the update region, is marked for painting when the next WM_PAINT message occurs. The invalidated areas accumulate in the update region until the region is processed when the next WM_PAINT message occurs, or the region is validated by using the ValidateRect or ValidateRgn function.

The *bErase* parameter specifies whether the background within the update area is to be erased when the update region is processed. If *bErase* is nonzero, the background is erased when the BeginPaint function is called; if *bErase* is zero, the background remains unchanged. If *bErase* is nonzero for any part of the update region, the background in the entire region is erased, not just in the given part.

Parameter	Type/Description
hWnd	HWND Identifies the window whose update region is to be modified.
hRgn	HRGN Identifies the region to be added to the update region. The region is assumed to have client coordinates.
bErase	BOOL Specifies whether the background within the update region is to be erased.

Return Value None.

Comments Windows sends a WM_PAINT message to a window whenever its update region is not empty and there are no other messages in the application queue for that window.

The given region must have been previously created by using one of the region functions (for more information, see Chapter 1, "Window Manager Interface Functions").

InvertRect

Syntax void InvertRect(*hDC*, *lpRect*)

This function inverts the contents of the given rectangle. On monochrome displays, the InvertRect function makes white pixels black, and black pixels white. On color displays, the inversion depends on how colors are generated for the display. Calling InvertRect twice with the same rectangle restores the display to its previous colors.

Parameter	Type/Description
hDC	HDC Identifies the device context.
lpRect	LPRECT Points to a RECT data structure that contains the logical coordinates of the rectangle to be inverted.

Return Value None.

Comments The InvertRect function compares the values of the top, bottom, left, and right fields of the specified rectangle. If bottom is less than or equal to top, or if right is less than or equal to left, the rectangle is not drawn.

InvertRgn

Syntax BOOL InvertRgn(*hDC*, *hRgn*)

This function inverts the colors in the region specified by the *hRgn* parameter. On monochrome displays, the InvertRgn function makes white pixels black, and black pixels white. On color displays, the inversion depends on how the colors are generated for the display.

Parameter	Type/Description
hDC	HDC Identifies the device context for the region.
hRgn	HRGN Identifies the region to be filled. The coordinates for the region are specified in device units.

Return Value The return value specifies the outcome of the function. It is nonzero if the function is successful. Otherwise, it is zero.

IsCharAlpha 3.0

Syntax BOOL IsCharAlpha(*cChar*)

This function determines whether a character is an alphabetical character. This determination is made by the language driver based on the criteria of the current language selected by the user at setup or with the Control Panel.

Parameter	Type/Description
cChar	char Specifies the character to be tested.

Return Value The return value is TRUE if the character is alphabetical. Otherwise, it is FALSE.

IsCharAlphaNumeric 3.0

Syntax BOOL IsCharAlphaNumeric(*cChar*)

This function determines whether a character is an alphabetical or numerical character. This determination is made by the language driver based on the criteria of the current language selected by the user at setup or with the Control Panel.

Parameter	Type/Description
cChar	char Specifies the character to be tested.

Return Value The return value is TRUE if the character is an alphanumeric character. Otherwise, it is FALSE.

IsCharLower 3.0

Syntax BOOL IsCharLower(*cChar*)

This function determines whether a character is a lowercase character. This determination is made by the language driver based on the criteria of the current language selected by the user at setup or with the Control Panel.

Parameter	Type/Description
cChar	char Specifies the character to be tested.

Return Value The return value is TRUE if the character is lowercase. Otherwise, it is FALSE.

IsCharUpper 3.0

Syntax BOOL IsCharUpper(*cChar*)

This function determines whether a character is an uppercase character. This determination is made by the language driver based on the criteria of the current language selected by the user at setup or with the Control Panel.

Parameter	Type/Description
cChar	char Specifies the character to be tested.

Return Value The return value is TRUE if the character is uppercase. Otherwise, it is FALSE.

IsChild

Syntax BOOL IsChild(*hWndParent*, *hWnd*)

This function indicates whether the window specified by the *hWnd* parameter is a child window or other direct descendant of the window specified by the *hWndParent* parameter. A child window is the direct descendant of a given parent window if that parent window is in the chain of parent windows that leads from the original pop-up window to the child window.

Parameter	Type/Description
hWndParent	HWND Identifies a window.
hWnd	HWND Identifies the window to be checked.

Return Value The return value specifies the outcome of the function. It is TRUE if the window identified by the *hWnd* parameter is a child window of the window identified by the *hWndParent* parameter. Otherwise, it is FALSE.

IsClipboardFormatAvailable

Syntax BOOL IsClipboardFormatAvailable(*wFormat*)

This function specifies whether data of a certain type exist in the clipboard.

