

Table of Contents

I. Writing the “Hello world” program.....	6
II. Data types in the Win32 API.....	15
Base types.....	15
Descriptor for data types.....	15
String data types.....	16
Utility data types.....	16
III. Creating a complete window procedure in the Win32 API.....	17
IV. WIN32 API. Keyboard shortcuts.....	27
Accelerator table.....	28
Creating tables accelerator keys.....	29
Assigning keystrokes - accelerator.....	30
Keys - accelerators and menus.....	31
Processing <code>wm_command</code> message.....	32
Creating a key - accelerator for font attributes.....	33
Use accelerator table, created during the program run.....	34
Creating editable user key - accelerator.....	35
Guide keyboard accelerators.....	37
FUNCTIONS <code>COPYACCELERATORTABLE</code>	38
FUNCTION <code>CREATEACCELERATORTABLE</code>	39
FUNCTION <code>DESTROYACCELERATORTABLE</code>	40
FUNCTION <code>LOADACCELERATORS</code>	41
FUNCTION <code>TRANSLATEACCELERATOR</code>	42
STRUCTURE KEYBOARD ACCELERATORS.....	44
MESSAGES KEYBOARD ACCELERATORS.....	44
<code>WM_COMMAND</code> MESSAGE.....	45
<code>WM_INITMENU</code> MESSAGE.....	46
<code>WM_INITMENUPOPUP</code> MESSAGE.....	47
<code>WM_MENUCHAR</code> MESSAGE.....	48
<code>WM_MENUSELECT</code> MESSAGE.....	49
<code>WM_SYSCHAR</code> MESSAGE.....	51
<code>WM_SYSCOMMAND</code> MESSAGE.....	53
V. WIN32 API. Icons.....	55
Types of pictograms.....	55
Dimensions of icons.....	56
Create icons.....	57
Display icons on the screen.....	57
Destruction of icons.....	57
Override icons.....	57
Use of pictograms.....	58
Create icons.....	59
Display icons on the screen.....	62
Sharing resources icons.....	63
Guide for icons.....	65
FUNCTION <code>COPYICON</code>	66
FUNCTION <code>CREATEICON</code>	67

FUNCTION CREATEICONFROMRESOURCE.....	69
FUNCTION CREATEICONFROMRESOURCEEX.....	71
FUNCTION CREATEICONINDIRECT.....	73
FUNCTION DESTROYICON.....	74
FUNCTION DRAWICON.....	75
FUNCTION DRAWICONEX.....	76
FUNCTION GETICONINFO.....	78
FUNCTION LOADICON.....	80
FUNCTION LOOKUPICONIDFROMDIRECTORY	82
FUNCTION LOOKUPICONIDFROMDIRECTORYEX.....	84
Structure of icons.....	86
STRUCTURE ICONMETRICS.....	88
VI. WIN32 API. Windows.....	89
Window of application program.....	90
Components of window of application program.....	91
Controls, dialog boxes and message windows.....	93
Z-sequence (z order).....	94
Creating a window.....	95
Attributes of window.....	96
Window handles.....	99
Creating main window.....	100
Messages creatingwindows.....	101
Multithreaded application program.....	102
General window style.....	103
Child window.....	104
Size of the window frame.....	107
Components of focus frame.....	108
Initial state of windows.....	109
Styles of parent and child windows.....	110
Extended styles.....	111
Relationship of windows.....	112
Blocking of windows.....	113
Foreground and background windows.....	114
State of the window show.....	115
Minimize, maximize and restored window.....	117
Size and position of windows.....	118
Size of the window.....	119
Position (location) of windows.....	120
Dimensions and position by default.....	121
System Commands.....	122
Functions of size and position.....	123
Message of the size and position.....	124
Destruction of windows.....	125
Creating the main window.....	126
Creating, enumeration and resizing of child windows.....	128
Destruction of windows.....	131
Window structures.....	133
STRUCTURE CLIENTCREATESTRUCT.....	133
STRUCTURE COPYDATASTRUCT.....	135

STRUCTURE CREATESTRUCT.....	136
STRUCTURE MDICREATESTRUCT.....	139
STRUCTURE MINMAXINFO.....	142
STRUCTURE NCCALCSIZE_PARAMS.....	144
STRUCTURE STYLESTRUCT.....	145
STRUCTURE WINDOWPLACEMENT.....	146
STRUCTURE WINDOWPOS.....	148
Window messages.....	151
WM_ACTIVATE MESSAGE.....	151
WM_ACTIVATEAPP MESSAGE.....	153
WM_CANCELMODE MESSAGE.....	154
WM_CHILDACTIVATE MESSAGE.....	155
WM_CLOSE MESSAGE.....	156
WM_COMPACTING MESSAGE.....	157
WM_COPYDATA MESSAGE.....	158
WM_CREATE MESSAGE.....	160
WM_DESTROY MESSAGE.....	161
WM_QUIT MESSAGE.....	162
WM_ENABLE MESSAGE.....	163
WM_ENTERSIZEMOVE MESSAGE.....	164
WM_EXITSIZEMOVE MESSAGE.....	165
WM_GETICON MESSAGE.....	166
WM_GETMINMAXINFO MESSAGE.....	168
WM_GETTEXT MESSAGE.....	169
WM_GETTEXTLENGTH MESSAGE.....	171
WM_INPUTLANGCHANGE MESSAGE.....	173
WM_INPUTLANGCHANGEREQUEST MESSAGE.....	174
WM_MOVE MESSAGE.....	175
WM_MOVING MESSAGE.....	176
WM_NCACTIVATE MESSAGE.....	178
WM_NCCALCSIZE MESSAGE.....	179
WM_NCDESTROY MESSAGE.....	181
WM_PARENTNOTIFY MESSAGE.....	182
WM_POWER MESSAGE.....	184
WM_SETICON MESSAGE.....	188
WM_SETTEXT MESSAGE.....	190
WM_SETTINGCHANGE MESSAGE.....	191
WM_SIZE MESSAGE.....	197
WM_SIZING MESSAGE.....	199
WM_STYLECHANGED MESSAGES.....	201
WM_STYLECHANGING MESSAGE.....	202
WM_USERCHANGED MESSAGE.....	203
WM_WINDOWPOSCHANGING MESSAGE.....	204
WM_WININICHANGE MESSAGE.....	206
WM_CLOSE MESSAGE.....	207
WM_COMPACTING MESSAGE.....	208
WM_COPYDATA MESSAGE.....	209
WM_CREATE MESSAGE.....	211
WM_DESTROY MESSAGE.....	212

Window functions.....	213
FUNCTION ADJUSTWINDOWRECT.....	213
FUNCTION ADJUSTWINDOWRECTEX.....	215
FUNCTION ARRANGEICONICWINDOWS.....	217
FUNCTION BEGINDEFERWINDOWPOS.....	218
FUNCTION BRINGWINDOWTOTOP.....	220
FUNCTION CASCADEWINDOWS.....	221
FUNCTION CHILDWINDOWFROMPOINT.....	223
FUNCTION CHILDWINDOWFROMPOINTEX.....	225
FUNCTION CLOSEWINDOW.....	227
FUNCTION CREATEWINDOW.....	228
FUNCTION CREATEWINDOWEX.....	245
FUNCTION DEFERWINDOWPOS.....	250
FUNCTION DESTROYWINDOW.....	254
FUNCTION ENABLEWINDOW.....	256
FUNCTION ENDDEFERWINDOWPOS.....	258
FUNCTION ENUMCHILDPROC.....	259
FUNCTION ENUMTHREADWINDOWS.....	261
FUNCTION ENUMWINDOWS.....	263
FUNCTION ENUMWINDOWSPROC.....	265
FUNCTION FINDWINDOW.....	267
FUNCTION FINDWINDOWEX.....	268
FUNCTION GETCLIENTRECT.....	270
FUNCTION GETDESKTOPWINDOW.....	271
FUNCTION GETFOREGROUNDWINDOW.....	272
FUNCTION GETLASTACTIVEPOPUP.....	273
FUNCTION GETNEXTWINDOW.....	274
FUNCTION GETPARENT.....	276
FUNCTION GETTOPWINDOW.....	277
FUNCTION GETWINDOW.....	278
FUNCTION GETWINDOWPLACEMENT.....	280
FUNCTION GETWINDOWRECT.....	282
FUNCTION GETWINDOWTEXT.....	283
FUNCTION GETWINDOWTEXTLENGTH.....	285
FUNCTION GETWINDOWTHREADPROCESSID.....	287
FUNCTION ISCHILD.....	288
FUNCTION ISICONIC.....	289
FUNCTION ISWINDOW.....	290
FUNCTION ISWINDOWUNICODE.....	291
FUNCTION ISWINDOWVISIBLE.....	292
FUNCTION ISZOOMED.....	293
FUNCTION MOVEWINDOW.....	294
FUNCTION OPENICON.....	296
FUNCTION SETFOREGROUNDWINDOW.....	297
FUNCTION SETPARENT.....	299
FUNCTION SETWINDOWLONG.....	301
FUNCTION SETWINDOWPLACEMENT.....	303
FUNCTION SETWINDOWPOS.....	305
FUNCTION SETWINDOWTEXT.....	309

FUNCTION SHOWOWNEDPOPUPS.....	311
FUNCTION SHOWWINDOW.....	313
FUNCTION SHOWWINDOWASYNC.....	316
FUNCTION TILEWINDOWS.....	318
FUNCTION WINDOWFROMPOINT.....	320
FUNCTION WINMAIN.....	321
FUNCTION ANYPOPUP.....	323
FUNCTION ENUMTASKWINDOWS.....	324
FUNCTION GETSYSMODALWINDOW.....	325
FUNCTION GETWINDOWTASK.....	326
FUNCTION SETSYSMODALWINDOW.....	327
FUNCTION CHILDWINDOWFROMPOINT.....	328
FUNCTION CHILDWINDOWFROMPOINTEX.....	330
FUNCTION CLOSEWINDOW.....	332
WIN32 API. MENU.....	333
Menu items and menu.....	333
Secondary menu.....	334
Menu of the window.....	335
Identification reference.....	336
Descriptors of the menu.....	337
Menu items.....	338
Command items and items which opens a submenu.....	339
Menu item identification.....	340
Position of menu item.....	341
Sets the default menu items.....	342
Menu items with and without tick.....	343
Included, inaccessibility and lock menu items.....	344
Selected menu item.....	345
Custom menu items.....	346
Menu divider and line breaks.....	347
Creating the menu.....	348
Resources of menu templates.....	349
Template of menu in memory.....	350
Functions of the menu.....	351
Show on-screen menu.....	352
Menu window class.....	353
Destruction of menu.....	354
Messages used by menu.....	355
Changing the menu.....	356
Using the menu.....	357
Resource usage of menu templates.....	358
Advanced format for menu templates.....	359
Old format of menu templates.....	360
Resource loading of menu templates.....	361
Creating menu class.....	362
Creating of a subsidiary (context) menu.....	365
On-screen display the context menu.....	366
Use the menu bitmap (icon).....	367
Checking the box type icon.....	368

Introduction to Win32 API

I. Writing the “Hello world” program

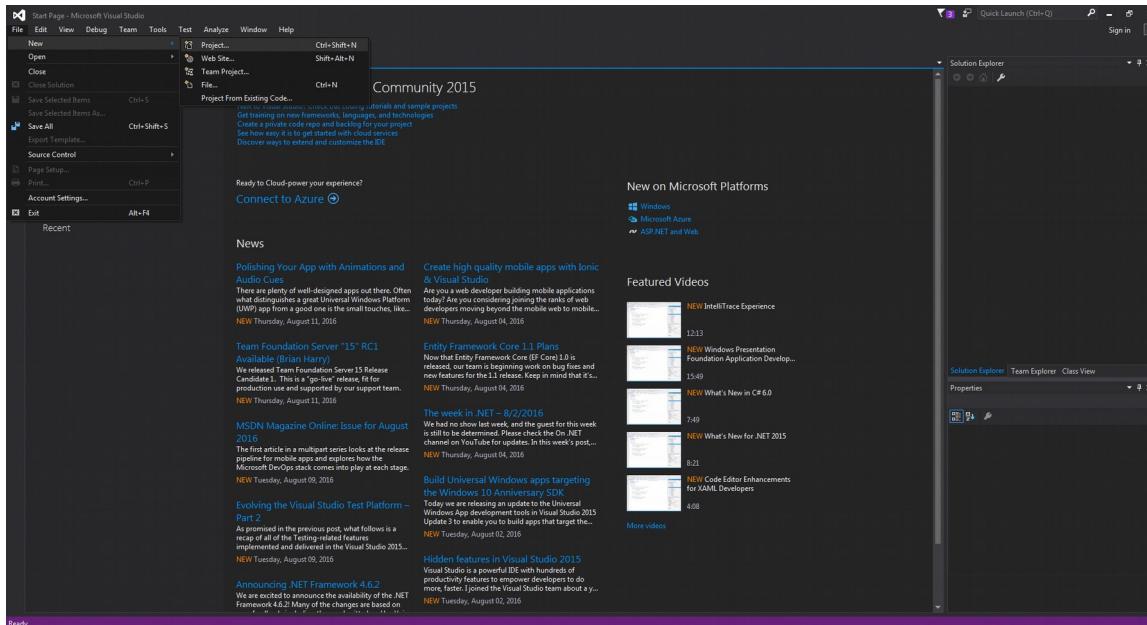
Win32 API (hereinafter WinAPI) - a set of functions (API - application programming interfaces), running under Windows environment. These functions are contained in windows.h library.

Windows API is designed to be used with C language to write the applications under MS Windows operating system environment. Windows API is the closest way to interact with operating system from the application level. The lower level access is required only for device drivers in the current versions of Windows and is available through the Windows Driver Model.

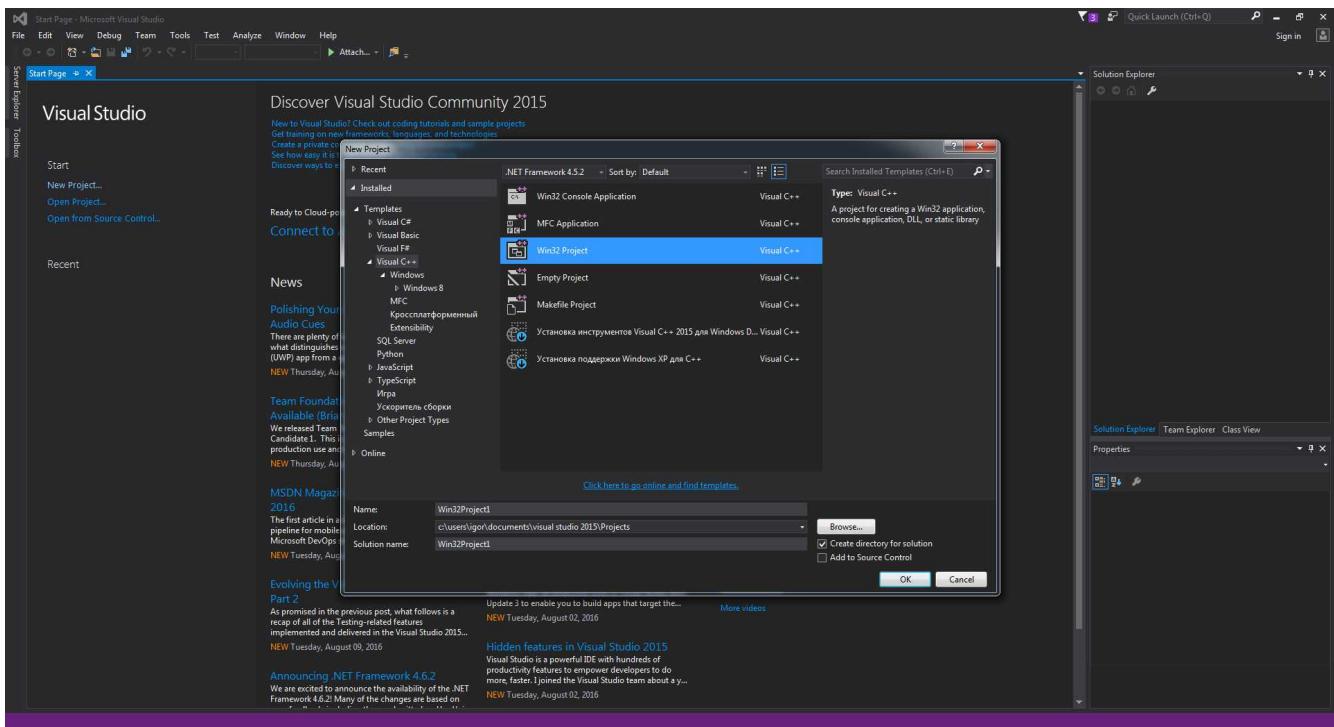
Windows API is a set of functions, data structures and numerical constants, that follow C language conventions. All programming languages that can call to such functions and to operate these types of data in the programs executed under Windows, can use this API. In particular, the languages C ++, Pascal, Visual Basic, and many others.

Using WinAPI functions you can easily create different window procedures, dialog boxes, software and even games. This library, so far, is the base for the development of Windows Forms programming, MFC, because these interfaces are add-ons for that library. If you master it, you will easily create forms and understand how this happens.

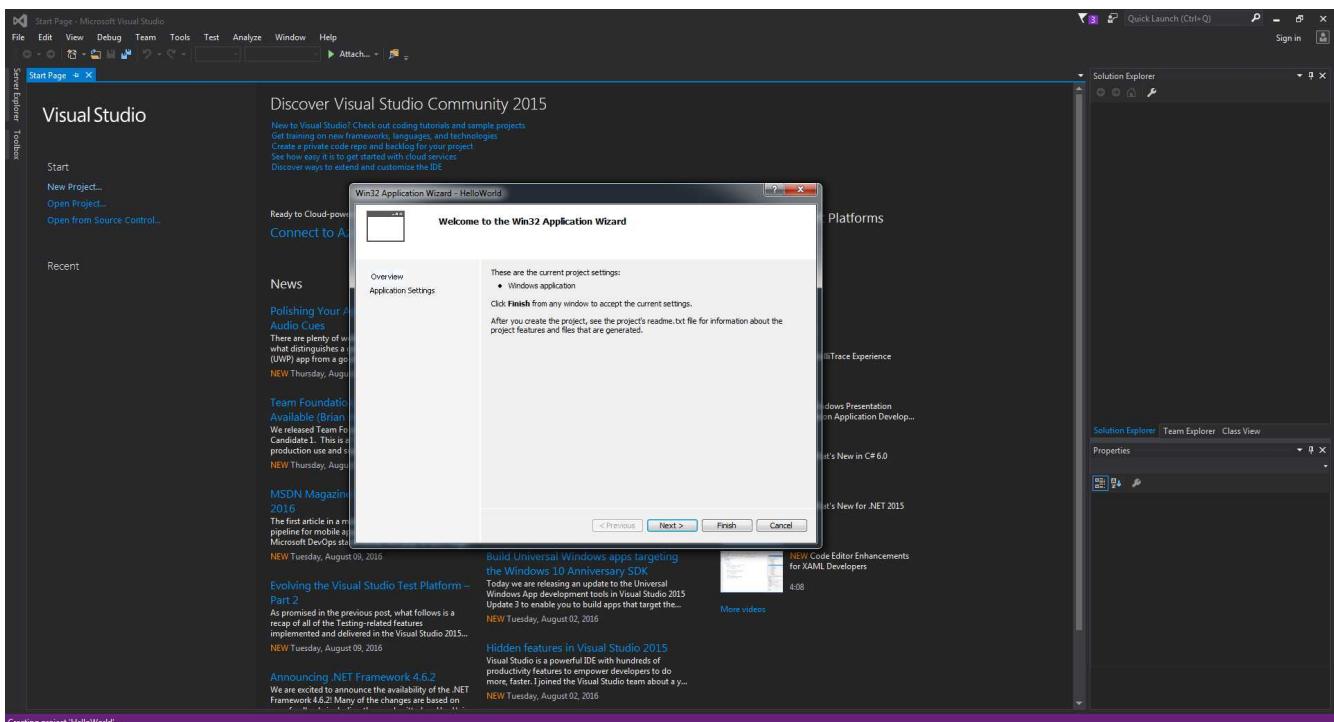
We will not go deep into the theory now. We are going to start the project in MVS, and to finish with the sample example at the end of this chapter. First, we will open Microsoft Visual Studio 2015, and open the "File" tab to create new project:



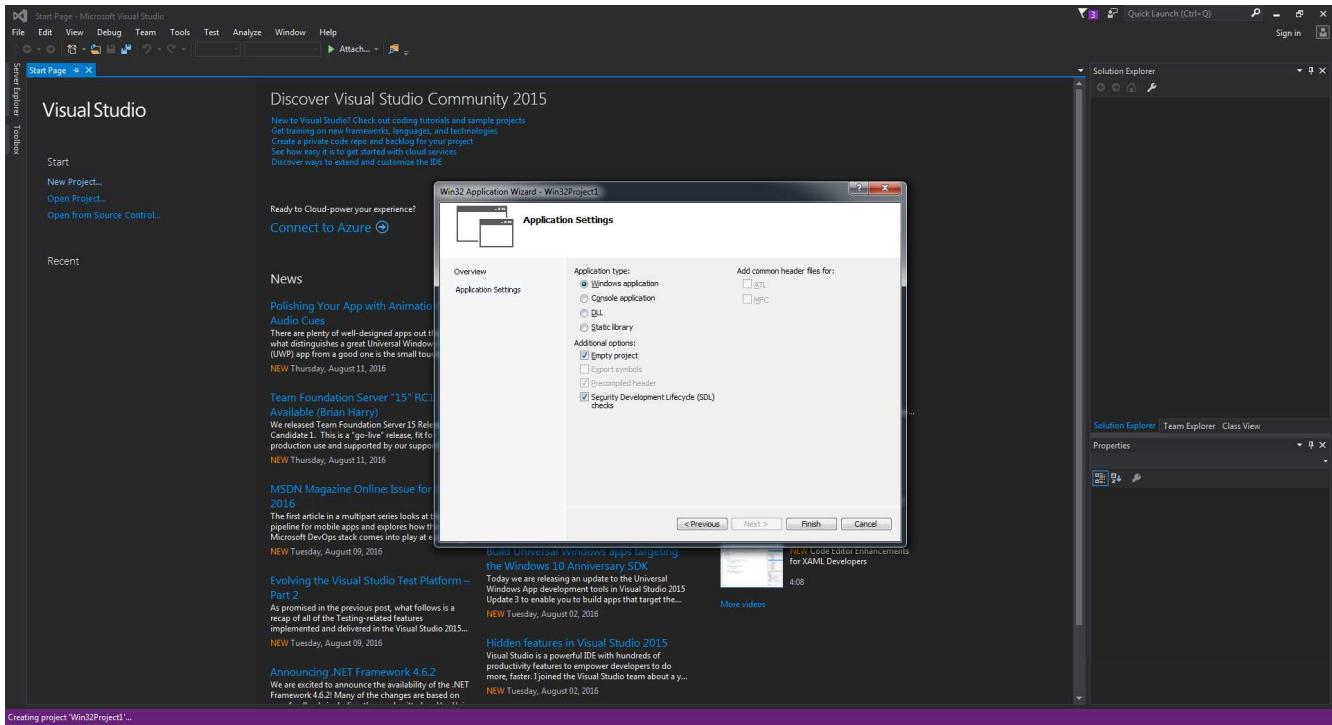
Then, in the drop-down list select Visual C ++, and in the window select «Win32 project.” Click on it.



Enter the name of the project, specify the path and click "OK". Next, you will get to "Win32 Application Wizard". Click "Next".

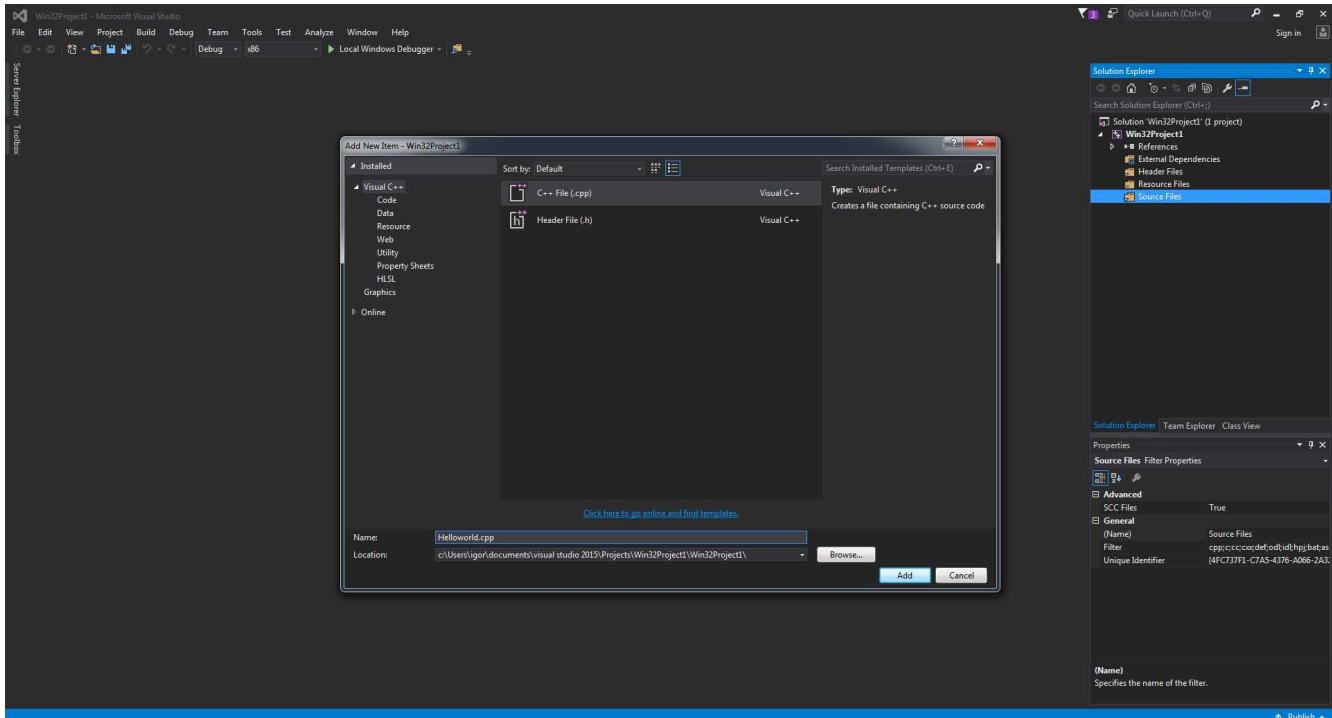


You have to choose “Windows application” and “Empty project”. By default, the option “Empty Project” is not selected. We need also to select “Windows application” as a type of our application. If everything is correct - click “Finish”.



Finally, let's start to write a simple demo program, which traditionally will display on the screen: "Hello World" inscription.

Essentially, you have to add "cpp" file to the created project. We right click on the "Source Files" and in the drop-down list, select the tab - "Add", then "New Item ...". As a result, the following window should appear:



Select "C ++ File", enter its name, press "Add". Then open the file and paste the following code into it

```
#include <windows.h> // header file, contains Windows API functions

// Main function - same as int main() in the console application:
int WINAPI WinMain(HINSTANCE hInstance, // handle of application instance
HINSTANCE hPrevInstance, // not used in Win32
LPSTR lpCmdLine, // needed to run the window mode in command line mode
int nCmdShow) // the windows display mode

{
    // The function to show window with "OK" button on the screen (talk about options later)
    MessageBox(NULL, L"Hello, world!!!", L"Window function", MB_OK);
    return NULL; // return the result of the function
}
```

The program should compile successfully and run, and show a window on the screen:



Now we will stop to study the program code in details.

In the first line, we include the header file windows.h. It contains all the necessary API function.

In the lines from 4 to 7 we have the description of the function int WINAPI WinMain (). Qualifier WINAPI is always needed for the WinMain main function. Just remember. WinMain is the name of the function. It has four parameters. The first is HINSTANCE hInstance () line number 4. hInstance is the window instance handler (it is a kind of code of the window procedure, the identifier by which it will be distinguished from other windows). Using it, you can address to this window while working in other functions (more about that later), change some settings of the window. HINSTANCE is one of many data types defined in WinAPI, same as int, for example. The HINSTANCE hInstance instruction tells us that we are going to create a new variable of type HINSTANCE with the name of hInstance.

We will talk about data types in detail in the next chapter, so move onto the next parameter: HINSTANCE hPrevInstance (line 5). As it is written in the comments, it is not used in Win32. Next we have the type LPSTR variable (line 6), named lpCmdLine. It is used when we launch window from the command line with the parameters. It is very exotic solution, so we will not dwell on it.

And the last parameter: integer, it determines how to display the window. We need ShowWindow function, that would be described later. For example, by using it, we can maximize the window to fill the screen, make it a certain height, transparent or put it on the top.

Now we move to the MessageBox () function (line 10). It has four parameters and it's used to display an error message, for example. In our case, we are using it to display the "Hello world" message. In

general this function looks as follows:

```
int MessageBox(HWND hWnd, // parent window handler  
LPCTSTR lpText, // a pointer to a string with message line  
LPCTSTR lpCaption, // a pointer to a string with header text  
UINT uType); // flags to display the buttons and other style icons
```

In our case, the first parameter is set to zero. That's because we do not have the parent window (our window does not run from any program). Next, we have two variables of type LPCWSTR: lpText and lpCaption. The first variable contains information that will be displayed in the window in a text form. The second contains the text of the header of the window. It is almost the same as char * str, but has a number of features. The text would be displayed properly if you put a the letter L (UNICODE string) before the main string. And the last type of data - UINT - 32-bit unsigned integer. It is an analogue of simple unsigned integer. In this parameter you can pass some values (about them later, too), due to which you can change the appearance of a button. In our case - it MB_OK - means that the window creates a button that says "OK" and its appropriate action when it is pressed (it will close the application).

On line number 11, we return the value of the main function, since it is not void type. Now you have general idea of Windows API, and how you can use it.

II. Data types in the Win32 API

WINAPI has defined its own set of data types, as they are defined in the C / C ++ language (int, char, float, etc.). Now you don't need to study all its definitions. It is enough to remember that they are available in Windows API, and be the time, when they will appear, or would be required to use them anywhere in the program, just return here to see their definitions. In the future, we will use all of them. They can be divided into several types: basic, descriptor, string, and utility.

Base types

- **BOOL** – it is similar to the type of data “bool”. It also has two possible values – 0 and 1. When you are using WINAPI, you better use 0 instead of NULL.
- **BYTE** – same as byte or 8-bit unsigned integer. Similar to unsigned char.
- **DWORD** — 32-bit unsigned integer. Similar to unsigned long int, UINT.
- **INT** – 32-bit integer. Similar to long int.
- **LONG** – 32-bit integer. Similar to long int.
- **NULL** – null pointer. This is how it is declared in the program:

```
void *NULL = 0;
```

- **UINT** – 32-bit unsigned integer. Similar to unsigned long int, DWORD.

Descriptor for data types

Descriptor, as mentioned earlier, is an identifier of any object. There are different descriptors for different types of objects. The descriptor of the object can be described as:

```
HANDLE h;
```

There are also brush descriptors, mouse cursors, fonts, etc. With their help, we can change any settings at the initialization time or during application work, thus as you can not do that in the console application. They are used for descriptive functions, of control types: CreateProcess (), ShowWindow (), etc. or as the return value of some functions:

```
// receiving the descriptor for input or output devices:  
HANDLE h = GetStdHandle(DWORD nStdHandle);
```

In this function, we have got the descriptor for std_in std_out streams and we can, for example, use it in some sort of condition.

- **HANDLE** – descriptor of object.
- **HBITMAP** – descriptor of bitmap. From the name of handle bitmap.
- **HBRUSH** – descriptor of brush. From the name of handle brush.
- **HCURSOR** – descriptor of cursor. From the name of handle cursor.
- **HDC** – descriptor of device context. From the name of handle device context.
- **HFONT** – descriptor of font. From the name of handle font.
- **HICONS** – descriptor of icons. From the name of handle icons.
- **HINSTANCE** – descriptor of the application instance. From the name of handle instance.

- **HMENU** – descriptor of menu. From the name of handle menu.
- **HPEN** – descriptor of pen. From the name of handle pen.
- **HWND** – descriptor of window. From the name of handle window.

String data types

To begin with we will look at what types of encoding exist in Windows OS. There are two types of character encodings: ANSI and UNICODE. Single-byte characters are ANSI, double-byte – UNICODE. We can easily connect the UNICODE character set in the project properties. After that in the code we can create a variable of char data type as follows:

```
// creating a string of 10 elements:
wchar_t str[10];
```

If we want to use the ANSI character set, we traditionally write:

```
// also creating a string of 10 elements:
char str[10];
```

In WINAPI, depending on whether Unicode is connected or not, two types of strings are used: UNICODE or TCHAR. The following describes the types of data strings.

- **LPCSTR** – a pointer to a constant string, ending with zero-interrupter. From the name of long pointer constant string.
- **LPCTSTR** – a pointer to a constant string, without UNICODE. From the name of long pointer constant TCHAR string. This add-in function to LPCSTR.
- **LPCWSTR** – a pointer to a constant UNICODE string. From the name of long pointer constant wide character string. This add-in function to LPCSTR.
- **LPSTR** – a pointer to a string, ending with zero-interrupter. From the name of long pointer string.
- **LPTSTR** – a pointer to a string without UNICODE. From the name of long pointer TCHAR string. This add-in function to LPSTR.
- **LPWSTR** – a pointer to a UNICODE string. From the name of long pointer wide character string. This add-in function to LPSTR.
- **TCHAR** – symbol data type — same as char and wchar_t.

Utility data types

Utility data types are used in some types of functions. In particular, the parameters described below are used when working with the window callback function like the following:

```
HRESULT CALLBACK NameOfFunction(HWND hWnd, UINT uMsg,
    WPARAM wParam, LPARAM lParam);
```

- **LPARAM** – type to describe lParam (long parameter). Used with wParam in some functions.
- **LRESULT** – value, returned by the window procedure has long data type.
- **WPARAM** – type to describe wParam (word parameter). Used with lParam in some functions.

III. Creating a complete window procedure in the Win32 API

General algorithm for creating the window procedure in WINAPI:

- Create two functions: WinMain() – the main function with parameters, as mentioned in the first chapter – same as the main for console application; the function that handles the processes (for example – WndProc()), message flows from and to Windows operation system.
- Create descriptor of the window hMainWnd and register the window class WNDCLASSEX (ie specify the characteristics of the window, which would be created). Must be contained in WinMain.
- Create a skin of our window. Must be contained in WinMain.
- Write cycles to process messages. Must be contained in WndProc.
- Create a function to display the windows. ShowWindow and other utility functions.

Now it should be difficult enough because there would be a variety of functions with a bunch of settings, classes, that you should know, understand, and some other wisdoms of C and C ++ languages.

To begin with we will describe the overall structure of WinMain () function:

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInst,
                    HINSTANCE hPreviousInst,
                    LPSTR lpCommandLine,
                    int nCommandShow
)
{
    // creating a window handle
    // describe a window class
    // create a window, show it on the screen
    // return value if it fails or when you exit
}
```

First, create a window handle:

```
HWND hMainWnd; // and create handle for future windows
```

To initialize the window class fields, you need to create an instance of it, then fill in the class fields through it.

Declaration of this class in windows.h looks like this:

```

struct tagWNDCLASSEX {
    UINT cbSize; // structure size (in bytes)
    UINT style; // window class style
    WNDPROC WndProc; // a pointer to a user-defined function name
    int cbWndExtra; // the number of bytes released at the end of the structure
    int cbClsExtra; // the number of bytes released when creating an application instance
    HICON hIcon; // icon descriptor
    HICON hIconMin; // .... small icon in tray
    HCURSOR hCursor; // .... the mouse cursor
    HBRUSH hbrBack; // .... the color of the window background
    HINSTANCE hInst; // .... the application instance
    LPCTSTR lpszClassName; // const-pointer to a string containing the name of the class
    LPCTSTR lpszMenuName; //const-pointer to a string containing the name of the menu
                           //to be used for class
}WNDCLASSEX;

```

That is, we need to create a variable of type WNDCLASSEX, usually wc, then use it to initialize class fields, like this:

```

WNDCLASSEX wc; // create an instance to access the class members WNDCLASSEX
wc.cbSize = sizeof(wc); // size of structure (in bytes)
wc.lpfnWndProc = WndProc; // a pointer to a user-defined function
                           // etc.

```

It is necessary if we want to use this class in the future. This will be the template for creating a heap of windows. Of course, at first we don't need so much windows. However, registration is definitely necessary! We can even set the default settings during initializing fields, rather than coming up with what should be the window (during the next chapter we will discuss the parameters that are necessary to initialize class fields).

So, again: if at least one field of the class will not be initialized, the window would not be created. To verify, there is a useful function RegisterClassEx (). This is the next step after filling WNDCLASSEX class fields: obligatory check of registration of class:

```

if (!RegisterClassEx(&wc)) {
    // in case of wrong registration:
    MessageBox(NULL,
               L"Cant register the class!",
               L"Error", MB_OK);
    return NULL; // return, exit the WinMain
}

```

The next thing we have to do in WinMain () function - call the CreateWindow () function and assign it a value of descriptor, which we have created in the first phase. Code snippet is here:

```
hMainWnd = CreateWindow(szClassName, // the class name
    L"Full windows procedure", // the window name (the one above)
    WS_OVERLAPPEDWINDOW | WS_VSCROLL, // window display mode
    CW_USEDEFAULT, // window position along the x axis (default)
    NULL, // window position along the y axis (default)
    CW_USEDEFAULT, // the width of the window (default)
    NULL, // the height of the window (default)
    HWND(NULL), // handle to the parent window (we do not have a parent window)
    NULL, // menu descriptor (we do not have it)
    HINSTANCE(hInst), // .... the application instance
    NULL); // We do not share anything from WndProc
```

Description of CreateWindow () function in windows.h looks like this:

```
HWND CreateWindow(
    LPCTSTR lpClassName, // the class name
    LPCTSTR lpWindowName, // the window name (the one above)
    DWORD dwStyle, // window style
    int x, // window position along the x axis
    int y, // window position along the y axis
    int nWidth, // the width of the window
    int nHeight, // the height of the window
    HWND hWndParent, // handle to the parent window
    HMENU hMenu, // ....menu
    HINSTANCE hInst, // the application instance
    LPVOID lParam
); // a pointer to the data transmitted from the user-defined function
```

If successful, the function hMainWnd will return not NULL value. It is easy to guess that it is necessary to do a background check (similar to RegisterClassEx()):

```
if (!hMainWnd) {
    // in case of wrong creation of the window (bad parameters for example and etc):
    MessageBox(NULL, L"Can't create the window!", L"Error", MB_OK);
    return NULL; // return, exit the WinMain
}
```

After that, we need to call two functions:

```
ShowWindow(hMainWnd, nCommandShow);
UpdateWindow(hMainWnd);
```

There is no need to describe them in detail, because they have only several parameters.

The first function displays a message box on the PC screen. Its first parameter - window descriptor (returned from CreateWindow () call). The second option - the display style. When you first launch the window should be equal to the last parameter of the WinMain () function, and the next time you can enter your own data. The second function – as the name suggests, is responsible for updating the window on the screen when it is minimized or by dynamic information. It's time to make a cycle and return value of the function:

```

    while (GetMessage(&msg, NULL, NULL, NULL)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

```

Finally, the modified program that outputs the message window (from the first chapter) with other buttons:

```

#include <windows.h>

// Основная функция:
int WINAPI WinMain(HINSTANCE hInst, // descriptor of the application instance
                    HINSTANCE hPreviousInst, // not used in Win32, but should be announced
                    LPSTR lpCommandLine, // needed to run window in command line mode
                    int nCommandShow) //the window display mode

{
    int result = MessageBox(NULL, L"Do you like WINAPI?!", L"Question",
                           MB_ICONQUESTION | MB_YESNO);
    switch (result)
    {
        case IDYES: MessageBox(NULL, L"Keep up the good work!!!",
                               L"Answer", MB_OK | MB_ICONASTERISK); break;
        case IDNO: MessageBox(NULL, L"It's a pitt!!!", L"Answer",
                               MB_OK | MB_ICONSTOP); break;
    }
    return NULL;
}

```



WinMain() in the last line contains two functions calls, interacting with our function WndProc() and return statement operator. The design of the function WndProc() should be conditionally like this:

```

LRESULT CALLBACK WndProc(HWND hWnd, // descriptor of window
                        UINT uMsg, // message sent by the OS
                        WPARAM wParam, // parameters
                        LPARAM lParam) // messages, for the next для последующего appeal

{
    // 1) create the necessary variables
    // 2) write conditions under which it is necessary to perform the desired action
    // 3) The return value of the function
}

```

LRESULT – it is the return value. CALLBACK should be written as this is a callback function. First, you need to create the required variables. First, create an instance of a device context HDC for the correct orientation of the text in the window. Besides, for the window field processing, we need two more variables RECT (rectangular window area) and PAINTSTRUCT (in the structure of the window to draw information) respectively:

```
HDC hDC;
PAINTSTRUCT ps;
RECT rect;
```

To change the color of the text (and not only) you need to create a variable of type COLORREF and assign its return value to the function RGB() with three parameters:

```
COLORREF colorText = RGB(255, 0, 0); // parameters int
```

This function converts the integer type in the color intensity and returns the value of mixing intensity of three colors (red + green + blue). As you can see via these color gradations it can be created $255 \times 255 \times 255 = 16,777,216$ of colors.

Windows OS and various functions of our program would send thousands of messages every second for every single application. These messages can be sent as result of handling the key or mouse button pressing etc. For these cases, we have a structure MSG, described in WinMain(), which stores information about these messages. In WndProc() there are conditions for the selection of these messages.

If the operating system sends a message WM_PAINT, for example, then something should be drawn in the window. These messages are processed by the condition switch() (multiple-choice operator), parameters of it are uMsg. uMsg we have created when describing our function WndProc(). The main value in the statement, without which the window will not be updated after folding and closing WM_DESTROY. Also there is WM_PAINT which is necessary to draw in the client area. WM - from words Window Message. Namely:

```
switch (uMsg) {
    case WM_PAINT:
        // draw something
    case WM_DESTROY:
        // We definitely make the condition of closing the window
    default:
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
```

And what we should do with these cases? When sending a WM_PAINT message - we call drawing functions, BeginPaint (), GetClientRect (), SetTextColor (), DrawText (), EndPaint (). It is clear by name of these functions what they are doing.

BeginPaint () function in the literal sense of the word begins to draw. Only for this purpose it is necessary to have a window handle and PAINTSTRUCT object (we have it ps). It returns a value of type HDC, so we need to give it a hDc.

GetClientRect () selects a region. Its argument is similar to the previous function: window handle and a pointer to a RECT object class (we have it rect). SetTextColor () function returns the color of the text. Its parameters are: the return value of function BeginPaint () - hDC and a pointer to the object COLORREF class. We even could not set the color of text separately, creating a variable colorText, but could do it right there. But in terms of readability and comprehension of code - it is fundamentally wrong. Always try to declare variables separately and write in the comments, why they are needed, and after some time you will not have any questions about them.

Also observe the Hungarian notation in programming, the essence of which: variable names should bear the meaning of their existence and to show the type of data. Declaration of function DrawText():

```
int DrawText(
    HDC hDC, // device context handle
    LPCTSTR lpchText, // our string pointer
    int nCount, // text length (if it is equal to -1, then defines itself)
    LPRECT lpRect, // a pointer to the object RECT
    UINT uFormat // text display format
);
```

The first 4 parameters have a clear meaning. Fourth uFormat - has several types. Commonly used DT_SINGLELINE, DT_CENTER and DT_VCENTER to display text in the center region, in one line. But you can use other options. EndPaint () function takes two parameters: a window handle and the object ps. Have you noticed the analogy with the BeginPaint ()? You know what to do when you call WM_PAINT (do not forget to use break in the end). WM_DESTROY is sent with the window function DestroyWindow (), which is caused when we closed it. This occurs in the default statement. Thus a call DefWindowProc () function happen, the parameters of which are the same as that of the WndProc (). In its body WM_DESTROY must have PostQuitMessage () function, which is sending the WinMain () message WM_QUIT. Its setting is usually NULL, interpreting for the main function WinMain (), as the WM_QUIT.

```
VOID WINAPI PostQuitMessage(int nExitCode);
```

Upon completion of the switch (), we return a null value with WndProc () function. And now we back to the WinMain (), in particular to the message handler.

```
while (GetMessage(&msg, NULL, NULL, NULL)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
```

GetMessage () function has the following description:

```
BOOL WINAPI GetMessage(LPMSG lpMsg, // a pointer to a structure MSG
    HWND hWnd, // descriptor of window
    UINT wMsgFilterMin, // filters
    UINT wMsgFilterMax // filters for messages sample
);
```

It handles messages sent by the OS. The first parameter is the address of the MSG structure, in which next message would be located. The second parameter - a window handle. The third and fourth parameters indicate the procedure for the selection of messages. Typically, zero values and any value function are removed from the queue. The cycle stops when it receives a message WM_QUIT. In this case, it returns FALSE, and we exit the program. TranslateMessage () and DispatchMessage () functions are needed in the loop to interpret the messages. Usually it is used in the processing of the pressed keys on the keyboard. At the end of the cycle, we return the operating system return code msg.wParam. As a result, we should get the following code:

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
TCHAR mainMessage[] = L"Some text to display!";

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmdLine, int nCmdShow)
{
    TCHAR szClassName[] = L"My class";
    HWND hWndWind;
    MSG msg;
    WNDCLASSEX wc;
    wc.cbSize = sizeof(wc);
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = WndProc;
    wc.lpszClassName = NULL;
    wc.lpszClassName = szClassName;
    wc.cbWndExtra = NULL;
    wc.cbClsExtra = NULL;
    wc.hIcon = LoadIcon(NULL, IDI_WINLOGO);
    wc.hIconSm = LoadIcon(NULL, IDI_WINLOGO);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wc.hInstance = hInst;
    if (!RegisterClassEx(&wc)) {
        MessageBox(NULL, L"Can't register the class!", L"Error", MB_OK);
        return NULL;
    }
    hWndWind = CreateWindow(
        szClassName,
        L"Window procedure",
        WS_OVERLAPPEDWINDOW | WS_VSCROLL,
        CW_USEDEFAULT,
        NULL,
        CW_USEDEFAULT,
        NULL,
        (HMENU)NULL,
        NULL,
        HINSTANCE(hInst),
        NULL);
    if (!hWndWind) {
        MessageBox(NULL, L"Can't create the window!", L"Error", MB_OK);
        return NULL;
    }
    ShowWindow(hWndWind, nCmdShow);
    UpdateWindow(hWndWind);
    while (GetMessage(&msg, NULL, NULL, NULL)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;
    COLORREF colorText = RGB(255, 0, 0);
    switch (uMsg) {
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        GetClientRect(hWnd, &rect);
        SetTextColor(hdc, colorText);
        DrawText(hdc, mainMessage, -1, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
    return NULL;
}
```

IV. WIN32 API. Keyboard shortcuts

In Microsoft Windows, the keyboard accelerator (keyboard accelerator) (or, simply, the accelerator) - a keystroke or combination of keystrokes that are generated for the application program a WM_COMMAND message or WM_SYSCOMMAND.

