam</code></td>
<td>Specifies the data-request item to be canceled.</td>
</tr>
</tbody>
</table>

**Argument**

<table>
<thead>
<tr>
<th><code>altItem</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lParam</code></td>
<td>High-order word of <code>lParam</code>. An atom that specifies the data for which the update request is being retracted. When <code>altItem</code> is NULL, all active...</td>
</tr>
</tbody>
</table>
WM_DDE_UNADVISE conversations associated with the client are to be terminated.

**Post**

Post the WM_DDE_UNADVISE message by calling the PostMessage function, not SendMessage.

Allocate *alten* by calling the GlobalAddAtom function.

**Receive**

Post the WM_DDE_ACK message to respond positively or negatively. When posting WM_DDE_ACK, reuse the *alten* atom or delete it and create a new one.
The following table shows the symbolic constant names, hexadecimal values, and descriptive information for Microsoft Windows virtual-key codes. The codes are listed in numeric order.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_LBUTTON</td>
<td>01H</td>
<td>Left mouse button</td>
</tr>
<tr>
<td>VK_RBUTTON</td>
<td>02H</td>
<td>Right mouse button</td>
</tr>
<tr>
<td>VK_CANCEL</td>
<td>03H</td>
<td>Used for control-break processing</td>
</tr>
<tr>
<td>VK_MBUTTON</td>
<td>04H</td>
<td>Middle mouse button (3-button mouse)</td>
</tr>
<tr>
<td></td>
<td>05H–07H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_BACK</td>
<td>08H</td>
<td>BACKSPACE key</td>
</tr>
<tr>
<td>VK_TAB</td>
<td>09H</td>
<td>TAB key</td>
</tr>
<tr>
<td>VK_CLEAR</td>
<td>0CH</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>VK_RETURN</td>
<td>0DH</td>
<td>RETURN key</td>
</tr>
<tr>
<td>VK_SHIFT</td>
<td>10H</td>
<td>SHIFT key</td>
</tr>
<tr>
<td>VK_CONTROL</td>
<td>11H</td>
<td>CONTROL key</td>
</tr>
<tr>
<td>VK_MENU</td>
<td>12H</td>
<td>MENU key</td>
</tr>
<tr>
<td>VK_PAUSE</td>
<td>13H</td>
<td>PAUSE key</td>
</tr>
<tr>
<td>VK_CAPITAL</td>
<td>14H</td>
<td>CAPITAL key</td>
</tr>
<tr>
<td></td>
<td>15H–19H</td>
<td>Reserved for Kanji systems</td>
</tr>
<tr>
<td></td>
<td>1AH</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_ESCAPE</td>
<td>1BH</td>
<td>ESCAPE key</td>
</tr>
<tr>
<td></td>
<td>1CH–1FH</td>
<td>Reserved for Kanji systems</td>
</tr>
<tr>
<td>VK_SPACE</td>
<td>20H</td>
<td>SPACEBAR</td>
</tr>
<tr>
<td>VK_PRIOR</td>
<td>21H</td>
<td>PAGE UP key</td>
</tr>
<tr>
<td>VK_NEXT</td>
<td>22H</td>
<td>PAGE DOWN key</td>
</tr>
<tr>
<td>VK_END</td>
<td>23H</td>
<td>END key</td>
</tr>
<tr>
<td>VK_HOME</td>
<td>24H</td>
<td>HOME key</td>
</tr>
<tr>
<td>VK_LEFT</td>
<td>25H</td>
<td>LEFT ARROW key</td>
</tr>
</tbody>
</table>