Parameter	Type/Description
wFormat	WORD Specifies a registered clipboard format. For information on clipboard formats, see the description of the SetClipboardData function, later in this chapter.

Return Value The return value specifies the outcome of the function. It is TRUE if data having the specified format are present. Otherwise, it is FALSE.

Comments This function is typically called during processing of the WM_INITMENU or WM_INITMENUPOPUP message to determine whether the clipboard contains data that the application can paste. If such data are present, the application typically enables the Paste command (in its Edit menu).

IsDialogMessage

Syntax BOOL IsDialogMessage(*hDlg*, *lpMsg*)

This function determines whether the given message is intended for the modeless dialog box specified by the *hDlg* parameter, and automatically processes the message if it is. When the IsDialogMessage function processes a message, it checks for keyboard messages and converts them into selection commands for the corresponding dialog box. For example, the TAB key selects the next control or group of controls, and the DOWN key selects the next control in a group.

If a message is processed by IsDialogMessage, it must not be passed to the TranslateMessage or DispatchMessage function. This is because IsDialogMessage performs all necessary translating and dispatching of messages.

IsDialogMessage sends WM_GETDLGCODE messages to the dialog function to determine which keys should be processed.

Parameter	Type/Description
hDlg	HWND Identifies the dialog box.
lpMsg	LPMSG Points to an MSG data structure that contains the message to be checked.

Return Value The return value specifies whether or not the given message has been processed. It is nonzero if the message has been processed. Otherwise, it is zero.

Comments Although IsDialogMessage is intended for modeless dialog boxes, it can be used with any window that contains controls to provide the same keyboard selection as in a dialog box.

IsDlgButtonChecked

Syntax WORD IsDlgButtonChecked(*hDlg*, *nIDButton*)

This function determines whether a button control has a checkmark next to it, and whether a three-state button control is grayed, checked, or neither. The IsDlgButtonChecked function sends a BM_GETCHECK message to the button control.

Parameter	Type/Description
hDlg	HWND Identifies the dialog box that contains the button control.
nIDButton	int Specifies the integer identifier of the button control.

Return Value The return value specifies the outcome of the function. It is nonzero if the given control has a checkmark next to it. Otherwise, it is zero. For three-state buttons, the return value is 2 if the button is grayed, 1 if the button has a checkmark next to it, and zero otherwise.

IsIconic

Syntax **BOOL IsIconic(*hWnd*)**

This function specifies whether a window is minimized (iconic).

Parameter	Type/Description
hWnd	**HWND** Identifies the window.

Return Value The return value specifies whether the window is minimized. It is nonzero if the window is minimized. Otherwise, it is zero.

IsRectEmpty

Syntax **BOOL IsRectEmpty(*lpRect*)**

This function determines whether or not the specified rectangle is empty. A rectangle is empty if the width and/or height are zero.

Parameter	Type/Description
lpRect	**LPRECT** Points to a **RECT** data structure that contains the specified rectangle.

Return Value The return value specifies whether or not the given rectangle is empty. It is nonzero if the rectangle is empty. It is zero if the rectangle is not empty.

IsWindow

Syntax **BOOL IsWindow(*hWnd*)**

This function determines whether the window identified by the *hWnd* parameter is a valid, existing window.

Parameter	Type/Description

hWnd	HWND	Identifies the window.

Return Value The return value specifies whether or not the given window is valid. It is nonzero if *hWnd* is a valid window. Otherwise, it is zero.

IsWindowEnabled

Syntax BOOL IsWindowEnabled(*hWnd*)

This function specifies whether the specified window is enabled for mouse and keyboard input.

Parameter	Type/Description	
hWnd	HWND	Identifies the window.

Return Value The return value specifies whether or not the given window is enabled. It is nonzero if the window is enabled. Otherwise, it is zero.

Comments A child window receives input only if it is both enabled and visible.

IsWindowVisible

Syntax BOOL IsWindowVisible(*hWnd*)

The IsWindowVisible function returns nonzero anytime an application has made a window visible by using the ShowWindow function (even if the specified window is completely covered by another child or pop-up window, the return value is nonzero).

Parameter	Type/Description	
hWnd	HWND	Identifies the window.

Return Value The return value specifies whether or not a given window exists on the screen. It is nonzero if the given window exists on the screen. Otherwise, it is zero.

IsZoomed

Syntax BOOL IsZoomed(*hWnd*)

IsZoomed

This function determines whether or not a window has been maximized.

Parameter	Type/Description
hWnd	HWND Identifies the window.

Return Value The return value specifies whether or not the given window is maximized. It is nonzero if the window is maximized. Otherwise, it is zero.