Regarding the keyboard accelerators

Accelerators are closely related to the menu - both provide the user with access to a set of application program control commands. Typically, users rely on the application menu to explore a set of instructions, and then switch to the use of accelerators as become more experienced in dealing with the application. Accelerators provide faster direct access to more commands than it does the menu. At a minimum, the application must provide accelerators for the most frequently used commands. Although accelerators typically generate commands that exist, such as menu items, they can also create commands that have no such equivalent menu items.

Accelerator table

Table key - Accelerator (accelerator table) consists of an array of ACCEL structures, each of which defines a particular accelerator. Each ACCEL structure includes the following information:

A combination of keystrokes - accelerator
accelerator identifier.

Various flags. These flags include and the one that determines whether Windows provides visual feedback when the accelerator is used, highlighting the menu item if it is available.

To handle the keystrokes - accelerators for a given flow, the developer must call TranslateAccelerator function in a cycle the message associated with the thread's message queue. TranslateAccelerator function monitors keyboard input in the message queue by checking combinations of keys that correspond to items in the table keys - accelerators. When TranslateAccelerator finds a match, it takes the input from the keyboard (ie, posts and WM_KEYUP WM_KEYDOWN) in a WM_COMMAND message or a WM_SYSCOMMAND, and then sends a message to the window procedure of the specified window.

WM_COMMAND message includes an identifier accelerator keys, which made TranslateAccelerator generate the message. The window procedure examines the identifier to determine the source of the message, and then processes the message accordingly.

In Windows accelerator table exist at two different levels. Windows supports a single, system-level accelerator table, which is used by all applications. An application can not change the system table accelerators. For a description of key - accelerator, accelerator table provided by the system, refer to the article Destination keystrokes - accelerators.

Windows also supports table keys - accelerators for each application. The application can define any number of accelerator tables for their own windows. A unique 32-bit descriptor (HACCEL) identifies each table. However, only one table shortcuts - accelerators can be simultaneously active for a given thread. Descriptor table - key accelerators transmitted in TranslateAccelerator function determines which accelerator table is active for the flow. The active accelerator table can be overridden at any time by transmitting a corresponding descriptor accelerator table in TranslateAccelerator.

Creating tables accelerator keys

Wanted a few steps to create an accelerator table for the application. First, use the resource compiler to create resources accelerators tables and add them to the executable file. During the launch of the program, LoadAccelerators function is used to load the accelerator table in memory and obtain its handle. This descriptor is transmitted in TranslateAccelerator function to activate the accelerator table.

accelerator table can also be created for the application and run-time, using an array of ACCEL structures transmission in CreateAcceleratorTable function. This method supports user-defined accelerators in the application program. Like the function LoadAccelerators, CreateAcceleratorTable returns a handle accelerator table, which can be transferred to the TranslateAccelerator, to actuate the accelerator table.

Windows will automatically destroys accelerator table loaded LoadAccelerators. Table accelerators created CreateAcceleratorTable must be destroyed before the application will be closed; otherwise, the table continues to exist in the memory and after the application is closed. Table accelerator is destroyed by calling the function DestroyAcceleratorTable.

The existing accelerator table can be copied and modified. The existing accelerator table is copied by using CopyAcceleratorTable function. After the copy has been modified, the new table descriptor is extracted by means of accelerators CreateAcceleratorTable call. In the end, the handle goes to the TranslateAccelerator, to activate the new table.

Assigning keystrokes - accelerator

To determine the key - the accelerator, ASCII character code may be used, or a virtual key code. ASCII character code makes the key - the accelerator case sensitive. ASCII symbol "C" as the accelerator may determine ALT + c, but not ALT + C. However, sensitive key register - accelerators can be complicated to use. For example, a key - ALT + C accelerator will be generated if the CAPS LOCK key or the SHIFT - pressed, but not if both are pressed.

As a rule, the keys - boosters should not be case sensitive, as most applications do not use the ASCII character codes, and the virtual key codes for the keys - accelerators.

Avoid shortcuts - accelerators, which are in contradiction with the mnemonics application menu because the key - the accelerator overrides the mnemonic character that may confuse the user. For more information about menus, see. Article menu.

If the application specifies the key - the accelerator, which is also defined in the system table accelerators, application-defined key - the accelerator overrides the system accelerator, but only within the context of the application. As you wish, avoid this practice because it prevents the system keys - accelerator in fulfilling its role in the standard Windows user interface. system-level accelerators are described below in the following list:

- ALT + ESC - Switches to the next application (for Windows version 3.1). Switches on the panel application in Windows 95 and higher goals.
- ALT + F4 - Closes the application or window.
- ALT + HYPHEN - Open the System menu for the document window.
- ALT + PRINT SCREEN - Copies the image of the active window to the clipboard.
- ALT + SPACEBAR - Opens the System menu of the application window.
- ALT + TAB - Switch to the next application.
- CTRL + ESC - Switches the task list (for Windows version 3.1) is equivalent to pressing the Start button (for Windows 95 and higher).
- CTRL + F4 - Closes the active group or document window.
- F1 - If the application has a Help (Help), launches Help.
- PRINT SCREEN - Copies of the screen image to the clipboard.
- SHIFT + ALT + TAB - Switch to the previous application. The user must press and hold ALT + SHIFT before pressing the TAB key.

Keys - accelerators and menus

Using the keys - the accelerator is the same action as a menu item: Both actions cause Windows to send a WM_COMMAND message or WM_SYSCOMMAND to the appropriate window procedure.

WM_COMMAND message includes an identifier that window procedure checks to determine the source of the message. If WM_COMMAND message created the accelerator, the identifier - for this key - the accelerator. Similarly, if the WM_COMMAND message in a menu item, the identifier - this menu item. Since the accelerator provides a shortcut to select a command from the menu, the application usually assigns the same identifier for the key - the accelerator and the corresponding menu item.

The application processes the WM_COMMAND message key - the accelerator in exactly the same way as the corresponding WM_COMMAND message menu. However, WM_COMMAND message contains a flag that determines whether the key message created - accelerator or a menu item if it accelerators, they must be processed differently from their corresponding menu items. Post WM_SYSCOMMAND does not contain this option.

ID determines whether a key is generated - the accelerator or a WM_COMMAND message WM_SYSCOMMAND. If the identifier has the same value as a menu item in the System menu, accelerator creates WM_SYSCOMMAND message. Otherwise, the key - the accelerator generates a WM_COMMAND message.

If the key - the accelerator has the same ID as the menu and the menu item is not available or is blocked, the accelerator is blocked and does not generate a WM_COMMAND message or WM_SYSCOMMAND. Key - accelerator also generates a command, if the corresponding window is minimized.

When the user uses the key - the accelerator, which corresponds to a menu item, the window procedure receives messages WM_INITMENU and WM_INITMENUPOPUP, as if the user has selected a menu item. For information on how to handle these messages, refer to the article. Menu.

Key - the accelerator, which corresponds to the menu item should be included in the text of the menu item.

Processing `wm_command` message

When the key - the accelerator, the window defined in TranslateAccelerator function receives a WM_COMMAND or WM_SYSCOMMAND. The low-order word of the wParam parameter contains the ID of the accelerator. The window procedure examines the identifier to determine which source sends WM_COMMAND message and processes the message accordingly.

As a rule, if the key - the accelerator corresponds to a menu item in an application, and the menu item it is assigned the same identifier. If you need to know whether the message WM_COMMAND created an accelerator or a menu item, you can check out the high word of the wParam parameter. If a message is generated key - the accelerator, the high word is 1; if the message has created a menu item, the leading word – 0.

Creating a key - accelerator for font attributes

The example in this section shows how to perform the following tasks:

- Create resource accelerator table.
- Download accelerator table during program run.
- Convert button - boosters in the message loop.
- Process WM_COMMAND messages generated by accelerators.

These tasks are displayed in relation to the application, which includes a menu symbol (Character) and related keys - accelerators, which allow the user to select the current font attributes.

The following part of the resource definition file defines the Character menu and associated accelerator table.

Attention !, that menu items show a keystroke - the accelerator and that each accelerator has the same identifier as the associated menu item.

```
#include <windows.h>
#include "acc.h"

MainMenu MENU
BEGIN
POPUP "&Character"
BEGIN
MENUITEM "&Regular\tF5", IDM_REGULAR
MENUITEM "&Bold\tCtrl+B", IDM_BOLD
MENUITEM "&Italic\tCtrl+I", IDM_ITALIC
MENUITEM "&Underline\tCtrl+U", IDM_ULINE
END
END

FontAccel ACCELERATORS
BEGIN
VK_F5, IDM_REGULAR, VIRTKEY
"B", IDM_BOLD, CONTROL, VIRTKEY
"I", IDM_ITALIC, CONTROL, VIRTKEY
"U", IDM_ULINE, CONTROL, VIRTKEY
END
```

Use accelerator table, created during the program run

Win32 application programming interface (API) allows you to create accelerator table during program run. The steps included in the creation and use of accelerators in the table the following program run:

Key definition - using accelerators fill an array ACCEL structures and then creating accelerator table by transmitting array CreateAcceleratorTable function.

Activating the accelerator table and process WM_COMMAND messages generated by accelerators.

Destruction accelerator table before the application closes.

CREATING accelerator table

The first step in the creation of accelerator table during program run is filling an array of ACCEL structures. Each structure of the array determines the key - the accelerator table. Determination of accelerators include their flags, their keys and IDs. ACCEL structure has the following form.

```
typedef struct tagACCEL { // accl
    BYTE fVirt;
    WORD key;
    WORD cmd;
} ACCEL;
```

You define a keystroke - accelerator, by setting the ASCII letters code or virtual-key code in the term structure of key ACCEL. If you specify a virtual key code, you must first enable the check box at the term FVIRTKEY fVirt; otherwise, Windows will understand the code as ASCII letters code. You can turn FCONTROL box, or FALT FSHIFT, or all three, to be combined with pressing CTRL key, ALT, or SHIFT.

To create an accelerator table, send the address of an array of ACCEL structures CreateAcceleratorTable function. CreateAcceleratorTable creates accelerator table and returns its handle.

PROCESSING accelerator

The loading process and call keys - accelerators under their table, created when you run the program, similar to the treatment of these keys provided resource accelerator table. For more information, see Article to load the resource accelerator table to article processing a WM_COMMAND message.

DESTRUCTION accelerator table

Before application to close, it must destroy the accelerator table, created during the program run. You can destroy the accelerator table and remove it from memory, by sending the descriptor table DestroyAcceleratorTable function.

Creating editable user key - accelerator

This example shows how to create a dialog box that allows the user to change the key - the accelerator associated with the menu item. The dialog consists of a combo box containing menus, combo box containing the names of keys and windows to flag "tick" to select the keys CTRL, ALT and SHIFT.

The following example shows how the dialog is defined in the resource definition.

```
EdAccelBox DIALOG 5, 17, 193, 114
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Edit Accelerators"
BEGIN
COMBOBOX IDD_MENUITEMS, 10, 22, 52, 53,
CBS_SIMPLE | CBS_SORT | WS_VSCROLL |
WS_TABSTOP
CONTROL "Control", IDD_CNTRL, "Button",
BS_AUTOCHECKBOX | WS_TABSTOP,
76, 35, 40, 10
CONTROL "Alt", IDD_ALT, "Button",
BS_AUTOCHECKBOX | WS_TABSTOP,
76, 48, 40, 10
CONTROL "Shift", IDD_SHIFT, "Button",
BS_AUTOCHECKBOX | WS_TABSTOP,
76, 61, 40, 10
COMBOBOX IDD_KEYSTROKES, 124, 22, 58, 58,
CBS_SIMPLE | CBS_SORT | WS_VSCROLL |
WS_TABSTOP
PUSHBUTTON "Ok", IDOK, 43, 92, 40, 14
PUSHBUTTON "Cancel", IDCANCEL, 103, 92, 40, 14
LTEXT "Select Item:", 101, 10, 12, 43, 8
LTEXT "Select Keystroke:", 102, 123, 12,
60, 8
END|
```

The dialog box uses an array of application-defined VKEY structures, each of which contains the text string of a key is pressed and the text string of key - accelerator. When the dialog box is created, it analyzes the array and adds a line of text on each key is pressed in the combo box selection by pressing the button (Select Keystroke). When the user clicks on the button Ok, the dialog box searches for text string to the selected key is pressed, and retrieves the appropriate text string of key - accelerator. The dialog box attaches a text string on the key - the accelerator to the text of the menu item that the user selected. The following example shows an array of structures VKEY:

```

#define MAXKEYS 26

typedef struct _VKEYS {
    char *pKeyName;
    char *pKeyString;
} VKEYS;

VKEYS vkeys[MAXKEYS] = {
    "BkSp", "Back Space",
    "PgUp", "Page Up",
    "PgDn", "Page Down",
    "End", "End",
    "Home", "Home",
    "Lft", "Left",
    "Up", "Up",
    "Rgt", "Right",
    "Dn", "Down",
    "Ins", "Insert",
    "Del", "Delete",
    "Mult", "Multiply",
    "Add", "Add",
    "Sub", "Subtract",
    "DecPt", "Decimal Point",
    "Div", "Divide",
    "F2", "F2",
    "F3", "F3",
    "F5", "F5",
    "F6", "F6",
    "F7", "F7",
    "F8", "F8",
    "F9", "F9",
    "F11", "F11",
    "F12", "F12"
};

```

The initialization routine dialog populates the combo box you select an item (Select Item) and the selection by pressing the button (Select Keystroke). Once the user selects a menu item and the related key - the accelerator, the dialog box checks the controls in the dialog box to get user selection, modify the text of a menu item, and then creates a new accelerator table that contains user-defined a new key - Accelerator . The following example shows the procedure of the dialog.

Guide keyboard accelerators

The following functions, structure and posts related to the keyboard accelerators.

FEATURES keyboard accelerators

The following functions are used with the keys - accelerators.

CopyAcceleratorTable

CreateAcceleratorTable

DestroyAcceleratorTable

LoadAccelerators

TranslateAccelerator

FUNCTIONS COPYACCELERATORTABLE

CopyAcceleratorTable function copies the specified accelerator table. This function is used to get the data accelerator table that corresponds to the descriptor of the table, or define the table size of the data.

Syntax

```
int CopyAcceleratorTable
(
    HACCEL hAccelSrc, // handle copied accelerator table
    LPACCEL lpAccelDst, // pointer to the structure receiving a copy
    int cAccelEntries // number of entries in the copied table
);
```

Options

hAccelSrc

Identifies the accelerator table, which is copied.

lpAccelDst

Points to an array of structures ACCEL, where information on the accelerator table to be copied.

cAccelEntries

Specifies the number of ACCEL structures to copy into the buffer specified by the parameter lpAccelDst.

Return values

If lpAccelDst value is empty (NULL), the return value specifies the number of input accelerator table in the original table. Otherwise, it specifies the number of input accelerator table that has been copied.

See also

ACCEL, CreateAcceleratorTable, DestroyAcceleratorTable, LoadAccelerators, TranslateAccelerator Accommodation and compatibility CopyAcceleratorTable

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION CREATEACCELERATORTABLE

CreateAcceleratorTable function creates an accelerator table.

Syntax

```
HACCEL CreateAcceleratorTable  
(  
LPACCEL lpaccl, // pointer to an array of structures with key data - accelerator  
int cEntries // number of structures in the array  
);
```

Options

lpaccl

Points to an array of structures ACCEL, which describes the accelerator table.

cEntries

Specifies the number of ACCEL structures in the array.

Return values

If the function succeeds, the return value - a handle to the created accelerator table; otherwise, the value is empty (NULL).

remarks

The application before its closure should use DestroyAcceleratorTable function to destroy each accelerator table that are created by using CreateAcceleratorTable function.

See also

ACCEL, CopyAcceleratorTable, DestroyAcceleratorTable, LoadAccelerators, TranslateAccelerator
Accommodation and compatibility CreateAcceleratorTable

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION DESTROYACCELERATORTABLE

DestroyAcceleratorTable function destroys the accelerator table. Before the application is closed, it must use this function to destroy each accelerator table that are created by using CreateAcceleratorTable function.

Syntax

```
BOOL DestroyAcceleratorTable  
(  
HACCEL hAccel // handle accelerator table  
);
```

Options

hAccel

Identifies the accelerator table, which is destroyed. This descriptor must be created using CreateAcceleratorTable function call.

Return values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get the detailed error information, call GetLastError.

See also

[CopyAcceleratorTable](#), [CreateAcceleratorTable](#), [LoadAccelerators](#), [TranslateAccelerator](#)
[Accommodation and compatibility](#) [DestroyAcceleratorTable](#)

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION LOADACCELERATORS

LoadAccelerators function loads the specified accelerator table.

Syntax

```
HACCEL LoadAccelerators
(
HINSTANCE hInstance, // handle to application instance
LPCTSTR lpTableName // address of string with the name of the table
);
```

Options

hInstance

It identifies the instance of the module whose executable file contains the accelerator table to load.

lpTableName

Specifies the line with the symbol of zero at the end, which calls accelerator table to load. Alternatively, this parameter can specify that the table Resource Identifier accelerator in the low word and zero in the high word. It can be used macro MAKEINTRESOURCE, to create this value.

Return values

If the function succeeds, the return value - handle loaded accelerator table.

If the function fails, the return value - NULL (NULL).

remarks

If the accelerator table has not yet been loaded, it is loaded from the specified executable.

Accelerator table loaded from resources are freed automatically when the application finishes its work.

See also

[CopyAcceleratorTable](#), [CreateAcceleratorTable](#), [DestroyAcceleratorTable](#), [MAKEINTRESOURCE](#)
Accommodation and compatibility [LoadAccelerators](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION TRANSLATEACCELERATOR

TranslateAccelerator function treats keys - accelerators for menu commands. The function translates the message WM_KEYDOWN or WM_SYSKEYDOWN in a WM_COMMAND message or WM_SYSCOMMAND (if there is an element to keys in the specified accelerator table), and then transmits the WM_COMMAND or WM_SYSCOMMAND message directly to the appropriate window procedure. TranslateAccelerator does not return as long as the window procedure does not process the message.

Syntax

```
int TranslateAccelerator
(
    HWND hWnd, // handle of destination window
    HACCEL hAccTable, // handle to accelerator table
    LPMMSG lpMsg // address of structure with message
);
```

Options

hWnd

Identifies the window whose messages should be translated.

hAccTable

Identifies the accelerator table. The table must be loaded using LoadAccelerators function call or set up a call CreateAcceleratorTable function.

lpMsg

Points to an MSG structure that contains message information retrieved from the calling thread's message queue by using the function GetMessage or PeekMessage.

Return values

If the function succeeds, the return value - TRUE (TRUE).

If the function fails, the return value - FALSE (FALSE). To get extended error information, call GetLastError.

remarks

To distinguish the message that this function sends from messages sent by menus or controls, the high word of wParam message parameter WM_COMMAND or a WM_SYSCOMMAND, contains the value 1.

Keyboard shortcuts - accelerators used for SELECT items from the window menu are translated into a message WM_SYSCOMMAND; all other shortcuts - accelerators are translated into WM_COMMAND messages.

When TranslateAccelerator returns a nonzero value, and the translated message, the application should not use TranslateMessage function to process the message again.

- Key accelerator does not necessarily have to conform to the menu command.

If the key team - the accelerator corresponds to the menu item, an application sends messages WM_INITMENU and WM_INITMENUPOPUP, as if the user tried to show the menu. However, these messages are not sent when it occurs in any of the following conditions:

The window is blocked.

The menu item is blocked.

The combination of keys - the accelerator does not correspond to an item in the window menu and the window is minimized.

In fact, it was produced mouse capture. For information about mouse capture, call SetCapture function.

If the specified window - active window, and no other window has the keyboard focus (this is usually the case if the window is minimized), TranslateAccelerator translates WM_SYSKEYUP and WM_SYSKEYDOWN messages instead WM_KEYDOWN and WM_KEYUP posts.

If a keystroke is happening - the accelerator, which corresponds to the menu item when the window that owns the menu is minimized, the TranslateAccelerator does not send WM_COMMAND message. However, if the keystroke occurs - the accelerator, which does not correspond to any of the menu items or menu window, the function sends a WM_COMMAND message, even if the window is minimized.

See also

CreateAcceleratorTable, GetMessage, LoadAccelerators, MSG, PeekMessage, SetCapture, TranslateMessage, WM_COMMAND, WM_INITMENU, WM_INITMENUPOPUP, WM_KEYDOWN, WM_SYSKEYDOWN, WM_SYSCOMMAND
Accommodation and compatibility TranslateAccelerator

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

STRUCTURE KEYBOARD ACCELERATORS

Below the following structure is used with the - key accelerators.
ACCEL

MESSAGES KEYBOARD ACCELERATORS

The following messages are used with keys - accelerators.

WM_COMMAND

WM_INITMENU

WM_INITMENUPOPUP

WM_MENUCHAR

WM_MENUSELECT

WM_SYSCHAR

WM_SYSCOMMAND

WM_COMMAND MESSAGE

The WM_COMMAND message is sent when the user selects an item from a menu command, when the control authority sends a notification message to its parent window, or when a keystroke translates - accelerator.

Syntax

```
WM_COMMAND  
wNotifyCode = HIWORD (wParam); // Notification code  
wID = LOWORD (wParam); // Identifier of the menu item, control, or  
// Accelerator keys  
hwndCtl = (HWND) lParam; // Handle of control
```

Options

wNotifyCode

The value of the upper word wParam. Specifies the notification code if the message is from a control. If the message from the key - the accelerator, this parameter is 1. If the message is from a menu, this option - 0.

wID

The value of the lower word wParam. Specifies the identifier of the menu item, the control authority or keys - the accelerator.

hwndCtl

The value of lParam. Identifies the control body that sends the message if the message is from a control. Otherwise, this parameter is set to NULL (NULL).

Return values

If an application processes this message, it should return zero.

Comment

Keystrokes - accelerators, which are selected window menu items are translated into WM_SYSCOMMAND posts.

If pressing takes place - the accelerator, which corresponds to the menu item when the window that owns the menu is minimized, a WM_COMMAND message is not sent. However, if a keystroke - accelerator, which does not correspond to any of the menu items or menu window, a WM_COMMAND message is sent, even if the window is minimized.

If the application includes a menu separator, the system sends a WM_COMMAND message with the low word of the wParam parameter set to zero when the user selects the separator.

See also

WM_SYSCOMMAND

Accommodation and compatibility WM_COMMAND

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_INITMENU MESSAGE

Post WM_INITMENU recovers when the menu is about to become active. This occurs when the user clicks on an item in the menu bar or presses the menu key. This allows an application to modify the menu before it is displayed on the screen.

Syntax

WM_INITMENU

hmenuInit = (HMENU) wParam; // Handle of menu that is initialized

Options

hmenuInit

The value of wParam. Identifies the menu to be initialized.

Return values

If an application processes this message, it returns zero.

remarks

WM_INITMENU message is sent only when the menu turn the first time; create only one message WM_INITMENU for each treatment. For example, moving the mouse over several menu items by holding down the button will not generate any new messages. WM_INITMENU does not provide information about the menu items.

See also

WM_INITMENUPOPUP

Accommodation and compatibility WM_INITMENU

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_INITMENUPOPUP MESSAGE

WM_INITMENUPOPUP message is sent when the pop-up menu or submenu is about to become active. This allows an application to modify the menu before it is displayed on the screen without changing the menu completely.

Syntax

```
WM_INITMENUPOPUP  
hmenuPopup = (HMENU) wParam; // Handle to the submenu  
uPos = (UINT) LOWORD (lParam); // Position of the sub-menu item  
fSystemMenu = (BOOL) HIWORD (lParam); // Check menu window
```

Options

hmenuPopup

The value of wParam. Identifies the pop-up menu or submenu.

uPos

The value of the lower word lParam. Specifies the zero-based relative position of the menu item that opens a pop-up menu or submenu.

fSystemMenu

The value of the upper word lParam. Determines whether the pop-up window menu (also known as the System menu or Control menu (Control)). If this menu - the window menu, this option - TRUE (TRUE); otherwise, it is FALSE (FALSE).

Return values

If an application processes this message, it returns zero.

See also

[WM_INITMENU](#)

Accommodation and compatibility WM_INITMENUPOPUP

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_MENUCHAR MESSAGE

WM_MENUCHAR message sent when the menu is activated and the user presses a key that does not match any character or key - accelerator. This message is sent to the window that owns the menu.

Syntax

```
WM_MENUCHAR  
chUser = (char) LOWORD (wParam); ASCII // symbol  
fuFlag = (UINT) HIWORD (wParam); // Check menu  
hmenu = (HMENU) lParam; // Handle to menu
```

Options

chUser

The value of the lower word wParam. It specifies the ASCII character that corresponds to the key pressed by the user.

fuFlag

The value of the upper word wParam. It specifies the type of the active menu. This parameter can be one of the following values:

MF_POPUP - A pop-up menu, submenu, or shortcut menu

MF_SYSCOMMAND - Window Menu (System Menu) or the Control menu (Control menu)

hmenu

The value of lParam. Identifies the active menu.

Return values

An application that processes this message should return one of the following values in the high word of the return value:

0 - Informs Windows, which operative system must reset the character the user pressed and create a short beep on the system dynamics.

1 - Tells Windows, which operative system should close the active menu.

2 - Tells Windows, which low-order word of the return value sets the zero-based relative position of the menu item. This element is selected Windows.

remarks

The low word is ignored if the high-order word contains 0 or 1. An application should process this message when the key - the accelerator is used to select a menu item that displays a bitmap.

Accommodation and compatibility WM_MENUCHAR

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_MENUSELECT MESSAGE

The message is sent to the window menu WM_MENUSELECT owner, when the user selects a menu item.

Syntax

```
WM_MENUSELECT  
uItem = (UINT) LOWORD (wParam); // Menu item or submenu index  
fuFlags = (UINT) HIWORD (wParam); // Menu flags  
hmenu = (HMENU) lParam; // Handle of menu, which clicked
```

Options

uItem

The value of the lower word wParam. If the selected item - the command post, this parameter contains the identifier of the menu item. If the selected item opens a popup has a menu or submenu, this parameter contains the menu index jumping out a menu or submenu in the main menu, and then hMenu parameter contains the handle of the main (at which clicked) menu; Use GetSubMenu function to get a handle jumping out the menu and sub-menu.

fuFlags

The value of the upper word wParam. Specifies one or more flags menu. This parameter can be a combination of the following values:

MF_BITMAP - item displays a bitmap.

MF_CHECKED - item checked.

MF_DISABLED - blocked item.

MF_GRAYED - item becomes unavailable.

MF_HILITE - highlighted item.

MF_MOUSESELECT - item is selected with the mouse.

MF_OWNERDRAW - Item drawn by the user.

MF_POPUP - item opens a pop-up menu or submenu.

MF_SYSMENU - item is contained in the window menu (also known as the System menu or Control).

Hmenu parameter identifies the window menu associated with the message.

hmenu

The value of lParam. Identifies the menu on which clicked.

Return values

If an application processes this message, it should return zero.

remarks

If fuFlags parameter contains 0xFFFF, and hmenu parameter contains an empty (NULL), then Windows has closed the menu.

Do not use the value for fuFlags - (minus) 1. This is not done because fuFlags defined as (UINT) HIWORD (wParam). If HIWORD (wParam) was 0xFFFF, fuFlags (due to actuation UINT) becomes 0x0000FFFF, rather than - (minus) 1.

See also

[GetSubMenu](#)

Accommodation and compatibility WM_MENUSELECT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_SYSCHAR MESSAGE

WM_SYSCHAR message sent to the window with the keyboard focus when the broadcast message feature WM_SYSKEYDOWN TranslateMessage. It determines the character code system of the letter keys - that is the key symbol, that was pressed at the time when the ALT key was pressed.

Syntax

```
WM_SYSCHAR  
chCharCode = (TCHAR) wParam; // Symbol code  
lKeyData = lParam; // Data on the key
```

Options

chCharCode

The value of wParam. Specifies the character code window menu keys.

lKeyData

The value of lParam. Specifies the repetition through, scan code, check the additional keys, context code, previous key state flag, and transition-state flag, as shown in the following table:

0-15 - Specifies the repeat count. Value - number of times of repeated keystrokes in a result that the user hold.

16-23 - Specifies the scan code. The value depends on the company - original equipment manufacturer (OEM).

24 - Determines whether the additional keys key, type right-ALT and the CTRL keys, which appeared on enhanced 101- or 102-key keyboards. The value is 1 if it - the expansion key, otherwise it - 0.

25-28 - Reserved, do not use.

29 - Specifies the context code. A value of 1 if the ALT key is held down while the key is pressed; otherwise, the value 0.

30 - Specifies the previous key state. A value of 1 if the key is at the bottom before sending the message, or is it - 0 if-not a key is pressed.

31 - Specifies the transition state. A value of 1 if the key release, or is it - 0 when the key is pressed.

Return values

An application should return zero if it processes this message.

remarks

When the context code is zero, the message can be sent to the TranslateAccelerator function, which will handle it as if it were a standard keypad instead of the message system message character keys. This allows the - key accelerators to be used the active window, even if it has no keyboard focus to the active window.

For enhanced 101- and 102-key keyboards, extended keys are ALT and CTRL keys on the right of the main keyboard section; INS, DEL, HOME, END, PAGE UP, PAGE DOWN, and arrow keys in groups to the left of the numeric keypad; key PRINT SCRN; BREAK button; NUMLOCK key; as well as the key factor (/) and ENTER keys in the numeric keypad. Other keyboards may support additional bit keys lKeyData parameter.

See also

TranslateAccelerator, TranslateMessage, WM_SYSKEYDOWN
Accommodation and compatibility WM_SYSCHAR

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_SYSCOMMAND MESSAGE

A window receives this message when the user selects a command from the window menu (also known as the System menu or Control menu) or when the user chooses the Maximize button (the Maximize) or the Minimize button (Minimize).

Syntax

WM_SYSCOMMAND

```
uCmdType = wParam; // The requested type of system commands  
xPos = LOWORD (lParam); // Horizontal position, in screen coordinates  
yPos = HIWORD (lParam); // Vertical position, in screen coordinates
```

Options

uCmdType

It specifies the type of system command requested. It can be one of these values:

- **SC_CLOSE** - Closes the window.
- **SC_CONTEXTHELP** - Changes the pointer to a pointer with a question mark. If the user then clicks the mouse button on the controls in the dialog box, the control receives a message **WM_HELP**.
- **SC_DEFAULT** - Selects the default item; the user double-clicked on the menu window.
- **SC_HOTKEY** - Activates the window associated with the sequence determined by the application accelerator "hot key". Low word of lParam identifies the window that is activated.
- **SC_HSCROLL** - Scroll window content horizontally.
- **SC_KEYMENU** - Retrieves the window menu as a result of pressing.
- **SC_MAXIMIZE** (or **SC_ZOOM**) - Expands the window.
- **SC_MINIMIZE** (or **SC_ICON**) - Minimizes the window.
- **SC_MONITORPOWER** - only for Windows 95: Set the display condition. This command supports devices that have energy-saving features, such as a personal computer with the battery.
- **SC_MOUSEMENU** - Retrieves the window menu as a result of clicking a mouse button.
- **SC_MOVE** - Moves the window.
- **SC_NEXTWINDOW** - Moves to the next screen.
- **SC_PREVWINDOW** - Moves to the previous screen.
- **SC_RESTORE** - Restores the window to the normal position and size.
- **SC_SCREENSAVE** - Starts the screen saver application, defined in the section [boot] (boot) SYSTEM.INI file.
- **SC_SIZE** - Changes the size of the window.
- **SC_TASKLIST** - Starts or activates the Windows Task Manager (Task Manager).
- **SC_VSCROLL** - Scroll window contents vertically.
- **xPos**
- It defines the horizontal position of the cursor, in screen coordinates, if a window menu command is chosen with the mouse. Otherwise, xPos parameter is not used.
- **yPos**

Defines the vertical position of the cursor, in screen coordinates, if a window menu command is chosen with the mouse. This parameter is the - (minus) 1, if the team is selected, using the key - the system accelerator, or zero if used mnemonics.

Return values

An application should return zero if it processes this message.

remarks

DefWindowProc function completes the window menu request for the predefined actions listed in the previous table.

Reports WM_SYSCOMMAND, the four least significant bits uCmdType parameters are used for internal needs of Windows. To get the correct result when checking the values uCmdType, the application must combine the value with the value 0xFFFF0 uCmdType by using bitwise operator AND.

The items on the window menu can be changed by using the functions GetSystemMenu, AppendMenu, InsertMenu, ModifyMenu, InsertMenuItem and SetMenuItem. Applications that modify the window menu must process WM_SYSCOMMAND posts.

An application can execute any command system at any time by sending a message to the DefWindowProc WM_SYSCOMMAND. Any messages WM_SYSCOMMAND, not treated with the program to be transferred to DefWindowProc. Any command values added by an application must be processed by the application and can not be transferred to DefWindowProc.

Accelerator, which are defined to select items from the window menu are translated into WM_SYSCOMMAND posts; all other keystrokes - accelerators are translated into WM_COMMAND messages.

See also

AppendMenu, DefWindowProc, GetSystemMenu, InsertMenu, ModifyMenu, WM_COMMAND
Accommodation and compatibility WM_SYSCOMMAND

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

V. WIN32 API. Icons

Icon (icon) - this pattern, which consists of a bitmap, combined with a mask that creates a transparent area in the figure. This overview describes how to create, display on the screen, and the destruction of duplicate icons. Windows uses icons throughout the user interface to represent objects such as files, folders, shortcuts, applications and documents. **Options** icon in the Microsoft Win32 application programming interface (API) enable applications to create, upload, display on the screen, organize, and to destroy icons. "HOT SPOT" ICONS

One of the pixels in the icon labeled as "hot spot" (hot spot), which is a point by which the system recognizes and sets, for example, the position of the icon. "Hot Spot" icon - usually pixels arranged in its center. If you are using CreateIconIndirect function to create an icon, you can define any pixel that is a "hot spot."

Types of pictograms

The operating system provides a set of standard pictograms (standard icons), which are available for any application to use them at any time. Windows header files contain identifiers for the standard icons - IDs begin with "IDI_" prefix.

Each standard has a corresponding icon is the default image associated with it. You can replace the default image icon associated with any standard cursor at any time.

Custom icons (custom icons) designed for use in a particular application and can be of any kind.

Dimensions of icons

Windows uses four sizes of icons: system small, system large, small and large for the shell to shell.

Small system icon (system small icon) is displayed in the title bar. To change the size of the system small icon, run the applet "Display" in the Control Panel, click the Appearance tab, select Headers Buttons from the list elements, and then set the Size box. To return the size of the system small icon, call the GetSystemMetrics function with the flag and SM_CXSMICON SM_CYSMICON.

Large system icon (system large icon) is mainly used by applications, as well as it appears in the dialog box, switch programs using the Alt + Tab. Functions CreateIconFromResource, DrawIcon, ExtractIcon LoadIcon and all use more system icons. The size of the system large icon is defined by the video driver, therefore it can not be changed. You can return the size of a large system icon by calling GetSystemMetrics with flags and SM_CXICON SM_CYICON.

Functions CreateIcon, CreateIconFromResourceEx CreateIconIndirect and may be used for working with icons and other dimensions, other than the large system icon.

A small icon is a shell (shell small icon) used in Windows Explorer and standard dialog boxes. Currently, this is the default value for the small size of the system. To get the size of a small envelope icon, use SHGetFileInfo function with flags SHGFI_SHELLICONSIZE | SHGFI_SMALLICON, which returns a handle to the system image list, and then function ImageList_GetIconSize, to get the size of the icons.

Large envelope icon (shell large icon) used on the desktop. To change the size of a large icon, run the applet "Display" in the Control Panel, click on the Effects tab (Windows 98), select the icon of a list item, and then set the field size (the size is stored in the registry under HKEY_CURRENT_USER directory \ CONTROL Panel, Desktop \ WindowMetrics \ Shell Icon Size). You can also click on the tab of the Plus! and you check the box marked Use large icons (Use large icons). To return the size of a large envelope icon, use SHGetFileInfo function SHGFI_SHELLICONSIZE, which returns a handle to the system image list, and then ImageList_GetIconSize function to get the size of the icon.

The Start menu (Start) uses a shell or small icons or large icons shell, depending on whether the check box next to Use large icons lettering set (Use large icons).

Your application must provide resources icons in the following sizes:

48x48, 256 colors

32x32, 16 colors

16x16 pixels, 16 colors

When filling WNDCLASSEX structure, which is used when registering your window class, set its Member hIcon a 32x32 icon, as well as a member of the hIconSm 16x16 icon. For more information about the classes icons, see. Article class icon.

Create icons

Standard icons are predetermined, so there is no need to create them. To use a standard icon, an application can get its handle using LoadImage function. Handle icon (icon handle) - a unique value type HICON, which identifies a standard or custom icon.

To create a custom icon for the application, developers usually use a program to work with graphics and include resource ICONS (ICON) in the application of the resource definition file. During startup, the application can call or the LoadImage LoadIcon, to get a handle icon. Resource icons contains data for several different display devices. LoadIcon LoadImage and automatically selects the most appropriate data for the current display device.

The application can also create a custom icon to start the program period, by using CreateIconIndirect function, which creates a thumbnail based on ICONINFO structure content. GetIconInfo function fills the structure coordinates of the "hot spot" and information on the bit mask and color bitmap icons.

Applications should perform custom icons as resources and must use LoadIcon or LoadImage, which is preferable than the creation of the icon during the start of the program. Resource Usage icon helps to avoid depending on the device, it simplifies locating and allows applications to share the image of the icon.

CreateIconFromResourceEx function allows an application to view the system resources and create icons and cursors based on resource data. CreateIconFromResourceEx function creates an icon based on the binary resource data from other executable files and dynamic link libraries (DLL). The application must before calling this function refer to LookupIconIdFromDirectoryEx features and multiple functions of the resource. LookupIconIdFromDirectoryEx function returns the identifier of the most relevant data icon for the current display device. For more information about resource functions, see Article resource functions.

Display icons on the screen

An application can get a thumbnail picture using GetIconInfo function, and can draw it using DrawIconEx function. To draw the default image icon, set DI_COMPAT flag when calling DrawIconEx. If you do not set DI_COMPAT box, DrawIconEx draws the icon using the image that the user has defined.

Destruction of icons

When the application no longer needs the icon, which she created using CreateIconIndirect function, it must destroy the icon. DestroyIcon function destroys the handle to the icon and frees any memory used for it. Applications should use this function only for icons, created with CreateIconIndirect; it is not necessary to destroy the other icons.

Override icons

CopyIcon function copies descriptor icons. This allows the application or dynamic-link library (DLL) to get your own descriptor icons belonging to another module. Then, if another module is released, an

application that copied the icon will still get the opportunity to use it.

CopyImage function creates a new icon based on the set of the original icon. The new icon may be greater or less than the original icon.

For information about adding, removing, or replacing an icon resource in an executable (.EXE) file, refer to the article Resources.

Use of pictograms

This following topics describe how to perform certain tasks related icons:

Creating thumbnails

Show icons on screen

Sharing icon resources

Create icons

To use the icon, your application must get its handle. The following example shows how to create two different descriptor icons: one for the standard icon with an exclamation mark and the second for the custom icons are included as a resource in the application's resource-definition file.

```
HICON hIcon1; // Handle icon
HICON hIcon2; // Handle icon
// Create a standard icon with a question mark.
hIcon1 = LoadIcon(NULL, IDI_QUESTION);
// Create a custom icon, based on the resource.
hIcon2 = LoadIcon(hinst, MAKEINTRESOURCE(460));
// Create a personalized icon of the program period.
```

The application must perform custom icons as resources and should be used LoadIcon LoadImage or function is preferred, but not to create an icon of the program period. This approach avoids depending on the device, it simplifies locating and allows programs to share bitmaps icons. However, the following example uses CreateIcon, to create a custom icon for the period of the program, based on raster bit masks; it is included in order to illustrate how the system interprets the bit-mask bitmap icons.

```
HICON hIcon3; // Handle icon
// Light (yang) AND bit mask icon
```

```
BYTE ANDmaskIcon [] = {
0xFF, 0xFF, 0xFF, 0xFF, // line 1
0xFF, 0xFF, 0xC3, 0xFF, // line 2
0xFF, 0xFF, 0x00, 0xFF, // line 3
0xFF, 0xFE, 0x00, 0x7F, // line 4
```

```
0xFF, 0xFC, 0x00, 0x1F, // line 5
0xFF, 0xF8, 0x00, 0x0F, // line 6
0xFF, 0xF8, 0x00, 0x0F, // line 7
0xFF, 0xF0, 0x00, 0x07, // line 8
```

```
0xFF, 0xF0, 0x00, 0x03, // line 9
0xFF, 0xE0, 0x00, 0x03, // line 10
0xFF, 0xE0, 0x00, 0x01, // line 11
0xFF, 0xE0, 0x00, 0x01, // line 12
```

```
0xFF, 0xF0, 0x00, 0x01, // line 13
0xFF, 0xF0, 0x00, 0x00, // line 14
0xFF, 0xF8, 0x00, 0x00, // line 15
0xFF, 0xFC, 0x00, 0x00, // line 16
```

```
0xFF, 0xFF, 0x00, 0x00, // line 17
0xFF, 0xFF, 0x80, 0x00, // line 18
0xFF, 0xFF, 0xE0, 0x00, // line 19
0xFF, 0xFF, 0xE0, 0x01, // line 20
```

```
0xFF, 0xFF, 0xF0, 0x01, // line 21
0xFF, 0xFF, 0xF0, 0x01, // line 22
0xFF, 0xFF, 0xF0, 0x03, // line 23
0xFF, 0xFF, 0xE0, 0x03, // line 24

0xFF, 0xFF, 0xE0, 0x07, // line 25
0xFF, 0xFF, 0xC0, 0x0F, // line 26
0xFF, 0xFF, 0xC0, 0x0F, // line 27
0xFF, 0xFF, 0x80, 0x1F, // line 28

0xFF, 0xFF, 0x00, 0x7F, // line 29
0xFF, 0xFC, 0x00, 0xFF, // line 30
0xFF, 0xF8, 0x03, 0xFF, // line 31
0xFF, 0xFC, 0x3F, 0xFF}; // Line 32

// Light (yang) XOR bit mask icon

BYTE XORmaskIcon [] = {
0x00, 0x00, 0x00, 0x00, // line 1
0x00, 0x00, 0x00, 0x00, // line 2
0x00, 0x00, 0x00, 0x00, // line 3
0x00, 0x00, 0x00, 0x00, // line 4

0x00, 0x00, 0x00, 0x00, // line 5
0x00, 0x00, 0x00, 0x00, // line 6
0x00, 0x00, 0x00, 0x00, // line 7
0x00, 0x00, 0x38, 0x00, // line 8

0x00, 0x00, 0x7C, 0x00, // line 9
0x00, 0x00, 0x7C, 0x00, // line 10
0x00, 0x00, 0x7C, 0x00, // line 11
0x00, 0x00, 0x38, 0x00, // line 12

0x00, 0x00, 0x00, 0x00, // line 13
0x00, 0x00, 0x00, 0x00, // line 14
0x00, 0x00, 0x00, 0x00, // line 15
0x00, 0x00, 0x00, 0x00, // line 16

0x00, 0x00, 0x00, 0x00, // line 17
0x00, 0x00, 0x00, 0x00, // line 18
0x00, 0x00, 0x00, 0x00, // line 19
0x00, 0x00, 0x00, 0x00, // line 20

0x00, 0x00, 0x00, 0x00, // line 21
0x00, 0x00, 0x00, 0x00, // line 22
0x00, 0x00, 0x00, 0x00, // line 23
0x00, 0x00, 0x00, 0x00, // line 24

0x00, 0x00, 0x00, 0x00, // line 25
```

```
0x00, 0x00, 0x00, 0x00, // line 26
0x00, 0x00, 0x00, 0x00, // line 27
0x00, 0x00, 0x00, 0x00, // line 28

0x00, 0x00, 0x00, 0x00, // line 29
0x00, 0x00, 0x00, 0x00, // line 30
0x00, 0x00, 0x00, 0x00, // line 31
0x00, 0x00, 0x00, 0x00}; // Line 32

hIcon3 = CreateIcon (hinst, // application instance
32 // width pictograms
32, // height of icons
1, // number of XOR planes
1, // number of bits per pixel
ANDmaskIcon, // bit mask AND
XORmaskIcon); // Bitmask XOR
```

Display icons on the screen

Your application can upload and create icons to display them in the working area of the application program, or child windows. The following example shows how to draw the icon on the window, the display device context (DC) which is identified by a parameter hdc.

```
HICON hIcon1; // Handle icon  
HDC hdc; // Handle to the display device  
DrawIcon (hdc, 10, 20, hIcon1);
```

Windows automatically displays on screen the icon (s) of the class window. Your application can assign a class icon with the window class registration. Your application can replace the class icons using SetClassLong function. This function changes the default window settings for all windows in this class. The following example changes the class icon on the icon whose resource identifier is 480.

```
HINSTANCE hinst; // Handle of current instance  
HWND hwnd; // Handle to the main window  
  
// Change the icon for the class hwnd's window  
  
SetClassLong (hwnd, // window handle changes the icon  
GCL_HICON,  
(LONG) LoadIcon (hinst, MAKEINTRESOURCE (480))  
);
```

For more information about window classes, see. Classes window.

