About Window Classes - Win32 apps

By Karl-Bridge-Microsoft

Archived: 2026-04-05 13:32:00 UTC

Each window class has an associated window procedure shared by all windows of the same class. The window procedure processes messages for all windows of that class and therefore controls their behavior and appearance. For more information, see Window Procedures.

A process must register a window class before it can create a window of that class. Registering a window class associates a window procedure, class styles, and other class attributes with a class name. When a process specifies a class name in the **CreateWindow** or **CreateWindowEx** function, the system creates a window with the window procedure, styles, and other attributes associated with that class name.

This section discusses the following topics.

Types of Window Classes

There are three types of window classes:

- System Classes
- Application Global Classes
- Application Local Classes

These types differ in scope and in when and how they are registered and destroyed.

System Classes

A system class is a window class registered by the system. Many system classes are available for all processes to use, while others are used only internally by the system. Because the system registers these classes, a process cannot destroy them.

The system registers the system classes for a process the first time one of its threads calls a User or a Windows Graphics Device Interface (GDI) function.

Each application receives its own copy of the system classes. All 16-bit Windows-based applications in the same VDM share system classes, just as they do on 16-bit Windows.

The following table describes the system classes that are available for use by all processes.

Class	Description
Button	The class for a button.
ComboBox	The class for a combo box.
Edit	The class for an edit control.
ListBox	The class for a list box.
MDIClient	The class for an MDI client window.
ScrollBar	The class for a scroll bar.
Static	The class for a static control.

The following table describes the system classes that are available only for use by the system. They are listed here for completeness sake.

Class	Description
ComboLBox	The class for the list box contained in a combo box.
DDEMLEvent	The class for Dynamic Data Exchange Management Library (DDEML) events.
Message	The class for a message-only window.
#32768	The class for a menu.
#32769	The class for the desktop window.
#32770	The class for a dialog box.
#32771	The class for the task switch window.
#32772	The class for icon titles.

Application Global Classes

An application global class is a window class registered by an executable or DLL that is available to all other modules in the process. For example, your .dll can call the **RegisterClassEx** function to register a window class that defines a custom control as an application global class so that a process that loads the .dll can create instances of the custom control.

To create a class that can be used in every process, create the window class in a .dll and load the .dll in every process. To load the .dll in every process, add its name to the **AppInit_DLLs** value in following registry key:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows

Whenever a process starts, the system loads the specified .dll in the context of the newly started process before calling its entry-point function. The .dll must register the class during its initialization procedure and must specify the **CS_GLOBALCLASS** style. For more information, see Class Styles.

To remove an application global class and free the storage associated with it, use the **UnregisterClass** function.

Application Local Classes

An application local class is any window class that an executable or .dll registers for its exclusive use. Although you can register any number of local classes, it is typical to register only one. This window class supports the window procedure of the application's main window.

The system destroys a local class when the module that registered it closes. An application can also use the **UnregisterClass** function to remove a local class and free the storage associated with it.

How the System Locates a Window Class

The system maintains a list of structures for each of the three types of window classes. When an application calls the **CreateWindow** or **CreateWindowEx** function to create a window with a specified class, the system uses the following procedure to locate the class.

1. Search the list of application local classes for a class with the specified name whose instance handle matches the module's instance handle. (Several modules can use the same name to register local classes in the same process.)
2. If the name is not in the application local class list, search the list of application global classes.
3. If the name is not in the application global class list, search the list of system classes.

All windows created by the application use this procedure, including windows created by the system on the application's behalf, such as dialog boxes. It is possible to override system classes without affecting other applications. That is, an application can register an application local class having the same name as a system class. This replaces the system class in the context of the application but does not prevent other applications from using the system class.

Registering a Window Class

A window class defines the attributes of a window, such as its style, icon, cursor, menu, and window procedure. The first step in registering a window class is to fill in a **WNDCLASSEX** structure with the window class information. For more information, see Elements of a Window Class. Next, pass the structure to the **RegisterClassEx** function. For more information, see Using Window Classes.

To register an application global class, specify the CS_GLOBALCLASS style in the **style** member of the **WNDCLASSEX** structure. When registering an application local class, do not specify the **CS_GLOBALCLASS** style.

If you register the window class using the ANSI version of **RegisterClassEx**, **RegisterClassExA**, the application requests that the system pass text parameters of messages to the windows of the created class using the ANSI character set; if you register the class using the Unicode version of **RegisterClassEx**, **RegisterClassExW**, the application requests that the system pass text parameters of messages to the windows of the created class using the Unicode character set. The **IsWindowUnicode** function enables applications to query the nature of each window. For more information on ANSI and Unicode functions, see Conventions for Function Prototypes.

The executable or DLL that registered the class is the owner of the class. The system determines class ownership from the **hInstance** member of the **WNDCLASSEX** structure passed to the **RegisterClassEx** function when the class is registered. For DLLs, the **hInstance** member must be the handle to the .dll instance.