Virtual-key codes

Appendix A, Virtual-key codes
<table>
<thead>
<tr>
<th>VK</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_UP</td>
<td>26H</td>
<td>UP ARROW key</td>
</tr>
<tr>
<td>VK_RIGHT</td>
<td>27H</td>
<td>RIGHT ARROW key</td>
</tr>
<tr>
<td>VK_DOWN</td>
<td>28H</td>
<td>DOWN ARROW key</td>
</tr>
<tr>
<td>VK_SELECT</td>
<td>29H</td>
<td>SELECT key</td>
</tr>
<tr>
<td>VK_EXECUTE</td>
<td>2BH</td>
<td>EXECUTE key</td>
</tr>
<tr>
<td>VK_SNAPSHOT</td>
<td>2CH</td>
<td>PRINTSCREEN key for Windows version 3.0 and later</td>
</tr>
<tr>
<td>VK_INSERT</td>
<td>2DH</td>
<td>INSERT key</td>
</tr>
<tr>
<td>VK_DELETE</td>
<td>2EH</td>
<td>DELETE key</td>
</tr>
<tr>
<td>VK_HELP</td>
<td>2FH</td>
<td>HELP key</td>
</tr>
<tr>
<td>VK_0</td>
<td>30H</td>
<td>0 key</td>
</tr>
<tr>
<td>VK_1</td>
<td>31H</td>
<td>1 key</td>
</tr>
<tr>
<td>VK_2</td>
<td>32H</td>
<td>2 key</td>
</tr>
<tr>
<td>VK_3</td>
<td>33H</td>
<td>3 key</td>
</tr>
<tr>
<td>VK_4</td>
<td>34H</td>
<td>4 key</td>
</tr>
<tr>
<td>VK_5</td>
<td>35H</td>
<td>5 key</td>
</tr>
<tr>
<td>VK_6</td>
<td>36H</td>
<td>6 key</td>
</tr>
<tr>
<td>VK_7</td>
<td>37H</td>
<td>7 key</td>
</tr>
<tr>
<td>VK_8</td>
<td>38H</td>
<td>8 key</td>
</tr>
<tr>
<td>VK_9</td>
<td>39H</td>
<td>9 key</td>
</tr>
<tr>
<td>VK_A</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_B</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_C</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_D</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_E</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_F</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_G</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_H</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_I</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_J</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_K</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_L</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_M</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_N</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_O</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_P</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_Q</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_R</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_S</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_T</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_U</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_V</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_W</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_X</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_Y</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_Z</td>
<td>3AH-40H</td>
<td>Undefined</td>
</tr>
<tr>
<td>VK_NUMPAD0</td>
<td>60H</td>
<td>Numeric key pad 0 key</td>
</tr>
<tr>
<td>VK_NUMPAD1</td>
<td>61H</td>
<td>Numeric key pad 1 key</td>
</tr>
<tr>
<td>VK_NUMPAD2</td>
<td>62H</td>
<td>Numeric key pad2 key</td>
</tr>
<tr>
<td>Virtual-key code</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>VK_NUPPAD3</td>
<td>63H</td>
<td>Numeric key pad 3 key</td>
</tr>
<tr>
<td>VK_NUPPAD4</td>
<td>64H</td>
<td>Numeric key pad 4 key</td>
</tr>
<tr>
<td>VK_NUPPAD5</td>
<td>65H</td>
<td>Numeric key pad 5 key</td>
</tr>
<tr>
<td>VK_NUPPAD6</td>
<td>66H</td>
<td>Numeric key pad 6 key</td>
</tr>
<tr>
<td>VK_NUPPAD7</td>
<td>67H</td>
<td>Numeric key pad 7 key</td>
</tr>
<tr>
<td>VK_NUPPAD8</td>
<td>68H</td>
<td>Numeric key pad 8 key</td>
</tr>
<tr>
<td>VK_NUPPAD9</td>
<td>69H</td>
<td>Numeric key pad 9 key</td>
</tr>
<tr>
<td>VK_MULTIPLY</td>
<td>6AH</td>
<td>Multiply key</td>
</tr>
<tr>
<td>VK_ADD</td>
<td>6BH</td>
<td>Add key</td>
</tr>
<tr>
<td>VK_SEPARATOR</td>
<td>6CH</td>
<td>Separator key</td>
</tr>
<tr>
<td>VK_SUBTRACT</td>
<td>6DH</td>
<td>Subtract key</td>
</tr>
<tr>
<td>VK_DECIMAL</td>
<td>6EH</td>
<td>Decimal key</td>
</tr>
<tr>
<td>VK_DIVIDE</td>
<td>6FH</td>
<td>Divide key</td>
</tr>
<tr>
<td>VK_F1</td>
<td>70H</td>
<td>F1 key</td>
</tr>
<tr>
<td>VK_F2</td>
<td>71H</td>
<td>F2 key</td>
</tr>
<tr>
<td>VK_F3</td>
<td>72H</td>
<td>F3 key</td>
</tr>
<tr>
<td>VK_F4</td>
<td>73H</td>
<td>F4 key</td>
</tr>
<tr>
<td>VK_F5</td>
<td>74H</td>
<td>F5 key</td>
</tr>
<tr>
<td>VK_F6</td>
<td>75H</td>
<td>F6 key</td>
</tr>
<tr>
<td>VK_F7</td>
<td>76H</td>
<td>F7 key</td>
</tr>
<tr>
<td>VK_F8</td>
<td>77H</td>
<td>F8 key</td>
</tr>
<tr>
<td>VK_F9</td>
<td>78H</td>
<td>F9 key</td>
</tr>
<tr>
<td>VK_F10</td>
<td>79H</td>
<td>F10 key</td>
</tr>
<tr>
<td>VK_F11</td>
<td>7AH</td>
<td>F11 key</td>
</tr>
<tr>
<td>VK_F12</td>
<td>7BH</td>
<td>F12 key</td>
</tr>
<tr>
<td>VK_F13</td>
<td>7CH</td>
<td>F13 key</td>
</tr>
<tr>
<td>VK_F14</td>
<td>7DH</td>
<td>F14 key</td>
</tr>
<tr>
<td>VK_F15</td>
<td>7EH</td>
<td>F15 key</td>
</tr>
<tr>
<td>VK_F16</td>
<td>7FH</td>
<td>F16 key</td>
</tr>
<tr>
<td>80H–87H</td>
<td></td>
<td>OEM specific</td>
</tr>
<tr>
<td>88H–8FH</td>
<td></td>
<td>Unassigned</td>
</tr>
<tr>
<td>VK_NUMLOCK</td>
<td>90H</td>
<td>NUM LOCK key</td>
</tr>
<tr>
<td>91H</td>
<td></td>
<td>OEM specific</td>
</tr>
<tr>
<td>92H–B9H</td>
<td></td>
<td>Unassigned</td>
</tr>
<tr>
<td>B9H–C0H</td>
<td></td>
<td>OEM specific</td>
</tr>
<tr>
<td>C1H–DAH</td>
<td></td>
<td>Unassigned</td>
</tr>
<tr>
<td>DBH–E4H</td>
<td></td>
<td>OEM specific</td>
</tr>
<tr>
<td>E5H</td>
<td></td>
<td>Unassigned</td>
</tr>
<tr>
<td>E6H</td>
<td></td>
<td>OEM specific</td>
</tr>
<tr>
<td>E7H–E8H</td>
<td></td>
<td>Unassigned</td>
</tr>
<tr>
<td>E9H–F5H</td>
<td></td>
<td>OEM specific</td>
</tr>
<tr>
<td>F6H–FEH</td>
<td></td>
<td>Unassigned</td>
</tr>
</tbody>
</table>
RC diagnostic messages

This appendix contains descriptions of diagnostic messages produced by the Resource Compiler (RC). Many of these messages appear when the Resource Compiler is not able to compile your resources. The descriptions in this appendix can help you determine the problem.

A (V) symbol at the beginning of a message description indicates that the message is displayed only if RC is run with the -V (verbose) option. These messages are generally informational and do not necessarily indicate errors.

The messages are listed in alphabetical order.

**Accelerator Type required (ASCII or VIRTKEY)**

The type field in the ACCELERATORS statement must contain either the ASCII or VIRTKEY value.

**BEGIN expected in Accelerator Table**

The BEGIN keyword must immediately follow the ACCELERATORS keyword.

**BEGIN expected in Dialog**

The BEGIN keyword must immediately follow the DIALOG keyword.

**BEGIN expected in menu**

The BEGIN keyword must immediately follow the MENU keyword.

**BEGIN expected in RCDATA**

The BEGIN keyword must immediately follow the RCDATA keyword.
BEGIN keyword expected in String or Error Table
The BEGIN keyword must immediately follow the STRINGTABLE or ERRTABLE keyword.

Cannot Reuse String Constants
You are using the same value twice in a STRINGTABLE or ERRTABLE statement. Make sure you are not mixing overlapping decimal and hexadecimal values.

Control Character out of range [^A – ^Z]
A control character in the ACCELERATORS statement is invalid. The character following the caret (^) must be between A and Z, inclusive.

copy of temp-file-2 to exe-file failed
The temporary file was not able to create the new .EXE file. Make sure that the TEMP environment variable is pointing to a drive that is not write-protected.

Copying segment id (size bytes)
(V) RC is copying the specified segment to the .EXE file.