Sharing resources icons

The following code uses CreateIconFromResourceEx function, DrawIcon
LookupIconIdFromDirectoryEx and some of the functions of the resource to create a pictogram
descriptor based on its data from another executable. Then, it displays an icon on the screen in a
window.

```
HICON hIcon1; // Handle icon
HINSTANCE hExe; // Handle downloaded .EXE file
HRSRC hResource; // Handle to FindResource (resource search)
HRSRC hMem; // Handle to LoadResource (load share)
BYTE * lpResource; // Address of resource data
int nID; // ID (identifier of the resource, which is best
// Matches the current screen

HDC hdc; // Handle to the display context

// Load the file from which the icon is copied.
hExe = LoadLibrary ( "myapp.exe");

// Search icon directory, where identifier - 440.
hResource = FindResource (hExe,
MAKEINTRESOURCE (440)
RT_GROUP_ICON);

// Load and view icons directory.
hMem = LoadResource (hExe, hResource);

lpResource = LockResource (hMem);

// Get the ID of the icon, which should correspond
// Video display.
nID = LookupIconIdFromDirectoryEx ((PBYTE) lpResource, TRUE,
CXICON, CYICON, LR_DEFAULTCOLOR);

// Find the bits nID (ID) icon.

hResource = FindResource (hExe,
MAKEINTRESOURCE (nID),
MAKEINTRESOURCE (RT_ICON));

// To download and view the icon.
hMem = LoadResource (hExe, hResource);

lpResource = LockResource (hMem);

// Create a handle icon.
```

```
hIcon1 = CreateIconFromResourceEx ((PBYTE) lpResource,  
SizeofResource (hExe, hResource), TRUE, 0x00030000,  
CXICON, CYICON, LR_DEFAULTCOLOR);
```

```
// Draw the icon in the work area.  
DrawIcon (hdc, 10, 20, hIcon1);
```

Guide for icons

Below are listed the functions and structure of messages used with pictograms.

FUNCTION PICTOGRAM

The following are the functions that are used icons:

[CopyIcon](#)
[CreateIcon](#)
[CreateIconFromResource](#)
[CreateIconFromResourceEx](#)
[CreateIconIndirect](#)
[DestroyIcon](#)
[DrawIcon](#)
[DrawIconEx](#)
[GetIconInfo](#)
[LoadIcon](#)
[LookupIconIdFromDirectory](#)
[LookupIconIdFromDirectoryEx](#)

FUNCTION COPYICON

CopyIcon function copies a defined icon from another module in the current module.

Syntax

```
HICON CopyIcon  
(  
HICON hIcon // handle to the icon, which is copied  
);
```

Options

hIcon

It identifies the icon that will be copied.

Return values

If the function succeeds, the return value - handle duplicate icons.

If the function fails, the return value - NULL (NULL). To get extended error information, call GetLastError.

remarks

CopyIcon function allows an application or dynamic-link library (DLL) to get your own handle for the icon belonging to another module. If another unit is released, the program icon will still get the opportunity to use the icon.

See also

[CopyCursor](#), [DrawIcon](#), [DrawIconEx](#)
[Accommodation and compatibility CopyIcon](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CREATEICON

CreateIcon function creates an icon that has a certain size, color and bit patterns.

Syntax

```
HICON CreateIcon
{
HINSTANCE hInstance, // handle to application instance
int nWidth, // width of the icon
int nHeight, // height of icons
BYTE cPlanes, // number of planes in the XOR bitmask
BYTE cBitsPixel, // number of bits per pixel in the XOR bitmask
CONST BYTE * lpbANDbits, // pointer to an array of bit masks AND
CONST BYTE * lpbXORbits // pointer to XOR bitmask array
);
```

Options

hInstance

It identifies the instance of the module that creates the icon.

nWidth

Specifies the width of the icon, in pixels.

nHeight

Specifies the height of the icon, in pixels.

cPlanes

Specifies the number of planes in the XOR bitmask of the icon.

cBitsPixel

Specifies the number of bits per pixel in the XOR bitmask of the icon.

lpbANDbits

Points to an array of bytes that contains the bit values for the AND bitmask of the icon. This bitmask describes a monochrome bitmap.

lpbXORbits

It points to an array of bytes that contains the bit values for the XOR bitmask of the icon. This bitmask describes a monochrome or color device-dependent bitmap.

Return values

If the function succeeds, the return value - a handle icon.

If the function fails, the return value is empty (NULL). To get extended error information, call

GetLastError.

remarks

Parameters nWidth and nHeight must determine the width and height supported by the current display driver, because the system can not create icons of other sizes. To determine the width and height supported by the display driver, use the GetSystemMetrics function, specifying a value or SM_CXICON SM_CYICON.

CreateIcon applies the following truth table for XOR bit mask and AND:

See also

[GetSystemMetrics](#)

[Accommodation and compatibility CreateIcon](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CREATEICONFROMRESOURCE

(SozdatPiktogrammuIzResursa)

CreateIconFromResource function creates an icon or cursor from resource bits describing the icon.

Syntax

```
HICON CreateIconFromResource
(
    PBYTE presbits, // pointer to the bits of the icon or cursor
    DWORD dwResSize, // number of bytes in the buffer bits
    BOOL fIcon, // flag icon or cursor
    DWORD dwVer // version of Windows format
);
```

Options

presbits

It points to the buffer that contains the resource bits of the icon or cursor. These bits are typically loaded by calling LookupIconIdFromDirectory functions (in Windows 95, you can also call LookupIconIdFromDirectoryEx) and LoadResource.

dwResSize

Specifies the size, in bytes, of a set of bits specified parameter presbits.

fIcon

Specifies that should be set up - the icon or cursor. If this option - TRUE (TRUE), the icon is to be created. If he - FALSE (FALSE), the cursor is to be created.

dwVer

It specifies the version number of the icon or cursor format for the resource bits specified parameter presbits. This parameter can be one of the following values:

Format dwVer

Windows 2.x 0x00020000

Windows 3.x 0x00030000

All applications based on the Microsoft Win32, using icons and cursors for Windows 3.x. format

Return values

If the function succeeds, the return value - a handle icon or cursor.

If the function fails, the return value - NULL (NULL). To get extended error information, call GetLastError.

remarks

Functions CreateIconFromResource, CreateIconIndirect, GetIconInfo and LookupIconIdFromDirectory (in Windows 95 and CreateIconFromResourceEx LookupIconIdFromDirectoryEx function) allow shell application and viewer icon to check and use the resources of the entire system.

See also

CreateIconFromResource, CreateIconFromResourceEx, CreateIconIndirect, GetIconInfo,

LoadResource, LookupIconIdFromDirectory, LookupIconIdFromDirectoryEx
Accommodation and compatibility CreateIconFromResource

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CREATEICONFROMRESOURCEEX

(SozdatPiktogrammuIzResursaRasshirennaya)

[Now Supported on Windows NT]

CreateIconFromResourceEx function creates an icon or cursor from resource bits describing the icon.

Syntax

```
HICON CreateIconFromResourceEx
(
    PBYTE pbIconBits, // pointer to the bits of the icon or cursor
    DWORD cbIconBits, // number of bytes in the buffer bits
    BOOL fIcon, // flag icon or cursor
    DWORD dwVersion, // version of Windows format
    int cxDesired, // desired width of the icon or cursor
    int cyDesired, // desired height of the icon or cursor
    UINT uFlags // flags downloadable resource
);
```

Options

pbIconBits

It points to the buffer that contains the resource bits of the icon or cursor. These bits are typically loaded by calling `LookupIconIdFromDirectoryEx` and `LoadResource` functions.

cbIconBits

Specifies the size, in bytes, bits are set, the specified parameter `pbIconBits`.

fIcon

Specifies that should be set up - the icon or cursor. If this option - TRUE (TRUE), the icon is to be created. If it - FALSE (FALSE), the cursor is to be created.

dwVersion

It specifies the version number of the icon or cursor format for the resource bits specified parameter `pbIconBits`. This parameter can be one of the following values:

Format `dwVersion`

Windows 2.x 0x00020000

Windows 3.x 0x00030000

All applications based on the Win32, use the Windows 3.x format for icons and cursors.

cxDesired

It specifies the desired width, in pixels, of the icon or cursor. If this parameter is zero, the function uses to set the width of the value of the system parameter or `SM_CXICON` `SM_CXCURSOR`.

cyDesired

It determines the desired height, in pixels, of the icon or cursor. If this parameter is zero, the function uses to set the height, the value of the system parameter or `SM_CYICON` `SM_CYCURSOR`.

uFlags

Specifies a combination of the following values:

`LR_DEFAULTCOLOR` - Uses the default color format.

`LR_MONOCHROME` - Creates a monochrome icon or cursor.

Return values

If the function succeeds, the return value - a handle icon or cursor.

If the function fails, the return value - NULL (NULL). To get extended error information, call GetLastError.

remarks

Functions CreateIconFromResourceEx, CreateIconFromResource, CreateIconIndirect, GetIconInfo and LookupIconIdFromDirectoryEx allow shell application and viewer icon to check and use the resources of the entire system.

See also

BITMAPINFOHEADER, CreateIconFromResource, CreateIconIndirect, GetIconInfo, LoadResource, LookupIconIdFromDirectoryEx

Accommodation and compatibility CreateIconFromResourceEx

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CREATEICONINDIRECT

CreateIconIndirect function creates an icon or cursor from ICONINFO structure.

Syntax

```
HICON CreateIconIndirect  
(  
PICONINFO piconinfo // pointer to structure information on the icon  
);
```

Options

piconinfo

Indicates CONINFO structure that the function uses to create the icon or cursor.

Return values

If the function succeeds, the return value - a handle icon or cursor that is created.

If the function fails, the return value - NULL (NULL). To get extended error information, call GetLastError.

remarks

The system copies the structure of bitmaps ICONINFO before creating the icon or cursor. The application must continue to manage the original bitmaps and delete them when they are no longer needed.

When you finish using the icon, destroy it using DestroyIcon function.

See also

DestroyIcon, ICONINFO

Accommodation and compatibility CreateIconIndirect

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION DESTROYICON

DestroyIcon function destroys an icon and frees any memory occupied by it.

Syntax

```
BOOL DestroyIcon  
(  
HICON hIcon // handle to the icon that should be destroyed  
);
```

Options

hIcon

It identifies the icon that will be destroyed. Icon should not be in use.

Return values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.
remarks

DestroyIcon function needs to be called only for icons created CreateIconIndirect function.

See also

CreateIconIndirect

Accommodation and compatibility DestroyIcon

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION DRAWICON

DrawIcon function draws an icon on the window specified device context.

Syntax

```
BOOL DrawIcon
(
    HDC hDC, // handle to device context
    int X, // x-coordinate of the upper left corner
    int Y, // y- coordinate of the upper left corner
    HICON hIcon // handle to the icon, which should draw
);
```

Options

hDC

Identifies the device context for the window.

X

Specifies the logical x-coordinate of the upper-left corner of the icon.

Y

Specifies the logical y-coordinate of the upper-left corner of the icon.

hIcon

It identifies the icon that will be drawn.

Windows NT: icon resource must be preloaded, by using LoadIcon function.

Windows 95: resource icon should be preloaded, by using LoadIcon LoadImage or function.

Return values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.
remarks

DrawIcon places the upper-left corner of the icon to the location determined by the X and Y parameters. Location is subject to the current mode of the display device context.

See also

CreateIcon, DrawIconEx, LoadIcon

Accommodation and compatibility DrawIcon

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION DRAWICONEX

DrawIconEx function draws an icon or cursor on the window, determine the device context, performing the actions defined bitmap, and stretching or compressing the icon or cursor as specified.

Syntax

```
BOOL DrawIconEx
(
    HDC hdc, // handle to device context
    int xLeft, // x-coordinate of the upper left corner
    int yTop, // y-coordinate of the upper left corner
    HICON hIcon, // handle to the icon, which should draw
    int cxWidth, // width of the icon
    int cyWidth, // height of icons
    UINT istepIfAniCur, // index of the frame in the living index
    HBRUSH hbrFlickerFreeDraw, // handle to the background brush
    UINT diFlags // flags being drawn icons
);
```

Options

hdc

Identifies the device context for the window.

xLeft

Specifies the logical x-coordinate of the upper-left corner of the icon or cursor.

yTop

Specifies the logical y-coordinate of the upper-left corner of the icon or cursor. hIcon

Identifies the icon or cursor that will be drawn. This parameter can identify an animated cursor. Source icons or the cursor must be preloaded by using LoadImage function.

cxWidth

It specifies the logical width of the icon or cursor. If this parameter is zero and the parameter diFlags - DI_DEFAULTSIZE, the function to set the width, uses a metric value SM_CXICON or SM_CXCURSOR system. If this parameter is zero, and DI_DEFAULTSIZE not used, the function uses the actual resource width.

cyWidth

It specifies the logical height of the icon or cursor. If this parameter is zero, and setting diFlags - DI_DEFAULTSIZE, the function to set the height, use the metric value or SM_CYICON SM_CYCURSOR system. If this parameter is zero, and DI_DEFAULTSIZE not used, the function uses the actual height of the resource.

istepIfAniCur

It specifies the index of the frame to draw if hIcon identifies an animated cursor. This parameter is ignored if hIcon does not identify an animated cursor.

hbrFlickerFreeDraw

Identifies a brush that the system uses for flicker-free picture. If hbrBkgnd - valid brush handle, the system creates an offscreen bitmap using the specified brush for the background color, draws the icon or cursor into a bitmap and then copy it to the device context identified by hdc. If hbrBkgnd matters BLANK (NULL), the system draws the icon or cursor directly into the device context.

diFlags

Specifies the drawing flags. This parameter can be one of the following values:

DI_COMPAT - Draws the icon or cursor using the image defined by the system, rather than a user-defined image.

DI_DEFAULTSIZE - If the parameters cxWidth and cyWidth set to zero, draws the icon or cursor using the width and height values determined metric system for cursors or icons. If this option is not selected, and the parameters and cxWidth cyWidth set to zero, the function uses the actual size of the resource.

DI_IMAGE - Performs raster operation defined ropImage element.

DI_MASK - Performs raster operation defined ropMask element.

DI_NORMAL - Combination DI_IMAGE and DI_MASK.

Return values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError
remarks

DrawIconEx function places the upper left corner of the icon to the position determined by the parameters and xLeft yTop. Location is subject to the current mode of the display device context.

See also

[CopyImage](#), [DrawIcon](#), [LoadImage](#)

Accommodation and compatibility [DrawIconEx](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETICONINFO

GetIconInfo function obtains information about the specified icon or cursor.

Syntax

```
BOOL GetIconInfo
(
    HICON hIcon, // handle to icon
    PICONINFO piconinfo // address of structure icons
);
```

Options

Identifies the icon or cursor. To obtain information about a standard icon or cursor, specify one of the following:

IDC_ARROW - arrow cursor
IDC_IBEAM - Cursor in a straight line - I
IDC_WAIT - cursor to an hourglass
IDC_CROSS - Cursor in the form of a cross
IDC_UPARROW - arrow cursor up
IDC_SIZENWSE - Set the cursor size, it points to the north-west and south-east
IDC_SIZENESW - Set the cursor size, points to the north-east and south-west
IDC_SIZEWE - Set the cursor size, points to the east and west
IDC_SIZENS - Set the cursor size, points to the north and south
IDC_SIZEALL - Set the cursor size, points north, south, east, and west
IDC_NO - Cursor "No"
IDC_APPSTARTING - Cursor run the application (arrow and hourglass)
IDC_HELP - reference cursor (arrow and question mark)
IDI_APPLICATION - application icon
IDI_HAND - icon with the "stop" signal
IDI_QUESTION - icon with a question mark
IDI_EXCLAMATION - An icon with an exclamation mark
IDI_ASTERISK - Icon with an asterisk (the letter "i" in a circle)
IDI_WINLOGO - icon with the Windows logo
piconinfo
Indicates structure ICONINFO. The function fills in the structure members.

Return values

If the function succeeds, the return value is nonzero and the function fills the elements defined ICONINFO structure.

If the function fails, the return value is zero. To get extended error information, call GetLastError.
remarks

GetIconInfo function creates bitmaps for members and hbmMask hbmColor ICONINFO structure. The calling application must manage these bitmaps and delete them when they are no longer needed.

See also

[CreateIcon](#), [CreateIconFromResource](#), [CreateIconIndirect](#), [DestroyIcon](#), [DrawIcon](#), [DrawIconEx](#),
[ICONINFO](#), [LoadIcon](#), [LookupIconIdFromDirectory](#)
Accommodation and compatibility [GetIconInfo](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION LOADICON

LoadIcon function loads the specified resource icons from the executable (.EXE) file that is associated with an instance of the application.

Syntax

```
HICON LoadIcon  
(  
HINSTANCE hInstance, // handle to application instance  
LPCTSTR lpIconName // string with the name or icon resource identifier  
);
```

Options

hInstance

Identifies the instance of the module whose executable file contains the icon that will be downloaded. This parameter must be set to NULL (NULL), when loaded standard icon.

lpIconName

Specifies the line with the symbol of zero at the end, which contains the name of the resource icon to be loaded. Alternatively, this parameter can contain the resource identifier in the low-order word and zero in the high word. Use macro MAKEINTRESOURCE, to create this value.

To use one of the predefined icons Windows, insert the hInstance parameter to NULL value (NULL), and lpIconName option in one of the following:

IDI_APPLICATION - Icon the default application.

IDI_ASTERISK - Asterisk (used in news reports).

IDI_EXCLAMATION - exclamation point (used in warning messages).

IDI_HAND - An icon that has a hand (used in severe warning messages).

IDI_QUESTION - Question mark (used for tooltips).

IDI_WINLOGO - Logo Windows.

Return values

If the function succeeds, the return value - handle to the newly loaded icon.

If the function fails, the return value - NULL (NULL). To get extended error information, call GetLastError.

remarks

LoadIcon resource loads icons only if it was not loaded; otherwise, the function receives a handle to an existing resource. The function searches the icon resource that best fits the current display. Resource icons can be in color or monochrome bitmap.

LoadIcon can only load an icon whose size corresponds to the metric system, and values SM_CXICON SM_CYICON. Use LoadImage feature to upload icons that are a different size.

See also

CreateIcon, LoadImage, MAKEINTRESOURCE

Accommodation and compatibility LoadIcon

Windows NT Yes
Win95 Yes
Yes Win32s
Imported Library user32.lib
Header file winuser.h
Unicode WinNT
Platform Notes None

FUNCTION LOOKUPICONIDFROMDIRECTORY

LookupIconIdFromDirectory function searches among the icons or cursors data that best fits the current display device.

Syntax

```
int LookupIconIdFromDirectory
(
PBYTE presbits, // address of resource data
BOOL fIcon // appearance of the icon or cursor
);
```

Options

presbits

Specifies the data directory icon or cursor. Since this function does not check the correctness of the data resource, it can cause a general protection fault (GP) or returns an undefined value if presbits indicates not correct resource data.

fIcon

Specifies that was wanted, the icon or cursor. If this option - TRUE (TRUE), the function searches for the icon; if the parameter - FALSE (FALSE), the function searches the cursor.

Return values

If the function succeeds, the return value - the integer resource identifier for the icon or cursor that best correspond to the current display device.

remarks

resource type File RT_GROUP_ICON (RT_GROUP_CURSOR designating cursors) contains data icon (or cursor) in several device-dependent and not dependent on the size of the equipment.

LookupIconIdFromDirectory looks for a file resource icon (or cursor) that best correspond to the current display device and returns an integer identifier. Functions FindResource FindResourceEx and this identifier is used macro MAKEINTRESOURCE, to determine the resource module.

Catalog icon is loaded from a resource file with resource type RT_GROUP_ICON (or RT_GROUP_CURSOR for cursors) and the integer value of the resource name for a particular icon to be loaded. LookupIconIdFromDirectory returns an integer identifier that is the name of a resource icon that best fits the current display device.

Functions LoadIcon, LoadCursor and LoadImage (in Windows 95) use this function to find the specified resource data for the icon or cursor that best correspond to the current display device.

See also

CreateIconFromResource, CreateIconIndirect, FindResource, FindResourceEx, GetIconInfo, LoadCursor, LoadIcon, LoadImage, LookupIconIdFromDirectoryEx, MAKEINTRESOURCE Accommodation and compatibility LookupIconIdFromDirectory

Windows NT Yes

Win95 Yes
Yes Win32s
Imported Library user32.lib
Header file winuser.h
No Unicode
Platform Notes None

FUNCTION LOOKUPICONIDFROMDIRECTORYEX

[Now Supported Windows NT]

LookupIconIdFromDirectoryEx function searches among the icons or cursors data that best fits the current display device.

Syntax

```
int LookupIconIdFromDirectoryEx
(
PBYTE presbits, // address of resource data
BOOL fIcon, // flag icon or cursor
int cxDesired, // desired width of the icon or cursor
int cyDesired, // desired height of the icon or cursor
UINT Flags // resource flags
);
```

Options

presbits

Specifies the data directory icon or cursor. Since this function does not check the correctness of the data resource, it can cause a general protection fault (GP) or returns an undefined value if presbits indicates no valid resource data.

fIcon

Specifies that are wanted, the icon or cursor. If this option - TRUE (TRUE), the function searches for the icon; if the parameter - FALSE (FALSE), the function searches the cursor.

cxDesired

It determines the desired width of the icon, in pixels. If this parameter is zero, the function uses the metric value or SM_CXICON SM_CXCURSOR system.

cyDesired

Icon determines the desired height in pixels. If this parameter is zero, the function uses the metric value or SM_CYICON SM_CYCURSOR system.

Flags

Specifies a combination of the following values:

Checkbox What it means

LR_DEFAULTCOLOR Uses the default color format.

LR_MONOCHROME Creates a monochrome icon or cursor.

Return values

If the function succeeds, the return value - the integer resource identifier for the icon or cursor that best fits the current display device.

remarks

resource type File RT_GROUP_ICON (RT_GROUP_CURSOR indicates cursors) contains data on the icon (or cursor) in several device-dependent and not dependent on the size of the equipment.

LookupIconIdFromDirectoryEx looks for a file resource icon (or cursor) that best fits the current

display device and returns an integer identifier. FindResource and FindResourceEx functions use MAKEINTRESOURCE macro with this identifier to determine the resource module.

Catalog icon is loaded from a resource file with resource type RT_GROUP_ICON (or RT_GROUP_CURSOR for cursors) and the integer value of the resource name for a particular icon to be loaded. LookupIconIdFromDirectoryEx function returns an integer identifier that is the name of a resource icon that best fits the current display device.

Functions LoadIcon, LoadImage LoadCursor and use this function to find data specified resource for the icon or cursor that best fits the current display device.

See also

CreateIconFromResourceEx, CreateIconIndirect, FindResource, FindResourceEx, GetIconInfo, LoadCursor, LoadIcon, LoadImage, LookupIconIdFromDirectory, MAKEINTRESOURCE Accommodation and compatibility LookupIconIdFromDirectoryEx

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

Structure of icons

The following structures are used with pictograms.

ICON INFO

ICON METRICS

STRUCTURE ICONINFO

ICONINFO structure contains information about the icon or cursor.

Syntax

```
typedef struct _ICONINFO
{
    BOOL fIcon;
    DWORD xHotspot;
    DWORD yHotspot;
    HBITMAP hbmMask;
    HBITMAP hbmColor;
} ICONINFO;
```

The members of the structure

fIcon

Sets that defines this structure, the icon or cursor. The value TRUE (TRUE) determines the icon; FALSE (FALSE) defines the cursor.

xHotspot

Specifies the x-coordinate of the tip ("hot spot") the cursor. If this structure defines an icon tip - always in the center of the icon, as this term is ignored.

yHotspot

Specifies the y-coordinate of the tip ("hot spot") the cursor. If this structure defines an icon tip - always in the center of the icon, as this term is ignored.

hbmMask

Sets the bit mask bitmap icons. If this structure defines a black and white icon, this bit mask is made so that the top half was a bit mask AND icons, and the bottom half - XOR bitmask of the icon. Under this condition, the height must be an even factor of two. If this structure defines a color icon, this mask only defines the AND bit mask icon.

hbmColor

It identifies the bitmap color icons. This member can be optional if this structure defines a black and white icon. Bitmask AND hbmMask structure member applied with a flag SRCAND to destination; subsequently, the color bitmap is applied (using XOR) to the destination by using SRCINVERT flag.

See also

[CreateIconIndirect](#), [GetIconInfo](#)

Accommodation and compatibility ICONINFO

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

STRUCTURE ICONMETRICS

ICONMETRICS structure contains scalable metrics associated with icons. This structure uses the SystemParametersInfo function, as determined by the action or SPI_GETICONMETRICS SPI_SETICONMETRICS.

Syntax

```
typedef struct tagICONMETRICS {  
    UINT cbSize;  
    int iHorzSpacing;  
    int iVertSpacing;  
    int iTitleWrap;  
    LOGFONT lfFont;  
} ICONMETRICS, FAR * LPICONMETRICS;
```

The members of the structure

cbSize

Sets the size of the structure in bytes.

iHorzSpacing and iVertSpacing

Horizontal and vertical space, in pixels, for each icon bred.

iTitleWrap

title transfer check box on the new line. If this term is not zero, icons headers automatically move to the next line. If this member is zero, the headlines do not move to a new line.

lfFont

It sets the font used for the icon titles.

See also

SystemParametersInfo

Accommodation and compatibility ICONMETRICS

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

Unicode WinNT

Platform Notes None

VI. WIN32 API. Windows

A window in an application, according to the description in the Microsoft Windows operating system - a rectangular area of the screen where the application displays outputted and accepts user input.

Window screen shared use with other windows, including those that are made from other applications.

At the same time, only one window may accept input data from the user. The user can use a mouse, keyboard or other input device to interact with the window and the application program to which it belongs. about windows

The windows are the main means of graphic application based on the Win32, which interact with the user and perform tasks, so one of the first tasks of graphic applications based on the Win32, is to create a window. This review describes the elements of an application programming interface (API) Microsoft Win32, which applications use to create and use a window; manage relations between them; and the size, movement and display windows on the display screen.

When you start Windows, it automatically creates the desktop window (desktop window). The desktop window - specified window system, which colors the background of the screen and serves as the nucleus for all windows displayed by all applications.

Window Desktop uses a bitmap to be drawn background screen. Pattern generated bitmap called homescreen wallpaper (desktop wallpaper). By default, the Desktop window uses a bitmap of .BMP file that is defined in the registry, as wallpaper.

GetDesktopWindow function returns a handle to the desktop window.

An application configuration system, such as Control Panel applet (Control Panel), change the desktop wallpaper, using the SystemParametersInfo function with wAction parameter set to parameter and SPI_SETDESKWALLPAPER lpvParam, determining the name of the bitmap file.

SystemParametersInfo then loads the bitmap from the specified file, using it to paint the background of the screen, and introduces a new file name in the registry.

Window of application program

Each graphic based on the Win32, an application creates at least one window, called the main window (main window), which serves as a main window for the application. This window serves as the primary interface between the user and the application program. Most applications also provide, either directly or indirectly, other windows, perform the tasks associated with the main window. Each window acts as part of the process in the display information displayed on the screen, and receiving input from the user.

When you run the application, the system connects to the same taskbar button with the application. taskbar button (taskbar button) contains the icon and the program header. When the application is active, its taskbar button appears pressed.

Components of window of application program

Window application program includes elements such as title bar, build-ka menu, the window menu (formerly known as the System menu), coagulation button box, the maximize button, restore button, close button, frame sizing, workspace, range of horizontal scrolling and vertical scrolling line. The main window of the application program typically includes all of these components.

The title bar (title bar) displays a predetermined application program icon and a line of text; usually text specifies the name of the application or indicate the purpose of the window. The application determines the icon and text, when a window is created. In addition, the title bar makes it possible for the user moving a window using a mouse or other pointing device.

Most applications include the menu bar (menu bar), which lists the commands supported by the application. The items in the menu bar are the main categories of teams. Selecting an item in the menu bar usually opens a pop-up menu, whose items are consistent with the objectives within the given category. Choosing the command, the user sends the application to perform the task.

Window Menu (window menu) is created and managed with the help of Windows. It contains a standard set of menu items, which, when selected by the user, set the size or position of the window, close the application or perform tasks. For more information about menus and menu window, see. Article menu.

When you click on a button expand or collapse the window, it affects the size and position of the window. When the user clicks the button deployment (maximize button) window, Windows increases the window to fit the screen and a window so that it covers the entire desktop, minus the taskbar. At the same time, Windows Deployment button replaces a window in the reset button the same size. Reset button (restore button) - bitmap, in which when schelknesh mouse restores the window to its previous size and position.

When the user clicks the button clotting (minimize button) window, Windows reduces the window to the size of its taskbar button, puts the box on the taskbar button, and displays a taskbar button in its normal state. To restore the application to its previous size and position, click on its button on the taskbar.

Frame sizing (sizing border) - zone on the perimeter of the window, which enables the user to change the largest window using a mouse or other pointing device.

Working area (client area) - part of the window, where the application displays the screen output information, such as text or graphics. For example, desktop publishing application program displays in the work area the current page of the document. The application must provide a function called window procedure to process the data entered through the window and display in the work area output information. For more information about window procedures, see. Article Window procedure.

Horizontal line (horizontal scroll bar) and vertical (vertical scroll bar) scroll convert data input from the

mouse or keyboard to the values that the application uses to move the contents of the workspace horizontally or vertically. For example, a word processing application program that displays a long document, typically provides a line vertical scrolling, to allow the user to navigate up and down the document.

The title bar, menu bar, a window menu, minimize button and deployment window, setting the frame size, and scroll bar are considered collectively as a region is not working (nonclient area) window. Windows manages most aspects of the working area is not; application manages all other regards window. Specifically, the application controls the look and behavior of the workspace.

Controls, dialog boxes and message windows

The application uses several types of windows in addition to its main window, including controls, dialog boxes and message boxes.

Control (control) - a window that an application uses to get konretno information from the user, such as name of the file to open it or nuzhnyyny size of the selected text in paragraphs. Applications also use the controls to get the information needed to control application specific properties. For example, the word processing application program typically provides a control that allows the user to enable or disable automatic word wrap. For more information about the controls, see. Article Controls.

Controls are always used together with another window - the usual dialog. Dialog (dialog box) is a box which contains one or more controls. The application uses the dialog box to prompt the user for data entry required to complete the command. For example, an application that includes a command to open a file should show on the dialog screen includes controls where the user specifically identifies the path and file name.

Window messages (message box) - a window that shows the note on the screen, warning, or warning to the user. For example, a message box may inform the user about the problem with which the application was faced with the task.

Dialog boxes and message boxes do not usually use the same set of Windows components, as does the main window. Most often, they have a title bar, a window menu, the frame (not resizable), and the work area, but they usually do not have a menu bar, minimize and maximize buttons. For more information about dialog boxes and message boxes, see. Dialogs.

Z-sequence (z order)

Z-order (Z order) box indicates the position of the window in a stack of overlapping windows. This window stack is oriented along an imaginary axis, the z-axis (Z-axis), leaving off the screen. The window at the top of Z-sequence overrides all other windows. Z-Window below all sequences overlapped by other windows.

Windows Z-sequence retains the usual list. Adding Z-window in the sequence, it is based on the fact that in any case the uppermost windows, the top-level window or child windows. The top window (topmost window) superimposed on any other windows are not the top, regardless of whether they are active or priority windows. The top window has WS_EX_TOPMOST style. All the topmost window in the Z-appear before any sequence not topmost windows. A child window is grouped with its parent in the Z-order.

When an application creates a window, Windows places it atop the sequence Z-window of the same type. You can use BringWindowToTop function to move a window to the top of Z-order for windows of the same type. You can rearrange the sequence Z-using function SetWindowPos and DeferWindowPos.

The user changes the Z-order of the sequence, activating different windows. Windows installs the active window at the top of Z-sequence for the windows of the same type. When the window moves to the top of Z-order, it is the child windows are also made. You can use GetTopWindow function to find all the child windows of the parent window and return the handle to the child window that is highest in the Z-order. GetNextWindow function retrieves the next or previous descriptor in the Z-box

Creating a window

The application creates its window (including the main window) Using the function CreateWindow or the CreateWindowEx, Windows and provides information that is required to determine the attributes of the window. CreateWindowEx function has a parameter, dwExStyle, which have no function CreateWindow; otherwise, the function is identical. In fact, simply calls the CreateWindow CreateWindowEx, setting parameter to zero dwExStyle. For this reason, the remainder of this brief review only relates to CreateWindowEx.

Win32 API provides additional features - including DialogBox, CreateDialog, and the MessageBox - windows to create a special purpose, such as dialog boxes and message boxes. For more information about these features, see. [Dialogs](#).

Attributes of window

An application that creates a window must provide the following information:

Class window (Window class)

The name of the window (Window name)

Style window (Window style)

The parent window or owner window

dimensions

placement

The position on the screen

Child window identifier or menu handle

Handle to the instance

Creation data

The sections below described the attributes.

window class

Every window belongs to a particular class of the window. The application must register a window class before creating any windows of that class. Class window (window class) defines most aspects of appearance and behavior. The main window class component - a window procedure (window procedure), whose function is to receive and process all input data and requests sent by the window. Windows provides the input and requests in the form of messages (messages). For more information about window classes, window procedures, or messages, see Window Classes, Window treatments, or Messages and Message Queues.

window Name

A window can have a name. The name of the window (window name) (also called window text (window text)) - is a text string that identifies a window for the user. The main window, dialog box or a message box, if present, usually shows its window name in its title bar. To control the appearance of the name of the window depends on the class of controls. Button, the edit field or static control displays its window name within the rectangle that takes control. The window with the list, combo box, or static control does not reveal the name of his window.

The program uses SetWindowText function to change the name of the window after a window is created. It uses the function GetWindowTextLength and GetWindowText, to make a selection of the current text window name.

Style window

Each window has one or more windows styles. Style window (window style) - this IME Nova constant that defines aspects of appearance and behavior of the window, which does not define the window class. For example, SCROLLBAR class creates a scroll bar, and SBS_HORZ SBS_VERT styles and define the establishment or a horizontal or vertical scroll bar. Several styles of windows are used all the windows, but most of them use the window of a particular window class. Windows and, to some extent, the window procedure class interpret styles.

Parent or self window

A window can have a parent window. The window that has a parent window is called the child window (child window). The parent window (parent window) provides a coordinate system that is used to position the child window. The presence of a parent window affects aspects of the appearance of the window; for example, a child window is clipped so that-be no part of the child window may not appear outside of its parent window. The window that has no parent window or a parent who is the main window is called the top-level window (top-level window). The application uses EnumWindows function to get a handle to each of its top-level windows. EnumWindows function, in turn, passes the handle of each top-level window to a specific program callback function EnumWindowsProc.

The window may or may belong to another window. An owned window always appears in front of its owner window, is hidden when its owner window is minimized and destroyed when its owner window is destroyed.

The location, size and position in the sequence Z-

Each window has a location, size and position in the Z-order. Location - the coordinates of the upper left corner of the window relative to the upper left corner of the screen or, in the case of a child window of the top left corner of the working area of the parent. Window Size - is its width and height, measured in pixels. The position of the window in Z-sequence (Z order) - this position of the window in a stack of overlapping windows. For more information, see "1.6 Z-sequence (Z Order)".

Child-window identifier or menu handle

A child window may have child-window identifier (child-window identifier), a unique value of a particular program associated with the child window. Identifiers child window is especially useful in applications that create multiple child windows. When you create a child window, the application determines the identifier of the child window. After creating a window, an application can change the identifier of the window, using the SetWindowLong function, or can find the ID by using the function GetWindowLong.

Each window except the child window can have a menu. The application program may include a menu, the menu descriptor by providing, when registration or window class, or create a window.

Handle to the instance

Each application is based on Win32 has a handle associated instance. Windows provides a program descriptor instance when it starts. Because it can run multiple copies of the same program, Windows

uses instance handles inward to distinguish one instance of an application from another. The application must determine the instance handle many different windows, including those which are windows.

data creation

Each window can be determined by the data creation program associated with it. When the first window created, Windows passes a pointer to the data in the window procedure of the window being created. The window procedure uses this information to initialize program variables are defined.

Window handles

After creating the window, creating a function returns a handle to the window (window handle), which uniquely identifies the window. The application program uses the handle to the other functions to direct their action on the window. The window handle is HWND data type; the application must use this type when you declare a variable that contains the handle of the window.

Win32 API includes several specific constants which may be substituted in the window handle certain functions. For example, an application can use in HWND_TOPMOST SendMessageTimeout, HWND_BROADCAST function SendMessage function, or in HWND_DESKTOP MapWindowPoints function.

Although the constant NULL (NULL) - not a window handle, you can use it in some functions to determine whether on any window of exposure. For example, setting the value NULL (NULL) in the parameter hwndParent CreateWindowEx function creates a window that does not have either a parent or the owner. Some functions can return the value NULL (NULL) instead of a handle, indicating that the action does not apply to any window.

An application can use the FindWindow function, to detect whether there is a window system with a specific class name or the name of the window. If such a window exists, FindWindow returns the handle of the window. To limit your search to a single child windows application, use FindWindowEx function. IsWindow function determines whether a window handle identifies an existing window correctly.

Creating main window

Each based on Win32 application must have a WinMain function as their entry point. WinMain function performs a number of tasks, including registering the window class for the main window and the establishment of the main window. WinMain function registers the class of the main window, by calling RegisterClass function, and creates the main window by calling CreateWindowEx function.

Mobility problems entry point into the program must not be called WinMain.

Your function WinMain can also limit your application to a single instance. Create the mutex named-object (object-statistician) using CreateMutex function. If GetLastError returns ERROR_ALREADY_EXISTS, another sample of your application exists (it was created mutex-object), and you have to get out of your WinMain.

Windows automatically displays the main window after it is created; to display the application must use the ShowWindow function. After creating the main window of the application program function WinMain calls the ShowWindow, to pass it two parameters: the handle of the main window and a flag that determines whether the main window should be minimized or maximized when it appears for the first time. Typically, the flag can be set for any of constants beginning SW_ prefix. However, when you call the ShowWindow function to show the main window of the application, the box should be set to SW_SHOWDEFAULT. This flag tells Windows to display a window as defined by the program that launched the application.

If the window is created as a Unicode window (Unicode), it only accepts messages in Unicode (Unicode). To determine whether the window - the window Unicode (Unicode), the function is called IsWindowUnicode.

Messages creating windows

When you create any window, Windows sends messages to the window procedure for the window. Windows sends a message after creating WM_NCCREATE broken pane and WM_CREATE message after creating the workspace. Both messages include a pointer to CREATESTRUCT structure that contains all the information specified in CreateWindowEx function. Typically, the window procedure executes initialization task after receiving these messages.

When you create a child window, Windows, after sending the messages and WM_NCCREATE WM_CREATE, sends the message to the parent window WM_PARENTNOTIFY. It also sends other message and to create a window. The number and order of these messages depend on the window class and style and features used to create the window. These messages are described in other sections of this file reference.

Multithreaded application program

the application is based on Win32 can have multiple streams of execution, and each thread can create windows. Stream which creates a window should contain the code for its window procedure.

An application can use EnumThreadWindows function to enumerate the window by a separate thread. This feature, in turn, passes the handle of each thread in the window defined by the application callback function, EnumThreadWndProc.

GetWindowThreadProcessId function returns the thread identifier, which created a separate window.

To set the state of the form window created by another thread, use the ShowWindowAsync.

General window style

Win32 API provides a common window styles and window styles specific class. Common window styles are represented by constants that begin with the prefix WS_; they can be combined by the operator OR (OR) to form various types of windows, including the main windows, dialog boxes and child windows. Certain styles of a window class defines the appearance and behavior of windows belonging to the predefined control classes such as list boxes and edit window. This review describes the common window styles.

An application typically installs window styles when creating a window. It can also set the styles after the window is created, using the SetWindowLong function.

The overlay window

The overlay window (overlapped window) - a top-level window that has a title bar, a frame, and the workspace; it is supposed to serve as the main window of the application program. It may also have a window menu, minimize and maximize buttons, and scroll bars of the window. The overlay window that is used as the main window typically includes all of these components.

Defining style WS_OVERLAPPED or WS_OVERLAPPEDWINDOW in the CreateWindowEx function, the application creates an overlay window. If you are using WS_OVERLAPPED style, the window has a title bar and frame. If you are using WS_OVERLAPPEDWINDOW style, the window has a title bar that sets the size of the window frame, window menu, and minimize and maximize buttons window.

A pop-up window

A pop-up window (pop-up window) - a special type of overlapping window used for dialog boxes, message boxes, and other temporary windows that appear on the outside of the main window of the application program. header lines are optional for pop-up windows; otherwise, pop-up windows - the same as the overlapping WS_OVERLAPPED style windows.

You create a pop-up window, defining WS_POPUP style in CreateWindowEx function. To include the title bar, specify WS_CAPTION style. Use style WS_POPUPWINDOW, to create a pop-up window that has a frame and a window menu. Style WS_CAPTION should be combined with the style WS_POPUPWINDOW, to make the menu visible window.

Child window

The child window (child window) has WS_CHILD style and limited work area of its parent window. An application typically uses child windows to divide the working area of the parent window to the functional areas. You create a child window by defining WS_CHILD style in CreateWindowEx function.

The child window must have a parent window. The parent window can be overlapping window popping up a window or even another child window. You specify the parent window when you call CreateWindowEx. If you define a style in the CreateWindowEx WS_CHILD, but do not specify a parent window, Windows does not create a window.

The child window has a working area, but no other functions, if they are not explicitly requested. An application can request a title bar, a window menu, minimize button and deployment window frame and scroll bars for the child window, but a child window can not have menus. If the application specifies a handle to the menu, or when it registers the class of the child window, or when creating a child window, the menu descriptor is ignored.

positioning

Windows always installs a child window relative to the upper left of the working area of the parent window's corner. No part of the child window ever occurs outside the scope of its parent window. If the application creates a child window that is larger than the parent window or child window is set so that some or all of the child windows are located outside the parent frames, Windows takes away a child window; that is, outside of the work area of the parent window does not appear. Actions that affect parent window can also affect the child window, as follows.

Parent window A child window

Destruction Destruction before the parent window is destroyed.

Hiding Hiding before the parent window will be hidden.

A child window can be seen only when the parent window is visible.

Move Move to the work area of the parent window.

A child window is responsible for brushing his work area after displacement.

Visibility is shown after it seemed to the parent window.

fixing

Windows does not automatically secures the child window with the working area of the parent window. This means that the parent window does paint over the child window, if it is to comply with any drawing, in the same place where the child window. However, Windows holds the child window with the working area of the parent window if the parent window is WS_CLIPCHILDREN style. If the child window is fixed, the parent window can not paint over it.

A child window can be superimposed on the other child windows within the same workspace. A child window that shares the same parent window with one or more other child windows, the window is called the sister (sibling window). Sister windows can draw in the workspace of each other if one of the child windows has no WS_CLIPSIBLINGS style. If the child window has this style, any part of his nursing window, which is located inside the child window is fixed.

If the window has a style or WS_CLIPCHILDREN WS_CLIPSIBLINGS, there is little loss in efficiency. Each window takes up system resources, so that the application should not use child windows randomly. For the best performance of applications that need to logically divide its main window to do it in the window procedure of the main window rather than use child windows.

The relationship with the parent window

An application can change the parent window of an existing child window by calling SetParent function. In this case, Windows removes the child window of the working area of the old parent window and moves it in the working area of the new parent window. If SetParent specifies the handle value NULL (NULL), the desktop window becomes the new parent window. In this case, the child window is drawn in the main window, outside of any other window. GetParent restores the handle of the parent window to child window.

The parent window gives part of his work area the child window and the child window receives all the information entered for this field. The window class should not be the same for each of the child windows of the parent window. This means that the application can populate the parent window child windows that look differently and perform different tasks. For example, a dialog box may contain many types of controls, each of them is a child window that accepts different types of data from the user.

The child window has only one parent window, but a parent can have any number of child windows. Each child window, in turn, may have daughter windows. In this chain of windows, each child window is called the window generated by the original parent window. The application uses IsChild function to detect whether a given window or child window generated by the window of the parent window.

EnumChildWindows function enumerates the child windows of the parent window. Then, EnumChildWindows passes the handle of each child window to a specific application program callback function. Descendant windows of the parent window is also listed.

messages

Windows passes a child window incoming messages directly to the child window; messages are not transmitted through the parent window. The only exception is that if the child window has been blocked function EnableWindow. In this case, Windows passes any input messages, which would go to the child window into the parent window. This allows the parent window to check incoming messages and, if necessary, to allow them to the child window.

A child window can have a unique integer identifier. Identifiers child window is important when working with windows controls. The application directs the action controls, sending him a message. The application uses the ID of the child window controls to send messages to the control. Furthermore,

the control sends notification messages to its parent window. Notification message includes an identifier of the child window of the control that a parent is used to determine what message is sent to the control. The application determines the child-window identifier for other types of child windows, by setting the parameter hmenu CreateWindowEx function rather than the menu descriptor.

Size of the window frame

Win32 API provides the following styles of the window frame.

WS_BORDER - Creates a window with a frame made of thin lines.

WS_DLGFRAME - Creates a window with a double border, the style commonly used dialog boxes. A window with this style can not have a title bar.

WS_EX_DLGMODALFRAME - Creates a window with a double border. Unlike WS_DLGFRAME style, an application can also specify WS_CAPTION style, to create a title bar for the window.

