**窗口特征**

[窗口类型 3](#_Toc415228996)

[重叠窗口 3](#_Toc415228997)

[弹出窗口 3](#_Toc415228998)

[子窗口 3](#_Toc415228999)

[分层窗口（透明窗口） 5](#_Toc415229000)

[信息窗口 5](#_Toc415229001)

[窗口关系 6](#_Toc415229002)

[前景和背景窗口 6](#_Toc415229003)

[奴隶窗口 7](#_Toc415229004)

[Z序 7](#_Toc415229005)

[窗口显示状态 7](#_Toc415229006)

[活动窗口 8](#_Toc415229007)

[禁用窗口 8](#_Toc415229008)

[界面窗口 8](#_Toc415229009)

[最小化、最大化和恢复窗口 9](#_Toc415229010)

[窗口大小和位置 10](#_Toc415229011)

[默认的大小和位置 10](#_Toc415229012)

[可调大小 10](#_Toc415229013)

[系统命令 10](#_Toc415229014)

[大小和位置函数 11](#_Toc415229015)

[大小和位置的消息 11](#_Toc415229016)

[窗口动画 12](#_Toc415229017)

[窗口布局和镜像 12](#_Toc415229018)

[镜像对话框和消息框 15](#_Toc415229019)

[镜像设备上下文不与窗口关联 15](#_Toc415229020)

[窗口的销毁 15](#_Toc415229021)

# 窗口类型

这部分包含下列主题，描述下列窗口类型。

## 重叠窗口

一个重叠窗口是一个顶级窗口的标题栏，边境和客户区；它可以作为应用程序的主窗口。它也可以有一个窗口菜单，最大化和最小化按钮以及滚动条。作为一个重叠窗口主窗口通常包括所有这些组件。

通过在CreateWindowEX函数中指定WS\_OVERLAPPED或WS\_OVERLAPPEDWINDOW风格，应用程序可以创建一个重叠窗口。如果使用WS\_OVERLAPPED风格，窗口有标题栏和边界，如果使用WS\_OVERLAPPEDWINDOW风格，窗口有标题栏，上边界，窗口菜单，最大化和最小化按钮。

## 弹出窗口

弹出窗口是一种特殊类型的重叠窗口，用于对话框，消息框和其他临时窗口出现的外部应用程序的主窗口。弹出窗口的标题栏是可选的；否则，弹出窗口拥有和重叠窗口一样的WS\_OVERLAPPED风格。

通过在CreateWindowEX函数中选择WS\_POPUP风格创建的弹出窗口，包含标题栏，指定的WS\_CAPTION风格。使用WS\_POPUPWINDOW风格创建的的弹出窗口，包含有一个边界和一个窗口菜单。WS\_CAPTION风格必须结合WS\_POPUPWINDOW风格才可以使窗口菜单可见。

## 子窗口

子窗口有WS\_CHILD风格仅限于它的父窗口的客户区。一个应用程序通常使用子窗口来使其客户区划分为多个功能区域。你可以通过在CreateWindowEX函数中指定WS\_CHILD风格来创建子窗口。

一个子窗口必须有一个父窗口，父窗口可以是重叠窗口、弹出窗口，甚至是另一个子窗口。当你调用CreateWindowEX函数时可以指定父窗口。如果在CreateWindowEX函数中选择了WS\_CHILD风格却没有指定父窗口，系统将不会创建这个子窗口。

子窗口有客户区但不包含其他功能，除非明确要求。应用程序可以为子窗口指定标题栏，窗口菜单，最大化和最小化按钮，滚动条，但子窗口不能有菜单。如果应用程序指定了一个窗口句柄，则当创建子窗口类或者创建子窗口时，窗口句柄会被忽略。如果没有指定边框样式，系统将创建一个无边界的窗口。应用程序可以使用无边界的子窗口来划分父窗口的客户区，同时可保证划分对于用户是隐藏的。