Could not find RCPP.EXE
RCPP.ERR must be in the current directory or a directory in the PATH environment.

Could not open in-file-name
RC could not open the specified file. Make sure the file exists and that you typed the filename correctly.

Couldn’t open resource-name
RC could not open the specified file. Make sure the file exists and that you typed the filename correctly.

Couldn’t write executable
The .EXE file could not be copied to the temporary file. Make sure that the TEMP environment variable is pointing to a drive that is not write-protected and that the .EXE file from the linker is correct. You can check the .EXE file with the EXEHDR program.

Creating resource-name
(V) RC is creating a new .RES file.

Empty menus not allowed
An END keyword appears before any menu items are defined in the MENU statement. Empty menus are not permitted by the Resource Compiler. Make sure you do not have any open quotation marks within the MENU statement.
END expected in Dialog
   The END keyword must occur at the end of a DIALOG statement. Make sure there are no open quotes left from the preceding statement.

END expected in menu
   The END keyword must come at the end of a MENU statement. Make sure you do not have any open quotation marks or a mismatched pair of BEGIN and END statements.

Error: Bitmap file resource-file is not in 3.00 format.
   Use SDKPaint to convert version 2.x resource files to the 3.0 format.

Error Creating resource-name
   Could not create specified .RES file. Make sure it is not being created on a read-only drive. Use the -V option to find out whether the file is being created.

Error: I/O error reading file.
   Read failed. Since this is a generic routine, no specific filename is supplied.

Error: I/O error seeking in file
   Seeking in file failed.

Error: I/O error writing file.
   Write failed. Since this is a generic routine, no specific filename is supplied.

Error: Old DIB in resource-name. Pass it through SDKPAINT.
   The resource file specified is not compatible with Windows 3.0. Make sure you have read and saved this file using the latest version of SDKPaint.

Error: Out of memory. Try not using resources with string identifiers.
   There is not enough memory to allocate for a table of string names. You can view these names are when you use the -V option. Try to replace the string names with numbers. For example, you can change

   MYICON ICON myicon.ico

   to

   1 ICON myicon.ico

   or provide the following statement in your header file:

   #define MYICON 1
Error: Resource file resource-name is not in 3.00 format.
Make sure your icons and cursors have been read and saved using the latest version of SDKPaint.

Errors in .EXE file
LINK failed. See the CodeView and Utilities manual in the Microsoft C 5.1 Optimizing Compiler documentation set for more information.

.EXE file too large; relink with higher /ALIGN value
The EXE file is too large. Relink the .EXE file with a larger /ALIGN value. If the .EXE file is larger than 800K, you should use the /ALIGN:32 value on your LINK line.

.EXE not created by LINK
You must create the .EXE file with a version of LINK that is from C version 5.1 or later.

Expected Comma in Accelerator Table
RC requires a comma between the event and idvalue fields in the ACCELERATORS statement.

Expected control class name
The class field of a CONTROL statement in the DIALOG statement must be one of the following types: BUTTON, COMBOBOX, EDIT, LISTBOX, SCROLLBAR, STATIC, or user-defined. Make sure the class is spelled correctly.

Expected font face name
The typeface field of the FONT option in the DIALOG statement must be an ASCII character string enclosed in double quotation marks. This field specifies the name of a font.

Expected ID value for Menuitem
The MENU statement must contain a menuID field, which specifies the name or number that identifies the menu resource.

Expected Menu String
Each MENUITEM and POPUP statement must contain a text field, which is a string enclosed in double quotation marks that specifies the name of the menu item or pop-up menu. A MENUITEM SEPARATOR statement requires no quoted string.

Expected numeric command value
RC was expecting a numeric idvalue field in the ACCELERATORS statement. Make sure you have used a #define constant to specify the value and that the constant is spelled correctly.
Expected numeric constant in string table
A numeric constant, defined in a `#define` statement, must immediately follow the `BEGIN` keyword in a `STRINGTABLE` or `ERRTABLE` statement.

Expected numeric point size
The `pointsize` field of the `FONT` option in the `DIALOG` statement must be an integer point size value.

Expected Numerical Dialog constant
A `DIALOG` statement requires integer values for the `x`, `y`, `width`, and `height` fields. Make sure these values are included after the `DIALOG` keyword and that they are not negative.

Expected String in `STRINGTABLE`/`ERRTABLE`
A string is expected after each `stringid` value in a `STRINGTABLE` or `ERRTABLE` statement.

Expected String or Constant Accelerator command
`RC` was not able to determine what kind of key is being set up for the accelerator. The `event` field in the `ACCELERATORS` statement might be invalid.

Expecting number for ID
Expecting a number for the `id` field of a control statement in the `DIALOG` statement. Make sure you have a number or `#define` statement for the control ID.

Expecting quoted string in dialog class
The `class` field of the `CLASS` option in the `DIALOG` statement must be an integer or a string, enclosed in double quotation marks.

Expecting quoted string in dialog title
The `captiontext` field of the `CAPTION` option in the `DIALOG` statement must be an ASCII character string enclosed in double quotation marks.

File not found: `filename`
The file specified in the `RC` command line was not found. Check to see whether the file has been moved to another directory and whether the `filename` or `pathname` is typed correctly.

Font names must be ordinals
The `pointsize` field in the `FONT` statement must be an integer, not a string.

Gangload area is [size] bytes at offset 0x[address]
(V) This is the size (in bytes) of all the segments that have one of the following attributes:
PRELOAD
DISCARDABLE

- Code segments that contain the entry point, WinMain
- Data segments (which should not be discardable)

The segments are placed in a contiguous area in the .EXE file for fast loading. The offset value is from the the beginning of the file. To disable gangloading, use the -k option.

Insufficient memory to spawn RCPP.EXE
There wasn't enough memory to run the preprocessor (RCPP). You can try not running any memory-resident software that might be taking up too much memory. Use the CHKDSK program to verify the amount of memory you have.

Invalid Accelerator
An event field in the ACCELERATORS statement was not recognized or was more than two characters in length.

Invalid Accelerator Type (ASCII or VIRTKEY)
The type field in the ACCELERATORS statement must contain either the ASCII or VIRTKEY value.

Invalid control character
A control character in the ACCELERATORS statement is invalid. A valid control character consists of one letter (only) following a caret (^).