WS_EX_STATICEDGE - Creates a window with a three-dimensional border style intended to be used elements, which do not have access to user-entered information.

WS_THICKFRAME - Creates a window with variable frame sizes.

Window with WS_OVERLAPPED WS_POPUPWINDOW style or has a default WS_BORDER style.

One of the other styles of frames to be combined with the style or WS_OVERLAPPED

WS_POPUPWINDOW, to give different overlying window frame styles.

If windows or WS_POPUP WS_CHILD style frame style is not specified, the system creates a window without frame. An application can use child windows without frames to divide the working area of the parent window to save the partition invisible to the user.

Components of focus frame

OverScan window may include a title bar, a window menu, minimize button and deployment window, set the size of the window frame, as well as horizontal and vertical scroll bars. An application can create a window with one or more of these components, by determining the below listed styles CreateWindowEx functions:

WS_CAPTION - Creates a window that has a title bar (includes WS_BORDER style).

WS_HSCROLL - Creates a window that has a horizontal scroll bar.

WS_MAXIMIZEBOX - Creates a window that has a maximize button of the window. Style can not be combined with WS_EX_CONTEXTHELP style.

WS_MINIMIZEBOX - Creates a window that has a minimize button of the window. Style can not be combined with WS_EX_CONTEXTHELP style.

WS_SYSMENU - Creates a window that has a window menu on its title bar. There must also be defined WS_CAPTION style.

WS_VSCROLL - Creates a window that has a vertical scroll bar.

Initial state of windows

Below listed styles define one of the two states of the window: it allowed ra-bot, or prohibited, it is visible or invisible, minimized or maximized.

WS_DISABLED - Creates a window that is initially disabled. Locked window can not accept user input.

WS_MAXIMIZE - Creates a window that is initially maximized.

WS_MINIMIZE - Creates a window that is initially minimized.

WS_VISIBLE - Creates a window that is initially visible.

Styles of parent and child windows

The following styles affect the relationship between the parent fix the window and its child windows, and between a child window and its sibling windows.

WS_CLIPCHILDREN - Excludes the area occupied by child windows when drawing within the parent window. Use this style when creating the parent window.

WS_CLIPSIBLINGS - Secures the child windows relative to each other, that is, when a single child window receives WM_PAINT message, style WS_CLIPSIBLINGS fixes all other overlapping child windows out of the region of the child window that you want to modify. If WS_CLIPSIBLINGS is not defined, and child windows overlap, it is possible that when drawing within the client area in one of the child windows, drawing occurs within the working area of another neighboring child window.

Extended styles

The following styles can be defined in parameter dwExStyle CreateWindowEx function:

WS_EX_ACCEPTFILES - Specifies that a window created with this style accepts files of information technology "drag and paste" (drag-and-drop).

WS_EX_CONTEXTHELP - Includes a question mark in the title bar of the window. When the user clicks the question mark, the cursor changes to a pointer with a question mark. If the user then clicks a child window, it receives a message **WM_HELP**. The child window should pass the message to the parent window procedure, which should call the **WinHelp** function using a command **HELP_WM_HELP**. Application Help (Help) displays a pop-up window that typically contains help for the child window. **WS_EX_CONTEXTHELP** can not be used with styles **WS_MAXIMIZEBOX** or **WS_MINIMIZEBOX**.

WS_EX_CONTROLPARENT - Allows the user to navigate through the child windows of the parent window, using the TAB key.

WS_EX_DLGMODALFRAME - Creates a window with a double border. Unlike **WS_DLGFRA**ME style, an application can also specify the **WS_CAPTION** style, to create a title bar for the window.

WS_EX_NOPARENTNOTIFY - Specifies that a child window created with this style will not send a message **WM_PAREN**TNOTIFY its parent window when created or destroyed.

WS_EX_TOPMOST - Specifies that a window created with this style should be placed above all non-topmost windows and stay above them, even when the window is deactivated.

WS_EX_TOOLWINDOW - Creates tools window; that is, the window will be used as a floating toolbar. A tool window has a title string shorter than the normal line of the header, and the window title is drawn using a smaller font. A tool window does not appear in the taskbar or in the window that appears when the user presses ALT + TAB.

Relationship of windows

The window can be in the possession of the window, the window locked, priority nym or background window. There are several different techniques that can be associated with a window or other user window.

Windows owned

Overlapped or pop-up window may belong to another overlapped or pop-window. The current owner of several limited operating window.

An owned window is always above its owner in the Z-order.

Windows will automatically destroy the owned window when its owner is destroyed.

An owned window is hidden when its owner is minimized.

Only overlapped or pop-up window may be the owner of the window; child window can not be him. The application creates an owned window, determining the window handle of the owner through setting hwndParent CreateWindowEx function when it creates a window in WS_OVERLAPPED or WS_POPUP style. HwndParent parameter must identify an overlapped or pop-up window. If hwndParent identifies the child window, Windows assigns the ownership of the child window, the parent top-level window. After creating a window is in property, an application can not transfer ownership of a window to another window.

Dialog boxes and message boxes - Default owned windows. The application determines the owner window when you call a function that creates a dialog box or message box.

An application can use the function to flag GetWindow GW_OWNER, to find the window handle of the owner.

Blocking of windows

The window can be locked. A locked box (disabled window) does not accept background information through the keyboard or mouse of the user, but it can receive messages from other windows from other applications and from Windows. An application typically disables the window to prevent the use of the user box. For example, an application can disable the command button in the dialog box to prevent the selection of a user. An application can open access to a locked window at any time; allowing the window to restore the normal data entry.

The default window when created, is included in the job. However, an application can define a style WS_DISABLED, to disable the new window. The application enables or disables an existing window by using function EnableWindow. Windows sends a message WM_ENABLE window when it's going to change the ON state. An application can determine whether the window is enabled by using the function IsWindowEnabled.

When the child window is blocked, Windows sends messages to input information from the mouse in the descendant window into the parent window. The parent uses the messages to determine whether or not a child window. For more information on how to enter the data from the mouse. See mouse input.

Only one window at a time can receive input data from the keyboard; in this window, as they say, is the keyboard focus. If the application uses EnableWindow function to disable keyboard focus window, the window, in addition to disabling loses keyboard focus. EnableWindow then sets the keyboard focus to the value NULL (NULL), not indicating which window has focus. If the child window or other window generated, the keyboard focus has generated a window loses focus when the parent window is blocked. For more information about the keyboard focus, see. Keyboard Input data.

Foreground and background windows

Each process can have a multi-threaded execution, and each thread can create windows. The thread that created the window with which the user is currently working, called the priority flow, and the window is called the foreground window (foreground window). All other threads are background and the windows they have created called background windows (background windows).

Each thread has a priority level that determines the amount of CPU time, which takes a stream. Although an application can set the priority level of their streams, usually priority thread has a slightly higher priority level than the background threads. Priority stream, because it has a higher priority takes more CPU time than the background threads. Priority stream has a normal base priority - 9; a background thread has a normal base priority - 7.

The user sets the foreground window by clicking the mouse on the window or by using the keyboard shortcut ALT + TAB or ALT + ESC. The application sets the foreground window using the function SetForegroundWindow. If the new foreground window - top-level window, Windows activates it; otherwise it activates the associated top-level window. The application retrieves the data descriptor Priority window using the function GetForegroundWindow. To check whether your window application active, compare the handle returned GetForegroundWindow a window handle your application.

State of the window show

At any given time, the window can be active or inactive; visible or hidden; minimize, maximize or restore. These qualities are referred to in short as the status display (show state) window.

Active window

The active window (active window) - top-level window of the application program with which the user is currently working. To allow the user to easily identify the active window, Windows places it at the top of Z-sequence and replaces the color of its title bar and borders defined color scheme of the active window. The active window can be only the top-level window. When the user is working with a child window, Windows activates the top-level parent window associated with the child window.

Simultaneously, the system can be active only one top-level window. The user activates the top-level window by clicking the mouse on it (or one of its child windows) or by using the keyboard shortcut ALT + ESC or ALT + TAB. The application activates the top-level window, causing the function SetActiveWindow. Among other functions, which can cause Windows to activate a variety of top-level windows are included SetWindowPos, DeferWindowPos, SetWindowPlacement and DestroyWindow. Although an application can activate a different top-level window at any time to avoid entanglement of the user, it does so only in response to a user action. The application uses GetActiveWindow function to find data on the descriptor of the active window.

When activation proceeds from the top-level window in one of the application programs to the top-level window of another program, Windows sends WM_ACTIVATEAPP both software applications, informing them of the change. When activating switches among the various top-level windows in the same application, Windows sends a message to both windows WM_ACTIVATE.

Visibility

The window can be visible or hidden. Windows displays a visible window (visible window) on the screen. It hides a hidden window (hidden window), not prorisovyyava it. If the visible window, the user can provide input window information, and view the output information in the window. If the window is hidden, it is in fact blocked. Hidden window can process messages from Windows or other windows, but it can not process the user input and displaying output information on the screen. The application sets the visible state of the window when you create a window. Later, the application can change the state of visibility.

Window apparently when the window is set to WS_VISIBLE style. By default, the CreateWindowEx function creates a hidden window if the application does not specify the WS_VISIBLE style. As a rule, the application sets the WS_VISIBLE style after it is set up window to keep hidden from the user details of the process of creating it. For example, an application can store a new window hidden while it customizes the window view. If WS_VISIBLE style specified in CreateWindowEx, Windows, after the creation of the window, the window sends a message WM_SHOWWINDOW, but before you show it on the screen.

The application program can determine whether the window is visible, using function IsWindowVisible. The application can show (make visible) or hidden window using the function ShowWindow, SetWindowPos, DeferWindowPos or SetWindowPlacement. These features show or hide the window, installing or removing the WS_VISIBLE style for the window. They also send a message WM_SHOWWINDOW window before showing or hiding it.

When the owner window is minimized, Windows automatically hides associated with the owner of the window. Similarly, when the owner of the window is restored, Windows automatically displays associated with the owner of the window. In both cases, Windows sends a message WM_SHOWWINDOW windows have window - "master" before hiding or showing them. Sometimes, the application may need to hide owned windows without the need to curtail or conceal owner. In this case, the application program uses ShowOwnedPopups function. This function sets or removes WS_VISIBLE style for all owned windows and sends a message WM_SHOWWINDOW owned windows before hiding or showing them. Hiding the owner window has no effect on the state of visibility windows owned.

When the parent window is visible, its associated child windows are also visible. Likewise, when the parent window is hidden, its child windows are also hidden. Collapsing the parent window has no effect on the state of child windows appear; ie child windows rolled up together with a parent, but WS_VISIBLE style does not change.

Even if the window has a style WS_VISIBLE, the user may not be able to see on the screen window; other windows are completely superimposed on it, or it may have been moved behind the screen. Also, the visible child window is subject to the rules of consolidation established for his parent and child relationship. If the parent window of the window is not visible, it will also be invisible. If the parent window is moved over the edges of the screen, the child window also moves because a child window is displayed relative to the upper left corner of the parent. For example, the user can move the parent window that contains the child window far enough from the edge of the screen so that the user may not be able to see the child window, even though the child window and its parent window both have WS_VISIBLE style.

Minimize, maximize and restored window

Maximize window (maximized window) - a window that has WS_MAXIMIZE style. By default, Windows increases the maximized window so that it fills the screen or, in the case of a child window, the parent window's workspace. Although the window size can be set to the same size of the expanded window, a maximized window is slightly different. Windows automatically moves the window's title bar at the top of the screen or into the top of the work area of the parent window. Windows also disables the option to install the frame size of the window and the possibility of positioning the window's title bar (so that the user can not move the window by dragging the title bar).

Minimized window (minimized window) - a window that has WS_MINIMIZE style. By default, Windows reduces a minimized window to the size of its taskbar button and moves the minimized window to the taskbar. Refurbished window (restored window) - a window that has been returned to its former size and position to clotting or to deployment. If the application determines the style WS_MAXIMIZE or WS_MINIMIZE in the CreateWindowEx function, the window is initially maximized or minimized. After creating a window, an application can use CloseWindow function to minimize the window. ArrangeIconicWindows function arranges the icons on the desktop, or it arranges minimized child windows in the parent window. OpenIcon function restores a minimized window to its previous size and position.

Function ShowWindow can roll, to deploy or restore the window. It can also set the visibility of the window and the status of activity. SetWindowPlacement function includes the same functionality as the ShowWindow, but it can cancel drilling down, rolling and restore window positions assigned by default.

Functions IsZoomed IsIconic and determine accordingly whether or minimized this window is maximized. GetWindowPlacement restores the minimized, maximized, and restored positions of the window, and determines the status display window.

When Windows receives a command to maximize or restore a minimized window, Windows sends a message window WM_QUERYOPEN. If the window procedure returns FALSE (FALSE), Windows ignores the Maximize command (Open) or the Restore (Restore).

Windows automatically sets the size and position of a maximized window on a certain system defaults for a maximized window. To override these defaults, an application can either call a function or process SetWindowPlacement the WM_GETMINMAXINFO message, which is received by a window when Windows is going to expand it. WM_GETMINMAXINFO includes MINMAXINFO pointer to a structure containing the Windows values used to set the maximized size and position. Replacing these values overrides the defaults.

Size and position of windows

The size and position of the window are expressed as a limited rectangle in coordinates relative to the screen or parent window. The coordinates of the top-level windows are relative to the upper left corner of the screen; coordinates of a child window are measured relative-enforcement the upper left corner of the parent window. The application program is the initial window size and position, when it creates a window, but it can change the size and location of the window at any time. For more information about the limited boxes, see. The completed forms.

Size of the window

Window size (width and height) are in pixels. The window may have zero width or height. If the application sets a zero width and height of the window, Windows sets the size to the default minimum size of the window. To find the default minimum size of the window, the application uses the GetSystemMetrics function with the flags and SM_CYMIN SM_CXMIN.

In the application it may be necessary to create a window with a separate work area size. AdjustWindowRect and AdjustWindowRectEx function calculates the required size based on the desired size of the workspace. The application program can transmit, resulting, size values CreateWindowEx function.

The application program may set the window size so that it was extremely large; however, it should not set the value of the window so that it is larger than the screen. Before installing the window size, the application should check the width and height of the screen, using the GetSystemMetrics with flags and SM_CYSCREEN SM_CXSCREEN.

Position (location) of windows

The position of the window is defined as the coordinates of its upper left corner. These coordinates, sometimes called the coordinates of the window, always counted relative to the upper left corner of the screen, or, for a child window on the upper left workspace parent window angle. For example, the top-level window having coordinates (10,10) placed on the right to 10 pixels from the upper left corner of the screen, and 10 pixels downward therefrom. A subsidiary window having coordinates (10,10) placed to the right by 10 pixels from the upper left corner of the working area of its parent window, and 10 pixels downward therefrom.

WindowFromPoint function retrieves the handle of the window, which occupies a special place on the screen. Similarly, the function ChildWindowFromPoint ChildWindowFromPointEx and seek out the window handle to the child occupying a special place in the work area of the parent window. Although ChildWindowFromPointEx function can ignore invisible, locked, and transparent child windows, the ChildWindowFromPoint it can not.

Dimensions and position by default

An application can allow Windows to calculate the initial size or position of the top-level window, by setting CW_USEDEFAULT in CreateWindowEx. If an application sets the coordinates of the window in CW_USEDEFAULT and does not create any other top-level windows, Windows installs the new position of the window relative to the upper left corner of the screen; otherwise, it sets the position relative to the position of the top-level window that the application has created recently. If the parameters are the width and height are set to CW_USEDEFAULT, Windows computes the new window size. If an application has created other top-level windows, Windows bases the size of the new window on the size of the top-level application window created recently. Determination CW_USEDEFAULT, when creating a child or pop-up windows causes Windows to set the window size to the default minimum size of the window.

System Commands

An application that has a window menu can change the size and position of the window, sending system commands. System commands are generated when the user selects a command from the Window menu. An application can mimic the action of the user by sending a message to the window WM_SYSCOMMAND. The following system commands affect the size and position of the window.

SC_CLOSE - Closes the window. This command sends the WM_CLOSE message window. Window performs any steps necessary to clear and destroy themselves.

SC_MAXIMIZE - Maximizes the window.

SC_MINIMIZE - Minimizes the window.

SC_RESTORE - Returns a minimized or maximized window to its previous size and position.

SC_SIZE - Runs the command size (Size). The user can resize the window using the mouse or keyboard.

Functions of size and position

After creating a window, an application can set the window size or position, causing one of a number of different functions, including SetWindowPlacement, MoveWindow, SetWindowPos and DeferWindowPos. SetWindowPlacement sets the position of a minimized window, the position of the expanded window, restores the size and position of the window and shows its status. MoveWindow and SetWindowPos functions are similar; both set the size or position of a separate application window. SetWindowPos function includes a set of flags that affect the state of the display window; MoveWindow does not include these flags. Use BeginDeferWindowPos function, the DeferWindowPos EndDeferWindowPos and, at the same time to set the position of a number of windows including size, position, position in the Z-order, and show state.

An application can find the coordinates of the bounding rectangle of the window, using GetWindowRect function. GetWindowRect fills a RECT structure coordinates of the upper left and lower right corners of the window. The coordinates are calculated relative to the upper left corner of the screen, the same for the child window. ScreenToClient or MapWindowPoints function displays the screen coordinates of the bounding rectangle of the child window relative to the working area of the parent window's origin.

GetClientRect function retrieves the coordinates of the workspace window. GetClientRect fills a RECT structure coordinates of the upper left and lower right corners of the workspace, and the coordinates are relative to the working area of its own. This means that the coordinates of the upper left corner of the work area - is always (0,0), and the coordinates of the lower right corner - the width and height of the working area.

CascadeWindows feature a cascade of windows on the desktop or a cascade of certain child windows of the parent window. TileWindows feature a window on the desktop or a certain child windows of the parent window in the non-overlapping positions ("mosaic").

Message of the size and position

Windows sends a message WM_GETMINMAXINFO window whose size or position must change. For example, a message is sent when the user selects from the window menu Move (Move) or size (Size), or clicks the installation frame window is resized or the title bar; a message is sent and when an application calls SetWindowPos function to move or set the value of the window.

WM_GETMINMAXINFO message includes a pointer to MINMAXINFO structure containing the default normal size and position of the window, as well as the minimum and maximum sizes set by default. An application can override the defaults by processing WM_GETMINMAXINFO and setting appropriate MINMAXINFO elements. To take WM_GETMINMAXINFO, the window must have a style or WS_THICKFRAME WS_CAPTION. Window with WS_THICKFRAME style receives this message during the process of creating a window, and when it is moved or set in size.

Windows sends a message WM_WINDOWPOSCHANGING window whose size, position, position in the Z-order, or show state is about to change. This message includes a pointer to WINDOWPOS structure that defines a new window size, position, the position in the Z-order and the status display. By setting the WINDOWPOS elements, the application can work on the new size and position of a window.

After you change the window size, position, position in the Z-order or status display, Windows sends a message window WM_WINDOWPOSCHANGED. This message includes a pointer to WINDOWPOS, which informs about the new window size, position, Z-position, and the sequence state display.

Inserting WINDOWPOS structure, which is transmitted together with WM_WINDOWPOSCHANGED, does not affect the window. A window that must process WM_SIZE and WM_MOVE messages should convey the DefWindowProc function

WM_WINDOWPOSCHANGED; otherwise, Windows does not send WM_SIZE and WM_MOVE message window.

Windows sends a message WM_NCCALCSIZE window when the window is created or established in size. Windows uses the message to calculate the size of the workspace window and the workspace position relative to the upper left corner of the window. The window typically passes this message to the default window procedure; However, this message may be useful in applications that set a non-working area of the window or keep the work area when the window is set by the size. For more information about window size, see. Painting of the article and drawing.

Destruction of windows

In general, the application must destroy all the windows that it creates. It does this by using the function DestroyWindow. When a window is destroyed, the system hides it, if it is visible, and then deletes all internal data associated with a window. This action invalidates the window handle, which can no longer be used by the application program.

The application eliminates many of the windows that it creates, shortly after their creation. For example, an application usually destroys the dialog box window, as soon as the application receives enough information entered by the user in order to continue its mission. An application program ultimately destroys the main window of the application program (during shutdown).

Before destruction window, the application must save or delete any data associated with the window, and it should release all the system resources allocated to the window. If the application does not release the resources, Windows will free any resources that are not freed by the application.

The destruction of the window does not affect the window class from which it was created. New windows can still be created using this class, and any existing windows of this class continue to function. The destruction also destroys the window generated by their windows. DestroyWindow function sends a message WM_DESTROY first window, and then his sister and generated windows. Thus, all generated by the window being destroyed and broken windows.

When the user selects Close (Close), a window with the window menu receives the message WM_CLOSE. By processing this message, an application can prompt the user to confirm this action before the destruction of the window. If the user confirms that the window should be destroyed, the application can call DestroyWindow function to destroy a window.

If destructible window - active, and the activity and the state of the focus moved to another window. The window that is active - the following window, as defined by the key combination ALT + ESC. Then determines the new active window, a window receives keyboard focus.

Creating the main window

The first window, which creates an application - usually the main window (main window). You create a main window by using the CreateWindowEx function, which defines the class name, window styles, size, position, menu handle, an instance handle and data creation. The main window belongs to a particular class of application program window, so you have to register a window class and provide a window procedure for the class before creating the main window.

Most applications are used to create the main window WS_OVERLAPPEDWINDOW style. This style gives the window title bar, a window menu, the frame window installation sizes, buttons minimize and maximize windows. CreateWindowEx function returns a handle that uniquely identifies the window.

The following example creates the main window, windows belonging to the class of a particular application. Name box, "Main Window" appears in the title bar of the window. Combining WS_VSCROLL and WS_HSCROLL styles with style WS_OVERLAPPEDWINDOW, the application creates the main window with horizontal and vertical scroll bars in addition to the components provided WS_OVERLAPPEDWINDOW style. Fourfold repetition CW_USEDEFAULT constant sets the initial size and position of the window in the knowledge-cheniya defined by the system. By setting the value NULL (NULL), instead of the menu handle, the window will get a menu for a specific window class.

```
HINSTANCE hinst;  
HWND hwndMain;  
  
// Create the main window.  
hwndMain = CreateWindowEx (   
    0, // no extension style  
    "MainWClass", // class name  
    "Main Window", // window name  
    WS_OVERLAPPEDWINDOW | // Overlay window  
    WS_HSCROLL | // Horizontal scroll bar  
    WS_VSCROLL, // vertical scroll bar  
    CW_USEDEFAULT, // horizontal position by default  
    CW_USEDEFAULT, // vertical position by default  
    CW_USEDEFAULT, // the default width  
    CW_USEDEFAULT, // the default height  
    (HWND) NULL, // window is not a parent or
```

```
// Having the windows property  
(HMENU) NULL, // class menu used  
hinstance, // instance handle  
NULL); // No data window creation  
if (! hwndMain)  
return FALSE;  
  
// Shows the window, using the check box, a specific program,  
// That launches the application and passes in the app  
// A WM_PAINT message.  
ShowWindow (hwndMain, SW_SHOWDEFAULT);  
UpdateWindow (hwndMain);
```

Note that the previous example calls ShowWindow function after creating the main window. This is because Windows does not automatically displays the main window after it is created. Transmitting SW_SHOWDEFAULT box in the ShowWindow, the application allows the program that launched the application, set the initial show state of the main window. UpdateWindow window function sends his first message WM_PAINT.

Creating, enumeration and resizing of child windows

You can share the workspace window to functionally different areas by using child windows. To create a child window, like creating the main window, you use the CreateWindowEx function. To create a window of a certain class of application program window, you need to register window class and provide a window procedure before creating a child window. You have to give the child window WS_CHILD style and determine the parent window to child window, when creating.

The following example divides the workspace of the main window of the application program into three functional areas, creating three child windows of equal size. Each child window has the same height as the work area of the main window, but each - a third of its width. The main window creates the child windows in response to the WM_CREATE message, which the main window receives during its own process of creation. Since each child window has WS_BORDER style, each of them has a thin frame line. And also, since WS_VISIBLE-style is not specified, each child window is initially hidden. Note also that each child window is assigned an identifier of the child window.

Main Window sets the size and position of child windows in response to the WM_SIZE message, which the main window receives when it is resized. In response to the WM_SIZE, the main window restores the size of their workspace using GetWindowRect function, and then passes the dimensions in EnumChildWindows function. In turn, EnumChildWindows passes the handle of each child window to a specific application program callback function EnumChildProc. This function sets the size and position of each child window by calling MoveWindow function; size and position based on the dimensions of the working area of the main window and the child window identifier. Later, EnumChildProc calls ShowWindow function to make the window visible.

```
#define ID_FIRSTCHILD 100
#define ID_SECONDCHILD 101
#define ID_THIRDCHILD 102
LONG APIENTRY MainWndProc (hwnd, uMsg, wParam, lParam)
HWND hwnd;
UINT uMsg;
UINT wParam;
LONG lParam;
{
RECT rcClient;
int i;
switch (uMsg)
```

```

{

case WM_CREATE: // creation of the main window
// Create three invisible child windows
for (i = 0; i <3; i ++)

CreateWindowEx (0, "ChildWClass", (LPCTSTR) NULL, WS_CHILD | WS_BORDER,
0,0,0,0, hwnd, (HMENU) (int) (ID_FIRSTCHILD + i), hinst, NULL);

return 0;

case WM_SIZE: // change the size of the main window
// Get the dimensions of the working area of the main window and lists the child windows.
// Sends the dimensions in the child windows during enumeration.

GetClientRect (hwnd, & rcClient);

EnumChildWindows (hwnd, EnumChildProc, (LPARAM) & rcClient);

return 0;

// Create other messages.

}

return DefWindowProc (hwnd, uMsg, wParam, lParam);
}

BOOL CALLBACK EnumChildProc (hwndChild, lParam)
HWND hwndChild;
LPARAM lParam;
{
LPRECT rcParent;
int i, idChild;
// Restore the child window ID.
// Use it to place the child window.

idChild = GetWindowLong (hwndChild, GWL_ID);
if (idChild == ID_FIRSTCHILD)
i = 0;
else if (idChild == ID_SECONDCHILD)

```

```
i = 1;  
else  
i = 2;  
  
// The size and position of the child window.  
rcParent = (LPRECT) lParam;  
  
MoveWindow (hwndChild, (rcParent-> right / 3) * i, 0, rcParent-> right / 3, rcParent-> bottom, TRUE);  
  
// Make sure the child window is visible  
ShowWindow (hwndChild, SW_SHOW);  
  
return TRUE;  
}
```

Destruction of windows

You can use DestroyWindow function to destroy a window. As a rule, the application before the destruction of the window sends the message WM_CLOSE, giving the window the opportunity to ask the user for confirmation before destroying it. A window that includes a menu window automatically takes message WM_CLOSE, when the user selects the CLOSE command (Close) from the menu. If the user confirms that the window should be destroyed, the application calls DestroyWindow. Windows sends a message window WM_DESTROY after removing it from the screen. In response to the WM_DESTROY, the window retains its data and releases any resources that it has allocated system. The main window has completed its processing WM_DESTROY, causing function PostQuitMessage, to exit the application.

The following example shows how to query the user for confirmation before destroying the window. In response to WM_CLOSE, the example displays a dialog box on the screen, which contains the button Yes (Yes), OK and Stop (Cancel). If the user clicks the button Yes (Yes), called DestroyWindow; otherwise, the window is not destroyed. As destructible window - the main window, the example calls PostQuitMessageWM_DESTROY.

```
case WM_CLOSE:  
    // Create a message box. If the user clicks  
    // The Yes button (Yes), the main window is destroyed.  
    if (MessageBox (hwnd, szConfirm, szAppName,  
        MB_YESNOCANCEL) == IDYES)  
        DestroyWindow (hwndMain);  
    else  
        return 0;  
  
case WM_DESTROY:  
    // Registers WM_QUIT message to complete the exit from the program.  
    PostQuitMessage (0);  
    return 0;
```


Window structures

STRUCTURE CLIENTCREATESTRUCT

CLIENTCREATESTRUCT structure contains information about the menu and the first operating window, the child window of the medium multi-document (MDI). The application passes a pointer to this structure as a parameter lpvParam CreateWindow function when creating mnogodokumentalnogo interface (MDI) window of the user.

Syntax

```
typedef struct tagCLIENTCREATESTRUCT  
{  
    HANDLE hWindowMenu;  
    UINT idFirstChild;  
};  
  
CLIENTCREATESTRUCT;
```

Elements hWindowMenu structure

Identifies the window handle MDIMDI menu window using the function GetSubMenu.

idFirstChild

Specifies the identifier of the child window created by the first child window (MDI). Windows increases the identifier for each additional child window Mnogodokumentalnogo Interface (MDI), the application creates and reassigns identifiers when the application destroys a window to save the continuous range of identifiers. These identifiers are used in WM_COMMAND messages sent Framework window (MDI) application when the child window is selected from the Window menu; they should not be in conflict with any other command identifiers.

See also

CreateWindow, GetSubMenu, MDICREATESTRUCT, WM_COMMAND

Accommodation and compatibility CLIENTCREATESTRUCT

Windows NT Yes

Win95 Yes

Yes Win32s

imported library

Header file winuser.h

No Unicode

Platform Notes None

STRUCTURE COPYDATASTRUCT

COPYDATASTRUCT structure contains data that will be transferred to another application in accordance with WM_COPYDATA message.

Syntax

```
typedef struct tagCOPYDATASTRUCT
```

```
{
```

```
    DWORD dwData;
```

```
    DWORD cbData;
```

```
    PVOID lpData;
```

```
} COPYDATASTRUCT;
```

elements

dwData

It sets up to 32 bits of data to be transferred to the host application.

cbData

Sets the size, in bytes, of data, indicated lpData structure element.

lpData

It specifies the data to be transferred to the host application. This member can be NULL value (NULL).

See also

[WM_COPYDATA](#)

Accommodation and compatibility COPYDATASTRUCT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

STRUCTURE CREATESTRUCT

CREATESTRUCT structure defines the initialization parameters passed to the window procedure of the application.

Syntax

```
typedef struct tagCREATESTRUCT  
{  
    LPVOID lpCreateParams;  
    HINSTANCE hInstance;  
    HMENU hMenu;  
    HWND hwndParent;  
    int cy;  
    int cx;  
    int y;  
    int x;  
    LONG style;  
    LPCTSTR lpszName;  
    LPCTSTR lpszClass;  
    DWORD dwExStyle;  
} CREATESTRUCT;
```

Elements lpCreateParams structure

Indicates information that you want to use to create the window. Windows NT: This element is the value of the address structure specifier SHORT (16_bit), which establishes the size, in bytes, of data to create a window. The value immediately followed by the creation data. For more information, see. The following Remarks section.

hInstance

It identifies the module that owns the new window.

hMenu

Identifies the menu that is created using a window.

hwndParent

Identifies the parent window, if you create a window - the child window. If the window is the owner,

this element identifies the owner window. If the window - not a subsidiary or not being in the property window, this element - BLANK (NULL).

cy

Sets the height of the new window in pixels.

cx

It sets the width of the new window in pixels.

y

Sets y-coordinate of the upper left corner of the new window. If a new window - child window coordinates - relative to the parent window. Otherwise, the coordinates - origin on the screen.

x

Sets the x-coordinate of the upper left corner of the new window. If a new window - child window coordinates - relative to the parent window. Otherwise, the coordinates - origin on the screen. style

Specifies the style for the new window. lpszName

It indicates a null-terminated string at the end, which is the name of the new window.

lpszClass

It indicates a null-terminated string at the end, which defines the class name of the new window.

dwExStyle

Defines the extended style for the new window.

remarks

Windows NT: With reference to the element lpCreateParams CREATESTRUCT structure as a pointer may not be aligned on DWORD boundary (DWORD), the application must access the data using the pointer, which was declared a cis-use of the type of misaligned (UNALIGNED), as shown in the following example:

```
typedef struct tagMyData
{
    . . .; // Define the data to create a window here
} MYDATA;

typedef struct tagMyDlgData {
    SHORT cbExtra;
    MYDATA myData;
} MYDLGDATA, UNALIGNED * PMYDLGDATA;
```

```
PMYDLGDATA pMyDlgdata = (PMYDLGDATA) (((LPCREATESTRUCT) lParam) ->  
lpCreateParams);
```

See also

[CreateWindow](#), [CreateWindowEx](#)

Accommodation and compatibility CREATESTRUCT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

Unicode WinNT

Platform Notes None

STRUCTURE MDICREATESTRUCT

MDICREATESTRUCT structure contains information about the class, title, owner, location and size of the child window of the medium multi-document (MDI).

Syntax

```
typedef struct tagMDICREATESTRUCT
```

```
{
```

```
    LPCTSTR szClass;
```

```
    LPCTSTR szTitle;
```

```
    HANDLE hOwner;
```

```
    int x;
```

```
    int y;
```

```
    int cx;
```

```
    int cy;
```

```
    DWORD style;
```

```
    LPARAM lParam;
```

```
} MDICREATESTRUCT;
```

elements

szClass

It indicates a null-terminated string at the end, which sets the window class name of the child window Mnogodokumentalnogo Interface (MDI). The class name must be registered in a previous call to RegisterClass function.

szTitle

Specifies the line to zero at the end of the symbol, which is the title of a child window Mnogodokumentalnogo Interface (MDI). Windows displays the title in the child window's title bar.

hOwner

It identifies the instance of the application program, create a working window Mnogodokumentalnogo Interface (MDI).

x

Sets the initial horizontal position of the MDI child window, in user coordinates. If this element - CW_USEDEFAULT, MDI child window takes the default horizontal position.

y

Sets the initial vertical position of the MDI child window, in user coordinates. If this element - CW_USEDEFAULT, MDI child window receives a default vertical position.

cx

Sets the initial width, in device units, MDI child windows. If this element - CW_USEDEFAULT, MDI child window receives a default width.

cy

It sets the initial height, in device units, MDI child windows. If this element - CW_USEDEFAULT, MDI child window takes the default height.

style

Specifies the style of a child window Mnogodokumentalnogo Interface (MDI). If MDI operating window was created with MDIS_ALLCHILDSTYLES window style, this member can be any combination of the window styles listed in the description of the CreateWindow function. Otherwise, this element can be one or more of the following values:

WS_MINIMIZE - Creates a child window Mnogodokumentalnogo Interface (MDI), which is initially minimized.

WS_MAXIMIZE - Creates an MDI child window that is initially maximized.

WS_HSCROLL - Creates an MDI child window that has a horizontal scroll bar.

WS_VSCROLL - Creates an MDI child window that has a vertical scroll bar.

lParam

Specifies an application-defined 32-bit value.

Notes When the MDI child window (MDI) created, Windows sends the WM_CREATE message window. Parameter lParam WM_CREATE contains a pointer to a structure CREATESTRUCT. An element of this structure contains a pointer to lpCreateParams MDICREATESTRUCT structure passed with the message WM_MDICREATE, which created child window multiple document interface (MDI).

See also

CLIENTCREATESTRUCT, CREATESTRUCT, WM_CREATE

Accommodation and compatibility MDICREATESTRUCT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

Unicode WinNT

Platform Notes None

STRUCTURE MINMAXINFO

MINMAXINFO structure contains information about the expanded window size and positions established by its minimum and maximum size.\

Syntax

```
typedef struct tagMINMAXINFO
```

```
{
```

```
POINT ptReserved;
```

```
POINT ptMaxSize;
```

```
POINT ptMaxPosition;
```

```
POINT ptMinTrackSize;
```

```
POINT ptMaxTrackSize;
```

```
} MINMAXINFO;
```

elements ptReserved

Reserved, do not use.

ptMaxSize

Sets the width of the unfolded (point.x) and detailed height (point.y) window.

ptMaxPosition

Sets the position of the left side of the maximized window (point.x) and the position of the top of the maximized window (point.y).

ptMinTrackSize

It specifies the minimum set by the width (point.x) and a minimum installed height (point.y) window.

ptMaxTrackSize

Specifies the maximum set by the width (point.x) and a maximum installed height (point.y) window.

See also

POINT, WM_GETMINMAXINFO

Accommodation and compatibility MINMAXINFO

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

STRUCTURE NCCALCSIZE_PARAMS

NCCALCSIZE_PARAMS structure contains information that an application can use while processing WM_NCCALCSIZE message to calculate the size, Posey and valid contents of the client area of a window.

Syntax

```
typedef struct _NCCALCSIZE_PARAMS
```

```
{
```

```
    RECT rgc [3];
```

```
    PWINDOWPOS lppos;
```

```
}
```

```
    NCCALCSIZE_PARAMS;
```

elements

rgc

Sets an array of rectangles. The first contains the new coordinates of the window that has been moved or changed. The second contains the coordinates of a window before it was moved or resized. The third contains the coordinates of a window before the window was moved or resized. If the window - the child window coordinates - relative to the working area of the parent window. If the window - the top-level window, the coordinates - relative to the screen origin.

lppos

Indicates WINDOWPOS structure that contains size and position values specified in the operations that move or resize the window.

See also

[MoveWindow](#), [RECT](#), [SetWindowPos](#), [WINDOWPOS](#), [WM_NCCALCSIZE](#)

[Accommodation and compatibility NCCALCSIZE_PARAMS](#)

[Windows NT Yes](#)

[Win95 Yes](#)

[Yes Win32s](#)

[Imported Library -](#)

[Header file winuser.h](#)

[No Unicode](#)

[Platform Notes None](#)

STRUCTURE STYLESTRUCT

STYLESTRUCT structure contains styles for the window.

Syntax

```
typedef struct tagSTYLESTRUCT  
{  
    DWORD styleOld;  
    DWORD styleNew;  
} STYLESTRUCT, * LPSTYLESTRUCT;  
  
elements  
style
```

Sets an array of window styles or flags improved style.

See also

[WM_STYLECHANGED](#), [WM_STYLECHANGING](#)

Accommodation and compatibility [STYLESTRUCT](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file [winuser.h](#)

No Unicode

Platform Notes None

STRUCTURE WINDOWPLACEMENT

WINDOWPLACEMENT structure comprises location information window on the screen.

Syntax

```
typedef struct _WINDOWPLACEMENT
```

```
{
```

```
    UINT length;
```

```
    UINT flags;
```

```
    UINT showCmd;
```

```
    POINT ptMinPosition;
```

```
    POINT ptMaxPosition;
```

```
    RECT rcNormalPosition;
```

```
} WINDOWPLACEMENT;
```

elements

length

It sets the length of the structure in bytes. Before calling or GetWindowPlacement SetWindowPlacement functions, set this item to sizeof (WINDOWPLACEMENT).

GetWindowPlacement SetWindowPlacement and will fail if this item is not installed properly.

flags

Sets flags that control the position of the minimized window and the method by which the window is restored. This element can have several of the following values:

WPF_RESTORETOMAXIMIZED - Specifies that the restored window will be maximized, regardless of whether it was maximized before it was minimized. This setting is valid only in the next time the window is restored. It does not change the default recovery behavior. This flag is correct only when the member for showCmd set SW_SHOWMINIMIZED.

WPF_SETMINPOSITION - Specifies that the coordinates of the minimized window can be determined. This box must be checked if the coordinates are set to ptMinPosition element.

showCmd

Specifies the current show state of the window. This member can be one of the following values:

SW_HIDE - Hides the window and activates another window.

SW_MINIMIZE - Minimizes the specified window and activates the top-level window in the system list.

SW_RESTORE - Activates and displays a window on the screen. If the window is minimized or maximized, Windows restores it to its original size and position (same as **SW_SHOWNORMAL**).

SW_SHOW - Activates the window and displays it in its current size and position.

SW_SHOWMAXIMIZED - Activates the window and displays it as a maximized window.

SW_SHOWMINIMIZED - Activates the window and displays it on the screen as an icon.

SW_SHOWMINNOACTIVE - Displays a window as an icon. The active window remains active.

SW_SHOWNA - Displays a window in its current state. The active window remains active.

SW_SHOWNOACTIVATE - Displays a window in its most modern size and position. The active window remains active.

SW_SHOWNORMAL - Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as **SW_RESTORE**).

ptMinPosition

Sets the coordinates of the upper-left corner of the window when it is minimized.

ptMaxPosition

Sets the coordinates of the upper left corner of the window when it is deployed.

rcNormalPosition

Sets the coordinates of the window when it is in the reduced position.

See also

ShowWindow, POINT, RECT

Accommodation and compatibility WINDOWPLACEMENT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

STRUCTURE WINDOWPOS

WINDOWPOS structure contains information about the size and position of the window.

Syntax

```
typedef struct _WINDOWPOS
```

```
{
```

```
    HWND hwnd;
```

```
    HWND hwndInsertAfter;
```

```
    int x;
```

```
    int y;
```

```
    int cx;
```

```
    int cy;
```

```
    UINT flags;
```

```
} WINDOWPOS;
```

elements

hwnd

It identifies the window.

hwndInsertAfter

Specifies the position of the window in the Z-order (position from the beginning to the end). This member can be a handle to the window, behind which a window is placed, or may be one of the special values listed with the SetWindowPos function.

x

Sets the position of the left edge of the window.

y

Sets the position of the upper edge of the window.

cx

Sets the width of the window in pixels.

cy

Sets the height of the window, in pixels.

flags

Set window position. This member can be one of the following values:

SWP_DRAWFRAME - Prints frame (defined in the description of the class of the window) around the window.

SWP_FRAMECHANGED - Sends a message window WM_NCCALCSIZE, even if does not change the window size. If this option is not specified, WM_NCCALCSIZE sent only when the window is resized.

SWP_HIDEWINDOW - Hides the window.

SWP_NOACTIVATE - Do not activate the window. If this is not checked, the window is activated and moved to the top of either the topmost or non-topmost group (depending on the setting hWndInsertAfter parameter).

SWP_NOCOPYBITS - Resets all the contents of the working area. If this is not checked, the valid contents of the work area are saved and copied back into the client area after the window is sized or repositioned (again set).

SWP_NOMOVE - Retains current position (ignores the X and Y parameters).

SWP_NOOWNERZORDER - Does not change the owner window's position in the Z-order.

SWP_NOREDRAW - not redraw changes. If this option is selected, repainting of any kind occurs. This applies both to the workspace overscan (including the header area, and scroll bars) or to any part of the parent window uncovered by the movement of the window. When this option is selected, the application must explicitly or cancel or redraw any parts of the window and parent window that must be redrawn.

SWP_NOREPOSITION - The same as the box SWP_NOOWNERZORDER.

SWP_NOSENDCHANGING - Prevents the window from receiving WM_WINDOWPOSCHANGING posts.

SWP_NOSIZE - Retains the current size (ignores the cx and cy parameters).

SWP_NOZORDER - Retains the current Z-order (ignores hWndInsertAfter option).

SWP_SHOWWINDOW - Displays the window.

See also

EndDeferWindowPos, SetWindowPos, WM_NCCALCSIZE

Accommodation and compatibility WINDOWPOS

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

Window messages

WM_ACTIVATE MESSAGE

WM_ACTIVATE message is sent when a window is activated or deactivated. This message is sent first to the window procedure deaktiviziruemogo top-level window; then it is sent to the window procedure of the top-level window being activated.

Syntax

WM_ACTIVATE

fActive = LOWORD (wParam); // Check activation

fMinimized = (BOOL) HIWORD (wParam); // Check to minimize

hwndPrevious = (HWND) lParam; // Window handle

Options

fActive

The value of low byte words wParam. Specifies whether the window is activated or deactivated it. This parameter can be one of the following values:

WA_ACTIVE - Activated by some method other than a mouse click (eg, function call SetActiveWindow or using the keyboard interface to select the window).

WA_CLICKACTIVE - Activated click.

WA_INACTIVE - Disable.

fMinimized

The value of the high byte word wParam. Sets the collapsed state of the window, or activatable deaktiviziruemogo. A value other than zero indicates that the window is minimized (minimized).

hwndPrevious

The value of lParam. It identifies the window or deaktiviziruemoe activated, depending on the value of fActive. If the value fActive - WA_INACTIVE, hwndPrevious - handle-activated window. If the value fActive - WA_ACTIVE or WA_CLICKACTIVE, hwndPrevious - handle deaktiviziruemogo window. This handle can be left blank (NULL).

Return values

If the program has processed the message, it should return zero.

The default action

If the window is activated and is not minimized (minimized), DefWindowProc function sets the keyboard focus to the window.

remarks

If the window is activated click, it also receives a WM_MOUSEACTIVATE.

See also

DefWindowProc, SetActiveWindow, WM_MOUSEACTIVATE, WM_NCACTIVATE

Accommodation and compatibility WM_ACTIVATE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_ACTIVATEAPP MESSAGE

WM_ACTIVATEAPP message is sent when a window belonging to another application, except for the active window is going to be activated. The message is sent to the application program whose window is activated and the application whose window is deactivated.

Syntax

WM_ACTIVATEAPP

fActive = (BOOL) wParam; // Activation flag

dwThreadID = (DWORD) lParam: // thread identifier

Options

fActive

The value of wParam. Sets whether or deactivated the window is activated. This option - TRUE (TRUE), if the window is activated; FALSE (FALSE), if the window is deactivated.

dwThreadID

The value of lParam. Specifies the thread identifier. If fActive option - TRUE (TRUE), dwThreadID - stream ID that owns deaktiviziruemym window. If fActive - FALSE (FALSE), dwThreadID - stream ID that has activated the window.

Return values

If an application processes this message, it should return zero.

See also

WM_ACTIVATE

Accommodation and compatibility WM_ACTIVATEAPP

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_CANCELMODE MESSAGE

Message sent WM_CANCELMODE focus window when displayed dialog box or a message box; it makes it possible to cancel window focus modes, such as data collection from mouse.

Syntax

WM_CANCELMODE

Options

This message has no parameters.

Return values

If an application processes this message, it should return zero.

The default action

DefWindowProc function cancels internal processing of standard scroll bar input, cancels internal menu processing, and refuses to collect data from the mouse.

See also

DefWindowProc, ReleaseCapture

Accommodation and compatibility WM_CANCELMODE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_CHILDACTIVATE MESSAGE

The message is sent to the child window WM_CHILDACTIVATE medium multi-document (MDI) when the user clicks on the title bar area, or when activated window is moved or installed in size.