The class is not destroyed when the .dll that owns it is unloaded. Therefore, if the system calls the window procedure for a window of that class, it will cause an access violation, because the .dll containing the window procedure is no longer in memory. The process must destroy all windows using the class before the .dll is unloaded and call the **UnregisterClass** function.

Elements of a Window Class

The elements of a window class define the default behavior of windows belonging to the class. The application that registers a window class assigns elements to the class by setting appropriate members in a **WNDCLASSEX** structure and passing the structure to the **RegisterClassEx** function. The **GetClassInfoEx** and **GetClassLong** functions retrieve information about a given window class. The **SetClassLong** function changes elements of a local or global class that the application has already registered.

Although a complete window class consists of many elements, the system requires only that an application supply a class name, the window-procedure address, and an instance handle. Use the other elements to define default attributes for windows of the class, such as the shape of the cursor and the content of the menu for the window. You must initialize any unused members of the **WNDCLASSEX** structure to zero or **NULL**. The window class elements are as shown in the following table.

Element	Purpose
Class Name	Distinguishes the class from other registered classes.
Window Procedure	Pointer to the function that processes all messages sent to windows in the class and defines the behavior of the window.

Element	Purpose
Address	
Instance Handle	Identifies the application or .dll that registered the class.
Class Cursor	Defines the mouse cursor that the system displays for a window of the class.
Class Icons	Defines the large icon and the small icon.
Class Background Brush	Defines the color and pattern that fill the client area when the window is opened or painted.
Class Menu	Specifies the default menu for windows that do not explicitly define a menu.
Class Styles	Defines how to update the window after moving or resizing it, how to process double-clicks of the mouse, how to allocate space for the device context, and other aspects of the window.
Extra Class Memory	Specifies the amount of extra memory, in bytes, that the system should reserve for the class. All windows in the class share the extra memory and can use it for any application-defined purpose. The system initializes this memory to zero.
Extra Window Memory	Specifies the amount of extra memory, in bytes, that the system should reserve for each window belonging to the class. The extra memory can be used for any application-defined purpose. The system initializes this memory to zero.

Class Name

Every window class needs a Class Name to distinguish one class from another. Assign a class name by setting the **lpszClassName** member of the **WNDCLASSEX** structure to the address of a null-terminated string that specifies the name. Because window classes are process specific, window class names need to be unique only within the same process. Also, because class names occupy space in the system's private atom table, you should keep class name strings as short as possible.

The **GetClassName** function retrieves the name of the class to which a given window belongs.

Window Procedure Address

Every class needs a window-procedure address to define the entry point of the window procedure used to process all messages for windows in the class. The system passes messages to the procedure when it requires the window to carry out tasks, such as painting its client area or responding to input from the user. A process assigns a window procedure to a class by copying its address to the **lpfnWndProc** member of the **WNDCLASSEX** structure. For more information, see Window Procedures.

Instance Handle

Every window class requires an instance handle to identify the application or .dll that registered the class. The system requires instance handles to keep track of all of modules. The system assigns a handle to each copy of a running executable or .dll.

The system passes an instance handle to the entry-point function of each executable (see **WinMain**) and .dll (see **DllMain**). The executable or .dll assigns this instance handle to the class by copying it to the **hInstance** member of the **WNDCLASSEX** structure.

Class Cursor

The *class cursor* defines the shape of the cursor when it is in the client area of a window in the class. The system automatically sets the cursor to the given shape when the cursor enters the window's client area and ensures it keeps that shape while it remains in the client area. To assign a cursor shape to a window class, load a predefined cursor shape by using the **LoadCursor** function and then assign the returned cursor handle to the **hCursor** member of the **WNDCLASSEX** structure. Alternatively, provide a custom cursor resource and use the **LoadCursor** function to load it from the application's resources.

The system does not require a class cursor. If an application sets the **hCursor** member of the **WNDCLASSEX** structure to **NULL**, no class cursor is defined. The system assumes the window sets the cursor shape each time the cursor moves into the window. A window can set the cursor shape by calling the **SetCursor** function whenever the window receives the **WM_MOUSEMOVE** message. For more information about cursors, see Cursors.

Class Icons

A *class icon* is a picture that the system uses to represent a window of a particular class. An application can have two class icons—one large and one small. The system displays a window's *large class icon* in the task-switch window that appears when the user presses ALT+TAB, and in the large icon views of the task bar and explorer. The *small class icon* appears in a window's title bar and in the small icon views of the task bar and explorer.

To assign a large and small icon to a window class, specify the handles of the icons in the **hIcon** and **hIconSm** members of the **WNDCLASSEX** structure. The icon dimensions must conform to required dimensions for large and small class icons. For a large class icon, you can determine the required dimensions by specifying the **SM_CXICON** and **SM_CYICON** values in a call to the **GetSystemMetrics** function. For a small class icon, specify the **SM_CXSMICON** and **SM_CYSMICON** values. For information, see Icons.