Invalid Control type
Each control statement in a DIALOG statement must be one of the following: CHECKBOX, COMBOBOX, CONTROL, CTEXT, DEFPUSHBUTTON, EDITTEXT, GROUPBOX, ICON, LISTBOX, LTEXT, PUSHBUTTON, RADIOBUTTON, RTTEXT, SCROLLBAR.

Make sure these control statements are spelled correctly.

Invalid .EXE file
The .EXE file is invalid. Make sure that the linker created it correctly and that the file exists. You can check the .EXE file with the EXEHDR program.

Invalid switch, option
You used an option that was not valid. Use RC -? for a list of the command-line options.

Invalid type
The resource type was not among the types defined in the windows.h file.
Invalid usage. Use rc --? for Help
Make sure you have at least one filename to work with. Use RC --? for a list of the command-line options.

No executable filename specified.
The -FE option was used, but no .EXE filename specified.

No resource binary filename specified.
The -FO option was used, but no .RES filename specified.

Not a Microsoft Windows format .EXE file
Make sure that the linker created the .EXE file correctly and that the file exists. You can check the .EXE file with the EXEHDR program.

Out of far heap memory
There wasn’t enough memory. Try not running any memory-resident software that might be taking up too much space. Use the CHKDSK program to find out how much memory you have.

Out of memory, needed n bytes
RC was not able to allocate the specified amount of memory.

RC: Invalid swap area size: -S string
Invalid swap area size. Check your syntax for the -S option on the RC command line. The following are acceptable command lines:

RC S123
RC S123K ;where K is kilobytes
RC S123p ;where p is paragraphs

RC: Invalid switch: option
You used an option that was not valid. Use RC --? for a list of the command-line options.

RC: RCPP preprocessor-command-string
(V) RC is passing the specified string to the preprocessor.

RC: RCPP.ERR not found
RCPP.ERR must be in the current directory or a directory in the PATH environment.

RC terminated by user
A CONTROL+C key combination was pressed, terminating RC.

RC terminating after preprocessor errors
See the Microsoft C 5.1 Optimizing Compiler documentation for information about preprocessor errors.

RCPP.EXE command line greater than 128 bytes
The command line was too long.
RCPP.EXE is not a valid executable
RCPP.EXE is not valid. The file might have been altered. Try copying the file from the SDK disks.

Reading resource-name
(V) RC is reading the .RES file.

Resources will be aligned on number byte boundaries
(V) The alignment is determined by the ALIGN:number option on the LINK line.

Sorting preload segments and resources into gangload section
(V) RC is sorting the preloaded segments so that they can be loaded quickly.

Text string or ordinal expected in Control
The text field of a CONTROL statement in the DIALOG statement must be either a text string or an ordinal reference to the type of control is expected. If using an ordinal, make sure that you have a #define statement for the control.

The EXETYPE of this program is not Windows
The EXETYPE WINDOWS statement did not appear in the .DEF file. Since the linker might make optimizations for OS/2 (the default EXETYPE) that are not appropriate for Windows, the EXETYPE WINDOWS statement must be specified.

Unable to create destination
RC was not able to create the destination file. Make sure there is enough disk space.

Unable to open exe-file
RC could not open this .EXE file. Make sure that the linker created it correctly and that the file exists.

Unbalanced Parentheses
Make sure you have closed every open parenthesis in the DIALOG statement.

Unexpected value in RCDATA
The raw-data values in the RCDATA statement must be integers or strings, each separated by a comma. Make sure you did not leave out a comma or leave out a quotation mark around a string.

Unknown DIB header format
The bitmap header is not a BITMAPCOREHEADER or BITMAPINFOHEADER structure.
Unknown error spawning RCPP.EXE
For an unknown reason, RCPP was not started. Try copying the file from the SDK disks, and use the CHKDSK program to verify the amount of available memory.

Unknown Menu SubType
The item-definition field of the MENU statement can contain only MENUITEM and POPUP statements.

Warning: ASCII character not equivalent to virtual key code
There is an invalid virtual-key code in the ACCELERATORS statement. The ASCII value for some characters (such as *, ^, &,) is not equivalent to the virtual-key code for the corresponding key. (In the case of the asterisk (*), the virtual-key code is equivalent to the ASCII value for 8, the numeric character on the same key. Therefore the statement

VIRTKEY ' * '

is invalid.) See Appendix A, "Virtual-key codes," and Appendix D, "Character tables," for these values.

Warning: Discardable segment id (hex-size bytes) is excessively large.
The segment is greater than 27FFh in size. RC displays this warning because very large segments can adversely affect memory usage. Check your map file listing for the exact size of your segments.

Warning: SHIFT or CONTROL used without VIRTKEY
The ALT, SHIFT, and CONTROL options apply only to virtual keys in the ACCELERATORS statement. Make sure you have used the VIRTKEY option with one of these other options.

Writing resource resource-name or ordinal-id resource type (resource size)
(V) RC is writing the resource name or ordinal ID, followed by a period and the resource type and size (in bytes).

Warning: string segment number set to PRELOAD
RC displays this warning when it copies a segment that must be preloaded but that is not marked PRELOAD in the linker .DEF file. All nondiscardable segments should be preloaded, including automatic data segments, fixed segments and the entry point of the code (WinMain). The attributes of your code segments are set by the .DEF file. Check your map file listing for more information.
Index

[[ ]] (double brackets)
   as document convention 3
. . . (ellipses)
   as document convention 3
{} (curly braces)
   as document convention 3
() (parentheses)
   as document convention 3
^ (caret)
   as document convention 3
& (ampersand)
   80, 81, 82, 83, 84, 86, 87, 88
| (vertical bar)
   as document convention 3
\bc169\ec \bc170\ec (quotation marks)
   as document convention[(quotation marks),
   as document convention 3
\bcB\ec Bold text \bcD\ec
   as document convention 2
\bcF105M\ec Monospaced type \bcF255D\ec
   as document convention 3
\bcMI\ec Italic text \bcD\ec
   as document convention 3
\bcS\ec BACKSPACE \bcD\ec key 95
\bcS\ec CONTROL \bcD\ec key 68
\bcS\ec SHIFT \bcD\ec key 68
\bcS\ec TAB \bcD\ec key 77
#define directive
   [define directive] resource compiler 103
   resource compiler 103
#elif directive
   [elif directive] resource compiler 106
#else directive
   [else directive] resource compiler 106
#endif directive
   [endif directive] resource compiler 107
#if directive
   [if directive] resource compiler 105
   [if directive] resource compiler 105
#endif directive
   [ifdef directive] resource compiler 104
   resource compiler 104
#else directive
   [ifndef directive] resource compiler 105
   resource compiler 105
#else directive
   [ifndef directive] resource compiler 103
   resource compiler 103
#undef directive
   [undef directive] resource compiler 104
   resource compiler 104