Syntax

WM_CHILDACTIVATE

Options

This message has no parameters.

Return values

If an application processes a message, it should return zero.

See also

MoveWindow, SetWindowPos

Accommodation and compatibility WM_CHILDACTIVATE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_CLOSE MESSAGE

WM_CLOSE message is sent as a signal to which the window or the application should complete its work.

Syntax

WM_CLOSE

Options

This message has no parameters.

Return values

If an application processes this message, it should return zero.

The default action

DefWindowProc function refers to DestroyWindow function to destroy a window.

notes

An application can prompt the user for confirmation before destroying the window, in the course of processing WM_CLOSE message and calls DestroyWindow function only if the user confirms the selection.

See also

DefWindowProc, DestroyWindow

Accommodation and compatibility WM_CLOSE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_COMPACTING MESSAGE

WM_COMPACTING message is sent to all top-level windows when Windows detects more than 12.5 percent of the system time for 30 - 60-second interval, is spent on the memory seal. This indicates that the system memory is insufficient.

Syntax

WM_COMPACTING

wCompactRatio = wParam; // compression ratio

Options

wCompactRatio

The value of wParam. Sets the coefficient of the current time, the central processing unit (CPU), memory spent Windows on the seal to the current CPU time spent Windows to perform other actions. For example, 0x8000 represents 50 percent of the CPU time spent on memory seal.

Return values

If an application processes this message, it should return zero.

remarks

When the application receives this message, it should release as much memory as possible, taking into account the current level of action applied programs and the total number of applications running on Windows.

Accommodation and compatibility WM_COMPACTING

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_COPYDATA MESSAGE

WM_COPYDATA message is sent when one program sends the data to another program.

Syntax

WM_COPYDATA

wParam = (WPARAM) (HWND) hwnd; // Handle the transmission window

lParam = (LPARAM) (PCOPYDATASTRUCT) pcds; // Pointer to the data structure

Options

hwnd

It identifies the window that transmits the data.

pcds

Indicates COPYDATASTRUCT structure that contains the data for transmission.

Return values

If the receiving application processes this message, it should return TRUE (TRUE); otherwise, it should return - FALSE (FALSE).

remarks

To send this message, the program must use the SendMessage function, not the function PostMessage. The data to be transmitted must not contain pointers or other references to objects that are not available for a program receiving the data.

Up until this message is valid due to data should not be modified by another current-transfer process. The host program must take into account the data to read-only. Parameter pcds is correct only for the processing of the message. The host program should not free the memory caused pcds. If the receiving program is applied to the data after the return value of the function SendMessage, it must copy the data to a local buffer.

See also

PostMessage, SendMessage, COPYDATASTRUCT

Accommodation and compatibility WM_COPYDATA

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_CREATE MESSAGE

WM_CREATE message is sent when the program asks, calling a function CreateWindowEx or CreateWindow be created window. The new window procedure receives the message window after the window is created, but before the window becomes visible. The message is sent to the return value of the function CreateWindowEx or CreateWindow.

Syntax

WM_CREATE

```
lpcs = (LPCREATESTRUCT) lParam; // Structure with creation data
```

Options

lParam

The value of lParam. Indicates CREATESTRUCT structure that contains information about the window being created. Members CREATESTRUCT identical parameters CreateWindowEx function.

Return values

If an application processes this message, it returns 0, to continue to create the window. If the application returns -1, the window is destroyed and the CreateWindowEx or CreateWindow function returns a NULL handle (NULL).

See also

CreateWindow, CreateWindowEx, CREATESTRUCT, WM_NCCREATE

Accommodation and compatibility WM_CREATE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_DESTROY MESSAGE

WM_DESTROY message is sent when the window is destroyed. It is sent to the window procedure of the window being destroyed after the window is removed from the screen. This message is sent first blasted the window, and then the child windows (if any) when they are destroyed. During the message processing, it can be taken as all child windows still exist.

Syntax

WM_DESTROY

Options

This message has no parameters.

Return values

If an application processes this message, it should return zero.

remarks

If destructible window - part of the chain of clipboard viewer window (set, SetClipboardViewer function call), the window should be removed from the chain by processing ChangeClipboardChain function before returning from WM_DESTROY posts.

See also

ChangeClipboardChain, DestroyWindow, PostQuitMessage, SetClipboardViewer, WM_CLOSE

Accommodation and compatibility WM_DESTROY

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_QUIT MESSAGE

WM_QUIT message indicates a request to complete the application, and is created when the application of the function is called PostQuitMessage. This forces the GetMessage function return zero.

Syntax

WM_QUIT

nExitCode = (int) wParam; // Code completion

Parameters nExitCode

The value of wParam. Specifies the exit code in the PostQuitMessage function.

Return values

This message has no return value, because it leads to end the cycle of posts before a message is sent to the window procedure of the application.

See also

GetMessage, PostQuitMessage

Accommodation and compatibility WM_QUIT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_ENABLE MESSAGE

WM_ENABLE message is sent when an application alters the ON state of the window. It is sent to a window whose enabled state changes. This message is sent to the return value of the function EnableWindow, but only after the change on state (WS_DISABLED style bit) window.

Syntax

WM_ENABLE

fEnabled = (BOOL) wParam; // Check box to enable / disable

Options

fEnabled

The value of wParam. Sets whether the window has been enabled or disabled. This option would be TRUE (TRUE), if the window is enabled or FALSE (FALSE), if the window was locked.

Return values

If an application processes this message, it should return zero.

See also

EnableWindow

Accommodation and compatibility WM_ENABLE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_ENTERSIZEMOVE MESSAGE

WM_ENTERSIZEMOVE message is sent only once the window when it enters the sizing or moving mode. Window enters navigation mode or set the size when the user clicks on the title bar area or sets the size of the window frame, or when the window passes WM_SYSCOMMAND message to the DefWindowProc function, and the wParam message parameter sets the SC_MOVE or SC_SIZE. WM_ENTERSIZEMOVE Windows sends a message regardless of whether the movement of the full window enabled.

Syntax

WM_ENTERSIZEMOVE

wParam = 0; // Not used, must be zero

lParam = 0; // Not used, must be zero

Options

This message has no parameters.

Return values

The program should return zero if it processes this application.

See also

DefWindowProc, WM_EXITSIZEMOVE, WM_SYSCOMMAND

Accommodation and compatibility WM_ENTERSIZEMOVE

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_EXITSIZEMOVE MESSAGE

WM_EXITSIZEMOVE message is sent only once the window after it came out of the mode size or move.

Syntax

WM_EXITSIZEMOVE

wParam = 0; // Not used should be zero

lParam = 0; // Not used, must be zero

Options

This message has no parameters.

Return values

The program should return zero if it processes this message.

See also

WM_ENTERSIZEMOVE

Accommodation and compatibility WM_EXITSIZEMOVE

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_GETICON MESSAGE

WM_GETICON message sent to a window to return back to handle large or small icon associated with the window. Windows returns back a large icon when drawing a minimized window, and a small icon when drawing the header area.

Syntax

WM_GETICON

fType = wParam; // Type of icons

Options

fType

The value of wParam. Sets the type of the returned back to icons. This parameter can be one of the following values:

ICON_BIG - Return back to a large icon for the window.

ICON_SMALL - Return back to a small icon for the window.

Return values

Return value - handle large or small icons, depending on the value fType. When an application receives the message, it can return a handle large or small icons, or send a message to DefWindowProc.

The default action

DefWindowProc returns a handle large or small icon associated with a window, depending on the fType.

notes

When an application receives the message, it can return a handle large or small icons, or send a message to DefWindowProc.

See also

DefWindowProc, WM_SETICON

Accommodation and compatibility WM_GETICON

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_GETMINMAXINFO MESSAGE

WM_GETMINMAXINFO message sent to a window when the size or position of the window, are collecting change. An application can use this message to cancel the maximization of the size and position of the window by default, or it sets the default minimum or maximum size.

Syntax

WM_GETMINMAXINFO

lpmmi = (LPMINMAXINFO) lParam; // Address of structure

Options

lpmmi

The value of lParam. Indicates MINMAXINFO structure that contains the value of the position and size of the default deployment and is the default minimum and maximum sizes. The application program can override the default settings elementovov this structure.

Return values

If an application processes this message, it should return zero.

remarks

Set the maximum size - the largest window size that can be produced using the frame to set the window size. Setting a minimum size - the smallest window size that can be produced using the frame to set the window size.

See also

MoveWindow, SetWindowPos, MINMAXINFO

Accommodation and compatibility WM_GETMINMAXINFO

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_GETTEXT MESSAGE

The application sends a message to the WM_GETTEXT, to copy the text that matches the text in the buffer window in which the caller provided.

Syntax

WM_GETTEXT

wParam = (WPARAM) cchTextMax; // Number of characters to copy

lParam = (LPARAM) lpszText; // Address of buffer for text

Options

cchTextMax

The value of wParam. Sets the maximum number of characters that will be copied, including the zero-terminator character is complete.

lpszText

The value of lParam. It points to the buffer that is to receive the text.

Return values

Return value - number of characters copied.

The default action

DefWindowProc function copies the text associated with the window into the specified buffer and returns the number of characters copied.

remarks

To edit an item, the text to be copied - contains modify elements. For a combo box, the text - editing content items (or static text) of the combo box. For buttons, text - name of the button. For other windows, the text - window title. To copy a text item in the list box, an application can use LB_GETTEXT message.

When a message is sent to the static element WM_GETTEXT control with SS_ICON style handle icon will be returned in the first four bytes lpszText pointer buffer. This is true only if WM_SETTEXT message used to set the icon.

The powerful editing tools, if the text to be copied exceeds 64K, use a message or EM_STREAMOUT or EM_GETSELTEXT.

See also

DefWindowProc, EM_GETSELTEXT, EM_STREAMOUT, GetWindowText, GetWindowTextLength, LB_GETTEXT, WM_GETTEXTLENGTH, WM_SETTEXT

Accommodation and compatibility WM_GETTEXT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_GETTEXTLENGTH MESSAGE

The application sends a message WM_GETTEXTLENGTH to determine the length of text, symbols associated with a window. The length does not include the complete character string (zero-terminator).

Syntax

WM_GETTEXTLENGTH

wParam = 0; // Not used, must be zero

lParam = 0; // Not used, must be zero

Options

This message has no parameters.

Return value

Return value - the length of the text, in characters.

The default action

DefWindowProc function returns the length of the text, in characters. Under certain conditions, this may actually be greater than the length of the text. For additional information Noi. See the following Remarks section.

remarks

To edit the elements, the text to be copied - the content of the editing elements. For a combo box, the text - editing content items (or static text) portion of the combo box. For buttons, text - name of the button. For other windows, the text - window title. To determine the length of the element in the list box, an application can use LB_GETTEXTLEN message.

Under certain conditions, the DefWindowProc function returns a value that is greater than the actual length of the text. This occurs with certain mixtures of ANSI and Unicode, and because of the operating system, which takes into account the possible existence of DBCS characters within the text. The return value, however, will always be at least as large as a fact-cal length of the text; Can thus always use it to determine the distribution-division in the buffer. This behavior can occur when an application uses both ANSI functions and common dialogs, which use Unicode. LB_GETTEXT, or CB_GETLBTEXT, or GetWindowText function.

See also

CB_GETLBTEXT, DefWindowProc, GetWindowText, GetWindowTextLength, LB_GETTEXT, LB_GETTEXTLEN, WM_GETTEXT.

Accommodation and compatibility WM_GETTEXTLENGTH

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_INPUTLANGCHANGE MESSAGE

WM_INPUTLANGCHANGE message is sent to the topmost window in the current after it has been changed to the scene task. It should be used for creating any application-specific settings and parameters of their transfer to the DefWindowProc function, which any child records to be transferred.

Syntax

WM_INPUTLANGCHANGE

charset = wParam;

hkl = (HKL) lParam;

Options

charset

The value of wParam. Specifies the character set of the new keyboard layout.

hkl

The value of lParam. Identifies the new keyboard layout.

Return values

An application should return a non-zero value if the processes this message.

See also

DefWindowProc, WM_INPUTLANGCHANGEREQUEST

Accommodation and compatibility WM_INPUTLANGCHANGE

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_INPUTLANGCHANGEREQUEST MESSAGE

Post WM_INPUTLANGCHANGEREQUEST announces top-level window of the application program when the user selects the language keyboard input, or change the language keyboard input using the "hot" button or from the menu system languages. Application must accept the change by passing the message to the DefWindowProc function or reject the change (and prevent it from happening) returning immediately.

Syntax

WM_INPUTLANGCHANGEREQUEST

fSysCharSet = (BOOL) wParam

hkl = (HKL) lParam;

Parameters wParam

The least significant bit of this parameter is set, if the handle symbol arrangement on the keyboard can be used with a set of system symbols. Other bits are reserved. For example, in the Russian version of Windows 95, this option sets the least significant bit for the symbols on the keyboard layout descriptors for English (US) and Russian language, but resets for other descriptors.

hkl

The value of lParam. Identifies the keyboard layout to switch between them.

Return values

This message informs that nothing is sent to the application program, so that the return value is ignored. To accept the changes, the application must pass the message to DefWindowProc. To reject the changes, the application should return zero without calling DefWindowProc.

See also

DefWindowProc, WM_INPUTLANGCHANGE

Accommodation and compatibility WM_INPUTLANGCHANGEREQUEST

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_MOVE MESSAGE

Post WM_MOVE passed after when the window is moved. Sntaksis

WM_MOVE

xPos = (int) LOWORD (lParam); // Horizontal position

yPos = (int) HIWORD (lParam); // The vertical position

Options

xPos

The value of low byte words lParam. Sets the x-coordinate of the upper-left corner of the workspace window.

yPos

The value of the upper word lParam. Sets the y-coordinate of the upper-left corner of the workspace window.

Return values

If an application processes this message, it should return zero.

remarks

xPos and yPos parameters are given in screen coordinates for overlapped and pop-up windows and user coordinates - parent to child windows. An application can use a macro MAKEPOINTS, to convert the lParam parameter to POINTS structure.

See also

MAKEPOINTS, POINTS

Accommodation and compatibility WM_MOVE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_MOVING MESSAGE

WM_MOVING message sent to a window that the user moves. By processing this message, an application can monitor the size and position of the drag rectangle and, if needed, change its size or position.

Syntax

fwSide = wParam; // The edge of the window to be moved

lprc = (LPRECT) lParam; // Screen coordinates of the drag rectangle

Options

fwSide

The value of wParam. Specifies which edge of the window moves. This parameter can be a combination of the following values:

WMSZ_BOTTOM - The lower edge

WMSZ_BOTTOMLEFT - left bottom corner

WMSZ_BOTTOMRIGHT - bottom right corner

WMSZ_LEFT - Left Edge

WMSZ_RIGHT - Right edge

WMSZ_TOP - The upper edge

WMSZ_TOPLEFT - The upper left corner

WMSZ_TOPRIGHT - Left-right corner

lprc

The value of lParam. Address RECT structure with the screen coordinates of the drag rectangle. To change the size or position of the drag rectangle, an application must change the members of this structure.

Return value

The program should return TRUE (TRUE), if it processes this message.

See also

RECT, WM_MOVE, WM_SIZING

Accommodation and compatibility WM_MOVING

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_NCACTIVATE MESSAGE

WM_NCACTIVATE message sent to a window when his non-working area must be changed to indicate its active or inactive state.

Syntax

WM_NCACTIVATE

fActive = (BOOL) wParam; // A new state of the header field or icon

Options

fActive

The value of wParam. It sets when the header area or the icon must be modified to indicate an active or inactive state of them. If the active area of the title or the icon must be displayed on the screen, fActive option - TRUE (TRUE). He - FALSE (FALSE) to inactive the header area or the icon.

Return values

When fActive option - FALSE (FALSE), the application should return TRUE (TRUE), to indicate that Windows should continue processing the default values, or it should return FALSE (FALSE), to prevent the header area or the icon of the deactivation. When fActive - TRUE (TRUE), the return value is ignored.

The default action

DefWindowProc function displays the header area or the icon in the header of its active colors when fActive option - TRUE (TRUE), and in its inactive colors when fActive - FALSE (FALSE).

See also

DefWindowProc

Accommodation and compatibility WM_NCACTIVATE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_NCCALCSIZE MESSAGE

WM_NCCALCSIZE message is sent when the size of the workspace window and the position must be calculated. By processing this message, an application can manage the contents of a window when the size or position of the window changes.

Syntax

WM_NCCALCSIZE

```
fCalcValidRects = (BOOL) wParam; // Check the current field  
lpnccsp = (LPNCCALCSIZE_PARAMS) lParam; // Pointer to size  
// Or calculated data  
lpnccsp = (LPRECT) lParam; // Pointer to the new position of the window
```

Options

fCalcValidRects

The value of wParam. This is a Boolean value, if TRUE (TRUE), it is determined that the application must specify which part of the work area contains valid information. The operating system copies the correct information in the specified area within the new client area. Also, if this option - TRUE (TRUE), lParam points to the structure NCCALCSIZE_PARAMS. Esli this option - FALSE (FALSE), the application should not specify a valid part of the workspace. Also, if this option - FALSE (FALSE), lParam points to a RECT structure.

lpnccsp

The value of lParam. If wParam - TRUE (TRUE), lParam points to NCCALCSIZE_PARAMS structure that contains the information that an application can use to calculate the new size and position of the client rectangle. If wParam - FALSE (FALSE), lParam points to a RECT structure that contains the coordinates of the new window that has been moved or changed. This option is equivalent to rgc [0] from NCCALCSIZE_PARAMS structure.

Return values

If fCalcValidRects option - FALSE (FALSE), the application should return zero. If fCalcValidRects - TRUE (TRUE), the application may return zero or a valid combination of the following values:

WVR_ALIGNTOP, WVR_ALIGNLEFT, WVR_ALIGNBOTTOM, WVR_ALIGNRIGHT - These values are used in combination-s, it is determined that the working area of the window must be preserved and RHR-aligned respectively on the new position of the window. For example, to align the work area at the bottom left corner, return values and WVR_ALIGNLEFT WVR_ALIGNTOP.

WVR_HREDRAW, WVR_VREDRAW - These values are used in combination with any other values, cause the window to be completely re-derived, if the user changes the size of the rectangle horizontally or vertically. These values are similar to the styles and class CS_HREDRAW CS_VREDRAW.

WVR_REDRAW - This value is forced to re-display the entire window. This is - a combination of values and WVR_HREDRAW WVR_VREDRAW.

WVR_VALIDRECTS - This value indicates that, on return from WM_NCCALCSIZE, rectangles specified elements rgrc [1] and rgrc [2] NCCALCSIZE_PARAMS structures contain the correct source and region targeting rectangles, respectively. Windows combines these rectangles to calculate the area of the window you want to save. Windows copies any part of the window image that is within the source rectangle and attach an image to a rectangle destination. Both rectangles are in the box - the parent or relative screen coordinates. This return value allows an application to perform more complex techniques of conservation work area, such as centering or preserving a subset of the workspace.

If fCalcValidRects - TRUE (TRUE), and the program returns zero, the old work area is maintained and aligned to the left upper corner of the new workspace.

Default Actions

The window can be re-displayed, depending on whether or style class CS_HREDRAW CS_VREDRAW installed. This is - the default, backward compatibility process the message function DefWindowProc (in addition to the usual client rectangle calculation described in the previous table).

See also

DefWindowProc, MoveWindow, SetWindowPos, NCCALCSIZE_PARAMS, RECT

Accommodation and compatibility WM_NCCALCSIZE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_NCDESTROY MESSAGE

WM_NCDESTROY message informs a window that it OverScan destroyed. DestroyWindow function sends a window WM_NCDESTROY WM_DESTROY message after message. WM_DESTROY used to free memory, which placed the object associated with the window.

Syntax

WM_NCDESTROY

Options

This message has no parameters

Return values

If an application processes this message, it should return zero.

remarks

This message frees any memory internally allocated for the window.

See also

DestroyWindow, WM_DESTROY, WM_NCCREATE

Accommodation and compatibility WM_NCDESTROY

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_PARENTNOTIFY MESSAGE

WM_PARENTNOTIFY message sent to the parent of a child window when the child window is created or destroyed, or when the user clicks the mouse button while the cursor - over the child window. When the child window is created, the system sends WM_PARENTNOTIFY just before the CreateWindow or the CreateWindowEx function, which creates conditions for the return of the window to its previous state. When the child window is destroyed, Windows sends a message to any action that takes place for the destruction of the window.

Syntax

WM_PARENTNOTIFY

fwEvent = LOWORD (wParam); // Flags event

idChild = HIWORD (wParam); // Identifier of the child window

lValue = lParam; // Handle to a child, or the coordinates of the cursor

Options

fwEvent

The value of low byte words wParam. Defines the event reported by the parent. This parameter can be one of the following values:

WM_CREATE - Creates a child window.

WM_DESTROY - child window is destroyed.

WM_LBUTTONDOWN - User Place the cursor over the child window and clicked the left mouse button.

WM_MBUTTONDOWN - User Place the cursor over the child window and clicked the middle mouse button.

WM_RBUTTONDOWN - User Place the cursor over the child window and clicked the right mouse button.

idChild

The value of the upper word wParam. If fwEvent setting is WM_CREATE or WM_DESTROY, idChild sets the identifier of the child window. Otherwise, idChild undefined.

lValue

It contains a handle to the child window when fwEvent setting is WM_CREATE or WM_DESTROY; otherwise, lValue comprises x- and y-coordinates of the cursor. x-coordinate in the low word and the y-coordinate is in the high word.

Return values

If an application processes this message, it should return zero.

remarks

This message is also sent to all windows, the ancestors of the child window, including the top-level window. All child windows, except for those boxes that have advanced the WS_EX_NOPARENTNOTIFY style, send this message to their parent windows. By default, child windows in a dialog box have WS_EX_NOPARENTNOTIFY style, if not caused by CreateWindowEx function to create a child window without this style.

See also

CreateWindow, CreateWindowEx, WM_CREATE, WM_DESTROY, WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN

Accommodation and compatibility WM_PARENTNOTIFY

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_POWER MESSAGE

WM_POWER message is sent when the system is typically a personal computer c battery is going to enter the pause mode. Post WM_POWER outdated. It is envisaged to simplify the portability of a 16-bit application based on Windows. New Win32-based applications should use WM_POWERBROADCAST message.

Syntax

WM_POWER

```
fwPowerEvt = wParam; // Notification message about lowering power
```

Options

fwPowerEvt

The value of wParam. Specifies the notification message about lowering diet. This parameter can be one of the following:

PWR_CRITICALRESUME - Indicates that the system is resuming operation after entering in pause mode without first sending a notification message PWR_SUSPENDREQUEST application. The application must perform any necessary recovery actions.

PWR_SUSPENDREQUEST - Indicates that the system is about to enter the pause mode.

PWR_SUSPENDRESUME - Indicates that the system resumes operation, after the introduction of the suspension of the normal mode, ie, the system sent a notification message PWR_SUSPENDREQUEST application before the system was suspended. The application must perform any necessary recovery actions.

Return values

The value returned by the application depends on the value of wParam. If wParam - PWR_SUSPENDREQUEST, the return value - PWR_FAIL, to prevent the system from entering the state of suspension of work; otherwise, this - PWR_OK. If wParam - PWR_SUSPENDRESUME or PWR_CRITICALRESUME, return value is zero.

remarks

This message is sent only to the application program that is executed by the system and meet the technical requirements Improved supply system (APM) basic deduces (BIOS) entry system. The message is sent to the power management driver to each window returned to the Board the function EnumWindows. Suspend Mode operation - a condition in which there is the greatest power efficiency savings, but all current data and settings are stored. Random access memory (RAM) retains its content, but many devices are likely to be turned off.

See also EnumWindows, WM_POWERBROADCAST

Accommodation and compatibility WM_POWER

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_QUERYDRAGICON MESSAGE

Message sent WM_QUERYDRAGICON collapsed (ikonizirovannomu) window. The window is going to make the movement with the help of the user, but does not specify the icon for the class. An application can return the value of the descriptor icon or cursor. The system displays the cursor or icon, while the user drags the icon.

Syntax

WM_QUERYDRAGICON

Options

This message has no parameters.

Return values

An application should return a handle to the cursor or icon that Windows should show, at a time when the user drags the icon. The cursor or icon must be compatible with the resolution of the display device. If the application returns an empty (NULL), the system displays the default cursor.

Default Actions

DefWindowProc function returns the handle to the specified cursor by default.

remarks

When the user drags the icon window without determining the class icon, Windows changes the icon to the default cursor. If the application requires that in moving from the system characterized displayed cursor, it must return a handle or pointer icon that is compatible with the resolution on the display device. If the application returns a handle color cursor or icon, the system converts the cursor or icon in Black and white. An application can call the function LoadCursor or LoadIcon, to get a cursor or icon from its executable (.EXE) file and return back to this descriptor.

See also

DefWindowProc, LoadCursor, LoadIcon

Accommodation and compatibility WM_QUERYDRAGICON

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_QUERYOPEN MESSAGE

Message sent WM_QUERYOPEN icon when the user requests that the window has been restored to its previous size and position.

Syntax

WM_QUERYOPEN

Options

This message has no parameters.

Return values

If the icon can be opened, the application that processes this message should return TRUE (TRUE); otherwise, it should return FALSE (FALSE), in order to prevent discovery of the icon.

Default Actions

DefWindowProc function returns TRUE (TRUE).

remarks

When processing this message, an application should not perform any action that forces become active and change the focus (for example, create a dialog box).

See also

DefWindowProc, LoadCursor, LoadIcon

Accommodation and compatibility WM_QUERYOPEN

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_SETICON MESSAGE

The application sends a message WM_SETICON, to compare a new large or small icon with a window. Windows displays a large icon when the window Sverre-NLRB (minimized) and a small icon in the title bar.

Syntax

WM_SETICON

wParam = (WPARAM) fType; // Type of icons

lParam = (LPARAM) (HICON) hicon; // Handle icon

Parameters fType

The value of wParam. It specifies the type of installed icons. This parameter can be one of the following values:

Meaning What it means

ICON_BIG Installing a large icon for the window.

ICON_SMALL Install a small icon for the window.

hicon

The value of lParam. Identifies new large or small icon. If this parameter - NULL (NULL), the icon marked in the fType parameter, is removed.

Return values

Return value - descriptor of the previous large or small icon, depending on the value fType. It - BLANK (NULL), if the window previously had no icon type, designated fType.

Default Actions

DefWindowProc function returns the handle of the previous large or small icon associated with a window, depending on the value fType.

See also

DefWindowProc, WM_GETICON

Accommodation and compatibility WM_SETICON

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_SETTEXT MESSAGE

The application sends a message WM_SETTEXT, to set the text box.

Syntax

WM_SETTEXT

wParam = 0; // Not used, must be zero

lParam = (LPARAM) (LPCTSTR) lpsz; // Address of text window line

Options

lpsz value of lParam. Specifies the line with the symbol of zero at the end, which is a text box.

Return values

Return value - TRUE (TRUE), if the text is set. It - FALSE (FALSE) (for the editing tools), LB_ERRSPACE (for the list box), or CB_ERRSPACE (for the combo box), if there is insufficient space available to set the text in the fields editorial-tirovanie. It - CB_ERR, if the message is sent without the combo box editing facilities.

Default Actions

DefWindowProc function sets and displays the text window.

remarks

For editing tools, text - content editing tools. For a combo box, the text - the content of the editing tools combo box. For buttons, text - name of the button. For other windows, the text - window title. This message does not change the current selection in the combo box list window. An application should use CB_SELECTSTRING message to select an item in the list box that matches the text in the edit tools.

See also

DefWindowProc, CB_SELECTSTRING, WM_GETTEXT

Accommodation and compatibility WM_SETTEXT

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_SETTINGCHANGE MESSAGE

The system sends a message WM_SETTINGCHANGE all top-level windows when the SystemParametersInfo function changes a system-level installation. The system sends this message only if the caller sets SystemParametersInfo SPIF_SENDCHANGE box. The application program may send WM_SETTINGCHANGE all top-level windows when it makes changes to system parameters. For example, you can send the message after the call to WriteProfileString functions, WriteProfileSection or SetLocaleInfo, or after making changes to the system parameters in the system reestre. Soobschenie WM_SETTINGCHANGE - the same as the old message WM_WININICHANGE.

Syntax

WM_SETTINGCHANGE

wParam = wFlag; // Check the parameters at the system level

lParam = (LPARAM) (LPCTSTR) pszSection; // Name of the partition or register

Options

wFlag

The value of wParam. When the system sends the message as a result of the treatment to the SystemParametersInfo, this option - a flag that indicates the system parameter that was changed. On the list of values, see. The SystemParametersInfo function. When an application sends co-communication, this parameter should be left blank (NULL).

pszMetrics

The value of lParam. A pointer to a string that specifies the domain containing the system parameter that was changed. For example, this line may be a registry key name or the name of the section in the WIN.INI file. This option is not particularly useful in determining which changed the system setting. For example, when the string - the name of the registry, it typically indicates only the tip of the registry, but not all the way. In addition, some applications send this message with the lParam parameter setting in an empty (NULL). Generally, when you accept this message, you need to check and reset all system configuration settings that are used by your application.

Return values

If you process this message, the return value - zero.

remarks

To send a message WM_SETTINGCHANGE all top-level windows, use the SendMessage function with the hwnd parameter set to HWND_BROADCAST. Call the function that change the WIN.INI file, which can be displayed instead of the registry. This mapping occurs when the WIN.INI file and modify partitions installed in the system registry under the following keys:

HKEY_LOCAL_MACHINE \ Software \ Microsoft \ Windows NT \ CurrentVersion \ IniFileMapping

The change in the memory cell has no influence on the behavior of the communication.

See also

SendMessage, SetLocaleInfo, SystemParametersInfo, WM_WININICHANGE, WriteProfileSection, WriteProfileString

Accommodation and compatibility WM_SETTINGCHANGE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

Unicode WinNT

Platform Notes None

WM_SETTINGCHANGE MESSAGE

The system sends a message WM_SETTINGCHANGE all top-level windows when the SystemParametersInfo function changes a system-level installation. The system sends this message only if the caller sets SystemParametersInfo SPIF_SENDCHANGE box. The application program may send WM_SETTINGCHANGE all top-level windows when it makes changes to system parameters. For example, you can send the message after the call to WriteProfileString functions, WriteProfileSection or SetLocaleInfo, or after making changes to the system parameters in the system reestre. Soobschenie WM_SETTINGCHANGE - the same as the old message WM_WININICHANGE.

Syntax

WM_SETTINGCHANGE

wParam = wFlag; // Check the parameters at the system level

lParam = (LPARAM) (LPCTSTR) pszSection; // Name of the partition or register

Options

wFlag

The value of wParam. When the system sends the message as a result of the treatment to the SystemParametersInfo, this option - a flag that indicates the system parameter that was changed. On the list of values, see. The SystemParametersInfo function. When an application sends co-communication, this parameter should be left blank (NULL).

pszMetrics

The value of lParam. A pointer to a string that specifies the domain containing the system parameter that was changed. For example, this line may be a registry key name or the name of the section in the WIN.INI file. This option is not particularly useful in determining which changed the system setting. For example, when the string - the name of the registry, it typically indicates only the tip of the registry, but not all the way. In addition, some applications send this message with the lParam parameter setting in an empty (NULL). Generally, when you accept this message, you need to check and reset all system configuration settings that are used by your application.

Return values

If you process this message, the return value - zero.

remarks

To send a message WM_SETTINGCHANGE all top-level windows, use the SendMessage function with the hwnd parameter set to HWND_BROADCAST. Call the function that change the WIN.INI file, which can be displayed instead of the registry. This mapping occurs when the WIN.INI file and modify partitions installed in the system registry under the following keys:

HKEY_LOCAL_MACHINE \ Software \ Microsoft \ Windows NT \ CurrentVersion \ IniFileMapping

The change in the memory cell has no influence on the behavior of the communication.

See also

SendMessage, SetLocaleInfo, SystemParametersInfo, WM_WININICHANGE, WriteProfileSection, WriteProfileString

Accommodation and compatibility WM_SETTINGCHANGE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

Unicode WinNT

Platform Notes None

WM_SHOWWINDOW MESSAGE

WM_SHOWWINDOW message sent to a window when the window is about to be hidden or visible.

Syntax

WM_SHOWWINDOW

fShow = (BOOL) wParam; // Check to show / hide

fnStatus = (int) lParam; // Check status

Options

fShow

The value of wParam. It determines whether the window is shown. TRUE (TRUE), if the window is shown, or false (FALSE), if the window is hidden.

fnStatus

The value of lParam. Specifies the state of the displayed window. Parameter fnStatus zero if the message is sent due reference to ShowWindow function; otherwise, fnStatus - one of the following:

SW_OTHERUNZOOM - Window opens because a maximized window was restored or minimized (minimized).

SW_OTHERZOOM - The window is covered by another window, which had been deployed.

SW_PARENTCLOSING - Window owner rolls up the window.

SW_PARENTOPENING - Window owner window is restored.

Return values

If an application processes this message, it should return zero.

Default Actions

DefWindowProc function hides or shows the window, as established in accordance with the message.

remarks

If the window when it is created, has the WS_VISIBLE style, the window receives this message after it is created, but before it is displayed. A window also receives this message when its visibility state is changed or the function ShowWindow ShowOwnedPopups. WM_SHOWWINDOW message is sent in the following circumstances:

When the top-level, overlapped window is created with WS_MAXIMIZE or WS_MINIMIZE style.

When installed SW_SHOWNORMAL flag in an address to ShowWindow function.

See also

DefWindowProc, ShowOwnedPopups, ShowWindow

Accommodation and compatibility WM_SHOWWINDOW

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_SIZE MESSAGE

WM_SIZE message sent to a window after its size has changed.

Syntax

WM_SIZE

```
fwSizeType = wParam; // Check resize  
nWidth = LOWORD (lParam); // The width of the work area  
nHeight = HIWORD (lParam); // Height of the work area
```

Options

fwSizeType

The value of wParam. It specifies the type of resizing requested. This parameter can take one of the following:

SIZE_MAXHIDE - Message is sent to all windows popping up when deployed to a different window.

SIZE_MAXIMIZED - The window was maximized.

SIZE_MAXSHOW - Message is sent to all windows popping up when some other window has been restored to its former size.

SIZE_MINIMIZED - The window was minimized (minimized).

SIZE_RESTORED - A window has been changed, but neither value nor SIZE_MINIMIZED SIZE_MAXIMIZED not apply.

nWidth

The value of the lower word lParam. Specifies the new width of the client area.

nHeight

The value of the upper word lParam. Specifies the new height of the work area.

Return values

If an application processes this message, it should return zero.

remarks

If the function or SetScrollPos MoveWindow called for a child window as a result of WM_SIZE message, bRedraw parameter must be different from zero to force a window to be repainted. Although the width and height of the window - 32-bit values, parameters and nWidth nHeight WM_SIZE message contain only the low 16 bits.

See also

MoveWindow, SetScrollPos

Accommodation and compatibility WM_SIZE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_SIZING MESSAGE

WM_SIZING message sent to a window in which the user resizes. By processing this message, an application can monitor the size and position of the Vai-peretaski rectangle and, if needed, change its size or position.

Syntax

fwSide = wParam; // Edge mounted on the window size

lprc = (LPRECT) lParam; // Screen coordinates of the drag rectangle

Options

fwSide

The value of wParam. It indicates that the edge of the window size is set. This parameter can be a combination of the following values:

WMSZ_BOTTOM - The lower edge

WMSZ_BOTTOMLEFT - Corner bottom left of the

WMSZ_BOTTOMRIGHT - Corner bottom right of the

WMSZ_LEFT - Left Edge

WMSZ_RIGHT - Right edge

WMSZ_TOP - The upper edge

WMSZ_TOPLEFT - Corner top left of the

WMSZ_TOPRIGHT - Angle the top right of

lprc

Address RECT structure with the screen coordinates of the drag rectangle. To change the size or position of the draggable rectangle, the application must change the members of this structure.

Return values

An application should return TRUE (TRUE), if it processes this message.

See also

RECT, WM_MOVING, WM_SIZE

Accommodation and compatibility WM_SIZING

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_STYLECHANGED MESSAGES

WM_STYLECHANGED message sent to a window, if after the SetWindowLong function has changed one or more window styles.

Syntax

WM_STYLECHANGED

wStyleType = wParam; // Extended or not extended styles

lpss = (LPSTYLESTRUCT) lParam; // Structure containing new styles

Options

wStyleType

The value of wParam. It determines whether expanded or unexpanded window styles changed. This parameter can be a combination of the following values:

GWL_EXSTYLE - Extended window styles changed.

GWL_STYLE - unextended window styles changed.

lpss

The value of lParam. Indicates STYLESTRUCT structure that contains the new styles for the window. The application program can examine the styles, but can not change them.

Return values

An application should return zero if it processes this message.

See also

SetWindowLong, STYLESTRUCT, WM_STYLECHANGING

Accommodation and compatibility WM_STYLECHANGED

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_STYLECHANGING MESSAGE

WM_STYLECHANGING message sent to a window when the function SetWindowLong going to change one or more window styles.

Syntax

WM_STYLECHANGING

wStyleType = wParam; // Advanced and advanced styles

lpss = (LPSTYLESTRUCT) lParam; // Structure containing new styles

Options

wStyleType

The value of wParam. It determines whether expanded or unexpanded window styles changed. This parameter can be a combination of the following values:

GWL_EXSTYLE - Extended window styles changed.

GWL_STYLE - unextended window styles changed.

lpss

The value of lParam. Indicates STYLESTRUCT structure that contains the proposed new styles for the window. The application program can examine styles and, if necessary, to modify them.

Return values

An application should return zero if it processes this message.

See also

SetWindowLong, STYLESTRUCT, WM_STYLECHANGED

Accommodation and compatibility WM_STYLECHANGING

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_USERCHANGED MESSAGE

WM_USERCHANGED message is sent to all windows, after the user has entered or left the system. When a user enters or leaves the system, the system modifies the user-specific settings. The system sends this message immediately after modifying settings.

Syntax

WM_USERCHANGED

wParam = 0; // Not used, must be - nil

lParam = 0; // Not used, must be - nil

Return values

An application should return zero if it processes this message.

Accommodation and compatibility WM_USERCHANGED

Windows NT Yes

Win95 Yes

No Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_WINDOWPOSCHANGING MESSAGE

WM_WINDOWPOSCHANGING message sent to a window whose size, position, or place in the Z-order is about to change as a result of recourse to the SetWindowPos function or another window management functions.

Syntax

WM_WINDOWPOSCHANGING

lpwp = (LPWINDOWPOS) lParam; // Indicates the position data and the size

Options

lpw

The value of lParam. Indicates WINDOWPOS structure that contains information from the relatively new window size and position.

Return values

If an application processes this message, it should return zero.

Default Actions

For windows WS_OVERLAPPED style or WS_THICKFRAME, DefWindowProc function sends a window message WM_GETMINMAXINFO. This is done in order to verify the correctness of the new size and position of the window and enforce CS_BYTEALIGNCLIENT styles and user CS_BYTEALIGNWINDOW. Not passing WM_WINDOWPOSCHANGING message to the DefWindowProc function, an application can override these defaults.

remarks

At the time when this message is processed by changing any of the values in WINDOWPOS, it acts on the new size or position of the window in the Z-order. An application can prevent changes in the window by setting or resetting the corresponding bits in the flags element WINDOWPOS.

See also

DefWindowProc, EndDeferWindowPos, SetWindowPos, WINDOWPOS, WM_GETMINMAXINFO, WM_MOVE, WM_SIZE, WM_WINDOWPOSCHANGED

Accommodation and compatibility WM_WINDOWPOSCHANGING

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_WININICHANGE MESSAGE

WM_WININICHANGE date. It is included for compatibility with earlier versions of Windows. New applications should use WM_SETTINGCHANGE message.

Accommodation and compatibility WM_WININICHANGE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

Unicode WinNT

Platform Notes None

WM_CLOSE MESSAGE

WM_CLOSE message is sent as a signal to which the window or the application should complete its work.

Syntax

WM_CLOSE

Options

This message has no parameters.

Return values

If an application processes this message, it should return zero.

The default action

DefWindowProc function refers to DestroyWindow function to destroy a window.

notes

An application can prompt the user for confirmation before destroying the window, in the course of processing WM_CLOSE message and calls DestroyWindow function only if the user confirms the selection.

See also

DefWindowProc, DestroyWindow

Accommodation and compatibility WM_CLOSE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_COMPACTING MESSAGE

WM_COMPACTING message is sent to all top-level windows when Windows detects more than 12.5 percent of the system time for 30 - 60-second interval, is spent on the memory seal. This indicates that the system memory is insufficient.

Syntax

WM_COMPACTING

wCompactRatio = wParam; // compression ratio

Options

wCompactRatio

The value of wParam. Sets the coefficient of the current time, the central processing unit (CPU), memory spent Windows on the seal to the current CPU time spent Windows to perform other actions. For example, 0x8000 represents 50 percent of the CPU time spent on memory seal.

Return values

If an application processes this message, it should return zero.

remarks

When the application receives this message, it should release as much memory as possible, taking into account the current level of action applied programs and the total number of applications running on Windows.

Accommodation and compatibility WM_COMPACTING

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_COPYDATA MESSAGE

WM_COPYDATA message is sent when one program sends the data to another program.

Syntax

WM_COPYDATA

wParam = (WPARAM) (HWND) hwnd; // Handle the transmission window

lParam = (LPARAM) (PCOPYDATASTRUCT) pcds; // Pointer to the data structure

Options

hwnd

It identifies the window that transmits the data.

pcds

Indicates COPYDATASTRUCT structure that contains the data for transmission.

Return values

If the receiving application processes this message, it should return TRUE (TRUE); otherwise, it should return - FALSE (FALSE).

remarks

To send this message, the program must use the SendMessage function, not the function PostMessage. The data to be transmitted must not contain pointers or other references to objects that are not available for a program receiving the data.

Up until this message is valid due to data should not be modified by another current-transfer process. The host program must take into account the data to read-only. Parameter pcds is correct only for the processing of the message. The host program should not free the memory caused pcds. If the receiving program is applied to the data after the return value of the function SendMessage, it must copy the data to a local buffer.

See also

PostMessage, SendMessage, COPYDATASTRUCT

Accommodation and compatibility WM_COPYDATA

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_CREATE MESSAGE

WM_CREATE message is sent when the program asks, calling a function CreateWindowEx or CreateWindow be created window. The new window procedure receives the message window after the window is created, but before the window becomes visible. The message is sent to the return value of the function CreateWindowEx or CreateWindow.

Syntax

WM_CREATE

```
lpcs = (LPCREATESTRUCT) lParam; // Structure with creation data
```

Options

lParam

The value of lParam. Indicates CREATESTRUCT structure that contains information about the window being created. Members CREATESTRUCT identical parameters CreateWindowEx function.

Return values

If an application processes this message, it returns 0, to continue to create the window. If the application returns -1, the window is destroyed and the CreateWindowEx or CreateWindow function returns a NULL handle (NULL).

See also

CreateWindow, CreateWindowEx, CREATESTRUCT, WM_NCCREATE

Accommodation and compatibility WM_CREATE

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

WM_DESTROY MESSAGE

WM_DESTROY message is sent when the window is destroyed. It is sent to the window procedure of the window being destroyed after the window is removed from the screen. This message is sent first blasted the window, and then the child windows (if any) when they are destroyed. During the message processing, it can be taken as all child windows still exist.

Syntax

WM_DESTROY

Options

This message has no parameters.

Return values

If an application processes this message, it should return zero.

remarks

If destructible window - part of the chain of clipboard viewer window (set, SetClipboardViewer function call), the window should be removed from the chain by processing ChangeClipboardChain function before returning from WM_DESTROY posts.

See also

ChangeClipboardChain, DestroyWindow, PostQuitMessage, SetClipboardViewer, WM_CLOSE

Accommodation and compatibility WM_DESTROY

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library -

Header file winuser.h

No Unicode

Platform Notes None

Window functions

The following functions are used to create and manage windows.

FUNCTION ADJUSTWINDOWRECT

AdjustWindowRect function calculates the required size of the window rectangle, based on the desired size of the rectangle by the user. The window rectangle can then be passed to the CreateWindowEx function to create a window, workspace kotorogo- desired size.

Syntax:

```
BOOL AdjustWindowRect  
(  
    LPRECT lpRect, // pointer to a structure  
    // User rectangle  
    DWORD dwStyle, // window style  
    BOOL bMenu // check menu display  
);
```

Options

lpRect

Pointer to a RECT structure that contains the coordinates of the upper left and lower right corners of the desired workspace. When the function returns, the structure contains the coordinates of the upper left and lower right corners of the window to set the allowable size of the desired workspace.

dwStyle

Specifies the window styles, the required size is to be calculated.

bMenu

It determines whether the window has a menu.

Return values

If the function succeeds, the return value is nonzero. If the function is not executed, the return value is zero. To get extended error information, call GetLastError.

remarks

User Rectangle - the smallest rectangle that completely encloses the work area. Rectangle box - the smallest rectangle that completely encloses the window. AdjustWindowRect function does not add extra space when a menu bar occupies two or more lines.

See also

AdjustWindowRectEx, CreateWindowEx, RECT

Accommodation and compatibility AdjustWindowRect

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ADJUSTWINDOWRECTEX

AdjustWindowRectEx function calculates the required size of the extended style of the window rectangle, based on the desired size of the working area of the window. The window rectangle can then be passed to the CreateWindowEx function to create a window, an area where the desired size.

Syntax:

```
BOOL AdjustWindowRect  
(  
LPRECT lpRect, // pointer to a structure  
// Workspace  
DWORD dwStyle, // window style  
BOOL bMenu // check menu display  
DWORD dwExStyle // extended style  
);
```

Options

lpRect

Pointer to a RECT structure that contains the coordinates of the upper left and lower right corners of the desired workspace. When the function returns, the structure contains the coordinates of the upper left and lower right corners of the window to set the allowable size of the desired workspace

dwStyle

Specifies the window styles, the required size is to be calculated.

bMenu

It determines whether the window has a menu.

dwExStyle

Defines the extended style of the window, which is required to be calculated size.