If an application sets the **hIcon** and **hIconSm** members of the **WNDCLASSEX** structure to **NULL**, the system uses the default application icon as the large and small class icons for the window class. If you specify a large class icon but not a small one, the system creates a small class icon based on the large one. However, if you specify a small class icon but not a large one, the system uses the default application icon as the large class icon and the specified icon as the small class icon.

You can override the large or small class icon for a particular window by using the **WM_SETICON** message. You can retrieve the current large or small class icon by using the **WM_GETICON** message.

Class Background Brush

A *class background brush* prepares the client area of a window for subsequent drawing by the application. The system uses the brush to fill the client area with a solid color or pattern, thereby removing all previous images from that location whether they belong to the window or not. The system notifies a window that its background should be painted by sending the **WM_ERASEBKGND** message to the window. For more information, see Brushes.

To assign a background brush to a class, create a brush by using the appropriate GDI functions and assign the returned brush handle to the **hbrBackground** member of the **WNDCLASSEX** structure.

Instead of creating a brush, an application can set the **hbrBackground** member to one of the standard system color values. For a list of the standard system color values, see **SetSysColors**.

To use a standard system color, the application must increase the background-color value by one. For example, **COLOR_BACKGROUND** + 1 is the system background color. Alternatively, you can use the **GetSysColorBrush** function to retrieve a handle to a brush that corresponds to a standard system color, and then specify the handle in the **hbrBackground** member of the **WNDCLASSEX** structure.

The system does not require that a window class have a class background brush. If this parameter is set to **NULL**, the window must paint its own background whenever it receives the **WM_ERASEBKGND** message.

A *class menu* defines the default menu to be used by the windows in the class if no explicit menu is given when the windows are created. A menu is a list of commands from which a user can choose actions for the application to carry out.

You can assign a menu to a class by setting the **lpszMenuName** member of the **WNDCLASSEX** structure to the address of a null-terminated string that specifies the resource name of the menu. The menu is assumed to be a resource in the given application. The system automatically loads the menu when it is needed. If the menu resource is identified by an integer and not by a name, the application can set the **lpszMenuName** member to that integer by applying the **MAKEINTRESOURCE** macro before assigning the value.

The system does not require a class menu. If an application sets the **lpszMenuName** member of the **WNDCLASSEX** structure to **NULL**, windows in the class have no menu bars. Even if no class menu is given, an application can still define a menu bar for a window when it creates the window.

If a menu is given for a class and a child window of that class is created, the menu is ignored. For more information, see Menus.

Class Styles

The class styles define additional elements of the window class. Two or more styles can be combined by using the bitwise OR (|) operator. To assign a style to a window class, assign the style to the **style** member of the **WNDCLASSEX** structure. For a list of class styles, see **Window Class Styles**.

Classes and Device Contexts

A *device context* is a special set of values that applications use for drawing in the client area of their windows. The system requires a device context for each window on the display but allows some flexibility in how the system stores and treats that device context.

If no device-context style is explicitly given, the system assumes each window uses a device context retrieved from a pool of contexts maintained by the system. In such cases, each window must retrieve and initialize the device context before painting and free it after painting.

To avoid retrieving a device context each time it needs to paint inside a window, an application can specify the **CS_OWNDC** style for the window class. This class style directs the system to create a private device context—that is, to allocate a unique device context for each window in the class. The application need only retrieve the context once and then use it for all subsequent painting.

The system maintains a **WNDCLASSEX** structure internally for each window class in the system. When an application registers a window class, it can direct the system to allocate and append a number of additional bytes of memory to the end of the **WNDCLASSEX** structure. This memory is called *extra class memory* and is shared by all windows belonging to the class. Use the extra class memory to store any information pertaining to the class.

Because extra memory is allocated from the system's local heap, an application should use extra class memory sparingly. The **RegisterClassEx** function fails if the amount of extra class memory requested is greater than 40 bytes. If an application requires more than 40 bytes, it should allocate its own memory and store a pointer to the memory in the extra class memory.

The **SetClassWord** and **SetClassLong** functions copy a value to the extra class memory. To retrieve a value from the extra class memory, use the **GetClassWord** and **GetClassLong** functions. The **cbClsExtra** member of the **WNDCLASSEX** structure specifies the amount of extra class memory to allocate. An application that does not use extra class memory must initialize the **cbClsExtra** member to zero.

The system maintains an internal data structure for each window. When registering a window class, an application can specify a number of additional bytes of memory, called *extra window memory*. When creating a window of the class, the system allocates and appends the specified amount of extra window memory to the end of the window's structure. An application can use this memory to store window-specific data.

Because extra memory is allocated from the system's local heap, an application should use extra window memory sparingly. The **RegisterClassEx** function fails if the amount of extra window memory requested is greater than 40 bytes. If an application requires more than 40 bytes, it should allocate its own memory and store a pointer to the memory in the extra window memory.

The **SetWindowLong** function copies a value to the extra memory. The **GetWindowLong** function retrieves a value from the extra memory. The **cbWndExtra** member of the **WNDCLASSEX** structure specifies the amount of extra window memory to allocate. An application that does not use the memory must initialize **cbWndExtra** to zero.