A
ABORTDOC printer escape 153
ACCELERATORS resource statement 67
Addition (+) operator 80, 81, 82, 83, 85, 86, 87, 88, 89, 90, 91, 92

B
BANDINFO printer escape 154
BEGIN_PATH printer escape 156
BITMAP data structure 10
BITMAP resource compiler key word 62
BITMAPCOREHEADER data structure 11, 13
BITMAPCOREINFO data structure 12
BITMAPFILEHEADER data structure 14
BITMAPINFO data structure 14, 17
BITMAPINFOHEADER data structure 15, 16
BOOL data type 7
Border
   window 76
Braces
   curly ({})
   as document convention 3
Brackets
   double ([ ])
      as document convention 3
BS_3STATE control style 96
BS_AUTO3STATE control style 96
BS_AUTOCHECKBOX control style 96
BS_AUTORADIOBUTTON control style 96
BS_CHECKBOX control style 96
BS_DEF_PUSHBUTTON control style 96
BS_GROUPBOX control style 96
BS_HATCHED brush style 38, 39
BS_HOLLOW brush style 39
BS_LEFTTEXT control style 96
BS_OWNедерDRAW control style 96
BS_PATTERN brush style 39
BS_PUSHBUTTON control style 96
BS_RADIOBUTTON control style 96
BS_SOLID brush style 39
Button
   owner-draw 36, 46
BUTTON control class 83, 85, 86, 95
BYTE data type 7

C
Capital letters
   small
   as document convention 3
CAPTION resource statement 77
Caret (^) 68, 95
Caret (\bc94\ec) 67
Carriage-return character 98
CB_ADDSTRING message 37, 47
CB_INSERTSTRING message 37, 47
CBS_AUTOHSCROLL control style 97
CBS_DROPDOWN control style 97
CBS_DROPDOWNLIST control style 97
CBS_HASSTRINGS control style 97
CBS_OEMCONVERT control style 97
CBS_OWNедерDRAWFIXED control style 97
CBS_OWNедерDRAWVARIABLE control style 97
CBS_SIMPLE control style 97
CBS_SORT control style 97
char data type 7
CHECKBOX resource statement 83, 84
CLASS resource statement 78
CLIENTCREATESTRUCT data structure 20
CLIP_TO_PATH printer escape 157
Clipping
   child window 76
CODE module-definition statement 134
Code segment attributes
   defining 134, 140
CODE statement 133
COLORREF data type 20
Combo box
   owner-draw 28
      sorting owner-draw 22
COMBOBOX control class 91, 95
COMBOBOX resource statement 91, 92
Communication devices 23, 25
COMPAREITEMSTRUCT data structure 22
COMSTAT data structure 23
CONTROL resource statement 94
Control window
   user-defined 94
CREATESTRUCT data structure 24
Creating windows 75
CS_BYTEALIGNCLIENT window class style 58
CS_BYTEALIGNWINDOW window class style 58
CS_CLASSDC window class style 58
CS_OWNедерDC window class style 59
CS_OWNедерDC window class style 59
CS_PARENTDC window class style 59
CS_SAVEBITS window class style 59
CS_VREDRAW window class style 59
CTEXT resource statement 82
Curly braces ({} )
   as document convention 3
CURSOR resource-compiler key word 62
CW_USEDEFAULT default window width 45

D
DATA module-definition statement 135
Data segment attributes
   defining 135, 140
DATA statement 133
Data types
   naming conventions 7, 9
DCB data structure 25
Default pushbutton control 87
DefDlgProc function 78
DEFPUSHBUTTON resource statement 87, 88
DESCRIPTION module-definition statement 135
DESCRIPTION statement 133
DEVICEDATA printer escape 158
Devices
communication 23, 25
DEVMODE data structure 29
Dialog box units 80, 81, 82, 83, 84, 85, 87, 88, 89, 90, 91
Dialog option statements 75
DIALOG resource statement 73
DIALOG template 73
DialogBox function 74
Disabled window 74
DISCARDABLE resource-compiler key word 62, 64, 65, 66, 69, 74
DLGITEMTEMPLATE data structure 34
DLGTEMPLATE data structure 32
Double brackets ([[]]) as document convention 3
Double quotation marks ("bc169\ec\bc170\ec) 66, 67, 70, 78
Double quotation marks (\bc170\ec\bc169\ec) 70
DRAFT_QUALITY font quality 41
DRAFTMODE printer escape 158
DRAWPATTERNRECT printer escape 158
Driver
printer initialization 29
DS_ABSALIGN dialog-box style 33, 74
DS_LOCALEEDIT dialog-box style 33, 76
DS_MODALFRAME dialog-box style 33, 76
DS_NOIDLEMSG dialog-box style 33, 76
DS_SETFONT dialog-box style 33
DS_SYSMODAL dialog-box style 33, 76
DWORD data type 8

E
Edit control 90, 98
EDIT control class 95
Editing keys 90
EDITTEXT resource statement 89, 90
EM_SETPASSWORDCHAR message 98
ENABLEDUPLEX printer escape 160
ENABLEPAIRKERNING printer escape 160
ENABLERELATIVEWIDTHS printer escape 161
END_PATH printer escape 162
ENDDOC printer escape 162
ENUMPAPERBINS printer escape 164
ENUMPAPERMETRICS printer escape 165
EPSPRINTING printer escape 166
ES_AUTOHSCROLL control style 99
ES_AUTOVSCROLL control style 98
ES_CENTER control style 97
ES_LEFT control style 97
ES_LOWER CASE control style 98
ES_MULTILINE control style 98
ES_NOHIDESEL control style 99
ES_OEMCONVERT control style 99
ES_PASSWORD control style 98
ES_RIGHT control style 97
ES_UPPERCASE control style 98
EVENPARITY parity type 26
EXETYPE module-definition statement 136
EXPORTS module-definition statement 136
EXPORTS statement 133
EXT_DEVICE_CAPS printer escape 166
EXTTEXTOUT printer escape 168

F
FAR data type 8
FARPROC data type 8
FF_DECORATIVE font family 42
FF_DONTCARE font family 42
FF_MODERN font family 42
FF_ROMAN font family 42
FF_SCRIPT font family 42
FF_SWISS font family 43
FIXED resource-compiler key word 62, 63, 65, 66, 69, 73
FLUSHOUTPUT printer escape 169
FONT resource-compiler key word 62
FONT resource statement 79
FONTINFO data structure 34