本节讨论如下：

位置：

系统总是将子窗口定位到与它关联的父窗口的客户区的左上角，不会有子窗口的一部分出现在其父窗口的边界外面。如果应用程序创建的子窗口大于父窗口或者定位子窗口的位置以至于其超出了父窗口的边界，系统将剪辑子窗口；也就是说，父窗口的客户区域外的部分不会显示。行为会影响父窗口也会影响子窗口,如下:

|  |  |
| --- | --- |
| 父窗口 | 子窗口 |
| 销毁 | 在父窗口被销毁之前销毁 |
| 隐藏 | 先于父窗口隐藏，子窗口可见仅限于父窗口可见 |
| 移动 | 在父窗口的客户区中移动，同时负责在移动后绘制父窗口的客户区 |
| 显示 | 在父窗口显示后显示 |

剪辑：

系统不会从父窗口的客户区自动剪辑子窗口，这意味着如果父窗口在子窗口所在的位置绘图，将会覆盖子窗口的相同区域。然而，如果父窗口是WS\_CLIPCHILDREN风格，系统将不会这么做。如果子窗口被剪辑了，父窗口不能再画它。

如果一个窗口有WS\_CLIPCHILDREN或者WS\_CLIPSIBLINGS风格，会有轻微的性能损失。每个窗口都占用系统资源，因此应用程序不能无限制的使用子窗口。为了最佳性能，需要在逻辑上划分其主窗口的应用程序应该在主窗口的窗口过程，而不是通过使用子窗口。

关系到父窗口:

应用程序可以改变现有的子窗口的父窗口通过调用SetParent函数。在这种情况下，系统将从旧的父窗口的客户区删除子窗口，同时将其移动到新的父窗口的客户区。如果SetParent函数指定一个空处理，桌面窗口将成为新的父窗口。在这种情况下，子窗口挂在桌面上，跳出所有的窗口边界外。GetParent函数用于检索子窗口的父窗口句柄。

父窗口放弃部分的子窗口客户区，子窗口接收所有的输入。窗口类不需要每个子窗口的父窗口相同，这意味着应用程序可以填补子窗口的父窗口，使其看起来不同，执行不同的任务。举个例子，一个对话框可以包含许多类型的控件，每个子窗口，接受来自用户的不同类型的数据。

子窗口只有一个父窗口，但父窗口可以有任意数量的子窗口。反过来，每个子窗口可以有子窗口。每个子窗口被称为后代原始父窗口的窗口。应用程序使用IsChild函数发现是否一个给定的窗口或者一个给定父窗口的子代窗口是否是一个子窗口。

EnumChildWindows函数列举一个父窗口的子窗口。然后EnumChildWindows函数将每个子窗口句柄传递给一个应用程序定义的回调函数。后代给定父窗口的窗口也列举。

信息：

系统直接将子窗口的输入数据传递给子窗口；不通过父窗口的消息。唯一的例外是如果EnableWindowfunction已经被禁用了。在这种情况下，系统传递给子窗口的任何消息，都会被子窗口的父窗口代替。这允许父窗口在必要情况下检查输入消息和启用子窗口。

子窗口可以有一个唯一的整数标识符，它在处理窗口控制时是很重要的。应用程序通过发送消息来指导控制活动。应用程序使用控件的子窗口标示符来进行消息控制。

此外，控件发送通知信息给它的父窗口。一个通知消息包含控件的子窗口标示符，父窗口用它来确定哪些控件发送了消息。应用程序指定其他类型的子窗口的子窗口标示符，可以通过在CreateWindowEX函数中设置hMenu成员来做而不是通过菜单句柄。

## 分层窗口（透明窗口）

对于窗口有复杂的形状、模拟其形状的、或希望使用alpha混合效果的，使用分层窗口可以显著提高性能效果和视觉效果。系统自动合成和重绘分层windows与windows底层应用程序。因此，分层窗口呈现平稳，没有复杂的窗口闪烁。此外，分层窗口可以部分半透明，alpha混合。