Return values

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call GetLastError.

remarks

User Rectangle - the smallest rectangle that completely encloses the work area. Rectangle box - the smallest rectangle that completely encloses the window. AdjustWindowRect function does not add extra space when a menu bar occupies two or more lines.

See also

AdjustWindowRect, CreateWindowEx, RECT

Accommodation and compatibility AdjustWindowRect

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Notes platform is not there

FUNCTION ARRANGEICONICWINDOWS

ArrangeIconicWindows function arranges all minimized (icon) child windows of the parent window is determined.

Syntax

```
UINT ArrangeIconicWindows  
(  
    HWND hWnd // handle to parent window  
);
```

Options

hWnd Identifies the parent window.

Return values

If the function has completed its work successfully, the return value is the height of one line of icons. If the function fails, the return value - zero. To get extended error information, call GetLastError.

remarks

An application that supports self minimized child windows can use ArrangeIconicWindows function to arrange the icons in the parent window. This function can also organize your desktop icons. To return the window handle to the main window, use the GetDesktopWindow. The application sends a message to the operating WM_MDIICONARRANGE box multi-document interface environment (MDI), to encourage the working window to organize child windows rolled multi-document interface environment (MDI).

See also

[CloseWindow](#), [GetDesktopWindow](#)

Accommodation and compatibility [ArrangeIconicWindows](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION BEGINDEFERWINDOWPOS

BeginDeferWindowPos function allocates memory for the structure of a multi-plant and returns a descriptor structure.

Syntax:

HDWP BeginDeferWindowPos

(

int nNumWindows // number of windows

);

Options

nNumWindows Specifies the initial number of windows which stores information about the location. Function DeferWindowPos, if necessary, to increase the size of the structure.

Return values

If the function succeeds, the return value identifies the structure of a multi-installation. If the available system resources are insufficient to distribute the structure, the return value is NULL (NULL).

remarks

Structure of a multi-installation - the internal structure; the application can not access it directly. DeferWindowPos function fills the structure of a multi-installation information about the location of the object for one or more windows are going to be displaced. EndDeferWindowPos function takes a handle to this structure, and again sets the window using the information stored in the structure. If the collateral to the windows in the structure of a multi-installation has a set of check boxes or SWP_HIDEWINDOW SWP_SHOWWINDOW, none of the windows is not installed again. If Windows is to increase the size of the structure of a multi-installation beyond the initial size specified parameter nNumWindows, but can not allocate enough memory to run it, Windows does not perform the task of the established procedure for the entire window (BeginDeferWindowPos, DeferWindowPos, and EndDeferWindowPos). In determining the required maximum size, the application can detect and process fails at the beginning of the process.

See also

DeferWindowPos, EndDeferWindowPos, SetWindowPos

Accommodation and compatibility BeginDeferWindowPos

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION BRINGWINDOWTOTOP

BringWindowToTop carries certain window function in the upper part of the Z-order. If the window - the top-level window is activated. If the window - a child window, the parent top-level window, associated with the child window is activated.

Syntax

```
BOOL BringWindowToTop  
(  
    HWND hWnd // handle of window  
);
```

Options

hWnd

It identifies the window that is transferred to the top of the Z-order.

Return values

If the function succeeds, the return value is zero. If the function fails, the return value is zero. To get extended error information, call GetLastError.

remarks

Use BringWindowToTop function to open any window that is partially or completely obscured by other windows. Calling this function call SetWindowPos similar functions to change the window position in Z-sequence. BringWindowToTop does not make the top-level window of the window. If the application is not in an active mode and wants to be in the active mode, it needs to call a function SetForegroundWindow.

See also

SetWindowPos, SetActiveWindow, SetForegroundWindow

Accommodation and compatibility BringWindowToTop

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CASCADEWINDOWS

CascadeWindows cascade function has certain window or child windows of the specified parent window.

Syntax

```
WORD WINAPI CascadeWindows  
(  
    HWND hwndParent, // handle to parent window  
    UINT wHow, // window types that are not ordered  
    CONST RECT * lpRect, // rectangle in which  
    // Ordered window  
    UINT cKids, // number of windows for ordering  
    const HWND FAR * lpKids // array of window descriptors  
)
```

Options

hwndParent

Identifies the parent window. If this parameter is set to NULL (NULL), accepted the desktop window.

wHow

Defines the box cascade. Currently available only check box, MDITILE_SKIPDISABLED, which prevents the blocking of child windows of MDI (Mnogodokumentalnogo Interface) from cascading.

lpRect

SMALL_RECT pointer to a structure that defines a rectangular area in screen coordinates, which are placed inside the window. This parameter can be NULL value (NULL), when using the work area of the parent window.

cKids

Specifies the number of elements in the array, a certain parameter lpKids. This parameter is ignored if lpKids - BLANK (NULL). lpKids

A pointer to an array of window handles that identify orderable window. If this parameter - NULL (NULL), child windows are placed in a certain parent window (or desktop window).

Return values

If the function succeeds, the return value - the number of orderable windows. If the function fails, the return value is zero.

See also

SMALL_RECT

Accommodation and compatibility CascadeWindows

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CHILDWINDOWFROMPOINT

ChildWindowFromPoint function determines which, if any, of the child windows belonging to the parent window contains the installation point (fixed).

Syntax

```
HWND ChildWindowFromPoint  
(  
    HWND hWndParent, // handle to parent window  
    POINT Point // structure with point coordinates  
)
```

Options

hWndParent

Identifies the parent window.

Point

Specifies a POINT structure that sets out to test the working coordinates of the point.

Return values

If the function succeeds, the return value - handle to the child window that contains the point, even if the child window is hidden or blocked. If the point is outside the parent window, the return value - NULL (NULL). If the point - in the parent window and not within any child window, the return value - the parent window handle.

remarks

Windows maintains an internal list that contains the descriptors of child windows of the parent window. The order of descriptors in the list depends on the Z-order of child windows. If more than one child window contains a fixed point, Windows returns a value in the first window descriptor list that contains the point.

See also

[ChildWindowFromPointEx](#), [POINT](#), [WindowFromPoint](#)

[Accommodation and compatibility ChildWindowFromPoint](#)

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CHILDWINDOWFROMPOINTEX

ChildWindowFromPointEx function determines which, if any, of the child windows belonging to a specific parent window contains the installation point. The function can ignore invisible, locked, and transparent child windows.

Syntax

```
HWND ChildWindowFromPointEx  
(  
    HWND hwndParent, // handle to parent window  
    POINT pt, // structure with point coordinates  
    UINT uFlags // skip flags  
);
```

Options

hwndParent

Identifies the parent window.

pt

Specifies a POINT structure that defines the work to verify the coordinates of the point.

uFlags

Specifies which child windows are skipped. This parameter can be a combination of the following values:

CWP_ALL - Do not skip any child windows.

CWP_SKIPINVISIBLE - Skip invisible child windows.

CWP_SKIPDISABLED - Ignore locked child windows.

CWP_SKIPTRANSPARENT - Skip transparent child windows.

Return values

If the function succeeds, the return value - the descriptor of the first child window that contains the point and meets the criteria set out in uFlags. If the point - in the parent window and not within any child window that meets the criteria, the return value - the parent window handle. If the point is outside the parent window, or if the function is not executed, the return value - NULL (NULL).

remarks

Windows maintains an internal list that contains the descriptors of the child windows of the parent window. The order of descriptors in the list depends on the Z-order of child windows. If more than one

child window contains a fixed point, Windows returns a handle to the first box in the list that contains the point and meets the criteria defined uFlags.

See also

[ChildWindowFromPoint](#), [POINT](#), [WindowFromPoint](#)

[Accommodation and compatibility ChildWindowFromPointEx](#)

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CLOSEWINDOW

Function CloseWindow rolls (but does not destroy) a certain window.

Syntax:

```
BOOL CloseWindow  
(  
    HWND hWnd // handle to the window, which coagulates  
);
```

Options

hWnd

It identifies the window that should be minimized.

Return values

If the function succeeds, the return value - not zero. If the function fails the return value - zero. To get extended error information, call GetLastError.

remarks

The window is rolled, reducing its size to the icon, and moves in the area of the screen icons. Windows displays the Windows icon instead of the window and displays the window title below the icon. To destroy a window, the application must use DestroyWindow function.

See also

ArrangeIconicWindows, DestroyWindow, IsIconic, OpenIcon

Accommodation and compatibility CloseWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CREATEWINDOW

CreateWindow function creates an overlapped, pop-up or child window. It defines the class, title, style, windows and (optional) to its original position and size of the window. The function also determines the parent window or owner, if any, and the window's menu.

Syntax:

```
HWND CreateWindow  
(  
LPCTSTR lpClassName, // pointer to registered class name  
LPCTSTR lpWindowName, // pointer to window name  
DWORD dwStyle, // window style  
int x, // horizontal position of window  
int y, // vertical position of window  
int nWidth, // width of the window  
int nHeight, // window height  
HWND hWndParent, // handle to parent or owner window  
HMENU hMenu, // handle menu or child-window identifier  
HANDLE hInstance, // handle to application instance  
LPVOID lpParam // pointer to window creation data  
);
```

Options

lpClassName

It indicates a null-terminated string at the end, or - to an integer atom. If this option - an atom, it must be shared atom created by a previous call to GlobalAddAtom function. Atom - a 16-bit value less than 0xC000, should be in the low lpClassName word; high word must be zero. If lpClassName - string, it specifies the name of the window class. The class name can be any name registered RegisterClass function or any of the predefined class names controls. Full list. See the next section for comments.

lpWindowName

It indicates a null-terminated string to the end that specifies the window name.

dwStyle

Specifies the style of the window being created. This parameter can be a combination of window styles and styles of dashboards listed in the next section below Notes.

x

Specifies the initial horizontal position of the window. For an overlapped or pop-up window, the parameter x - the initial x-coordinate of the upper left corner of the window, in screen coordinates. For a child window x - x-coordinate of the upper-left corner of the window relative to the left upper workspace corner of the parent window. If this parameter is set to CW_USEDEFAULT, Windows selects the default position for the upper left corner and ignores the y parameter. CW_USEDEFAULT parameter is valid only for overlapped windows; if it is specified for a pop-up or child window, x and y parameters are set to zero.

y

Specifies the initial vertical position of the window. For an overlapped or pop-up window, the parameter y - initial y-coordinate of the upper left corner of the window, in screen coordinates. For a child window, y - the initial y-coordinate of the upper left corner of the child window relative to the left upper workspace corner of the parent window. For a list box, y - the initial y-coordinate of the upper left corner of the workspace window with a list of the upper left area of the parent window of the working angle. If an overlapped window is created with a set WS_VISIBLE style a bit, and x is set as the CW_USEDEFAULT, Windows ignores the parameter y.

nWidth

Specifies the width of the window in the measuring unit for the device. For overlapped windows, nWidth is, or the width of the window in screen coordinates or parameter CW_USEDEFAULT. If nWidth - CW_USEDEFAULT, Windows selects the default width and height for the window; the default width extends from the initial x-coordinate to the right edge of the screen and the default height extends from the initial y-coordinate to the top of the icons. Meaning CW_USEDEFAULT allowed only for overlapped windows; if CW_USEDEFAULT defined for pop-up or child window, nWidth nHeight and set to zero.

nHeight

Specifies the window height, in device units. For overlapped windows, nHeight - the height of the window, in screen coordinates. If the parameter is set to nWidth CW_USEDEFAULT, Windows ignores nHeight.

hWndParent

Identifies the parent or owner window of the window being created. The correct window handle must be given when creating a child window or proprietor of the window. A child window is limited to the work area of its parent window. Located in the window of ownership - an overlay window is destroyed when the window is hidden or destroyed its owner when the owner rolled; It is always displayed at the top of the window to its owner. Despite the fact that this option must define a valid descriptor, if dwStyle option includes WS_CHILD style, it is not necessary, if dwStyle includes WS_POPUP style.

hMenu

Identifies the menu or determine the child-window identifier depending on the window style. For an overlapped or pop-up window, hMenu identifies the menu to be used with the window; if the class menu is to be used, it can be a NULL value (NULL). For a child window, hMenu determines the child-window identifier, an integer value used by a dialog box control to notify its parent about events. The application determines the child-window identifier; it must be unique for all child windows with the same parent window.

hInstance

It identifies an instance of the module, which is associated with the window.

lpParam

It indicates that the value passed to the window through CREATESTRUCT structure caused by using lpParam parameter of the WM_CREATE message. If an application calls CreateWindow, to create a multi-document user interface environment window (MDI), lpParam should point to the structure CLIENTCREATESTRUCT.

Return values

If the function succeeds, the return value - a descriptor created window. If the function fails the return value - NULL (NULL). To get extended error information, call GetLastError.

remarks

Before returning values, CreateWindow sends the WM_CREATE message to the window procedure.

For overlapped, pop-up windows and subsidiaries CreateWindow sends the WM_CREATE message window, and the WM_GETMINMAXINFO WM_NCCREATE. Parameter lpParam WM_CREATE message contains a pointer to a structure CREATESTRUCT. If you define a style WS_VISIBLE, CreateWindow sends the window all the messages required to activate and display a window.

If the window style specifies a title bar, window title specified in lpWindowName displayed in the title bar. When CreateWindow is used to create the controls, such as buttons, switches and static controls, lpWindowName used to determine the text control.

If the layout of your application you specify the version of Windows - 4.x, its windows can not have a button in the title bar if they also do not have a window menu. This is not required for the application, which you have determined the layout of the Windows version – 3.x.

The following predefined control classes can be defined in lpClassName parameter:

BUTTON

(BUTTON) Designates a small rectangular child window that represents the button, and the user can click the mouse on it to enable or disable it. The control buttons can be used alone or in groups, and they can either be signed or appear without text. Buttons usually change their form when the user clicks on them.

COMBOBOX

(Combo box) Represents a control consisting of a list box and select the field, similar to the element of text editing. When using this style, the application must display all the time or a window with a list or include a drop-down list. Depending on the style of the combo box, the user will be able to edit or select the contents of field. If a window with a list of probably typed characters within a field selection is highlighted by first entering the window with the list that matches the printed characters. On the contrary, the choice element in the list box displays the selected text in the selection box.

EDIT

(Editable WINDOW) Designates a rectangular child window within which the user can type text with the keyboard. The user selects the control and gives it the keyboard focus by clicking the mouse on it or moving it by pressing the TAB key (TAB). The user can print the text when you are editing window control displays a flashing caret (caret); Use the mouse to move the cursor, select the characters that will be replaced, or move the cursor to insert characters; or use the KEY RETURN TO POSITION (BACKSPACE), to delete the characters. edited window controls using the system font with variable pitch and show the on-screen characters from the ANSI character set. WM_SETFONT message can also be sent to the edit control's window to change the default font. The elements of the editing window management increase according to the number of tab characters so many spaces as they are required to move the carriage to the next tab stop. Tabs are taken such as to be in every eighth familiarity.

LISTBOX

(Window with a list) Indicates a list of character strings. This control is determined whenever the application must provide a list of names, such as file names from which the user can choose. The user can select a line by clicking the mouse on it. The selected line is highlighted, and a notification message is sent to the parent window. To scroll lists that are too long for the window of the control, use vertical or horizontal window with a list of the scroll bar. A window with a list of automatically hides or shows the scroll bar, as appropriate. MDICLIENT

Indicates working window MNOGODOKUMENTALNOGO INTERFACE (MDI). This window receives messages that control the child windows application MNOGODOKUMENTALNOGO interface. Recommended for its long - WS_CLIPCHILDREN and WS_CHILD. To create MDI operating window, which allows the user when a scroll MDI child windows, and determine WS_HSCROLL WS_VSCROLL styles. SCROLLBAR

(Scroll bar) Indicates a rectangle that contains a slider and has an arrow directed at both ends. The scroll bar sends a notification message to its parent window whenever the user clicks the control. If necessary, the parent window is responsible for the modification of the position of the slider. scroll bar controls are of the same type and use features as the scroll bar used in conventional windows. However, unlike the window scroll bars, scroll control line can be installed for use anywhere in the window when scrolling input information required for the window. The class also includes scroll bars and the size of the window controls. Window resizable - a small box that the user can pull to resize the window.

STATIC

(Static members) represents a single text box, box or rectangle used to label, box, or separate other controls. Static controls do not take any input information and do not provide any output information.

The following window styles can be defined in dwStyle parameter:

- **WS_BORDER** - Creates a window that has a thin frame line.
- **WS_CAPTION** - Creates a window that has a title bar (includes WS_BORDER style).
- **WS_CHILD** - Creates a child window. This style can not be used with WS_POPUP style.
- **WS_CHILDWINDOW** - Same as WS_CHILD style.
- **WS_CLIPCHILDREN** - Excludes the area occupied by child windows when drawing occurs within the parent window. This style is used when creating the parent window.
- **WS_CLIPSIBLINGS** - Secures the child windows relative to each other, that is, when a single child window receives WM_PAINT message, style WS_CLIPSIBLINGS fixes all other overlapping child windows out of the region of the child window that you want to modify. If WS_CLIPSIBLINGS style is not specified, and child windows overlap, it is possible that when drawing within the client area of the child window to be displayed inside the working area of a neighboring child window.
- **WS_DISABLED** - Creates a window that is initially disabled. Locked window can not accept user input.
- **WS_DLGFRADE** - Creates a window that has a border style typically used with dialog boxes. A window with this style can not have a title bar.
- **WS_GROUP** - Specifies the first control in a group of controls. The group consists of this first control and all controls defined after it, up to the next control with WS_GROUP style. The first control member of each group has usually WS_TABSTOP style, the user can move from one group to another. The user can then transfer the keyboard focus from one group of controls in the next group of controls using the arrow keys.
- **WS_HSCROLL** - Creates a window that has a horizontal scroll bar.
- **WS_ICONIC** - Creates a window that is initially minimized. The same style that WS_MINIMIZE.
- **WS_MAXIMIZE** - Creates a window that is initially maximized.
- **WS_MAXIMIZEBOX** - Creates a window that has a Maximize button (Maximize). It can be combined with WS_EX_CONTEXTHELP style. Also to be determined WS_SYSMENU style.
- **WS_MINIMIZE** - Creates a window that is initially minimized. The same style that WS_ICONIC.
- **WS_MINIMIZEBOX** - Creates a window that has a Close button (Minimize). It can be

combined with WS_EX_CONTEXTHELP style. Also to be determined WS_SYSMENU style.

- WS_OVERLAPPED - Creates an overlapped window. The overlay window has a title bar and frame. The same style that WS_TILED.
- WS_OVERLAPPEDWINDOW - Creates an overlapped window with styles WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX and WS_MAXIMIZEBOX. Same as WS_TILEDWINDOW style.
- WS_POPUP - Creates a pop-up window. This style can not be used with WS_CHILD style.
- WS_POPUPWINDOW - Creates a pop-up window with the styles WS_BORDER, WS_POPUP and WS_SYSMENU. WS_CAPTION WS_POPUPWINDOW and styles must be combined to make the window menu (window) is visible.
- WS_SIZEBOX - Creates a window that has the installation dimensions of the window frame. The same thing as WS_THICKFRAME style.
- WS_SYSMENU - Creates a window that has a window menu (window-menu) in its title bar. Also to be determined WS_CAPTION style.
- WS_TABSTOP - Specifies a control that can receive the keyboard focus when the user presses the TAB key (TAB). Pressing the tab key transfers keyboard focus to the next control with WS_TABSTOP style.
- WS_THICKFRAME - Creates a window that has the installation dimensions of the window frame. Same as WS_SIZEBOX style.
- WS_TILED - Creates an overlapped window. The overlay window has a title bar and frame. The same as the style WS_OVERLAPPED.
- WS_TILEDWINDOW - Creates an overlapped window with styles WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX and WS_MAXIMIZEBOX. The same as the style WS_OVERLAPPEDWINDOW.
- WS_VISIBLE - Creates a window that is initially visible.
- WS_VSCROLL - Creates a window that has a vertical scroll bar.

The following styles of buttons (BUTTON in class) that can be identified in the parameter dwStyle:

- BS_3STATE - Creates a button that is the same as the window for a check box, except that the window box can become inaccessible just as it is done when you select ("tick") checking (checked Only), or if you cancel it. Use the unavailable state to show that the state of the window for the check box is not defined.
- BS_AUTO3STATE - Creates a button that is in the same switch with three states, except that the window field changes its state when the user selects it. Condition cycles through the

installation verification flag phase, inaccessibility and cancel the installation.

- BS_AUTOCHECKBOX - Creates a button, which is also a window for a check box, except that the flag state of the installation check will automatically switch between the set and do not set the parameter, each time the user selects the button.
- BS_AUTORADIOBUTTON - Creates a button that is the same as the "radio" button, except that when the user selects it, Windows automatically sets the state of the buttons in the check box control regime, noting its "tick" and automatically sets the check state for all other buttons in the same group without checking flag.
- BS_CHECKBOX - Creates small, blank window for the check box with text. By default, the text is displayed to the right of the window. To display the text to the left of the window, combine this option with BS_LEFTTEXT style (or equivalent BS_RIGHTBUTTON style).
- BS_DEFPUSHBUTTON - Creates a command button that behaves like BS_PUSHBUTTON style button and also has a thick black border. If the button is in a dialog box, the user can select the button by pressing ENTER, and even when the button does not have input focus. This style is useful for enabling the user to quickly select the most appropriate (the default) option.
- BS_GROUPBOX - Creates a rectangle in which other control elements can be grouped. Any text associated with this style is displayed in the upper left corner of the rectangle.
- BS_LEFTTEXT - Places the text to the left of the "radio" button or the small window switch, when combined with a selector style or the "radio" button. The same thing as style BS_RIGHTBUTTON.
- BS_OWNERDRAW - Creates a button representing the owner. The owner window receives a message WM_MEASUREITEM, when the button is created and a message WM_DRAWITEM, when the appearance of the button has changed. Do not combine BS_OWNERDRAW style with any other button styles.
- BS_PUSHBUTTON - Creates a command button that sends a WM_COMMAND message of the owner window when the user selects the button.
- BS_RADIOBUTTON - Creates a small circle with the text. By default, the text is displayed to the right of the circle. To display the text to the left of the circle, combine this option with BS_LEFTTEXT style (or its equivalent - BS_RIGHTBUTTON style). Use the "Radio" button for a group of related, but mutually exclusive choices.
- BS_USERBUTTON - outdated, but provides for compatibility with 16-bit versions of Windows. Based on Win32 applications should use this option instead BS_OWNERDRAW.
- BS_BITMAP - Specifies that the button displays a bitmap.
- BS_BOTTOM - Places the text at the bottom of the button rectangle.

- BS_CENTER - Align the text horizontally in the center of the rectangle button.
- BS_ICON - Specifies that the button is displayed as an icon.
- BS_LEFT - Aligns text to the left of the button rectangle. However, if the button - the window switch or the "radio" button, which does not BS_RIGHTBUTTON style, the text is aligned to the right of the switch or the "radio" button.
- BS_MULTILINE - Wrap Text button in additional lines if the text string is too long to fit on one line in the rectangle button.
- BS_NOTIFY - Enables the button to send notification messages BN_DBLCLK, BN_KILLFOCUS and BN_SETFOCUS in its parent window. Note that buttons send a notification message BN_CLICKED regardless of whether it has this style.
- BS_PUSHLIKE - Creates a button (such as a switch, the switch to the three-state or "Radio" button) having a look and acts like a command button. The convex form of the button when it is pressed or not selected and submerged when it is clicked or selected.
- BS_RIGHT - Right Aligned text in the rectangle button. However, if the button - a window for the check box or the "Radio" button, which does not BS_RIGHTBUTTON style, the text is aligned to the right edge of the right side of the window for a check box or the "Radio" button.
- BS_RIGHTBUTTON - Sets the circle of "Radio" button or a window box for the flag to the right of the button rectangle. The same style that BS_LEFTTEXT.
- BS_TEXT - Specifies that the button displays text.
- BS_TOP - Stir the text at the top of the button rectangle.
- BS_VCENTER - Places the text in the middle (vertically) of the rectangle button.

Below are combined styles of windows (in the class COMBOBOX), which can be defined in the parameter dwStyle:

- CBS_AUTOHSCROLL - Automatically scrolls the text in the text edit box to the right when the user enters a character from the keyboard end of the line. If this style is not set, it accepted only the text that fits within the rectangular boundary of the field.
- CBS_DISABLENOSCROLL - The list box shows the vertical scroll bar is locked when the field box does not contain enough items to scroll. Without this style, the scroll bar is hidden if the window with the list does not contain enough items.
- CBS_DROPDOWN - CBS_SIMPLE is similar, except that the window with the list is not displayed until the user selects the icon next to the text editing.
- CBS_DROPDOWNLIST - Similar to CBS_DROPDOWN, except that the text edit field is replaced by a static text element that displays the current selection in the list box.

- CBS_HASSTRINGS - Specifies that the owner of the combo box contains items consisting of strings. Combo Box supports memory and address lines, so that the application can use CB_GETLBTEXT message to restore the text for a particular item.
- CBS_LOWERCASE - Converts to lowercase any uppercase characters entered in the edit combo box text.
- CBS_NOINTEGRALHEIGHT - Specifies that the size of the combo box - this is the exact size specified by the application when it created the combo box. Typically, Windows sets the size of the combo box so that it does not display the part.
- CBS_OEMCONVERT - Converts text entered in the edit field of the combo box text. The text is converted from the Windows character set to the OEM character set and then back to the Windows set. This ensures that the appropriate character conversion when the application calls CharToOem function to convert a Windows string in the combo box to OEM characters. This style is most useful for combo boxes that contain file names and apply only to combo boxes created with style CBS_SIMPLE or CBS_DROPDOWN.
- CBS_OWNERDRAWFIXED - Specifies that the owner of the list box is responsible for drawing its contents and that the items in the window with a list of all of equal height. The owner window receives a WM_MEASUREITEM, when the combo box is created, and the message WM_DRAWITEM, when the appearance of the combo box has changed.
- CBS_OWNERDRAWVARIABLE - Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are variable in height. The owner window receives WM_MEASUREITEM message for each item combo box when you create the combo box; owner window receives a message WM_DRAWITEM when changed the look of the combo box.
- CBS_SIMPLE - Always display a window with a list. The current selection in the list box is displayed in the text editing.
- CBS_SORT - Automatically sorts strings entered in the list window.
- CBS_UPPERCASE - Converts all lowercase characters to uppercase characters typed in the edit box the combo box text.

The following styles of text editing field (in the classroom EDIT) can be defined in dwStyle parameter:

- ES_AUTOHSCROLL - Automatically scrolls text to the right by 10 characters when the user types the symbol at the end of lines. When the user presses ENTER, and control scrolls all text back to set zero.
- ES_AUTOVSCROLL - automatically moves the text up one page when the user presses the ENTER key on the last line.
- ES_CENTER - is centered in the text field multiline text editing.

- **ES_LEFT** - Align text to the left.
- **ES_LOWERCASE** - Converts all characters to lowercase as they are printed in the text edit field.
- **ES_MULTILINE** - Indicates a multi-line text editing window. The default setting - single-line text editing window. When a multi-line edit box is located in the dialog box, the default response to pressing the ENTER key must activate the default button. To use the ENTER key to a newline, use style **ES_WANTRETURN**. When a multi-line edit box is not in the dialog box and define the style **ES_AUTOVSCROLL**, the edit box shows many lines as possible and scrolls vertically when the user presses the ENTER key. If you do not specify **ES_AUTOVSCROLL**, editing window shows as many lines as possible and beeps if the user presses ENTER, and yet more than a single line can not be displayed in the window. If you specify **ES_AUTOHSCROLL** style, multi-line edit box 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 you do not specify **ES_AUTOHSCROLL**, control, when necessary, automatically transfers without breaking words at the beginning of the next line. A new line is formed and then, if the user presses the ENTER key. The window size determines the position of the word transition to a new line. If the window size is changed, the position changes of transition to a new line, and the text is restored. Multi-line text edit box can have scroll bars. The editing window with scroll bars treated with their own messages on the scroll bar. Note that the editing window without scroll bars, scroll through the text, as described in the previous paragraphs and process any scroll messages sent by the parent window.
- **ES_NOHIDESEL** - Negates the default behavior for an edit text field. The default behavior hides the selection when the control loses the input focus and inverts the selection when the control panel receives the input focus. If you specify **ES_NOHIDESEL**, the selected text is inverted, even if the control panel does not have focus.
- **ES_NUMBER** - Allows you to enter into the edit field only numbers.
- **ES_OEMCONVERT** - Converts text entered in the edit box. The text is converted from the Windows character - in the OEM character set and then back - set in Windows. This ensures that the appropriate character conversion when the application of the function is called **CharToOem**, to convert a Windows string in the edit box to OEM characters. This style is most useful for text editing windows that contain file names.
- **ES_PASSWORD** - Displays an asterisk (*) instead of each character entered from the keyboard in the edit window. You can use the message **EM_SETPASSWORDCHAR**, to replace its symbol, which is displayed.
- **ES_READONLY** - Do not allow the user to enter or edit text in the edit box.
- **ES_RIGHT** - right-aligned text in a multiline edit box.

- **ES_UPPERCASE** - Converts all characters to uppercase characters as they are entered in the editing window.
- **ES_WANTRETURN** - Specifies that the service carriage return code is inserted when the user presses the ENTER key while entering text into a multiline text edit box in the dialog box. If you do not specify this style, pressing the ENTER, and you get the same effect as if clicked the default command button of the dialog box. This style has no effect in a single edit window.

The following window control styles with the list (in the LISTBOX class) can be defined in dwStyle parameter:

- **LBS_DISABLENOSCROLL** - Shows blocked vertical scrollbar in a window with a list of when the window box does not contain enough items to scroll. If you do not specify this style, the scroll bar is hidden when the window with the list does not contain enough items.
- **LBS_EXTENDEDSEL** - Allows multiple items to be selected, by using the SHIFT key and the mouse or special key combinations.
- **LBS_HASSTRINGS** - Specifies that the window with the list contains items consisting of strings. A window with a list of strings and saves memory addresses, so that an application can use LB_GETTEXT message to restore the text for a particular item. By default, all windows with the list except for windows with the list provided by the owner have this style. You can create a window provided by the owner of the list with or without this style.
- **LBS_MULTICOLUMN** - Identifies a multiple-window with a list that scrolls horizontally. Post LB_SETCOLUMNWIDTH sets the width of the columns.
- **LBS_MULTIPLESEL** - Enable or disable the selection of a sequence of characters each time the user one or double-click to activate a string of characters in the list box. The user can select any number of rows.
- **LBS_NODATA** - Defines the "lack of data" in the list box. This style is determined by when the number of items in the list box may exceed one thousand. A window with a list of "missing data" must also have style LBS_OWNERDRAWFIXED, but it should not have style or LBS_SORT LBS_HASSTRINGS. A window with a list of "missing data" is similar to a window with a list provided by the owner, except that it contains no string or bitmap data for the item. Commands "add", "insert" or "delete" item always ignore any transmitted data elements; a request to search for a string in a window always fails. Windows sends a message WM_DRAWITEM owner window when an item to be traced. ElementID (itemID) member DRAWITEMSTRUCT structure passed with the message WM_DRAWITEM, determines the row number to be traced. A window with a list of "missing data" does not send a message WM_DELETEITEM.
- **LBS_NOINTEGRALHEIGHT** - Specifies that the size of the window with the list corresponds to the size specified by the application when it created the list box. Typically, Windows sets the

combo box so that it does not display the part.

- LBS_NOREDRAW - Specifies that the window view from the list is not modified when changes are made. You can at any time change this style by sending a message WM_SETREDRAW.
- LBS_NOSEL - Specifies that a window contains a list of items that can be viewed, but not selected.
- LBS_NOTIFY - Notifies whenever the user clicks or double-clicks a row in the list box, the parent of an incoming message window.
- LBS_OWNERDRAWFIXED - Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box appear the same height. The owner window receives a WM_MEASUREITEM when the window with the list created and WM_DRAWITEM message when the window's appearance changed.
- LBS_OWNERDRAWVARIABLE - Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box appearing on the height variable. The owner window receives WM_MEASUREITEM message for each item in the list window when it is created, and WM_DRAWITEM message when the window's appearance changed.
- LBS_SORT - Sorts the strings in the list box alphabetically.
- LBS_STANDARD - Sorts the strings in the list box alphabetically. The parent window receives an incoming message whenever the user clicks or double-clicks on the line. The window with the list is framed on all sides.
- LBS_USETABSTOPS - Enables a list box to recognize and expand the characters in a table when drawing its strings. The default table occupies 32 dialog box units. Unit dialog box - horizontal or vertical distance. One horizontal dialog box unit is equal to the fourth part of the current measurement units of the dialog box size. Windows calculates these units, based on the font height and width of the existing system. GetDialogBaseUnits function returns the current base unit of measure of the dialog box, in pixels.
- LBS_WANTKEYBOARDINPUT - Specifies that the owner of the list box receives messages WM_VKEYTOITEM whenever the user presses a key, and a window with the input focus is. This enables the application to perform special processing when entering from the keyboard.

The following styles scrollbars (a class SCROLLBAR) can be defined in dwStyle parameter:

- SBS_BOTTOMALIGN - Aligns the bottom edge of the scroll bar to the bottom edge of the rectangle defined by the parameters x, y, nWidth, and nHeight. The scroll bar has the default height for system scroll bars. Use this style to SBS_HORZ style.
- SBS_HORZ - Indicates the horizontal scroll bar. If no styles SBS_BOTTOMALIGN, nor SBS_TOPALIGN not defined, the scroll bar has the height, width, and position specified x, y,

nWidth and nHeight.

- SBS_LEFTALIGN - Aligns the left edge of the scroll bar to the left edge of the rectangle defined by the parameters x, y, nWidth and nHeight. The scroll bar has the default width for system scroll bars. Use this style to SBS_VERT style.
- SBS_RIGHTALIGN - Aligns the right edge of the scroll bar to the right edge of the rectangle defined by the parameters x, y, nWidth, and nHeight. The scroll bar has the default width for system scroll bars. Use this style to SBS_VERT style.
- SBS_SIZEBOX - Indicates the size of the window. If you do not specify any SBS_SIZEBOXBOTTOMRIGHTALIGN, nor SBS_SIZEBOXTOPLEFTALIGN style, window size has a height, width and position determination of the parameters x, y, nWidth and nHeight.
- SBS_SIZEBOXBOTTOMRIGHTALIGN - Aligns the size of the lower-right corner of the window to the lower right corner of the rectangle defined by the parameters x, y, nWidth, and nHeight. The window size is the default size for the window size of the system. Use this style to SBS_SIZEBOX style.
- SBS_SIZEBOXTOPLEFTALIGN - Aligns the size of the upper-left corner of the window to the upper left corner of the rectangle defined by the parameters x, y, nWidth, and nHeight. The window size is the default size for the window size of the system. Use this style to SBS_SIZEBOX style.
- SBS_SIZEGRIP - Similar to the style SBS_SIZEBOX, but with a convex frame.
- SBS_TOPALIGN - Aligns the top edge of the scroll bar to the upper edge of the rectangle defined by the parameters x, y, nWidth, and nHeight. The scroll bar has the default height for system scroll bars. Use this style to SBS_HORZ style.
- SBS_VERT - Designates a vertical scroll bar. If you do not specify any SBS_RIGHTALIGN, nor SBS_LEFTALIGN style scroll bar has the height, width and position determination of the parameters x, y, nWidth and nHeight.

The following static control styles (in the STATIC class) can be defined in dwStyle parameter. Static control can only have one of these styles:

- SS_BITMAP - Specifies that static control should display the bitmap. The text of the error code - the name of the bitmap (not a filename) defined elsewhere in the resource file. Style and ignores the nWidth nHeight; control automatically sets its own dimensions, to place a bitmap.
- SS_BLACKFRAME - Specifies the window frame, using the same color as that of the frame of the main window. This color is black in the default Windows color scheme.
- SS_BLACKRECT - Specifies a rectangle filled with the current window frame color. By default, the color black in the Windows color scheme.

- SS_CENTER - Specifies a simple rectangle and centers the error code in the text box. The text is formatted before it is displayed on the screen. Words that extend beyond the end of the line is automatically transferred to the beginning of the next centered line.
- SS_CENTERIMAGE - Specifies that the middle point of the static control with SS_BITMAP style or SS_ICON to remain fixed when the control is changed. The four sides are adjusted so as to put a new bitmap or icon. If a static control has SS_BITMAP style and bitmap less than the working area of the control, workspace filled with the color of the pixel of the upper left corner of the bitmap. If a static control has SS_ICON style icon appears, but does not stain the work area.
- SS_GRAYFRAME - Specifies the window field frame derived the same color as the screen background (desktop). By default, the Windows color scheme, this color is gray.
- SS_GRAYRECT - Specifies a rectangle filled with the current screen background color. By default, the Windows color scheme, this color is gray.
- SS_ICON - Specifies the icon that is displayed in the dialog box. This text - the name of the icon (not a filename) defined elsewhere in the resource file. Style and ignores the nWidth nHeight; icon automatically sets its value.
- SS_LEFT - Specifies a simple rectangle and left-aligned text in a rectangle. The text is formatted before it is displayed. Words that extend beyond the end of the line is automatically transferred to the beginning of the next aligned with the left margin.
- SS_LEFTNOWORDWRAP - Specifies a simple rectangle and left-aligned text in a rectangle. The plates are expanded, but words are not tolerated. The text that extends beyond the end of the line is cut off.
- SS_METAPICT - Specifies that the metafile image should be displayed in the static control. This text - the name of the metafile images (not a filename) defined elsewhere in the resource file. Static metafile element management has a fixed size; metafile image is scaled to fit the working area static control.
- SS_NOPREFIX - Prevents interpretation of any ampersand symbol (&) in the control's text as a symbol of an accelerator prefix. They are displayed with a remote ampersand and followed by the underlined character in the string. This static control style may be included with any of the defined static controls. An application can be combined with other styles SS_NOPREFIX using bitwise OR = OR (|). This may be useful when the file names, or other lines that may contain an ampersand (&) to be displayed in a static dialog control.
- SS_NOTIFY - Sends notification messages to the parent window and STN_CLICKED STN_DBCLK, when the user clicks or double-clicks the control.
- SS_RIGHT - Specifies a simple rectangle and right-aligned text in a rectangle. The text is formatted before it is displayed on the screen. Words that extend beyond the end of the line is

automatically transferred to the beginning of the next aligned with the right edge of the line.

- SS_RIGHTIMAGE - Specifies that the angle of the bottom right of a static control with the style or SS_BITMAP SS_ICON must remain fixed when the control is changed. Only the top and left side are adjusted to fit the new bitmap or icon.
- SS_SIMPLE - Specifies a simple rectangle and displays a single line of left-aligned text in the rectangle border. The text string can not be shortened or changed in any way. Parental Control Panel window or dialog box should not process a message WM_CTLCOLORSTATIC.
- SS_WHITEFRAME - Specifies the window field frame derived the same color as the window background. By default, in Windows Color System - the color white.
- SS_WHITERECT - Specifies a rectangle filled with the current color of the window background. By default, in Windows Color System - the color white.

Listed below dialog styles that can be identified in the parameter dwStyle:

- DS_3DLOOK - Provides a dialog box is not bold, and displays three-dimensional border around the window controls in the dialog box.
- DS_3DLOOK - This style is required only on Win32-based application programs compiled for versions of Windows earlier than Windows 95 or Windows NT 4.0. The system automatically applies the three-dimensional look to dialog boxes created by applications compiled for current versions of Windows.
- DS_ABSALIGN - Indicates that the coordinates of the dialog box - screen coordinates; otherwise, Windows takes them for user coordinates.
- DS_CENTER - is aligned with the center of the dialog box in the work area; ie in the area, do not obstruct the panel.
- DS_CENTERMOUSE - is aligned with the center of the mouse cursor in the dialog box.
- DS_CONTEXTHELP - Includes a question mark in the title bar of the dialog box. When the user clicks the question mark, the cursor changes to a question mark with an arrow-pointer. If the user then clicks a control in the dialog box, the control receives a message WM_HELP. The control should pass the message to the dialog procedure, which should call the WinHelp function using a command HELP_WM_HELP. Application Help (Help) displays a pop-up window that typically contains help for the control. Note that DS_CONTEXTHELP - just a notch - filler. When the dialog box is created, the system checks for DS_CONTEXTHELP and, if available, WS_EX_CONTEXTHELP adds to the extended style of the dialog box.
WS_EX_CONTEXTHELP can not be used with styles WS_MAXIMIZEBOX or WS_MINIMIZEBOX.
- DS_CONTROL - Creates a dialog box that works well as a child window of another dialog box, much like a page in the properties window. This style allows the user to navigate among the

controls child dialog use it accelerator keys and so on.

- DS_FIXEDSYS - Uses SYSTEM_FIXED_FONT instead SYSTEM_FONT.
- DS_LOCALEEDIT - Applies only for 16-bit applications. This style edit control in the dialog box to allocate memory in the data segment of the application program. Otherwise, edit elements reserved memory the global memory object.
- DS_MODALFRAME - Creates a dialog box with a modal dialog box frame that can be combined with a title bar and window menu by specifying WS_CAPTION WS_SYSMENU and styles.
- DS_NOFAILCREATE - Creates even dialog box if errors occur - for example, if a child window can not be created or if the system can not create a special data segment for editing elements.
- DS_NOIDLEMSG - Suppresses WM_ENTERIDLE messages that Windows would otherwise be sent to the owner of the dialog box, while the displayed dialog box.
- DS_RECURSE - Style dialog box to dialog boxes like the control.
- DS_SETFONT - Specifies that the dialog box template (DLGTEMPLATE structure) contains two additional elements that define the font name and size in points. The corresponding font used to display text within the client area of the dialog box and in the dialog box controls. Windows passes the handle of the font dialog box and to each control by sending them a message WM_SETFONT.
- DS_SETFOREGROUND - Not Applicable B16-bit versions of Microsoft Windows. This style causes the dialog box to the active mode. Inside Windows calls for dialog SetForegroundWindow function.
- DS_SYSMODAL - Creates a system-modal dialog box. This style causes the dialog box to have the WS_EX_TOPMOST style, but otherwise, it has no influence on the dialog box or the behavior of other windows in the system, when a dialog box appears.

Windows 95: The system can support a maximum of 16,364 window handles.

See also

CharToOem, CLIENTCREATESTRUCT, CreateDialog, CREATESTRUCT, CreateWindowEx, DialogBox, DLGTEMPLATE, DRAWITEMSTRUCT, GetDialogBaseUnits, GlobalAddAtom, LB_GETTEXT, LB_SETCOLUMNWIDTH, MessageBox, RegisterClass, SetForegroundWindow, WM_COMMAND, WM_CREATE, WM_DELETEITEM, WM_DRAWITEM, WM_ENTERIDLE, WM_GETMINMAXINFO, WM_MEASUREITEM, WM_NCCREATE, WM_PAINT, WM_SETFONT, WM_SETREDRAW, WM_VKEYTOITEM

Accommodation and compatibility CreateWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION CREATEWINDOWEX

CreateWindowEx function creates an overlapped, jumps or child window with an extended style; otherwise, the function is identical to CreateWindow. For more information about creating a window and for full descriptions of the other parameters of CreateWindowEx, see Article the CreateWindow.

Syntax

```
HWND CreateWindowEx
(
    DWORD dwExStyle, // window style improved
    LPCTSTR lpClassName, // pointer to registered class name
    LPCTSTR lpWindowName, // pointer to window name
    DWORD dwStyle, // window style
    int x, // horizontal position of window
    int y, // vertical position of window
    int nWidth, // width of the window
    int nHeight, // window height
    HWND hWndParent, // handle to the parent window, or
    // owner
    HMENU hMenu, // menu descriptor or identifier
    // Child window
    HINSTANCE hInstance, // handle to application instance
    // programs
    LPVOID lpParam // pointer to window creation data
);
```

Options

dwExStyle

Specifies the extended window style. This parameter can be one of the following:

- WS_EX_ACCEPTFILES - Specifies that a window created with this style accepts files with the help of information technology "drag-and-paste".
- WS_EX_APPWINDOW - Activates the top-level window on the taskbar when the window is minimized.

- **WS_EX_CLIENTEDGE** - Specifies that a window has a border with intensive margin.
- **WS_EX_CONTEXTHELP** - Includes a question mark in the title bar of the window. When the user clicks the question mark, the cursor changes to a question mark with a pointer. If the user then clicks a child window, the child receives a message WM_HELP. The child window should pass the message to the parent window procedure, which should call the WinHelp function using the command HELP_WM_HELP. Help application displays a pop-up window that typically contains help for the child window. WS_EX_CONTEXTHELP can not be used with styles WS_MAXIMIZEBOX or WS_MINIMIZEBOX.
- **WS_EX_CONTROLPARENT** - Allows the user to navigate among the child windows of the main window using the Tab key (TAB).
- **WS_EX_DLGMODALFRAME** - Creates a window that has a double border; box can be created (optional) with a title bar, which determines the style in WS_CAPTION dwStyle parameter.
- **WS_EX_LEFT** - Window has generic properties "left-aligned". This - the default.
- **WS_EX_LEFTSCROLLBAR** - If the shell language is Hebrew, Arabic, or another language that supports reading order, the vertical scroll bar (if present) - to the left of the workspace. For other languages, the style is ignored and not treated as an error.
- **WS_EX_LTRREADING** - text window is displayed using the property reading order Left - Right. This - the default.
- **WS_EX_MDICHILD** - Creates an MDI child window.
- **WS_EX_NOPARENTNOTIFY** - Specifies that a child window created with this style does not send a message WM_PARENTNOTIFY parent window when it is created or destroyed.
- **WS_EX_OVERLAPPEDWINDOW** - Combines WS_EX_CLIENTEDGE
WS_EX_WINDOWEDGE and styles.
- **WS_EX_PALETTEWINDOW** - Combines WS_EX_WINDOWEDGE styles,
WS_EX_TOOLWINDOW and WS_EX_TOPMOST.
- **WS_EX_RIGHT** - Window has generic properties of the "right-aligned". It depends on the window class. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports a different procedure for the alignment for reading; otherwise, the style is ignored and not treated as an error.
- **WS_EX_RIGHTSCROLLBAR** - Vertical scroll bar (if present) - the right of the workspace. This - the default.
- **WS_EX_RTLREADING** - If the shell language is Hebrew, Arabic, or another language that supports different alignment order to read the text in the window is displayed using the

properties of the reading order, Right - Left. For other languages, the style is ignored and not treated as an error.