G
GETCOLORTABLE printer escape 169
Local stack 141
LOCALHANDLE data type 8
LOGBRUSH data structure 38
LOGFONT data structure 40
LOGPALETTE data structure 43, 52
LOGPEN data structure 44
LONG data type 8
long data type 8
LPBITMAP data type 8
LPBITMAPCOREHEADER data type 8
LPBITMAPCOREINFO data type 8
LPBITMAPFILEHEADER data type 8
LPBITMAPINFO data type 8
LPBITMAPINFOHEADER data type 8
LPCOMPAREITEMSTRUCT data type 8
LPCREATESTRUCT data type 8
LPDELETEITEMSTRUCT data type 9
LPDRAWITEMSTRUCT data type 9
LPHANDLETABLE data type 9
LPINT data type 9
LPLOGBRUSH data type 9
LPLOGFONT data type 9
LPLOGPALETTE data type 9
LPLOGPEN data type 9
LPMETRIESTRUCT data type 9
LPMETAFILEPICT data type 9
LPMENUITEMTEMPLATE data structure 47
Metafile picture format 49
METAFILEPICT data structure 49
MF_CHECKED menu option 48
MF_END menu option 48
MF_HELP menu-item option 48
MF_MENUBARBREAK menu-item option 48
MF_MENUBREAK menu-item option 48
MF_OWNERDRAW menu-item option 48
MF_POPUP menu-item option 48
MFCOMMENT printer escape 183
MIDCREATESTRUCT menu flag 46
Minimize box 76
Mnemonic 70, 80, 81, 82, 83, 84, 86, 87, 88
MOVEABLE resource-compiler key word 62, 64, 65, 66, 69, 73
MSG data structure 50
MULTIKEYHELP data structure 50
Multiple-line edit control 98

N
n\& (ampersand)\#ampersand]
use in MENUITEM statement 70
n\a See Escape character
n.DEF file See Module-definition file
n#include directive[include directive]
when required with STYLE statement 75
n\\ See Escape character
nAccelerator See ACCELERATORS resource statement
NAME module-definition statement 139
NAME statement 133
nAmpersand (&)
adding a mnemonic with 70, 80, 81, 82, 83, 84, 86, 87, 88
nASCII character
use with ACCELERATORS statement 67
nBitmap
device-independent
BITMAPCOREHEADER data structure 12
BITMAPCOREINFO data structure 13
BITMAPINFO data structure 15
BITMAPINFOHEADER data structure 17
color 55
described 12, 13, 15, 16

M
MakeProcInstance function 8
MARKPARITY parity type 26
Maximize box 76
MIDCREATESTRUCT data structure 45
MEASUREITEMSTRUCT data structure 46, 47
MENU resource statement 68, 69, 70, 71, 73, 78
MENUITEM SEPARATOR statement 73

Index
file format 109
file format 14
mouse cursor shape 62
resource 62
nBITMAPCOREINFO See also RGBTRIPLE
nBITMAPINFO See also RGBQUAD
nBrush
creating 38
nBUTTON control class
control styles 83, 87, 88, 89, 96
nCharacter
  escape\a 70
  escape\t 70
nCHECKBOX resource statement
  DIALOG resource statement 79
nCHECKED option
  MENUITEM resource statement 71
  POPUP resource statement 72
nChild window
  clipping 76
nClient area
  painting 52
nClipboard
  file format 113
nColor
  data types 20
  explicit RGB 20
  logical-palette index 20
  palette-relative RGB 20
  specifying 20
nColor palette See also Logical palette
nCOMBOBOX control class
  control styles 91, 97
nCOMBOBOX resource statement
  DIALOG resource statement 79
nControl
  owner-draw
    drawing 36
      item deleted from 28
    size-box 95
nControl class
  BUTTON@control styles 96
  BUTTON@described 95
  COMBOBOX@described 95
  control styles@described 96
  described 94
  EDIT 95
  control styles 97
  LISTBOX 95
  LISTBOX@control styles 99
  SCROLLBAR 95
  SCROLLBAR@control styles 100
  STATIC 95
nControl edit See edit control
nCONTROL option
  ACCELERATORS resource statement 68
nCONTROL resource statement
  DIALOG resource statement 79
nControl styles
  BUTTON class 96
  COMBOBOX class 97
  described 96
  EDIT class 97
  LISTBOX class 99
  SCROLLBAR class 100
nControl text
  centered 82
  left-justified 80
  right-justified 81
nCreateWindow function
  creating a window with dialog-box attributes 75
nCursor
  file format 111
  resource 62
nDDE
  messages 206
  protocol 205
nDEFPUSHBUTTON resource statement
nDELETEITEMSTRUCT data structure
  described 28
nDevice-independent bitmap See Bitmap, device-independent
nDEVICEDATA printer escape See
  PASSTHROUGH printer escape
nDialog box
  creating 32, 73
  items 34
  template 73
  text font 34
  window style 75
nDIALOG resource statement
  control class@control styles 96

Software development kit
dialog control statements@CHECKBOX 79, 83
dialog control statements@COMBOBOX 79, 91
dialog control statements@CONTROL 79, 94
dialog control statements@Control classes 94
dialog control statements@CTEXT 79, 82
dialog control statements@CTEXT statement 79
dialog control statements@DEFPUSHBUTTON 79, 87
dialog control statements@EDITTEXT 79, 89
dialog control statements@GROUPBOX 79, 86
dialog control statements@ICON 79, 92
dialog control statements@LISTBOX 79, 85
dialog control statements@LTEXT 79, 80
dialog control statements@PUSHBUTTON 79, 84
dialog control statements@RADIOBUTTON 79, 88
dialog control statements@RTEXT 79, 81
dialog control statements@RTEXT statement 79
dialog control statements@SCROLLBAR 93
dialog control statements@SCROLLBAR 93
dialog control statements@CAPTION 75, 77
dialog control statements@CLASS 75, 78
dialog control statements@FONT 75, 79
dialog control statements@MENU 75, 78
dialog control statements@STYLE 75
dialog control statements@STYLE 74
nDIB_PAL_COLORS
device-independent bitmap color table
option 13, 16, 39, 121
nDIB_RGB_COLORS
device-independent bitmap color table
option 39, 121
nDirective
resource compiler
#define 103
#define 104
described 103
nDLGTEMPLATE
DLGITEMTEMPLATE data structure 34
FONTINFO data structure 34
nDocument conventions
\bc\ecbold text\bc\ec 2
\bcFi105M\ecmonospaced type\bcF255D\ec 3
\bcMi\ecitalic text\bcD\ec 3
double brackets ([[]]) 3
horizontal ellipses (…) 3
parentheses ( ) 3
quotation marks (""") 3
small capital letters 3
vertical bar (|) 3
vertical ellipses 3
nDRAWITEMSTRUCT data structure
described 36
nDrop-down menu See Pop-up menu
nDynamic Data Exchange See DDE
NEAR data type 9
nEDIT control class
control styles 90, 97
nEDITTEXT resource statement
style option 90
nEDITTEXT statement
DIALOG resource statement 79
nEllipses
horizontal
as document convention 3
vertical
as document convention 3
nEscape character
\a 70
\t 70
nEscapes
printer 153
NEWFRAME printer escape 184
nExporting
function 139
NEXTBAND printer escape 184
nFile
bitmap
device-independent@format 109
clipboard@format 113