KillTimer

Syntax BOOL KillTimer(*hWnd*, *nIDEvent*)

This function kills the timer event identified by the *hWnd* and *nIDEvent* parameters. Any pending WM_TIMER messages associated with the timer are removed from the message queue.

Parameter	Type/Description
hWnd	HWND Identifies the window associated with the given timer event. This must be the same value passed as the *hWnd* parameter to the SetTimer function call that created the timer event.
nIDEvent	int Specifies the timer event to be killed. If the application called SetTimer with the *hWnd* parameter set to NULL, this must be the event identifier returned by SetTimer. If the *hWnd* parameter of SetTimer was a valid window handle, *nIDEvent* must be the value of the *nIDEvent* parameter passed to SetTimer.

Return Value The return value specifies the outcome of the function. It is nonzero if the event was killed. It is zero if the KillTimer function could not find the specified timer event.

_lclose

Syntax int _lclose(*hFile*)

This function closes the file specified by the *hFile* parameter. As a result, the file is no longer available for reading or writing.

The *hFile* argument is returned by the call that created or last opened the file.

Parameter	Type/Description
hFile	int Specifies the MS-DOS file handle of the file to be closed.

Return Value The return value indicates whether the function successfully closed the file. It is zero if the function closed the file, or –1 if the function failed.

_lcreat

Syntax int _lcreat(*lpPathName*, *iAttribute*)

This function opens a file with the name specified by the *lpPathName* parameter. The *iAttribute* parameter specifies the attributes of the file when the function opens it. If the file does not exist, the function creates a new file and opens it for writing. If the file does exist, the function truncates the file size to zero and opens it for reading and writing. When the function opens the file, the pointer is set to the beginning of the file.

Parameter	Type/Description
lpPathName	LPSTR Points to a null-terminated character string that names the file to be opened. The string must consist of characters from the ANSI character set.
iAttribute	int Specifies the file attributes. The parameter must be one of these values:

Value	Meaning
0	Normal; can be read or written without restriction.
1	Read-only; cannot be opened for write; a file with the same name cannot be created.
2	Hidden; not found by directory search.
3	System; not found by directory search.

Return Value The return value specifies an MS-DOS file handle if the function was successful. Otherwise, the return value is –1.

LimitEmsPages

Syntax void LimitEmsPages (*dwKbytes*)

This function limits the amount of expanded memory that Windows will assign to an application. It does not limit the amount of expanded memory that the application can get by directly calling INT 67H.

Parameter	Type/Description
dwKbytes	DWORD Specifies the number of kilobytes of expanded memory to which the application is to have access.

Return value None.

Comments LimitEmsPages has an effect only if expanded memory is installed and being used by Windows. If Windows is not using expanded memory, then the function has no effect.

LineDDA

Syntax void LineDDA(*X1*, *Y1*, *X2*, *Y2*, *lpLineFunc*, *lpData*)

This function computes all successive points in a line starting at the point specified by the *X1* and *Y1* parameters and ending at the point specified by the *X2* and *Y2* parameters. The endpoint is not included as part of the line. For each point on the line, the LineDDA function calls the application-supplied function pointed to by the *lpLineFunc* parameter, passing to the function the coordinates of the current point and the *lpData* parameter.

Parameter	Type/Description
X1	int Specifies the logical *x*-coordinate of the first point.
Y1	int Specifies the logical *y*-coordinate of the first point.
X2	int Specifies the logical *x*-coordinate of the endpoint.
Y2	int Specifies the logical *y*-coordinate of the endpoint.
lpLineFunc	FARPROC Is the procedure-instance address of the application-supplied function. See the following "Comments" section for details.

Parameter	Type/Description
lpData	LPSTR Points to the application-supplied data.

Return Value

None.

Comments

The address passed by the *lpLineFunc* parameter must be created by using the MakeProcInstance function.

The callback function must use the Pascal calling convention and must be declared FAR.

Callback Function

void FAR PASCAL *LineFunc*(*X*, *Y*, *lpData*)
int *X*;
int *Y*;
LPSTR *lpData*;

LineFunc 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.

Parameter	Definition
X	Specifies the *x*-coordinate of the current point.
Y	Specifies the *y*-coordinate of the current point.
lpData	Points to the application-supplied data.

Return Value

The function can perform any task. It has no return value.

LineTo

Syntax

BOOL LineTo(*hDC*, *X*, *Y*)

This function draws a line from the current position up to, but not including, the point specified by the *X* and *Y* parameters. The line is drawn with the selected pen. If no error occurs, the position is set to (*X*,*Y*).

Parameter	Type/Description
hDC	HDC Identifies the device context.