- WS_EX_STATICEDGE - Creates a window with a three-dimensional border style intended to be used for items that do not accept user input.
- WS_EX_TOOLWINDOW - Creates a tool window; that is, the window will be used as a floating toolbar. A tool window has a title bar that is shorter than a normal title bar, and the window title is drawn using a smaller font. A tool window does not appear in the taskbar or in the dialog box that appears when the user presses ALT + TAB.
- WS_EX_TOPMOST - Specifies that a window created with this style should be placed above all non-topmost windows and should stay above them, even when the window is deactivated. To add or remove this style, use the SetWindowPos.
- WS_EX_TRANSPARENT - Specifies that a window created with this style should be transparent. That is, any windows that appear out of the window, not obscure them. A window created with this style receives WM_PAINT messages only after all the nursing window underneath modified.
- WS_EX_WINDOWEDGE - Specifies that a window has a border with a raised edge.

Using WS_EX_RIGHT style for static or editable controls has the same effect as using SS_RIGHT style or ES_RIGHT, respectively. Use this style with the command buttons has the same effect as the use of styles and BS_RIGHT BS_RIGHTBUTTON.

lpClassName

It indicates a null-terminated string to the end or an integer atom. If lpClassName - an atom, it must be a global atom created by a previous call GlobalAddAtom function. The atom, a 16-bit value less than 0xC000, should be in the low word lpClassName; high word must be zero. If lpClassName - string, it specifies the name of the window class. The class name can be any name registered RegisterClass function or any of the predefined class names controls.

lpWindowName

It indicates a null-terminated string to the end that specifies the window name.

dwStyle

Specifies the style of the window being created. As for the rest of the list box styles that can be defined in dwStyle, cm. CreateWindow.

x

Specifies the initial horizontal position of the window. For an overlapped or pop-up window parameter x - the initial x-coordinate of the upper left corner of the window, in screen coordinates. For a child window x - x-coordinate of the upper-left corner of the window relative to the left upper workspace

corner of the parent window. If x is set to CW_USEDEFAULT, Windows selects the default position for the upper left corner and ignores the y parameter. CW_USEDEFAULT is valid only for overlapped windows; if it is specified for a pop-up or child window, the x and y parameters are set to zero.

y

Specifies the initial vertical position of the window. For an overlapped or pop-up window, the parameter y - initial y-coordinate of the upper left corner of the window, in screen coordinates. For a child window, y - the initial y-coordinate of the upper left corner of the child window relative to the left upper workspace corner of the parent window. For a list box, y - the initial y-coordinate of the upper left corner of the workspace window with a list of the upper left area of the parent window of the working angle. If an overlapped window is created in the style of WS_VISIBLE bit set and the x parameter is set to CW_USEDEFAULT, Windows ignores the parameter y.

nWidth

Specifies the width of the window, in device units. For overlapped windows nWidth - the width of the window in screen coordinates or CW_USEDEFAULT. If nWidth - CW_USEDEFAULT, Windows selects the default width and height for the window; the default width extends from the initial x-coordinate to the right edge of the screen; The default height extends from the initial y-coordinate to the top of the icon area. CW_USEDEFAULT is valid only for overlapped windows; if CW_USEDEFAULT specified for a pop-up or child window, the parameters and nWidth nHeight set to zero.

nHeight

Specifies the window height, in device units. For overlapped windows, nHeight - the height of the window in screen coordinates. If the parameter is set to nWidth CW_USEDEFAULT, Windows ignores nHeight.

hWndParent

Identifies the parent or owner window of the window being created. The valid window handle should be given when the child window or an owned window is created. A child window is limited to the work area of the parent window. An owned window - an overlay window that is destroyed when its owner window is destroyed or hidden when its owner is minimized; It is always displayed at the top of the window of the owner. Despite the fact that this option must determine the correct descriptor, if dwStyle option includes the WS_CHILD style, it is not necessary if dwStyle includes WS_POPUP style.

hMenu

Identifies the menu or, depending on the window style specifies the identity of the child window. For an overlapped or pop-up window, hMenu identifies the menu that will be used by the window; This parameter can be NULL value (NULL), if the class menu is used. For a child window, hMenu determines the child-window identifier, an integer value used by a dialog box control is to inform parents about the events. The application determines the child-window identifier; it must be unique for all child windows with the same parent window.

hInstance

It identifies the instance of the module that will be associated with a window.

lpParam

It indicates that the value passed to the window through CREATESTRUCT structure caused by the parameter lParam WM_CREATE message. If the application calls CreateWindow, to create a custom multi-document window environment, lpParam should point to the structure CLIENTCREATESTRUCT.

Return values

If the function successfully completed its work, the return value - a handle to the created window. If the function fails, the return value - NULL (NULL).

Notes CreateWindowEx function sends the created window WM_NCCREATE posts, WM_NCCALCSIZE and WM_CREATE. For information about the classes of the window controls, window styles and control styles used with this function, refer to the description of the CreateWindow function. Windows 95: The system can support a maximum of 16,364 window handles.

See also

CLIENTCREATESTRUCT, CREATESTRUCT, CreateWindow, GlobalAddAtom, RegisterClass, SetWindowPos, WM_CREATE, WM_NCCALCSIZE, WM_NCCREATE, WM_PAINT, WM_PARENTNOTIFY

Accommodation and CreateWindowEx compatibility:

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Notes hMenu platform is limited in size (WORD) WORD in Win32s

FUNCTION DEFERWINDOWPOS

DeferWindowPos function modifies the structure defining the position of multiple windows in a particular window. The function then returns a handle modifiable structure. EndDeferWindowPos function uses the information in this structure to change the position and size of a number of windows simultaneously. BeginDeferWindowPos function creates a structure.

HDWP DeferWindowPos

```
(  
    HDWP hWinPosInfo, // handle to the internal structure  
    HWND hWnd, // handle to window placement  
    HWND hWndInsertAfter, // handle the order placement  
    int x, // horizontal position  
    int y, // vertical position of the  
    int cx, // width  
    int cy, // height  
    UINT uFlags // window positioning flags  
);
```

Options

hWinPosInfo

It identifies a plurality of windows are placed structure that contains information about the size and position for one or more windows. This structure is returned by BeginDeferWindowPos or most recent call DeferWindowPos function.

hWnd

Identifies the window for which the modification information is stored in the structure.

hWndInsertAfter

It identifies the window that is preceded by a window installed in Z-sequence. This parameter must be a window handle or one of the following:

HWND_BOTTOM - Places the window at the bottom of the Z - order. If the hWnd parameter identifies a topmost window, the window loses its topmost status and is placed at the bottom of all other windows.

HWND_NOTOPMOST - Places the window above all non-topmost windows (that is, behind all topmost windows). This flag has no effect if the window - is not the topmost window.

HWND_TOP - Places the window at the top of the Z - order.

HWND_TOPMOST - Places the window above all non-topmost windows. Window supports the topmost position even when it is deactivated.

This parameter is ignored if the flag is set to **SWP_NOZORDER** uFlags parameter.

x

Sets the x-coordinate of the upper left corner of the window.

y

Sets the y-coordinate of the upper-left corner of the window.

cx

Specifies the new width of the window, in pixels.

cy

Specifies the new height of the window, in pixels.

uFlags

Sets the following sequence of values that affect the size and position of the window:

- **SWP_DRAWFRAME** - Prints frame (defined in the description of the class of the window) around the window.
- **SWP_FRAMECHANGED** - Sends a message window WM_NCCALCSIZE, even if does not change the window size. If this option is not specified, WM_NCCALCSIZE sent only when the window is resized.
- **SWP_HIDEWINDOW** - Hides the window.
- **SWP_NOACTIVATE** - Do not activate the window. If this is not checked, the window is activated and moved to the top of either the topmost or non-topmost group (depending on the setting hWndInsertAfter parameter).
- **SWP_NOCOPYBITS** - Resets all the contents of the working area. If this is not checked, the valid contents of the work area are saved and copied back into the client area after the window is sized or repositioned.
- **SWP_NOMOVE** - Retains current position (ignores the X and Y).
- **SWP_NOOWNERZORDER** - Does not change the owner window's position in the Z-order.
- **SWP_NOREDRAW** - not redraw changes. If this flag is set, no repainting of any kind occurs. This applies to the workspace overscan (including the header area and a scroll bar), and any part of the parent window uncovered by the movement of the window. When this option is selected, the application must explicitly invalidate or redraw any parts of the window and parent window

that need to be redone.

- SWP_NOREPOSITION - The same as the box SWP_NOOWNERZORDER.
- SWP_NOSENDCHANGING - Prevents the window from receiving WM_WINDOWPOSCHANGING posts.
- SWP_NOSIZE - Retains the current size (ignores the cx and cy).
- SWP_NOZORDER - Retains the current Z-order (ignores hWndInsertAfter option).
- SWP_SHOWWINDOW - Shows window.

Return values

The return value identifies the streamlined structure of the composite window location. The handle returned by this function may differ from the handle passed to the function. The new handle that this function returns should be passed during the next call to the function or DeferWindowPos EndDeferWindowPos. If the missing system resources are available for the function to succeed, the return value - NULL (NULL).

remarks

If an appeal to DeferWindowPos fails, the application should abandon the window positioning operation and not call EndDeferWindowPos. If SWP_NOZORDER not installed, Windows places the window identified by the hWnd parameter in the position after the window identified by the parameter hWndInsertAfter. If hWndInsertAfter - BLANK (NULL) or HWND_TOP, Windows places the hWnd window at the top of Z-sequence. If hWndInsertAfter set HWND_BOTTOM, Windows places the hWnd window at the bottom of Z-sequence. All coordinates for child windows - relative to the left upper workspace corner of the parent window. The window can be made at the top of the window or by setting the flag hWndInsertAfter HWND_TOPMOST and ensuring that SWP_NOZORDER flag is not set, or set the position of the window in the Z-order, so that it is above any existing topmost windows. When a non-topmost window is made topmost, its owned windows are also made topmost. Its owners, however, do not change. If neither flag nor SWP_NOACTIVATE SWP_NOZORDER check box is not selected (ie when the application's request to at the same time the window has been activated, and its position in the Z-order is changed), the value set in the hWndInsertAfter, used only in the following circumstances:

No HWND_TOPMOST HWND_NOTOPMOST box or box is not checked in hWndInsertAfter.

The window identified by hWnd - is not the active window.

The application program can not activate an inactive window without also bringing it to the top of Z-order. The application program can change the position of a window activated in the sequence Z-without restriction, or it can activate a window and then move it to the top of the uppermost or uppermost windows. The topmost window uppermost longer if it is reset to the lower part (HWND_BOTTOM) Z-after any sequence or not the uppermost window. When a topmost window is

made not at the top, its owners and its owned windows are also made non-topmost windows. Not topmost window may have at the top of the window, but not vice versa. Any window (for example, a dialog box) owned by a topmost window is itself made a topmost window to ensure that all owned windows are above their owner.

See also

BeginDeferWindowPos, EndDeferWindowPos, ShowWindow

Accommodation and compatibility DeferWindowPos

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION DESTROYWINDOW

DestroyWindow function destroys the specified window. The function sends a message and WM_DESTROY WM_NCDESTROY window to deactivate it and remove the keyboard focus from it. The function also destroys the window's menu, clears the queue streams, destroys timers, removes the ownership of the clipboard and breaks the clipboard viewer chain of windows (if the window is at the top of the chain of views). If a particular window - the parent or owner window, the DestroyWindow automatically destroys the associated child or owned windows when it destroys the parent or owner window. The function first destroys child or owned windows, and then it destroys the parent or owner window. DestroyWindow also destroys modeless dialog boxes created function CreateDialog.

Syntax

```
BOOL DestroyWindow  
(  
    HWND hWnd // handle to break the window  
);
```

Options

hWnd

window identifier, which will be destroyed.

Return values

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call GetLastError.

remarks

The stream can not use DestroyWindow, to destroy a window created by another thread. If the window being destroyed - a child window that has WS_EX_NOPARENTNOTIFY style, WM_PARENTNOTIFY message sent to the parent.

See also

CreateDialog, CreateWindow, CreateWindowEx, WM_DESTROY, WM_NCDESTROY, WM_PARENTNOTIFY

Accommodation and compatibility DestroyWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ENABLEWINDOW

EnableWindow function enables or disables mouse and keyboard input in a specific window or control. When the input is blocked, the window does not accept input such as mouse clicks and keystrokes. When the input is turned on, the window assumes all entered information.

Syntax

```
BOOL EnableWindow  
(  
    HWND hWnd, // handle of window  
    BOOL bEnable // check box to enable or disable input information  
);
```

Options

hWnd

window identifier, which will be enabled or disabled.

bEnable Specifies enabled or disabled window. If this option - TRUE (TRUE), the window is enabled. If the parameter - FALSE (FALSE), the window is blocked.

Return values

If the window was previously disabled, the return value is zero. If not previously been blocked by the window, the return value is zero. To get extended error information, call GetLastError.

remarks

If the ON state of the window is changed, a message is sent WM_ENABLE before returning function values EnableWindow. If the window is already locked, all its child windows are potentially blocked, although they have not sent a message WM_ENABLE. The window must be enabled before it can be activated. For example, if the application displays a non-modal dialog box and disconnect the main window, the application must include the main window before destroying the dialog box. Otherwise, another window will take the keyboard focus and be activated. If the child window is blocked, it is ignored when Windows tries to determine which window should accept mouse messages. By default, the window is enabled when it is created. To create a window that is initially disabled, the application can determine WS_DISABLED style in CreateWindow or CreateWindowEx function. Once the window is created, the application can use EnableWindow, to enable or disable the window. An application can use this function to enable or disable the control in the dialog box. Locked element of control can not take the keyboard focus, and the user can not access it.

See also

[CreateWindow](#), [CreateWindowEx](#), [IsWindowEnabled](#), [WM_ENABLE](#)

Accommodation and compatibility EnableWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ENDDEFERWINDOWPOS

EndDeferWindowPos function simultaneously updates the position and size in one or more windows in a single cycle refresh.

Syntax

```
BOOL EndDeferWindowPos  
(  
    HDWP hWinPosInfo // handle to the internal structure  
);
```

Options

hWinPosInfo

It identifies the location of the composite structure of the window that contains information about the size and location of one or more windows. This internal structure is returned BeginDeferWindowPos or the latest appeal to DeferWindowPos function.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value is zero. To get extended error information, call GetLastError.

remarks

EndDeferWindowPos function sends each window identified in the internal structure, posts and WM_WINDOWPOSCHANGING WM_WINDOWPOSCHANGED.

See also

BeginDeferWindowPos, DeferWindowPos, WM_WINDOWPOSCHANGED,
WM_WINDOWPOSCHANGING

Accommodation and compatibility EndDeferWindowPos

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ENUMCHILDPROC

EnumChildProc function - callback function defined by the application program that accepts child window handles as a result of recourse to EnumChildWindows function.

Syntax

```
BOOL CALLBACK EnumChildProc  
(  
    HWND hwnd, // handle to a child window  
    LPARAM lParam // value determined by the application  
);
```

Options

hwnd

It identifies the child window parent window specified in EnumChildWindows.

lParam

It sets the value determined by the application, given in EnumChildWindows.

Return Values.

To continue enumeration, the callback function should return TRUE (TRUE); to stop enumeration, it must return FALSE (FALSE).

remarks

The callback function can execute any desired task. An application must register this callback function by passing its address EnumChildWindows. EnumChildProc - mark - a placeholder for the function name defined by the application program.

See also

EnumChildWindows

Accommodation and compatibility EnumChildProc

Windows NT Yes

Win95 Yes

Yes Win32s

The imported user-defined library

Header file winuser.h

No Unicode

Notes WNDENUMPROC platform

FUNCTION ENUMTHREADWINDOWS

EnumThreadWindows function enumerates all child windows are not associated with a stream by passing the handle of each window, in turn, to a certain application program callback function. EnumThreadWindows valid as long as the most recent window, or until the callback function returns FALSE (FALSE). To enumerate child windows of a single window, use EnumChildWindows function. This function replaces EnumTaskWindows function.

Syntax

```
BOOL EnumThreadWindows  
(  
    DWORD dwThreadId, // thread identifier  
    WNDENUMPROC lpfn, // pointer to the callback function  
    LPARAM lParam // value determined by the application program  
);
```

Options

dwThreadId

Identifies the thread that the window should be listed.

lpfn

It indicates a specific application callback function. For more information on callback functions, see. EnumThreadWndProc callback function.

lParam

It specifies a 32-bit value defined by the application, which will be passed to the callback function.

Return values

If the function succeeds, the return value is nonzero. If the function fails, the return value - zero.

See also

EnumChildWindows, EnumThreadWndProc, EnumWindows

Accommodation and compatibility EnumThreadWindows

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ENUMWINDOWS

EnumWindows function enumerates all top-level windows on the screen by passing the handle of each window, in turn, to an application-defined callback function. EnumWindows valid as long as the last top-level window will not be transferred, or until the callback function returns FALSE (FALSE).

Syntax

```
BOOL EnumWindows  
(  
    WNDENUMPROC lpEnumFunc, // pointer to the callback function  
    LPARAM lParam // determined by the application  
)
```

Options

lpEnumFunc

Specifies an application-defined callback function. For more information, see. [EnumWindowsProc](#) callback function.

lParam

Sets the 32-bit, software-defined value to be passed to the callback function.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero.

remarks

EnumWindows function does not enumerate child windows. This feature is more reliable than in the function call GetWindow cycle. An application that calls GetWindow, to accomplish this task, it risks falling into an infinite loop or a reference to the handle of the window, which was destroyed.

See also

[EnumChildWindows](#), [EnumWindowsProc](#), [GetWindow](#)

Accommodation and compatibility [EnumWindows](#)

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ENUMWINDOWSPROC

EnumWindowsProc function - defined program callback function that takes a top-level window handles by calling EnumWindows or EnumDesktopWindows function.

Syntax

```
BOOL CALLBACK EnumWindowsProc  
(  
    HWND hwnd, // handle to parent window  
    LPARAM lParam // determined by the application  
);
```

Options

hwnd

Identifikator lParam top-level window

Sets the value of an application-defined or transmitted in EnumWindows EnumDesktopWindows.

Return values

To continue enumeration, the callback function should return TRUE (TRUE); to stop enumeration, it must return - FALSE (FALSE).

remarks

The callback function can execute any desired task. An application must register this callback function by passing address or EnumWindows EnumDesktopWindows. EnumWindowsProc - mark - a placeholder for the application-defined function name. WNDENUMPROC Type - function pointer EnumWindowsProc.

See also

EnumWindows, EnumDesktopWindows

Accommodation and compatibility EnumWindowsProc

Windows NT Yes

Win95 Yes

Yes Win32s

The imported library is determined by the user

Header file winuser.h

No Unicode

Notes WNDENUMPROC platform

FUNCTION FINDWINDOW

FindWindow function searches data descriptor top-level window whose class name and window name match the specified strings. This feature is not looking for child windows.

Syntax

```
HWND FindWindow  
(  
LPCTSTR lpClassName, // pointer to class name  
LPCTSTR lpWindowName // pointer to window name  
);
```

Options

lpClassName

It indicates a null-terminated string at the end, which defines the name of the class or - an atom that identifies the class-name string. If this option - an atom, it must be shared atom created by a previous call GlobalAddAtom function. Atom - a 16-bit value, which should be placed in the low lpClassName word; high word must be zero.

lpWindowName

It indicates a null-terminated string to the end that specifies the window name (the title bar). If this parameter - NULL (NULL), full compliance with the name of the window.

Return values

If the function succeeds, the return value - window handle that has defined the class name and window name. If the function fails, the return value - NULL (NULL). To get extended error information, call GetLastError.

See also

EnumWindows, FindWindowEx, GetClassName, GlobalAddAtom

Accommodation and compatibility FindWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

FUNCTION FINDWINDOWEX

FindWindowEx function retrieves information about a window handle, class name and window name match the specified strings. child windows search function starts from the first to the last of the specified child window.

Syntax

```
HWND FindWindowEx
(
    HWND hwndParent, // handle to parent window
    HWND hwndChildAfter, // handle to a child window
    LPCTSTR lpszClass, // pointer to class name
    LPCTSTR lpszWindow // pointer to window name
);
```

Options

hwndParent

Identifies the parent window whose child windows are to be found. If hwndParent - BLANK (NULL), the function uses the desktop window as the parent window. The function searches among windows that are child windows of the desktop.

hwndChildAfter

It identifies the child window. The search begins with the next child window in the Z - order. hwndChildAfter must be a direct subsidiary hwndParent window, not a mere child of the window. If hwndChildAfter - BLANK (NULL), the search begins with the first child window defined by parameter hwndParent. Please note that, if hwndParent and hwndChildAfter - BLANK (NULL), the function looks for all top-level windows.

lpszClass

It indicates a null-terminated string at the end, which defines the name of the class or - an atom that identifies the class-name string. If this option - an atom, it must be a common atom created by a previous call to GlobalAddAtom function. The atom, a 16-bit value, which should be placed in the younger part of the word - lpszClass; high word must be zero.

lpszWindow

It indicates a null-terminated string to the end that specifies the window name (the title bar). If this parameter is blank (NULL), the names of all the relevant boxes.

Return values

If the function succeeds, the return value - window handle that has a certain class and window names.
If the function fails, the return value - NULL (NULL). To get extended error information, call
GetLastError.

See also

EnumWindows, FindWindow, GetClassName, GlobalAddAtom

Accommodation and compatibility FindWindowEx

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETCLIENTRECT

GetClientRect function retrieves the coordinates of the workspace window. Work coordinates define the upper left and lower right corners of the working area. Since the work coordinates are defined relative to the upper-left corner of a window, coordinates of the upper left corner - (0,0).

Syntax

```
BOOL GetClientRect  
(  
    HWND hWnd, // handle of window  
    LPRECT lpRect // address of structure coordinates workers  
);
```

Options

hWnd

Identifies the window whose client coordinates to be returned.

lpRect

It points to a RECT structure that receives the coordinates of the workers. Left and upper elements - zero. The right and lower elements comprise the width and height of the window.

Return values

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call GetLastError.

See also

GetWindowRect, RECT

Accommodation and compatibility GetClientRect

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETDESKTOPWINDOW

GetDesktopWindow function returns the window handle of the Windows desktop. The desktop window covers the entire screen. Desktop window is an area on top of which are shown all the icons and other windows.

Syntax

HWND GetDesktopWindow (VOID)

Options

This function has no parameters.

Return values

Return value - handle desktop windows.

See also

GetWindow

Accommodation and compatibility GetDesktopWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETFOREGROUNDWINDOW

GetForegroundWindow function returns the handle to the foreground window (the window with which the user is currently working). The system assigns a little bit more high priority thread that creates the foreground window than the one that it gives other threads.

Syntax

HWND GetForegroundWindow (VOID)

Options

This function has no parameters.

Return values

Return value - handle the foreground window.

See also

[SetForegroundWindow](#)

[Accommodation and compatibility GetForegroundWindow](#)

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETLASTACTIVEPOPUP

GetLastActivePopup function determines what the latest pop-up window that belongs to the specified window was active.

Syntax

```
HWND GetLastActivePopup  
(  
    HWND hWnd // handle to owner window  
);
```

Options

hWnd

Owner ID box.

Return values

The return value identifies the most recently active, a pop-up window. The return value is the same as the hWnd parameter, if you do any of the following conditions:

The window identified by the hWnd, the most recent was active.

The window identified by the hWnd, does not have any pop-up windows.

The window identified by the hWnd, is not a top-level window, or it belongs to another window-tains.

See also

AnyPopup, ShowOwnedPopups

Accommodation and compatibility GetLastActivePopup

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETNEXTWINDOW

GetNextWindow function retrieves the data descriptor of the next or previous window in the Z - order. The next window - below the specified window; previous window above. If the specified window - a topmost window, the function retrieves the handle of the next (or pre-preceding) the top of the window. If the specified window - the top-level window, the function retrieves the handle of the next (or previous) top-level window. If the specified window - a child window, the function searches for a handle of the next (or previous) child window.

Syntax

```
HWND GetNextWindow  
(  
    HWND hWnd, // handle to the current window  
    UINT wCmd // direction flag  
)
```

Options

hWnd

It identifies the window. Found window handle that refers to this window, based on the value of the parameter wCmd.

wCmd

It determines whether the function returns a handle to the next screen or previous screen. This parameter can be any of the following values:

GW_HWNDNEXT - Returns the handle of the window below the window.

GW_HWNDPREV - Returns the window handle above the window.

Return values

If the function succeeds, the return value - the descriptor of the next (or previous) window. If there is no next (or previous) window, the return value - NULL (NULL). To get extended error information, call GetLastError.

Notes

The use of this function is the same as a challenge GetWindow function with a check mark or GW_HWNDNEXT GW_HWNDPREV.

See also

[GetTopWindow](#)

Accommodation and compatibility GetNextWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETPARENT

GetParent function is a handle to the parent window of the specified child window.

Syntax

```
HWND GetParent  
(  
    HWND hWnd // handle to a child window  
);
```

Options

hWnd

It identifies the window handle of the parent window which is to be found.

Return values

If the function succeeds, the return value - the parent window handle. If the window has no parent window, the return value - NULL (NULL). To get a more detailed error information, call GetLastError.

See also

SetParent

Accommodation and compatibility GetParent

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETTOPWINDOW

GetTopWindow function examines the Z - a sequence of child windows associated with a particular parent window and returns a handle to the child window back to the top of the Z - order.

Syntax

```
HWND GetTopWindow  
(  
    HWND hWnd // handle to parent window  
);
```

Options

hWnd Identifies the parent window whose child windows should be checked. If this parameter - NULL (NULL), the function returns a handle to the window from the top of the Z - order.

Return values

If the function succeeds, the return value - handle to the child window at the top of the Z - order. If the specified window has no child windows, the return value - NULL (NULL). Use the GetLastError function to get extended error information.

See also

GetNextWindow, GetWindow

Accommodation and compatibility GetTopWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETWINDOW

GetWindow function retrieves the window handle that has a certain relation (Z - or the owner of the sequence) to the target window.

Syntax

HWND GetWindow

(

 HWND hWnd, // handle to the original window

 UINT uCmd // check relations

);

Options

hWnd

It identifies the window. Retrieves the window handle relating to this window, based on the value of the parameter uCmd.

uCmd

It specifies the ratio between the defined window and a window whose handle is to be found. This parameter can be one of the following values:

- GW_CHILD - Found handle identifies the child window at the top of the Z - order, if the specified window - the parent window; somehow found a descriptor value is empty (NULL). Funktsiya checks only child windows of the specified window. It does not check the box - descendants.
- GW_HWNDFIRST - Found handle identifies a window of the same type, which is the highest in the Z - order. If the specified window - a topmost window, the handle identifies the topmost window, which is the highest in the Z - order. If the specified window - top-level window, the handle identifies the top-level window that is highest in the Z - order. If a particular window - a child window, the handle identifies the nursing window that is highest in the Z - order.
- GW_HWNDLAST - Found handle identifies a window of the same type, which is the lowest in the Z - order. If the specified window - a topmost window, the handle identifies the topmost window that is lowest in the Z - order. If the specified window - top-level window, the handle identifies the top-level window that is lowest in the Z - order. If the specified window - a child window, the handle identifies the nursing window that is lowest in the Z - order.
- GW_HWNDFNEXT - Found handle identifies the window below the specified window in the Z - order. If the specified window - a topmost window, the handle identifies the topmost window below the specified window. If the specified window - top-level window, the handle identifies

the top-level window below the specified window. If the specified window - a child window, the handle identifies the nursing window below the specified window.

- GW_HWNDNEXT - Found handle identifies the window below the specified window in the Z - order. If the specified window - a topmost window, the handle identifies the topmost window below the specified window. If the specified window - top-level window, the handle identifies the top-level window below the specified window. If the specified window - a child window, the handle identifies the nursing window below the specified window.
- GW_OWNER - Found descriptor identifies the owner window of the specified window, if any.

Return values

If the function succeeds, the return value - a window handle. If a given ratio of window to the specified window does not exist, the return value - NULL (NULL). To get extended error information, call GetLastError.

See also

GetActiveWindow, GetNextWindow, GetTopWindow

Accommodation and compatibility GetWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETWINDOWPLACEMENT

Function GetWindowPlacement looking for information on installing the appearance and recovery position, folding and unfolding of the specified window.

Syntax

```
BOOL GetWindowPlacement  
(  
    HWND hWnd, // handle of window  
    WINDOWPLACEMENT * lpwndpl // address of structure for location data  
);
```

Options

hWnd

It identifies the window.

lpwndpl

Indicates WINDOWPLACEMENT structure that receives information about the position and status of the species. Before calling GetWindowPlacement, set the length of the element (length) WINDOWPLACEMENT structures as sizeof (WINDOWPLACEMENT). GetWindowPlacement fail if lpwndpl-> length (length) is not installed correctly.

Return values

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero. To get extended error information, call GetLastError.

remarks

Element flags (flags) WINDOWPLACEMENT, found this function - is always zero. If the window identified by the hWnd parameter, deployed, showCmd element - SW_SHOWMAXIMIZED. If the window is minimized (minimized), showCmd - SW_SHOWMINIMIZED. Otherwise, he - SW_SHOWNORMAL. Element length (length) WINDOWPLACEMENT must be set to sizeof (WINDOWPLACEMENT). If this element is not installed correctly, the function returns FALSE (FALSE).

See also

SetWindowPlacement, WINDOWPLACEMENT

Accommodation and compatibility GetWindowPlacement

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETWINDOWRECT

GetWindowRect function retrieves the dimensions of the frame bounding rectangle of the specified window. Dimensions are given in screen coordinates, which are arranged relative to the upper left corner of the screen.

Syntax

```
BOOL GetWindowRect  
(  
    HWND hWnd, // handle of window  
    LPRECT lpRect // address of structure for the coordinates of the window  
);
```

Options

hWnd

It identifies the window.

lpRect

Points to a RECT structure that receives the screen coordinates of the upper left and lower right corners of the window.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero. For more information about the error call GetLastError.

See also

GetClientRect, RECT

Accommodation and compatibility GetWindowRect

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION GETWINDOWTEXT

GetWindowText function copies of the title bar text of the specified window (if it has it) to the clipboard. If the specified window - control, the text of the control is copied.

Syntax

```
int GetWindowText  
(  
    HWND hWnd, // handle to the window or control with text  
    LPTSTR lpString, // address of buffer for text  
    int nMaxCount // maximum number of characters to copy  
);
```

Options

hWnd

Identifies the window or control containing the text

lpString

It points to the buffer that will accept text.

nMaxCount

Sets the maximum number of characters to be copied to the clipboard. If the text exceeds this limit, it is truncated.

Return values

If the function succeeds, the return value - the length, in characters, of the copied string, not including the end of line character (zero-terminator). If no window title bar or text if the title bar is empty or if a window handle or control are unacceptable, return value is zero. To get extended error information, call GetLastError. This function can not find the item text for editing in another application.

remarks

This feature makes WM_GETTEXT send a message to the specified window or control. This function can not return the item text to edit in other applications.

See also

GetWindowTextLength, SetWindowText, WM_GETTEXT

Accommodation and compatibility GetWindowText

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION GETWINDOWTEXTLENGTH

GetWindowTextLength function returns back length, in characters, the title bar text of the specified window (if the window has a title bar). If the specified window - control, the function returns the length of the back of the text inside the control.

Syntax

```
int GetWindowTextLength  
(  
    HWND hWnd // handle of window or control panel with text  
);
```

Options

hWnd

Identifies the window or control.

Return values

If the function succeeds, the return value - the length, in characters, of the text. Under certain conditions, this value may actually be greater than the length of the text. For more information, see. The following Remarks section. If the window has no text, the return value is zero. To get extended error information, call GetLastError.

remarks

This feature forces WM_GETTEXTLENGTH send a message to the specified window or control. Under certain conditions, GetWindowTextLength function can return a value that is greater than the actual length of the text. This occurs with certain mixtures of ANSI and Unicode, and with an operating system, which allows for the possible existence of DBCS characters within the text. The return value, however, will always be at least as large as the actual length of the text; You may thus, always use this to guide the buffer allocation. This behavior can occur when an application uses both ANSI functions and common dialogs, which use Unicode. It can also occur when an application uses a variety of ANSI GetWindowTextLength with a window whose window procedure - Unicode, or Unicode kind with a window whose window procedure - ANSI. To obtain the exact length of the text, use message WM_GETTEXT, LB_GETTEXT, or CB_GETLBTEXT, or GetWindowText function.

See also

CB_GETLBTEXT, GetWindowText, LB_GETTEXT, SetWindowText, WM_GETTEXT,
WM_GETTEXTLENGTH

Accommodation and compatibility GetWindowTextLength

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION GETWINDOWTHREADPROCESSID

GetWindowThreadProcessId function returns back flow identifier created specified window and, optionally, a process identifier, which created window. This function replaces the function GetWindowTask.

Syntax

```
DWORD GetWindowThreadProcessId  
(  
    HWND hWnd, // handle of window  
    LPDWORD lpdwProcessId // address of the variable to the process ID  
);
```

Options

hWnd

It identifies the window.

lpdwProcessId

It indicates a 32-bit value, which takes a process ID. If this option - not NULL (NULL), GetWindowThreadProcessId copies the process ID in a 32-bit value; otherwise, it does not.

Return values

Return Value - stream ID that creates the window.

remarks

This function replaces the function GetWindowTask version 3.x Windows.

Accommodation and compatibility GetWindowThreadProcessId

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ISCHILD

IsChild function checks whether the window is a child window or a window generated by the determined parent window. The child window - determined by a direct descendant of the parent window if the parent window is in the chain of parent windows; chain of parent windows controls in a range from the original overlapped or pop-up window to the child window.

Syntax

```
BOOL IsChild  
(  
    HWND hWndParent, // handle to parent window  
    HWND hWnd // handle to the window to check  
)
```

Options

hWndParent

Identifies the parent window.

hWnd

It identifies the window that will be checked.

Return values

If the window - a subsidiary of, or generated by the specified window of the parent window, the return value is nonzero. If the window - not a subsidiary or return a null value generated by the specified window of the parent window.

See also

IsWindow, SetParent

Accommodation and compatibility IsChild

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ISICONIC

IsIconic function determines if not minimized (minimized) specified window (to the type icons).

Syntax

BOOL IsIconic

(

 HWND hWnd // handle of window

);

Options

hWnd

It identifies the window.

Return values

If the window as an icon, the return value is nonzero. If the window is not an icon, the return value - zero.

See also

IsZoomed

Accommodation and compatibility IsIconic

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ISWINDOW

IsWindow function determines whether the specified window handle identifies an existing window.

Syntax

```
BOOL IsWindow  
(  
    HWND hWnd // handle of window  
);
```

Options

hWnd

Sets the handle to the window.

Return values

If the window handle identifies an existing window, the return value is nonzero. If the window handle does not identify an existing window, the return value is zero.

See also

IsWindowEnabled, IsWindowVisible

Accommodation and compatibility IsWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ISWINDOWUNICODE

IsWindowUnicode function determines whether the specified window is a native Unicode window.

Syntax

```
BOOL IsWindowUnicode  
(  
    HWND hWnd // handle to the window  
);
```

Parameters hWnd

It identifies the window.

Return values

If the window - a native Unicode window, the return value is nonzero. If the window - not a native Unicode window, the return value is zero.

remarks

The system automatically makes the two-way translation (Unicode to ANSI-ASCII) for window messages. For example, if the ANSI-ASCII window message sent to the window with the UNICODE, the system converts the message to Unicode message before calling the window procedure. The system calls IsWindowUnicode to determine whether it is necessary to convert the message. When this function returns FALSE (FALSE), the window - native ANSI-ASCII window.

Accommodation and compatibility IsWindowUnicode

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library Specifies the user

Header file winuser.h

No Unicode

Notes WNDENUMPROC platform

FUNCTION ISWINDOWVISIBLE

IsWindowVisible function finds the data on the state of visibility of the specified window.

Syntax

```
BOOL IsWindowVisible  
(  
    HWND hWnd // handle of window  
);
```

Options

hWnd

It identifies the window.

Return values

If the specified window and its parent window have WS_VISIBLE style, the return value is nonzero. If the specified window and its parent window have WS_VISIBLE style, the return value is zero. Since the return value determines whether the window has WS_VISIBLE style, it can be different from zero even if the window is completely obscured by other windows.

remarks

Status window visibility is indicated by bit WS_VISIBLE style. When WS_VISIBLE style set, the window visible and the subsequent drawing up displays within it until the window has WS_VISIBLE style. Any drawing windows WS_VISIBLE style will not be displayed if the window is obscured by other windows or clipped by its parent window.

See also

ShowWindow

Accommodation and compatibility IsWindowVisible

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ISZOOMED

IsZoomed function determines whether the window is maximized.

Syntax

```
BOOL IsZoomed  
(  
    HWND hWnd // handle of window  
);
```

Options

hWnd

It identifies the window.

Return values

If the window is changed in scale, the return value is nonzero. If the window is not changed in scale, the return value is zero.

See also

IsIconic

Accommodation and compatibility IsZoomed

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION MOVEWINDOW

MoveWindow function changes the position and dimensions of the specified window. To the top of the window, his level, the position and size - relative to the upper-left corner of the screen. For a child window, they are - relative to the left upper workspace corner of the parent window.

Syntax

```
BOOL MoveWindow  
(  
    HWND hWnd, // handle of window  
    int X, // horizontal position  
    int Y, // vertical position of the  
    int nWidth, // width  
    int nHeight, // height  
    BOOL bRepaint // check repainting  
);
```

Options

hWnd

It identifies the window.

X

Sets the new position of the left side of the window.

Y

Sets the new position of the top of the window.

nWidth

Specifies the new width of the window.

nHeight

Specifies the new height of the window.

bRepaint

Specifies whether the window should be repainted. If this option - TRUE (TRUE), the window receives a WM_PAINT. If the parameter - FALSE (FALSE), no repainting of any sort happens. This applies to the work area, overscan (including the title bar and scroll bars), and any part of the parent window uncovered as a result of movement of the child window. If this option - FALSE (FALSE), the

application must explicitly cancel or redraw any parts of the window and parent window that need redrawing.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero.

remarks

If bRepaint option - TRUE (TRUE), Windows sends a WM_PAINT message to the window procedure immediately after moving a window (ie MoveWindow function calls UpdateWindow function). If bRepaint - FALSE (FALSE), Windows puts WM_PAINT message in the message queue associated with a window. Posts loop sends WM_PAINT message only after dispatching all other messages in the queue. MoveWindow function sends a message box WM_WINDOWPOSCHANGING, WM_WINDOWPOSCHANGED, WM_MOVE, WM_SIZE and WM_NCCALCSIZE.

See also

SetWindowPos, UpdateWindow, WM_GETMINMAXINFO, WM_PAINT

Accommodation and compatibility MoveWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION OPENICON

OpenIcon function restores a minimized (ikonizirovannoe) window to its previous size and position; then it activates it.

Syntax

```
BOOL OpenIcon  
(  
    HWND hWnd // handle of window  
);
```

Options

hWnd

It identifies the window that will be restored and activated.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero. To get extended error information, call GetLastError.

remarks

OpenIcon function sends this message window WM_QUERYOPEN.

See also

CloseWindow, IsIconic, ShowWindow.

Accommodation and compatibility OpenIcon

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION SETFOREGROUNDWINDOW

SetForegroundWindow function converts the stream that created the specified window into the foreground and activates the window. Keyboard input is directed through the window, and various visual cues are changed for the user.

Syntax

```
BOOL SetForegroundWindow  
(  
    HWND hWnd // handle to the window, which is translated in the foreground  
);
```

Options

hWnd

It identifies the window to be activated and transferred to the foreground.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero. To get extended error information, call GetLastError.

remarks

Priority window - a window at the top of Z-sequence. This is - the window with which the user is working. Among the priority multitasking, you should generally allow the user to control which window is the foreground window. However, the application can call the SetForegroundWindow, if you want to translate itself into active mode, to display critical errors or information that requires immediate attention of the user. A good example - a debugger, when it detects a breakpoint stop the program. The system assigns a slightly higher priority to the thread that created the foreground window than it does so in relation to other threads.

See also

[GetForegroundWindow](#)

[Accommodation and compatibility SetForegroundWindow](#)

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION SETPARENT

SetParent function replaces the parent window of the specified child window.

Syntax

HWND SetParent

```
(  
    HWND hWndChild, // Handle to the window whose parent is changed  
    HWND hWndNewParent // Handle to the new parent window  
)
```

Options

hWndChild

It identifies the child window.

hWndNewParent

Identifies the new parent window. If this parameter - NULL (NULL), the desktop window becomes the new parent window.

Return values

If the function succeeds, the return value - the previous parent window handle. If the function fails, the return value - NULL (NULL). To get extended error information, call GetLastError.

remarks

An application can use the SetParent function to set the parent window of a pop-up, overlapped, or child window. The new parent window and the child window must belong to the same application. If the window identified parameter hWndChild visible, Windows performs the appropriate change and redraw.

See also

GetParent

Accommodation and compatibility SetParent

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION SETWINDOWLONG

The SetWindowLong function changes the attributes of the specified window. The function also sets a 32-bit (long) value at the specified offset into the extra space on the memory window.

Syntax

```
LONG SetWindowLong  
(  
    HWND hWnd, // handle of window  
    int nIndex, // offset value, which is set  
    LONG dwNewLong // new value  
)
```

Options

hWnd

Identifies the window and, indirectly, the class to which the window belongs.

nIndex

Defines the value of the offset, measured from the ground to be found. Valid values are in the range from zero to the number of additional bytes of space in memory, minus 4; for example, if you set 12 or more bytes of additional memory space, the value 8 would index to a third 32-bit integer. To set any other value, specify one of the following:

GWL_EXSTYLE - Sets a new extended window style.

GWL_STYLE - Sets a new window style.

GWL_WNDPROC - Sets a new address for the window procedure.

GWL_HINSTANCE - Sets a new application instance handle.

GWL_ID - Sets a new identifier of the window.

GWL_USERDATA - Sets the 32-bit value associated with the window. Each window has a corresponding 32-bit value intended for use by the application that created the window.

The following values are also available when the hWnd parameter identifies a dialog box:

DWL_DLGPROC - Sets the new address of the dialog box procedure.

DWL_MSGRESULT - Sets the return value of a message processed in the dialog box procedure.

DWL_USER - Sets new extra information that is private to the application, such as handles or pointers.

dwNewLong

Sets the reduced value.

Return values

If the function succeeds, the return value - the previous value of the specified 32-bit signed integer. If the function fails, the return value is zero. To get extended error information, call GetLastError. If the previous value of the specified 32-bit integer is zero, and the function succeeds, the return value is zero, but the function does not clear the last error information. It is difficult to determine success or failure. To combat this, you must clear the last error information by calling SetLastError (0) before calling SetWindowLong. Then, function failure will be indicated by a return value of zero and the GetLastError result, which is different from zero.

remarks

SetWindowLong function fails if the window specified by the hWnd parameter does not belong to the same process as the calling thread. If you use the SetWindowLong function and GWL_WNDPROC index to replace the window procedure, the window procedure must conform to the guidelines specified in the description of the callback function WindowProc. Calling SetWindowLong with GWL_WNDPROC index creates a subclass of the window class that is used to create a window. The application must not be a subclass of a window created by another process. SetWindowLong function creates the window subclass, replace the window procedure associated with a single window, forcing Windows to call the new window procedure instead of the previous one. The application must pass any messages not processed by the new window procedure to the previous window procedure by calling CallWindowProc. This allows the application to create a chain of window procedures. Backing extra space in the memory, it sets a nonzero value in the element cbWndExtra WNDCLASS structure used with the function RegisterClass. You must not call SetWindowLong with the index GWL_HWNDPARENT, to replace the parent of a child window. Instead, use the SetParent function.

See also

CallWindowProc, GetWindowLong, GetWindowWord, RegisterClass, SetParent, SetWindowWord, WindowProc, WNDCLASS.

Accommodation and compatibility SetWindowLong

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION SETWINDOWPLACEMENT

SetWindowPlacement function sets the show state and restore, expand and collapse the position of the specified window.

Syntax

```
BOOL SetWindowPlacement
```

```
(
```

```
    HWND hWnd, // handle of window
```

```
    CONST WINDOWPLACEMENT * lpwndpl // address of structure with information about the  
    // position
```

```
);
```

Options

hWnd

It identifies the window.

lpwndpl

Indicates WINDOWPLACEMENT structure that specifies the new show state and position of the window. Before calling SetWindowPlacement, set the length of the element (length) WINDOWPLACEMENT structure in the value of sizeof (WINDOWPLACEMENT).

SetWindowPlacement fails, if lpwndpl-> length (length) is not installed correctly.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero. To get extended error information, call GetLastError.

remarks

Element length (length) WINDOWPLACEMENT must be set to sizeof (WINDOWPLACEMENT). If this element is not installed correctly, the function returns FALSE (FALSE).

See also

GetWindowPlacement, WINDOWPLACEMENT

Accommodation and compatibility SetWindowPlacement

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION SETWINDOWPOS

SetWindowPos function changes the size of the position and Z-sequence child, pop-up or top-level window. Child, pop-up and top-level windows are placed in order according to their appearance on the screen. The topmost window receives the highest rank and becomes the first window in the Z-order.