cursor@format 111

cursor@format 111

icon@format 110

initialization@WINDOWS.H 75

metafile@format 113

nFile format

module-definition file 133

nFont

resource 62

nGRAYED option

MENUITEM resource statement 71

POPUP resource statement 71

nGroup box

BUTTON class 86

nGROUPBOX resource statement

DIALOG resource statement 79

nHEAPSIZE statement

syntax 137

nIcon

file format 110

nICON resource statement

DIALOG resource statement 79

nINACTIVE option

MENUITEM resource statement 71

POPUP resource statement 71

nKey

\bc\ecBACKSPACE\bcD\ec 95

\bc\ecCONTROL\bcD\ec 68

\bc\ecSHIFT\bcD\ec 68

\bc\ecTAB\bcD\ec 77

editing 90

nKey word

resource-compiler

BITMAP 62

CURSOR 62

DISCARDABLE 62, 64, 65, 66, 69, 74

FIXED 62, 63, 65, 66, 69, 73

FONT 62

ICON 62

LOADONCALL 62, 63, 65, 66, 69, 73

MOVEABLE 62, 64, 65, 66, 69, 73

PRELOAD 62, 63, 65, 66, 69, 73

nList box

owner-draw 28

owner-draw@measuring 46

owner-draw@sorting 22

nLISTBOX control class

control styles 86, 99

nLISTBOX resource statement

DIALOG resource statement 79

nLogical palette See also LOGPALETTE data

structure

creating 43

nTEXT resource statement

DIALOG resource statement 79

nMDI See Multiple document interface (MDI)

nMenu

loading 47

owner-draw@drawing 36

owner-draw@measuring 46

data resource 68

nMENUBARBREAK option

MENUITEM statement 71

POPUP statement 72

nMENUBREAK option

MENUITEM statement 71

POPUP statement 71

nMetafile

file format 113

nModule-definition file

CODE statement 134

DATA statement 135

DESCRIPTION statement 135

EXETYPE statement 136

EXPORTS statement 136

HEAPSIZE statement 137

IMPORTS statement 138

LIBRARY statement 139

module statement@description 133

NAME statement 139

SEGMENTS statement 140

STACKSIZE statement 141

STUB statement 141

nModule statement

in module definition file@description 133

nMultiple Document Interface (MDI)

child window@creating 45

nMultiple document interface (MDI)

child window 20

nMultiple-line resource statement

ACCELERATORS 67

DIALOG 73

MENU 68
ncoding

executable module 139
imported functions 138
library module 139

naming conventions

data types 9

nNOCOMMENT option

ACCELERATORS resource statement 68
NOPARITY parity type 26

nOption

menu-item

CHECKED 71, 72
GRAYED 71
HELP 71
INACTIVE 71
MENUBARBREAK 72
MENUBARBREAK 71
MENUBREAK 71
SHIFT 68

nOwner-draw button See Button owner-draw
nOwner-draw control See Control owner-draw
nOwner-draw menu See Menu, owner-draw
nPalette See logical Logical palette

Pen

creating 44

nPens

described 71

nested 72

NPSTR data type 9

nPUSHBUTTON resource statement

DIALOG resource statement 79

nRADIOBUTTON resource statement

DIALOG resource statement 79

nRaw-data resource See RCDATA resource statement

nResource

bitmap 62
cursor 62
text 62
icon 62
loading 62, 63, 65, 66, 73
raw data 64
string 66
user-defined 63

nRGB See also Color

explicit 20
palette-relative 20

Index

243
nScroll bar  
  horizontal 76  
  vertical 77  

nSCROLLBAR control class  
  control styles 100  

nSELECTPAPERSOURCE printer escape  See  
  GETSETPAPERBINS printer escape  

nSETENDCAP printer escape  See  
  SETLINECAP printer escape  

nStatement  See specific statement  
  module-definition file@EXETYPE 136  
  module-definition file@LIBRARY 139  
  module-definition file@NAME 139  

nString resource  See also STRINGTABLE  
  resource statement,  See also RCDATA  
  resource statement  

nStyle  
  control  
    BUTTON class 83, 85, 87, 88, 89  
    COMBOBOX class 91  
    default@CHECKBOX statement 84  
    default@COMBOBOX statement 92  
    default@CTEXT statement 82  
    default@DEFPUSHBUTTON statement 88  
    default@EDITTEXT statement 90  
    default@GROUPBOX statement 87  
    default@ICON statement 92  
    default@LISTBOX statement 86  
    default@LTEXT statement 80  
    default@PUSHBUTTON statement 85  
    default@RADIOBUTTON statement 89  
    default@RTEXT statement 81  
    DS_ABSALIGN 74  
    EDIT class 90  
    LISTBOX class 86  
    STATIC class 92  

window  
  listing 75  
  WS_BORDER 76, 86  
  WS_CAPTION 76  
  WS_CHILD 74, 76  
  WS_CHILDWINDOW 76  
  WS_CLIPCHILDREN 76  
  WS_CLIPSIBLINGS 76  
  WS_DISABLED 76, 85, 87, 88, 89, 90  
  WS_DLGFRAME 76  