创建分层窗口，当调用CreateWindowEX函数时要指定WS\_EX\_LAYERED扩展窗口风格，或者在窗口被创建后调用SetWindowLong函数来设置WS\_EX\_LAYERED风格。当CreateWindowEX函数被调用后,分层窗口要等到SetLayeredWindowAttributes或者UpdateLayeredWindow 被调用才能可见。

注意从windows 8开始，WS\_EX\_LAYERED可以被用于子窗口和顶层窗口。之前的windows版本支持WS\_EX\_LAYERED风格的只有顶层窗口。

对于给定的水平分层窗口调用SetLayeredWindowAttributes函数可以设置不透明或者透明色键。调用后，系统仍然可以在窗口显示或者重定大小时要求窗口绘图。然而，由于系统存储分层窗口的图像，如果部分相对窗口的显示结果在桌面上调用，系统不会要求窗口画。遗留应用程序代码不需要重组其绘画。如果想为窗口添加透明或半透明效果，因为系统重定向绘画调用SetLayeredWindowAttributes到离屏内存和主旨进行重新编排，以达到期望的目的。

更快和更高效的动画或如果需要alpha,调用UpdateLayeredWindow.UpdateLayeredWindow函数被用于直接提供一个分层窗口的形状和内容，而不是用重定向机制调用SetLayeredWindowAttributes.此外，直接使用UpdateLayeredWindow能更有效的使用内存，因为所需的系统不需要额外的内存存储重定向的形象窗口。为了最大效率的动画窗口，可以调用UpdateLayeredWindow改变一个窗口的大小和位置。要注意在调用了SetLayeredWindowAttributes后,后续的UpdateLayeredWindow调用将失败，直到分层风格被清除并重新设置。

分层窗口的冲击测试是基于窗口的形状和透明度。这意味着窗口区域是color-keyed或者alpha值是空的会让鼠标消息通过。然而，如果分层窗口有WS\_EX\_TRANSPARENT扩展风格，这个分层窗口的形状将被忽略，鼠标事件将被传递到在其他窗口下面的分层窗口中。

## 信息窗口

一个信息窗口可以让你发送和接收消息。他是不可见的，没有Z值，无法枚举，不接受广播消息，只是简单的派送消息而已。

为了创建一个信息窗口，可以指定HWND\_MESSAGE常数或者在CreateWindowEX函数中的一个hWndParent成员的句柄。也可以通过在SetParent函数中为hWndNewParent成员指定HWND\_MESSAGE来更改信息窗口.

为了找到信息窗口，可以在FindWindowEX函数中为hwndParent成员指定HWND\_MESSAGE。此外，如果在FindWindowEX函数中hwndParent和hwndChild都为空则其会搜索信息窗口和顶级窗口。

# 窗口关系

有很多方法可以让一个窗口与用户或另一个窗口关联。一个窗口可能是奴隶窗口，前景窗口，或者后景窗口。一个窗口也会相对于其他窗口有Z序。有关更多信息，请参照下列主题：

## 前景和背景窗口

每一个进程可以有多个线程并发执行，每个线程都可以创建窗口，用户当前工作时创建的线程被称为前台线程，窗体被前台线程调用。其他线程都是后台线程，windows后台线程创建的窗体被称为背景窗口。

每个线程的优先级决定了每个线程得到CPU的时间，尽管应用程序可以设置线程的优先级，通常前台线程的优先级略高于后台线程。因为他有一个更高的优先级，前台线程接受CPU的时间比后台线程要多。前台线程有一个正常情况下的基本优先级9，后台线程有一个正常情况下的基本优先级7.

