**Window Classes**

A *window class* defines a set of behaviors that several windows might have in common. For example, in a group of buttons, each button has a similar behavior when the user clicks the button. Of course, buttons are not completely identical; each button displays its own text string and has its own screen coordinates. Data that is unique for each window is called *instance data*.

Every window must be associated with a window class, even if your program only ever creates one instance of that class. It is important to understand that a window class is not a "class" in the C++ sense. Rather, it is a data structure used internally by the operating system. Window classes are registered with the system at run time.

You must set the following structure members:

* **lpfnWndProc** is a pointer to an application-defined function called the *window procedure* or "window proc." The window procedure defines most of the behavior of the window. We'll examine the window procedure in detail later. For now, just treat this as a forward reference.
* **hInstance** is the handle to the application instance. Get this value from the *hInstance* parameter of **wWinMain**.
* **lpszClassName** is a string that identifies the window class.

Class names are local to the current process, so the name only needs to be unique within the process. However, the standard Windows controls also have classes. If you use any of those controls, you must pick class names that do not conflict with the control class names. For example, the window class for the button control is named "Button".

Next, pass the address of the [**WNDCLASS**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms633576(v=vs.85).aspx) structure to the [**RegisterClass**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms633586(v=vs.85).aspx) function. This function registers the window class with the operating system.

To create a new instance of a window, call the [**CreateWindowEx**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms632680(v=vs.85).aspx) function.

You can read detailed parameter descriptions on MSDN, but here is a quick summary:

* The first parameter lets you specify some optional behaviors for the window (for example, transparent windows). Set this parameter to zero for the default behaviors.
* CLASS\_NAME is the name of the window class. This defines the type of window you are creating.
* The window text is used in different ways by different types of windows. If the window has a title bar, the text is displayed in the title bar.
* The window style is a set of flags that define some of the look and feel of a window. The constant WS\_OVERLAPPEDWINDOW is actually several flags combined with a bitwise **OR**. Together these flags give the window a title bar, a border, a system menu, and **Minimize** and **Maximize** buttons. This set of flags is the most common style for a top-level application window.
* For position and size, the constant CW\_USEDEFAULT means to use default values.
* The next parameter sets a parent window or owner window for the new window. Set the parent if you are creating a child window. For a top-level window, set this to NULL.
* For an application window, the next parameter defines the menu for the window. This example does not use a menu, so the value is NULL.
* *hInstance* is the instance handle, described previously. (See [WinMain: The Application Entry Point](http://msdn.microsoft.com/en-us/library/windows/desktop/ff381406(v=vs.85).aspx).)
* The last parameter is a pointer to arbitrary data of type **void\***. You can use this value to pass a data structure to your window procedure. We'll show one possible way to use this parameter in the section [Managing Application State](http://msdn.microsoft.com/en-us/library/windows/desktop/ff381400(v=vs.85).aspx).

[**CreateWindowEx**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms632680(v=vs.85).aspx) returns a handle to the new window, or zero if the function fails. To show the window—that is, make the window visible —pass the window handle to the [**ShowWindow**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms633548(v=vs.85).aspx) function.

The hwnd parameter is the window handle returned by [**CreateWindowEx**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms632680(v=vs.85).aspx). The nCmdShow parameter can be used to minimize or maximize a window. The operating system passes this value to the program through the **wWinMain** function.

A GUI application must respond to events from the user and from the operating system.

* **Events from the user** include all of the ways that someone can interact with your program: mouse clicks, key strokes, touch-screen gestures, and so forth.
* **Events from the operating system** include anything "outside" of the program that can affect how the program behaves. For example, the user might plug in a new hardware device, or Windows might enter a lower-power state (sleep or hibernate).

These events can occur at any time while the program is running, in almost any order. How do you structure a program whose flow of execution cannot be predicted in advance?

To solve this problem, Windows uses a message-passing model. The operating system communicates with your application window by passing messages to it. A message is simply a numeric code that designates a particular event.