  WS_GROUP 76, 80, 81, 82, 83, 85, 88, 89,  
          90  
  WS_HSCROLL 76, 90  
  WS_ICONIC 76  
  WS_MAXIMIZE 76  
  WS_MAXIMIZEBOX 76  
  WS_MINIMIZE 76  
  WS_MINIMIZEBOX 76  
  WS_OVERLAPPED 77  
  WS_OVERLAPPEDWINDOW 77  
  WS_POPUP 77  
  WS_POPUPWINDOW 77  
  WS_SIZEBOX 77  
  WS_SYSMENU 77  
  WS_TABSTOP 77, 80, 81, 82, 83, 85, 87,  
            88, 89, 90  
  WS_THICKFRAME 77  
  WS_VISIBLE 77  
  WS_VSCROLL 77, 86, 90  

nSTYLE resource statement  
  when #include directive required with 75  

nSTYLE statement  
  DIALOG resource statement 74, 75  

nText control  
  left-justified 80  
  right-justified 81  

nWindow  
  border 76  
  child 76  
  control  
    user-defined 94  
  creating 24, 75  
  disabled 74, 76  
  iconic 76  
  overlapping 77  
  pop-up 77  
  size 76, 77  
  style  
    dialog box 75  
    visible 77  
    zoom 76  

nWindow style  
  listing 75  
  WS_CHILD 74
ODA_DRAWENTIRE drawing action 37
ODA_FOCUS drawing action 37
ODA_SELECT drawing action 37
ODDPARITY parity type 26
ODS_CHECKED owner-draw control status 37
ODS_DISABLED owner-draw control status 37
ODS_FOCUS owner-draw control status 37
ODS_GRAYED owner-draw control status 37
ODS_SELECTED owner-draw control status 37
ODT_BUTTON owner-draw control type 36, 47
ODT_COMBOBOX owner-draw control type 23, 28, 36, 47
ODT_LISTBOX owner-draw control type 23, 28, 36, 47
ODT_MENU owner-draw control type 36, 47
OFSTRUCT data structure 51
ONESTOPBITS stop-bits type 26
ONESTOPBIT stop-bits type 26
OR operator 72, 80, 81, 83, 85, 86, 87, 89, 90, 91

PAINTSTRUCT data structure 52
PALETTEENTRY data structure 43, 52
Parentheses ( )
  as document convention 3
PASSTHROUGH printer escape 185
PC_EXPLICIT palette-entry option 53
PC_NOCOLLAPSE palette-entry option 53
PC_RESERVED palette-entry option 53
PINT data type 9
Plus (+) operator 80, 81, 82, 83, 85, 86, 87, 88, 89, 90, 91, 92
POINT data structure 54
POPUP resource statement 70, 71, 72
PRELOAD resource-compiler key word 62, 63, 65, 66, 69, 73
Printer driver
  initialization 29
PROOF_QUALITY font quality 42
PSTR data type 9
PUSHBUTTON resource statement 84, 85
PWORD data type 9

QUERYESCSUPPORT printer escape 186
Quotation marks
double (\bc170\ec) 70
double (\bc169\ec\bc170\ec) 66, 67, 70, 78
Quotation marks (\bc169\ec \bc170\ec)
as document convention[Quotation marks ( ),
as document convention] 3

R2_BLACK raster drawing mode 144, 145
R2_COPYPEN raster drawing mode 144, 145
R2_MASKNOTPEN raster drawing mode 144, 145
R2_MASKPEN raster drawing mode 144, 145
R2_MASKPENNOT raster drawing mode 144, 145
R2_MERGENOTPEN raster drawing mode 144, 145
R2_MERGEPPEN raster drawing mode 144, 145
R2_MERGEPPENNOT raster drawing mode 144, 145
R2_NOP raster drawing mode 144, 145
R2_NOT raster drawing mode 144, 145
R2_NOTCOPYPEN raster drawing mode 144, 145
R2_NOTMASKPEN raster drawing mode 144, 145
R2_NOTMERGEPPEN raster drawing mode 144, 145
R2_NOTXORPEN raster drawing mode 144, 145
R2_WHITE raster drawing mode 144, 145
R2_XORPEN raster drawing mode 144, 145
Radio-button control 88
RADIODIUTTON resource statement 88, 89
RCDATA resource statement 64, 65
RECT data structure 54
RESTORE_CTM printer escape 186
RGBQUAD data structure 15, 55
RGBTRIPLE data structure 12, 55
RTEXT resource statement 80, 81

SAVE_CTM printer escape 187
Software development kit
User-defined resource statement 63, 64

V
Variable
  environmental
  INCLUDE 103
Vertical bar (|)
  as document convention 3
Virtual-key character 67
void data type 9

W
WINDOWS.H initialization file 75
WM_COMMAND message 20
WM_COMMAND message message 67
WMCOMPAREITEM message 22
WM_DDE_ACK message 209
WM_DDE_ADVISE message 211
WM_DDE_DATA message 213
WM_DDE_EXECUTE message 214
WM_DDE_INITIATE message 216
WM_DDE_POKE message 217
WM_DDE_REQUEST message 218
WM_DDE_TERMINATE message 219
WM_DDE_UNADVISE message 219
WM_DELETEITEM message 28
WM_DRAWITEM message 36
WM_ENTERIDLE message 33
WM_MEASUREITEM message 46
WM_SETFONT message 33
WM_SYSCOMMAND message 67
WM_SYSMENU window style 33
WNDCLASS data structure 58
WORD data type 9
Wordwrap 98
WS_BORDER window style 76, 86, 91
WS_CAPTION window style 33, 76
WS_CHILD window style 74, 76
WS_CHILDWINDOW window style 76
WS_CLIPCHILDREN window style 76
WS_CLIPSIBLINGS window style 76
WS.Disabled window style 76, 85, 87, 88, 89, 90
WS_DLGFRAME window style 76
WS_GROUP control style 76, 80, 81, 82, 83, 85, 88, 89, 90
WS_HSCROLL window style 46, 76, 90
WS_ICONIC window style 76
WS_MAXIMIZE window style 46, 76
WS_MAXIMIZEBOX window style 76
WS.MINIMIZE window style 46, 76
WS_MINIMIZEBOX window style 76
WS_OVERLAPPED window style 77
WS_OVERLAPPEDWINDOW window style 77
WS_POPUP window style 77
WS_POPUPWINDOW window style 77
WS_SIZEBOX window style 77
WS_SYSCOMMAND window style 76, 77
WS_TABSTOP window style 77, 80, 81, 82, 83, 85, 87, 88, 89, 90
WS_THICKFRAME window style 77
WS_VISIBLE window style 77
WS_VSCROLL window style 46, 77, 86, 90, 91