Syntax

```
BOOL SetWindowPos  
(  
    HWND hWnd, // handle of window  
    HWND hWndInsertAfter, // handle the order placement  
    int X, // horizontal position  
    int Y, // vertical position of the  
    int cx, // width  
    int cy, // height  
    UINT uFlags // window positioning flags  
)
```

Options

hWnd

It identifies the window.

hWndInsertAfter

It identifies the window that is preceded by a window installed in Z-sequence. This parameter must be a window handle or one of the following:

HWND_BOTTOM - Places the window at the bottom of the Z-order. If the hWnd parameter identifies a topmost window, the window loses its topmost status and is placed at the bottom of all other windows.

HWND_NOTOPMOST - Places the window in front of all not in the topmost windows (that is, behind all topmost windows). This flag has no effect if the window - is not the topmost window.

HWND_TOP - Places the window at the top of the Z-order.

HWND_TOPMOST - Places the window in front of non-topmost windows. The window maintains its topmost position even when it is inactive.

For more information about how to use this option, see. Following Remarks section.

X

Establishes a new position on the left side of the window.

Y

Sets the new position of the top of the window.

CX

Specifies the new width of the window, in pixels.

cy

Specifies the new height of the window, in pixels.

uFlags

Specifies the flags that establish the size and positioning of windows. This parameter can be a combination of the following values:

- SWP_DRAWFRAME - Prints frame (defined in the description of the class of the window) around the window.
- SWP_FRAMECHANGED - Sends a message WM_NCCALCSIZE window, even when the window size does not change. If this option is not specified, WM_NCCALCSIZE sent only when the window is resized.
- SWP_HIDEWINDOW - Hides the window.
- SWP_NOACTIVATE - Do not activate the window. If this is not checked, the window is activated and moved to the top of either the upper or topmost group (depending on the setting hWndInsertAfter parameter).
- SWP_NOCOPYBITS - Resets all the contents of the working area. If this is not checked, the valid contents of the work area are saved and copied back into the client area after the window is sized or repositioned.
- SWP_NOMOVE - Retains current position (ignores the X and Y parameters).
- SWP_NOOWNERZORDER - Does not change the owner window's position in the Z-order.
- SWP_NOREDRAW - not redraw changes. If this option is selected, it is not going on no repainting of any kind. This applies to the work area, overscan (including the title bar and scroll bars), and any part of the parent window uncovered as a result of movement of the window. When this option is selected, the application must explicitly invalidate or redraw any parts of the window and parent window that need redrawing.
- SWP_NOREPOSITION - The same as the box SWP_NOOWNERZORDER.
- SWP_NOSENDCHANGING - Prevents the window from receiving WM_WINDOWPOSCHANGING posts.

- SWP_NOSIZE - Retains the current size (ignores the cx and cy parameters).
- SWP_NOZORDER - Retains the current Z-order (ignores hWndInsertAfter option).
- SWP_SHOWWINDOW - Displays the window.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero. To get extended error information, call GetLastError.

remarks

If the boxes SWP_SHOWWINDOW or SWP_HIDEWINDOW, the window can not be moved or resized. All coordinates for child windows - working position (relative to the working area of the parent window's upper-left corner). The window can be done at the top of the window, or by setting the parameter in hWndInsertAfter HWND_TOPMOST to guarantee that SWP_NOZORDER flag is not set, or by setting the position of the window in the Z-order, so that it is above any existing topmost window. When a non-topmost window is made topmost, its owned windows are also made topmost. Its owners, however, do not change. If no box SWP_NOACTIVATE, or check SWP_NOZORDER not set (that is, when the application program required that the window was at the same time, and activated, and that its position has changed in the Z-order), the value specified in the hWndInsertAfter, used only in the following circumstances:

No HWND_TOPMOST HWND_NOTOPMOST box or box is not checked in hWndInsertAfter.

The window identified by hWnd - is not the active window.

The application program can not activate an inactive window without also bringing it to the top of Z-order. Applications can change the position of a window activated in the sequence Z-without limitation, or they can activate a window and then move it to the top of the uppermost or uppermost windows. If the topmost window at the bottom of reset (HWND_BOTTOM) Z-or after any sequence, it is no longer uppermost not the topmost window. When a topmost window is made not at the top, its owners and its owned windows are also made non-topmost windows. Not topmost window may own a topmost window, but on the contrary can not occur. Any window owned by a topmost window (such as a dialog box) is itself made a topmost window to ensure that all the windows are in the possession of its owner above. If the application is not in an active mode, but must be in active mode, it must call the function SetForegroundWindow.

See also

[MoveWindow](#), [SetActiveWindow](#), [SetForegroundWindow](#)

[Accommodation and compatibility](#) [SetWindowPos](#)

[Windows NT](#) Yes

[Win95](#) Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION SETWINDOWTEXT

SetWindowText function changes the title bar text of the specified window (if any). If the specified window - control, the text is changed.

Syntax

```
BOOL SetWindowText  
(  
    HWND hWnd, // handle to the window or control  
    LPCTSTR lpString // address lines  
);
```

Options

hWnd

Identifies the window or control whose text is to be changed.

lpString

It indicates a null-terminated string to the end, to be used as a new title or control text.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero. To get extended error information, call GetLastError.

remarks

SetWindowText WM_SETTEXT function causes a message to be sent to the specified window or control. If the window - the window control with a list created in WS_CAPTION style is not looking at it, SetWindowText sets the text for the control, and not to enter in a list box. SetWindowText function does not increase the size of tabs (ASCII 0x09) code. Tab characters are displayed as the pipe character (|).

See also

[GetWindowText, WM_SETTEXT](#)

[Accommodation and compatibility SetWindowText](#)

[Windows NT Yes](#)

[Win95 Yes](#)

[Yes Win32s](#)

[Imported Library user32.lib](#)

Header file winuser.h

Unicode WinNT

Platform Notes None

FUNCTION SHOWOWNEDPOPUPS

ShowOwnedPopups function shows or hides all pop-up windows that belong to the specified window.

Syntax

```
BOOL ShowOwnedPopups  
(  
    HWND hWnd, // handle of window  
    BOOL fShow // window display box  
);
```

Options

hWnd

Identifies the window that owns the pop-up window that will be shown or hidden.

fShow

Specifies whether the pop-up window to be displayed or hidden. If this option - TRUE (TRUE), all the hidden pop-up window displays. If this option - FALSE (FALSE), all visible pop-up windows are hidden.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero. To get extended error information, call GetLastError.

remarks

ShowOwnedPopups shows only the windows that were hidden by a previous call to ShowOwnedPopups function. For example, if a pop-up window has been hidden by using the ShowWindow function, later calling the function ShowOwnedPopups installing it fShow parameter to TRUE (TRUE) will force the window to be displayed.

See also

IsWindowVisible, ShowWindow

Accommodation and compatibility ShowOwnedPopups

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION SHOWWINDOW

ShowWindow function sets the show state of the specified window.

Syntax

BOOL ShowWindow

(

 HWND hWnd, // handle of window
 int nCmdShow // show state of the window
);

Parameters hWnd

It identifies the window.

nCmdShow

It specifies how the window should be shown. This option is initially ignored when the application calls the ShowWindow, if the program that launched the application, provides a framework STARTUPINFO. Otherwise, the first call to ShowWindow function, this value should be the value obtained by the WinMain function in its parameter nCmdShow. In subsequent calls, this parameter can be one of the following values:

- SW_HIDE - Hides the window and activates another window.
- SW_MAXIMIZE - Maximizes the specified window.
- SW_MINIMIZE - Minimizes the specified window and activates the next top-level window in the Z-order.
- SW_RESTORE - Activates and displays a window. If the window is minimized or maximized, Windows restores to its original size and position. An application should specify this flag when restoring a minimized window.
- SW_SHOW - Activates the window and displays its current size and position.
- SW_SHOWDEFAULT - Sets the show state based on the flag SW_, in particular STARTUPINFO structure passed in the CreateProcess function program that launched the application.
- SW_SHOWMAXIMIZED - Activates the window and displays it as a maximized window.
- SW_SHOWMINIMIZED - Activates the window and displays it as a minimized window.
- SW_SHOWMINNOACTIVE - Displays the window as a minimized window. The active window remains active.

- SW_SHOWNA - Displays a window in its current state. The active window remains active.
- SW_SHOWNOACTIVATE - Displays a window in its most modern size and position. The active window remains active.
- SW_SHOWNORMAL - Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when displaying the window for the first time.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero.

remarks

The first time the program calls the ShowWindow, it uses the parameter nCmdShow WinMain function as its parameter nCmdShow. Subsequent calls to ShowWindow must use one of the values in this list, instead of the specified function parameter WinMain nCmdShow. As noted in the discussion of parameter nCmdShow, nCmdShow value is ignored when the first call to the ShowWindow, if the program that launched the application, determines the startup information in STARTUPINFO structure. In this case, ShowWindow uses the information in a predetermined structure STARTUPINFO to display a window. On subsequent calls, the application should call ShowWindow with the installation nCmdShow SW_SHOWDEFAULT in, to use the startup information provided by the program that launched the application. For example, Program Manager program provides that applications start with a minimized main window. This behavior is designed for the following situations:

Applications create their main window by calling CreateWindow with the installation WS_VISIBLE flag.

Applications create their main window by calling CreateWindow with the WS_VISIBLE dumped flag, and at a later address to the ShowWindow with the installation Check this item ka-SW_SHOW, to make it visible.

See also

CreateProcess, CreateWindow, ShowOwnedPopups, STARTUPINFO, WinMain

Accommodation and compatibility ShowWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION SHOWWINDOWASYNC

ShowWindowAsync function sets the show state of a window created by different threads.

Syntax

```
BOOL ShowWindowAsync  
(  
    HWND hWnd, // handle of window  
    int nCmdShow // show state of the window  
);
```

Options

hWnd

It identifies the window.

nCmdShow

It specifies how the window should be shown. For a list of possible values, see. ShowWindow function description.

Return values

If the function succeeds, it returns a nonzero value. If the function fails, the return value - zero.

remarks

This function notifies the event display window message queue of this window. An application can use this function to avoid hang-up state when waiting for an unresponsive application to complete the process window display event.

See also

ShowWindow

Accommodation and compatibility ShowWindowAsync

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION TILEWINDOWS

TileWindows feature a "mosaic" defined window or child windows of a given parent window.

Syntax

WORD WINAPI TileWindows

(

```
    HWND hwndParent, // handle to parent window  
    UINT wHow, // window types that are not ordered  
    CONST RECT * lpRect, // rectangle ordered where the window  
    UINT cKids, // number of windows orderability  
    const HWND FAR * lpKids // array of window descriptors
```

);

Options

hwndParent

Identifies the parent window. If this parameter - NULL (NULL), assumed the desktop window.

wHow Defines the types of windows that are not ordered, or left to right, or horizontally. This parameter can be one of the following values, combined with zero or more values listed in CascadeWindows functions:

MDITILE_HORIZONTAL - Features a "mosaic" of the window horizontally.

MDITILE_VERTICAL - Features a "mosaic" of the window vertically.

lpRect

Indicates SMALL_RECT structure that specifies the rectangular area, in screen coordinates, which are placed inside the box. If this parameter - NULL (NULL), used work area of the parent window.

cKids

Sets the number of elements in the array, a given parameter lpKids. This parameter is ignored if lpKids - BLANK (NULL).

lpKids

Points to an array of window handles identifying windows that are ordered. If this parameter - NULL (NULL), placed the child windows specified parent window (or the desktop window).

Return values

If the function succeeds, the return value - the number of hosted windows. If the function fails, the

return value is zero.

See also

CascadeWindows, SMALL_RECT

Accommodation and compatibility TileWindows

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION WINDOWFROMPOINT

WindowFromPoint function retrieves the handle of the window that contains the specified point.

Syntax

```
HWND WindowFromPoint  
(  
POINT Point // structure with a mark  
);
```

Options

Point

Specifies a POINT structure that defines the point which was marked.

Return values

If the function succeeds, the return value - handle to the window that contains the mark. If any window does not exist in this mark, the return value - NULL (NULL).

remarks

WindowFromPoint not function retrieves a handle to hide or lock the window, even if the item - within the window. An application for an unlimited search is to use ChildWindowFromPoint function.

See also

ChildWindowFromPoint, POINT, WindowFromDC

Accommodation and compatibility WindowFromPoint

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION WINMAIN

WinMain function is called by the system as the initial entry point for the Win32-based, application.

Syntax:

```
int WINAPI WinMain
(
    HINSTANCE hInstance, // handle to the current instance of the window
    HINSTANCE hPrevInstance, // handle of the previous instance of the window
    LPSTR lpCmdLine, // pointer to command line
    int nCmdShow // shows the state of the window
);
```

Options

hInstance

Identifies the current sample application.

hPrevInstance

Identifies the previous sample application. For a Win32-based application, this parameter is always set to NULL (NULL). If you need to find out whether there is another sample of an existing program, create a named mutex-object using CreateMutex function. If GetLastError returns ERROR_ALREADY_EXISTS message, another instance of your application exists (it is created mutex-object).

lpCmdLine

It indicates a null-terminated string to the end of determining the command line for the application.

nCmdShow

It specifies how the window should be shown. This parameter can be one of the following values:

- SW_HIDE - Hides the window and activates another window.
- SW_MINIMIZE - Minimizes the specified window and activates the top-level window in the system list.
- SW_RESTORE - Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_SHOWNORMAL).
- SW_SHOW - Activates and displays a window on the screen in its current size and position.
- SW_SHOWMAXIMIZED - Activates the window and displays it as a maximized window.

- SW_SHOWMINIMIZED - Activates the window and displays it as an icon.
- SW_SHOWMINNOACTIVE - Displays a window as an icon. The active window remains active.
- SW_SHOWNA - Displays a window in its current state. The active window remains active.
- SW_SHOWNOACTIVATE - Displays a window in its most recent size and position. The active window remains active.
- SW_SHOWNORMAL - Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).

Return values

If the function has reached its goal, it ends when will WM_QUIT message, it should return the output value in wParam parameter of this message. If the function before entering the message loop, it should return 0.

remarks

WinMain initializes the application program displays its main window on the screen, and then enters the message loop "Search and Send to destination (dispatching)", which is the top-level management structure for the implementation of the other elements of the application. Message cycle is completed when the message is received WM_QUIT. At this point, the WinMain exits the application, returning the value passed to the parameter wParam WM_QUIT posts. If WM_QUIT was obtained from a call to PostQuitMessage, wParam value - value nExitCode PostQuitMessage function. For more information, see. Creating a message loop (Creating a Message Loop).

See also

CreateMutex, DispatchMessage, GetMessage, PostQuitMessage, TranslateMessage

Accommodation and compatibility WinMain

Windows NT Yes

Win95 Yes

Yes Win32s

imported library

Header file winbase.h

No Unicode

Platform Notes None

FUNCTION ANYPOPUP

AnyPopup function indicates whether there is an owned, visible, top-level pop-up, or an overlay window on the screen. Function searches across the Windows screen, and not only in the work area caused by the application program.

Syntax

BOOL AnyPopup (VOID)

Options

This function has no parameters

Return values

If there is a pop-up window, the return value is non-zero, even if a pop-up window is completely blocked by other windows. If the pop-up window does not exist, the return value - zero.

remarks

AnyPopup - feature version of Windows 1.x and retained for compatibility purposes. In general, it is useless. This function does not detect that are not owned by the pop-up windows or window, in which no bit set WS_VISIBLE style.

See also

GetLastActivePopup, ShowOwnedPopups

Accommodation and compatibility AnyPopup

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION ENUMTASKWINDOWS

EnumTaskWindows function obsolete. It was replaced by a function EnumThreadWindows. To maintain backward compatibility for 16-bit applications, EnumTaskWindows function has been replaced by a macro that causes EnumThreadWindows. Older applications can continue to cause EnumTaskWindows, as before it was documented, but new applications should use EnumThreadWindows.

FUNCTION GETSYSMODALWINDOW

Function GetSysModalWindow old. This feature is only available for compatibility with 16-bit versions of Windows.

FUNCTION GETWINDOWTASK

Function GetWindowTask old. This feature is only available for compatibility with 16-bit versions of Windows. Based on Win32 applications should use GetWindowThreadProcessId function.

FUNCTION SETSYSMODALWINDOW

Function SetSysModalWindow old. This function is only used for compatibility with 16-bit versions of Windows. New data-entry model does not take into account the window System Modal.

FUNCTION CHILDWINDOWFROMPOINT

ChildWindowFromPoint function determines which, if any, of the child windows belonging to the parent window contains the installation point (fixed).

Syntax

```
HWND ChildWindowFromPoint  
(  
    HWND hWndParent, // handle to parent window  
    POINT Point // structure with point coordinates  
)
```

Options

hWndParent

Identifies the parent window.

Point

Specifies a POINT structure that sets out to test the working coordinates of the point.

Return values

If the function succeeds, the return value - handle to the child window that contains the point, even if the child window is hidden or blocked. If the point is outside the parent window, the return value - NULL (NULL). If the point - in the parent window and not within any child window, the return value - the parent window handle.

remarks

Windows maintains an internal list that contains the descriptors of child windows of the parent window. The order of descriptors in the list depends on the Z-order of child windows. If more than one child window contains a fixed point, Windows returns a value in the first window descriptor list that contains the point.

See also

[ChildWindowFromPointEx](#), [POINT](#), [WindowFromPoint](#)

[Accommodation and compatibility ChildWindowFromPoint](#)

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CHILDWINDOWFROMPOINTEX

ChildWindowFromPointEx function determines which, if any, of the child windows belonging to a specific parent window contains the installation point. The function can ignore invisible, locked, and transparent child windows.

Syntax

```
HWND ChildWindowFromPointEx  
(  
    HWND hwndParent, // handle to parent window  
    POINT pt, // structure with point coordinates  
    UINT uFlags // skip flags  
);
```

Options

hwndParent

Identifies the parent window.

pt

Specifies a POINT structure that defines the work to verify the coordinates of the point.

uFlags

Specifies which child windows are skipped. This parameter can be a combination of the following values:

- CWP_ALL - Do not skip any child windows.
- CWP_SKIPINVISIBLE - Skip invisible child windows.
- CWP_SKIPDISABLED - Ignore locked child windows.
- CWP_SKIPTRANSPARENT - Skip transparent child windows.

Return values

If the function succeeds, the return value - the descriptor of the first child window that contains the point and meets the criteria set out in uFlags. If the point - in the parent window and not within any child window that meets the criteria, the return value - the parent window handle. If the point is outside the parent window, or if the function is not executed, the return value - NULL (NULL).

remarks

Windows maintains an internal list that contains the descriptors of the child windows of the parent

window. The order of descriptors in the list depends on the Z-order of child windows. If more than one child window contains a fixed point, Windows returns a handle to the first box in the list that contains the point and meets the criteria defined uFlags.

See also

ChildWindowFromPoint, POINT, WindowFromPoint

Accommodation and compatibility ChildWindowFromPointEx

Windows NT Yes

Win95 Yes

No Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

FUNCTION CLOSEWINDOW

Function CloseWindow rolls (but does not destroy) a certain window.

Syntax:

```
BOOL CloseWindow  
(  
    HWND hWnd // handle to the window, which coagulates  
);
```

Options

hWnd

It identifies the window that should be minimized.

Return values

If the function succeeds, the return value - not zero. If the function fails the return value - zero. To get extended error information, call GetLastError.

remarks

The window is rolled, reducing its size to the icon, and moves in the area of the screen icons. Windows displays the Windows icon instead of the window and displays the window title below the icon. To destroy a window, the application must use DestroyWindow function.

See also

ArrangeIconicWindows, DestroyWindow, IsIconic, OpenIcon

Accommodation and compatibility CloseWindow

Windows NT Yes

Win95 Yes

Yes Win32s

Imported Library user32.lib

Header file winuser.h

No Unicode

Platform Notes None

WIN32 API. MENU

Menu (menu) consists of a list of menu items (menu items). Selecting a menu item opens a submenu or forcing application to execute a command. This overview describes the menus and explains how to use them in applications developed using Microsoft's application programming interface Win32 (API).

Menu items and menu

Menu is built hierarchically. line is a horizontal menu at the top level of the hierarchy (menu bar); vertical or a menu (menus) jump down from the menu bar, and in the lower levels - sub-menus (submenus). The menu bar is sometimes called a top-level menu (top-level menu), and the menus and submenus are also known as pop-up menus (pop-up menus).

The menu item can either execute a command or enter the submenu. An item that executes the command is called the command post (command item) or command (command).

Item in the horizontal menu is almost always opens a menu. Menu bars rarely contain command posts. Menu, open the menu bar pops down from it and is sometimes called a pop-up menu (drop-down menu). When the pop-up menu appears on the screen, it is associated with the menu bar. menu item in the menu bar, which opens a pop-up menu, the name is also called (menu name) menu.

The names of the menu in the menu bar are the main categories of commands that allows an application program. Select a menu name from the menu bar usually opens the menu items that correspond to commands in the category. For example, the menu bar can contain the name of the File menu (File) which, when selected by the user, activates the menu with items such as the New (New), Open (Open) and Save (Save).

Only overlapped or pop-up window can contain a menu bar; child window can not contain it. If the window has a title bar, Windows sets the menu bar directly beneath it. The menu bar is always visible. Submenu is not visible, but only up until the user selects a menu item, which activates it. For more information about overlapping and pop-up windows, see. Common window styles.

Each menu must have an owner window. Windows sends messages to the window menu, the owner, when the user selects a menu item or elect the menu. These messages are described in the Messages menu used.

Secondary menu

Windows also provides support and menus (shortcut menus). Secondary menu is not associated with the menu bar; it may appear anywhere on the screen. An application typically connects the sub-menu with the parts of the window, such as a work area or to a specific object, such as an icon. For this reason, these menus are also called "context menus".

Secondary menu remains hidden as long as the user does not activate it, usually choosing the right mouse button, for example, a toolbar or taskbar button. Menu is normally displayed on the screen at the cursor position of the mouse or a carriage.

Menu of the window

Window Menu (window menu) (also known as the System menu (System menu or Control menu)) - pop-up menu, defined and controlled almost exclusively by the operating system. The user can open the window menu by clicking on the program icon in the header area or by right-clicking anywhere in the header area.

Window Menu provides a standard set of menus that the user can elect to change the size or position of the window, or close the application. Items in the menu window can be added, deleted and modified, but the majority of applications using only a standard set of menu items. The overlay, pop-up or child window can have a window menu. Rarely happens to overlap, or pop-up windows are not included window menu.

When the user selects a command from the Window menu, Windows sends a message to the window menu WM_SYSCOMMAND owner. In most applications, the window procedure handles the messages is not from the menu window. Instead, it simply sends the message to the DefWindowProc function to process messages by the system. If an application adds a command in the Window menu, the window procedure should handle this command.

An application program that modifies the Windows menu, GetSystemMenu can use the function to create a copy of the default window menu. Any window that does not use GetSystemMenu function to do its own copy of the window menu receives the standard window menu.

Identification reference

Each menu bar, menus, sub-menus and sub menus linked to a 32-digit reference ID. If the user presses the F1, while, when the menu is activated, the value recovering window WM_HELP owner as part of the message. For more information about the reference IDs, see chap. Help.

Descriptors of the menu

The system creates a unique handle to each menu. Handle to the menu (menu handle) is a value type `HMENU`. The application must identify the menu handle many of the Windows menu functions. You get the handle of the menu bar when you create a menu, or upload menu resource. For more information on creating and loading menus, see. [Creating menus](#).

To retrieve a handle to the menu bar to the menu that has been created or downloaded, used `GetMenu` function. To retrieve a handle to a submenu associated with the menu item, use the `GetSubMenu` or `GetMenuItemInfo`. To extract the data window menu descriptor, use the `GetSystemMenu`.

Menu items

The following sections discuss what Windows does when the user selects a menu item and the manner in which an application can control the appearance and functionality of the menu item.

Command items and items which opens a submenu

When the user selects a command post, Windows sends a message to the team about the window that owns the menu. If the command center is located in the window menu, Windows sends a message WM_SYSCOMMAND. Otherwise, it sends a WM_COMMAND message.

The descriptor of the corresponding sub-menus associated with each menu item, which opens it. When a user points to a point, Windows opens a submenu. Window owner no reports of the team is not sent. However, Windows sends a message to the owner WM_INITMENUPOPUP window menu to display on the screen submenus. You can get a handle to a submenu associated with the menu item by using the function or GetSubMenu GetMenuItemInfo.

The menu bar includes menu names usually, but it can also contain command points. Submenu typically contains command posts, but it can also contain items that open nested submenus. Adding such items in a submenu, you can put the menu to any depth. To provide the user with a visual signal, Windows automatically displays on the screen a small arrow to the right of the text of the menu item that opens a submenu.

Menu item identification.

The unique, application-defined integer associated with each menu item is called a menu item identifier (menu-item identifier). When the user selects a command from the menu, Windows sends the ID of the owner of the item window as part of the WM_COMMAND messages. The window procedure checks the ID to identify the source of the message and processes the message accordingly. In addition, you can define a menu item by using its ID when calling the menu functions; for example, to enable or disable a menu item.

ID menu item must be a value from 0 to 65,535, even though it is - a 32-bit integer. This is so because it sends WM_COMMAND message identifier of the menu item as the low-order word of the wParam parameter.

Menu items that open submenus have IDs are exactly the same as the command posts. However, Windows does not send a message to the team when a point is selected from the menu. Instead, Windows opens a submenu associated with the menu item.

To retrieve the data on the ID of the menu item at a given position, use the GetMenuItemID or GetMenuItemInfo.

Position of menu item

In addition to having a unique identifier and has a unique value position of each menu item in the menu bar or menu. The most extreme left point in the menu bar, or the top item on the menu has a zero position. The position value is incremented to the next menu. Windows assigns the value of the position of all the items on the menu, including separators.

When you call the function menu, which modifies or returns information about a particular menu item, you can specify a point using either the ID or position. For more information about changing menu content. See menu modifications.

Sets the default menu items

The submenu can contain one default menu item. When a user opens a submenu, double-clicking the mouse, Windows sends a message to the team owner menu window closes and the menu was chosen as the default command post. If there is no specified default command post, a submenu is open. To retrieve the data, and set the default item for a submenu, use and function GetMenuItemDefaultItem SetMenuItemDefaultItem.

Menu items with and without tick

The menu item can be or is marked "check" whether or not "tick". Windows displays on-screen bitmap next to the menu items marked to indicate their condition marked with "tick". Windows does not show on the screen bitmap next to no "tick" if the application-defined bitmap "no" tick "" is not defined. The menu items can only be marked "check"; in command centers could not be marked on the menu bar "checkbox."

Applications usually say "check" or remove the mark "check" in the menu to specify which of several parameters is valid. For example, assume that the application has a toolbar, the user can show or hide using the Toolbar command from the menu use. When the toolbar is hidden, at the Toolbar menu item is not flagged "tick". When the user selects, the application says "check" menu item and displays the toolbar.

Attribute tags "check" checks whether the menu item is marked "check". You can set the attribute mark "check" menu item by using CheckMenuItem function. You can use GetMenuItemState function to determine whether or not marked is marked in a menu item is currently "tick".

Instead CheckMenuItem and GetMenuItemState, you can use the functions and GetMenuItemInfo SetMenuItemInfo, to retrieve data and to establish the existence of a mark "check" in the menu.

Sometimes, a group of menu items corresponds to a set of mutually exclusive options. In this case, you can designate the selected parameter by using the mark menu item "radioselektorom" ("radioselektor" similar control body "radio button"). Markers menu items "radioselektorom" displays the marker dot pattern instead of bitmap notes "tick". To mark a menu item and make it a point to "radioselektorom", use the CheckMenuRadioItem.

By default, Windows displays on the screen marked "check" or bitmap "radioselektora" next to the marked menu items and no bitmap next to uncheck "check" menu items. However, you can use SetMenuItemBitmaps function to associate the program defined bitmaps mark and unmark "check" to the menu item. Windows then uses the specified bitmaps to indicate the state of the menu item with the "tick" mark or no "check".

Determine the program bitmaps associated with the menu item should be the same size as the default bitmap "tick", the dimensions of which may vary depending on the screen resolution. To retrieve information about the right size, use the GetMenuCheckMarkDimensions. You can create multi-bitmap resources for different screen resolutions; Create a bitmap resource, and, if necessary, to scale it; or create a bitmap at runtime and paint the image in it. Bitmaps can be either monochrome or color. However, since the menu items that stand out are inverted, the appearance of some inverted color bitmaps may not be the best. For more information, see. Bitmap.

Included, inaccessibility and lock menu items

The menu item can be included, inaccessible or locked. By default, the menu item is enabled. When a user elects included menu, Windows sends a command message to the owner or the window on the screen shows the corresponding sub-menu, depending on what type of the menu item.

When menu items are not accessible to the user, they should be blocked or gray. Unavailable menu items are locked and can not be selected. Locked item looks just like the inclusion of the item. When the user clicks on a locked item, it is not selected, and nothing happens. Banned items can be useful, for example, the manual shows the menu that appears active, but (for training purposes) is not.

Application grayed unavailable menu to provide a visual cue to the user that the command is not available. You can use the item is not available when the action is not suitable (for example, you can paint in gray Print command (the Print) from the File menu (the File) when the printer is not installed system).

EnableMenuItem function includes, painted gray or disabled menu item. To determine how a menu item is enabled, blocked or unavailable, use the GetMenuItemInfo.

Instead GetMenuItemInfo, you can also use GetMenuState function to install on, is not available or blocked menu item.

Selected menu item

Windows automatically selects menu items when the user selects them. However, the allocation may be explicitly added or removed from the name of the item in the menu bar by using the function HiliteMenuItem. This function has no influence on the items in the menu. When HiliteMenuItem used to select a menu title, in spite of this, the name only appears when it is selected. If the user presses the ENTER, and the highlighted item is not selected. This feature may be useful, for example, in the preparation of the application program that shows the use of menus.

Custom menu items

An application can completely control the appearance of the menu item by using the user-drawn (own) of paragraph (owner-drawn item). User Drawn items require that the application was taking full responsibility for the state of the selection draw (selection), "tick" and unmark "tick". For example, if the application provides a font menu, it can register each menu item by using the appropriate font; point for the Latin font will be registered in Latin, for the point will be registered in italics italics, and so on.

For more information, see. [Creating a user-drawn menu items](#).

Menu divider and line breaks

Windows provides a special type of menu item, called a separator (separator), which appears as a horizontal line. You can use a splitter to divide the menu into groups of related items. The separator can not be used in the menu bar, the user can not select a separator.

When the menu bar contains more names of menu items than can be placed on one line, Windows moves according to the menu bar, automatically breaking it into two or more lines. You can get to make a line break in a specific section of the horizontal menu, assigning the item box type MFT_MENUBREAK. Windows places that item and all subsequent items on a new line.

When a menu contains more items than can fit in a single column, Windows automatically closes the menu into two or more columns. You can get to make a column break at a specific point in the menu, assigning the item box type MFT_MENUBREAK. Windows places that item and all subsequent items in a new column. MFT_MENUARBREAK type flag has the same effect, except that a vertical line appears between the new and the old column.

When you use AppendMenu, InsertMenu or ModifyMenu, to designate line breaks, you must assign standard or flags MF_MENUBREAK MF_MENUARBREAK.

Creating the menu

You can create menus using either a menu template or function of its creation. Menu templates are usually defined as resources. Resources menu template can be downloaded directly or designated as the default menu for the window class. You can also create resources menu template in memory dynamically.

Resources of menu templates

Most applications creates a menu using the resources of the menu template. Template menu (menu template) determines the menu, including an items in the menu bar and all menus. For information about creating a resource menu template. See the documentation included in your development tools.

After you create a resource of the menu template, and add it to the executable (.EXE) file of your application, you can use LoadMenu function to load the resource into memory. This function returns a handle to the menu, which you can then assign the window by using SetMenu function.

Implementation of the menu as a resource for making the application easier to tie it to a particular country for use in many countries. Only the resource definition file must be tied to a specific country for each language, and not the source code application.

Template of menu in memory

The menu can be created from the menu template, which is formed in the memory during program execution. For example, an application that allows the user to customize their menus, can create a menu template in memory, based on the standard set by the user. The application can then save the template in a file or in the registry for future use. To create a menu template in memory, use LoadMenuIndirect function. For descriptions of the menu format template, refer to the article Using the Resource menu template.

Standard menu template consists of MENUITEMTEMPLATEHEADER structure, followed by one or more structures MENUITEMTEMPLATE.

Advanced menu template consists of MENUEX_TEMPLATE_HEADER structure, followed by one or more structures MENUEX_TEMPLATE_ITEM.

Functions of the menu

Using options to create a menu, you can create a menu at runtime or add menu items to the existing menu. You can use CreateMenu function to create an empty menu bar and CreatePopupMenu function to create an empty menu. To add items to the menu, use the InsertMenuItem. Obsolete AppendMenu InsertMenu and features are still supported, but new applications InsertMenuItem function should be used.

Show on-screen menu

Once the menu has been downloaded or created, it must be associated with the window before Windows will display it on the screen. You can dedicate the menu, using the class definition menu. For more information on the class menu, see. [Menu window class](#). You can also dedicate a menu for the window, using the definition of the menu descriptor as a parameter hMenu function CreateWindow or CreateWindowEx or by calling SetMenu function.

To show the on-screen menu, use the auxiliary TrackPopupMenuEx function. Secondary menu, also called floating, pop-up or context menu is normally displayed on the screen when a message is processed WM_CONTEXTMENU.

Outdated TrackPopupMenu function is still supported, but new applications should use TrackPopupMenuEx function.

Menu window class

You can define the default menu, the menu is called a class (class menu), when you register the window class. To do so, set the template name of the resource menu item lpszMenuName in the WNDCLASS structure, used for class registration.

By default, each window class menu is assigned to the window class, so you do not need to explicitly load the menu and assign it to each window. You can replace the menu class, by determining the different menu descriptor when calling CreateWindowEx function. You can also change the window's menu after it was created by using the function SetMenu. For more information, see [Classes window](#).

Destruction of menu

If the menu associated with a window, and that window is destroyed, Windows automatically deletes the menu, freeing the menu descriptor and the memory occupied by the menu. Windows does not, the automatic destruction of the menu, which is not associated with a window. The application must destroy mission canceled by calling the menu function DestroyMenu. Otherwise, the menu continues to exist in the memory even after the application is closed.

Messages used by menu

Windows reports that the action associated menu by sending messages to the window procedure of the window that owns the menu. Windows sends a series of messages, when a user selects items in the menu bar or clicks the right mouse button to display the on-screen menu support.

When the user activates the item in the menu bar, a window owner first gets a message WM_SYSCOMMAND. This message includes a flag that indicates whether the user has activated the menu by using the keyboard (SC_KEYMENU) or mouse (SC_MOUSEMENU). For more information about the keyboard interface to menus, see. Access to the menu via the keyboard.

Then, before the show on the screen of any menu, Windows sends a message WM_INITMENU window procedure so that the application can change the menu before the user sees them. Windows sends WM_INITMENU message only once to enter the Menu.

When a user points to a menu item that opens a submenu, the Windows, a window sends a message to the owner WM_INITMENUPOPUP display on the screen submenus. This message gives the application the ability to change a submenu before it will appear on the screen.

Each time the user moves the selection from one item to another, Windows sends a message to the window procedure of the window WM_MENUSELECT menu holder. This report identifies the current selected menu item. Many applications provide an information area at the bottom of its main window and use this message to show on the screen for more information about the selected menu item.

When the user selects a command from the menu, Windows sends a WM_COMMAND message to the window procedure. Low word of wParam WM_COMMAND message parameter contains the identifier of the selected item. The window procedure is to check ID and process the message accordingly.

Not all menus are accessible via the menu bar of the window. Many of the applications are shown on the screen sub-menu when the user clicks the right mouse button in a particular place. Such applications, if they tend to have to handle WM_CONTEXTMENU message and display on the screen sub-menu. If the application does not display the sub menu, it needs to send a message WM_CONTEXTMENU DefWindowProc function for default processing.

Changing the menu

Several functions allow you to change the menu after it was uploaded or created. These changes can include adding or removing menu items and modifying existing menu items.

To add a menu item, use the InsertMenuItem. You can use SetMenuItemInfo function to change the attributes of an existing menu item. Parameter lpmii indicates MENUITEMINFO structure that contains the new attributes and specifies which attributes are changed. Attributes menu items include the type, state, identifier, submenu, bitmaps, item data and text.

Obsolete AppendMenu InsertMenu and functions can also be used to add menu items, but new applications should use InsertMenuItem. AppendMenu function appends a menu item or submenu; InsertMenu function inserts a menu item at the specified position in the menu or submenu. Both features allow the attributes of a menu item to be defined, including an in any case, including blocking, inaccessibility, mark "or tick removal marks" check "menu item".

To change the appearance or attributes of an existing menu item, use the ModifyMenu. For example, a text string or bitmap menu item can be included, blocked, unavailable, marked "check" or removed from the "tick". ModifyMenu function replaces the specified menu item to the next item.

To retrieve information about a menu item, use the GetMenuItemInfo. Parameter lpmii indicates MENUITEMINFO structure that defines attributes recoverable and receives their current values.

To remove a menu item from the menu, use the or DeleteMenu RemoveMenu. If you delete the item that opens a submenu, DeleteMenu removes the associated submenu, discarding the menu handle and freeing the memory used by the submenu. RemoveMenu function removes the menu, but if the item opens a submenu, the function does not destroy the submenu or handle, allowing the submenu reused.

To redraw the menu bar, after the menu bar has been modified, use DrawMenuBar function. Otherwise, the changes do not appear until Windows redraws the owner window.

Using the menu

Using the resource menu template

Establishment of a subsidiary (context) menus

Using bitmap menu items

Creating a user-drawn menu items

Using custom bitmaps "tick"

Resource usage of menu templates

You typically include a menu in an application program, through the creation of a resource menu template, and then load the menu at run time. This section describes the format of the menu template, and explains how to load the resource menu template and use it in your application. For information about creating a resource menu template. See the documentation included with your software development tools.

Advanced format for menu templates

Advanced Format menu template support function menu features made for Windows 95 and Windows NT 4.0. Like Resources menu template, used with earlier versions of the Windows, enhanced resources are of type menu template RT_MENU resource. Windows distinguishes between two resource format version number, which is the first element of the resource header.

Advanced menu template consists of MENUEX_TEMPLATE_HEADER structure, followed by another structure MENUEX_TEMPLATE_ITEM defining points.

Old format of menu templates

Old menu template (for versions of Windows earlier than Windows 95 and Windows NT 4.0) sets the menu, but it does not support the new functionality. Old menu template resource is of type RT_MENU resource.

Old menu template consists of MENUITEMTEMPLATEHEADER structure, followed by one or more MENUITEMTEMPLATE structures.

Resource loading of menu templates

To download the resource menu template, use LoadMenu function, specifying the module handle, which contains a resource and an identifier of the menu template. LoadMenu function returns a handle to the menu that you can use to bind to the menu window. This window becomes the owner of the window menu, accepting all posts by menu.

To create a menu from a menu template that is already in memory, use LoadMenuIndirect function. This is useful if, if your application generates menu templates dynamically.

To assign a menu to a window, use the SetMenu, or define a menu handle in hMenu parameter CreateWindowEx function to create the window. Another way that you can bind a menu to a window, a menu template definition when you register the window class; template identifies the specified menu as the class menu for that window class.

To have Windows automatically linked with a menu window, specify menu template, when you register the window class. The template identifies the specified menu as the class menu for that window class. Then, when you create a window of this class, Windows automatically connects the specified menu to the window.

To create class menu, include menu template resource ID as a member of the WNDCLASS lpszMenuName structures, and then pass the address of a structure in RegisterClass function.

Creating menu class

The following example shows how to create a menu class for the application, create a window that uses the class menu and handle menu commands to the window procedure.

The following is an important part of the header file of the application program:

```
// Resource identifier menu template
```

```
#define IDM_MYMENURESOURCE 3
```

The following is an important part of the application:

```
HINSTANCE hinst;
```

```
int APIENTRY WinMain (hinstance, hPrevInstance, lpCmdLine, nCmdShow)
```

```
HINSTANCE hinstance;
```

```
HINSTANCE hPrevInstance;
```

```
LPSTR lpCmdLine;
```

```
int nCmdShow;
```

```
{
```

```
MSG msg; // message
```

```
WNDCLASS wc; // Window class data
```

```
HWND hwnd; // Handle to the main window
```

```
// Create the window class for the main window.
```

```
// Define template resource identifier as a menu
```

```
// Element lpszMenuName WNDCLASS structure to
```

```
// Create class menu.
```

```
wc.style = 0;
```

```
wc.lpfnWndProc = (WNDPROC) MainWndProc;
```

```
wc.cbClsExtra = 0;
```

```
wc.cbWndExtra = 0;
```

```
wc.hInstance = hinstance;
```

```
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
```

```
wc.hCursor = LoadCursor (NULL, IDC_ARROW);
```

```
wc.hbrBackground = GetStockObject (WHITE_BRUSH);
```

```
wc.lpszMenuName = MAKEINTRESOURCE (IDM_MYMENURESOURCE);
wc.lpszClassName = "MainWClass";
if (! RegisterClass (& wc))
    return FALSE;
hinst = hinstance;
// Create the main window. We set the parameters
// Hmenu to a null value (NULL) so that Windows
// Use the class menu for the window.
hwnd = CreateWindow ( "MainWClass", "Sample Application",
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hinstance,
NULL);
if (hwnd == NULL)
    return FALSE;
// Make the window visible and give a message
// WM_PAINT to the window procedure.
ShowWindow (hwnd, nCmdShow);
UpdateWindow (hwnd);
// Run the main message loop.
while (GetMessage (& msg, NULL, 0, 0)) {
    TranslateMessage (& msg);
    DispatchMessage (& msg);
}
return msg.wParam;
UNREFERENCED_PARAMETER (hPrevInstance);
}
LRESULT APIENTRY MainWndProc (hwnd, uMsg, wParam, lParam)
HWND hwnd;
UINT uMsg;
```

```
WPARAM wParam;
LPARAM lParam;
{
switch (uMsg) {
// Process other window messages.

case WM_COMMAND:
// Check command menu item identifier.
switch (LOWORD (wParam)) {
case IDM_FI_OPEN:
DoFileOpen (); // Defined program
break;
case IDM_FI_CLOSE:
DoFileClose (); // Defined program
break;

. // Processing menu commands.

.

default:
break;
}
return 0;

.

. // Process other window messages.

.

default:
return DefWindowProc (hwnd, uMsg, wParam, lParam);
}

return NULL;
}
```

Creating of a subsidiary (context) menu

To use the context menu in the application, send it to handle TrackPopupMenuEx function. An application typically calls TrackPopupMenuEx in the window procedure in response to a user-generated message, such as WM_LBUTTONDOWN or WM_KEYDOWN.

In addition to the pop-up menu descriptor, TrackPopupMenuEx requires that you define a handle of the owner window, the context menu item (in screen coordinates) and the mouse button, the user can use to select an item.

Outdated TrackPopupMenu function is still supported, but new applications should use TrackPopupMenuEx function. TrackPopupMenuEx function requires the same parameters as the TrackPopupMenu and, in addition, allows you to specify the screen that the menu should not be closed. An application typically calls this function in the window procedure when processing WM_CONTEXTMENU posts.

You can determine the position of the context menu, giving the x- and y-coordinates with the flag TPM_CENTERALIGN, TPM_LEFTALIGN or TPM_RIGHTALIGN. The checkbox determines the position of the shortcut menu with respect to x- and y-coordinates.

You must enable the user to select an item from the context menu through the use of the same mouse button, which is used to display the menu screen. To do this, define or check TPM_LEFTBUTTON or TPM_RIGHTBUTTON, to indicate which mouse button the user can use to select the menu item.

On-screen display the context menu

The function shown in the following example shows the on-screen context menu.

The application includes a menu resource identified by the string "ShortcutExample". The menu bar contains a title menu. The application uses TrackPopupMenu function to show the menu screen, the associated menu item. (Itself the menu bar is not displayed because it requires TrackPopupMenu descriptor menu, submenu, or shortcut menu.)

```
VOID APIENTRY DisplayContextMenu (HWND hwnd, POINT pt)
{
    HMENU hmenu; // Top-level menu
    HMENU hmenuTrackPopup; // Context menu>
    // Load the menu resource.
    if ((hmenu = LoadMenu (hinst, "ShortcutExample")) == NULL)
        return;
    // TrackPopupMenu may not show on screen
    // Menu bar, as received descriptor
    // The first pop-up menu
    hmenuTrackPopup = GetSubMenu (hmenu, 0);
    // Will show the context menu. Track down the right mouse button.
    TrackPopupMenu (hmenuTrackPopup,
                    TPM_LEFTALIGN | TPM_RIGHTBUTTON,
                    pt.x, pt.y, 0, hwnd, NULL);
    // Destroy the menu.
    DestroyMenu (hmenu);
}
```

Use the menu bitmap (icon)

The Windows, to show the on-screen menu, can use a bitmap (icon) instead of a text string. To use an icon, you have to set MFT_BITMAP box for the menu item and determine the bitmap handle that Windows should display for the menu item. This section describes how to install and remove the check MFT_BITMAP data descriptor icon.

Application programs written in earlier versions of the Windows, can be installed with the old box MF_BITMAP feature sets.

Checking the box type icon

MFT_BITMAP checkbox or MF_BITMAP reported to predominantly use Windows bitmap (icon) and not the text string to display on-screen menu. Checkbox menu item or MFT_BITMAP MF_BITMAP must be set during the program; You can not install it in a file resource definition.

For new applications, you can use the function or the InsertMenuItem SetMenuItemInfo, to check the box type MFT_BITMAP. To replace the text on the menu icon menu item, use the SetMenuItemInfo. To add a new icon to a menu item, use the InsertMenuItem.

Application programs written for earlier versions of the Windows, to install MF_BITMAP box can continue to use ModifyMenu function, or InsertMenu AppendMenu. To replace the text string on the menu icon menu item, use the ModifyMenu. To add a new icon menu item, use the check function to MF_BITMAP InsertMenu or AppendMenu.