用户通过点击窗口来设置前台窗口，或者通过ALT\_TAB与ALT+ESC键来组合。使用GetForegroundWindow函数来检索前台窗口的句柄。可以通过比较GetForegroundWindow函数返回的句柄来确定窗口是否是前台窗口。

应用程序通过使用SetForeGroundWindow函数来设置前台窗口。

系统限制哪些进程可以设置前台窗口，一个进程能够设置前台窗口仅当下列情形之一为真：

进程是前台进程。

进程通过前台进程产生。

进程收到了最后一个输入事件。

没有前台进程。

前台进程正在调试。

前台没有锁。

前台锁超时。

没有菜单活动。

进程可以设置前台窗口，也可以由其他进程通过调用AllowSetForegroundWindow函数来做，或者用BSF\_ALLOWSFW标志来调用BroadcastSystemMessage函数，前台进程也可以通过调用LockSetForegroundWindow函数来禁用SetForegroundWindow函数。

## 奴隶窗口

一个重叠窗口或弹出窗口可以归属于其他重叠窗口或弹出窗口。但是归属在窗口中是有限制的。

在Z序中奴隶窗口总是高于它的主人窗口。

当主人窗口被摧毁时奴隶窗口也会被摧毁。

当主人窗口最小化时奴隶窗口也会被隐藏。

只有重叠或弹出窗口可以做主人窗口，子窗口不能做主人窗口。应用程序在用WS\_OVERLAPPED或者WS\_POPUP风格创建窗口时，在CreateWindowEX函数中为hwndParent成员指定主人窗口的句柄可以创建一个奴隶窗口。hwndParent成员必须是重叠窗口或者弹出窗口。如果hwndParent识别了一个子窗口，系统会将其所有权分配给子窗口的顶级窗口。在创建了奴隶窗口后，应用程序不会将其所有权过户给其他窗口。

对话框和消息框在默认情况下属于奴隶窗口。在调用函数创建对话框或者消息框时应用程序会为其指定主人窗口。

应用程序可以使用GetWindow函数中的GW\_OWER标志来获取主人窗口的句柄*。*

## Z序

窗口的Z序可以在重叠窗口栈中表明窗口的位置。这个窗口栈被调整为一个假想的轴，Z轴，从屏幕上向外延伸。窗口顶部的Z值重叠其他所有窗口，窗口底部的Z值被其他窗口重叠。

系统会将Z值维持在一张列表中，他可以基于他们是否是最顶级窗口，顶层窗口，子窗口来添加Z值。最顶层窗口重叠其他非最顶层窗口，不管他是活跃窗口还是前台窗口。最顶层窗口有WS\_EX\_TOPMOST风格。最顶层窗口在Z序中处于最前面的位置，子窗口被分组到它的父窗口的Z序中。

当应用程序创建了一个窗口，系统会把它放置到其他相同类型的窗口的前面。你可以使用BringWindowToTop函数来把一个窗口放置到Z序中与它类型相同的窗口的前面。你可以使用SetWindowPos与DeferWindowPos函数对Z序进行重排。

用户让不同窗口活动会改变Z序。系统会在Z序中将活动窗口定位到其他相同类型的窗口的前面。当一个窗口跑到Z序的前面的时候，它的子窗口也如此。你可以使用GetTopWindow函数来搜索一个父窗口的所有子窗口并返回Z值最高的子窗口的句柄。GetNextWindow函数可以检索下一个或者以前的一个窗口的句柄。

# 窗口显示状态

在任何给定的时间，一个窗口可以活跃或者非活跃，隐藏或者非隐藏，最小化，最大化或者恢复。这些特质被统称为窗口的显示状态。下列主题讨论窗口显示状态：

## 活动窗口

活动窗口是用户当前操作的应用程序的顶级窗口。允许用户随意指定活动窗口，系统会将其放置到Z序的前面并改变其标题栏的颜色，同时使其颜色与系统明确的活动窗口的颜色接壤。当用户正在子窗口上工作时，系统会激活与子窗口相关联的顶层父窗口。

同一时间系统中只有一个顶层窗口处于活动状态。用户激活一个顶层窗口可以通过点击，或者通过使用ALT+TAB或者ALT+ESC键组合。应用程序通过调用SetActiveWindow函数来激活一个顶层窗口。其它函数如SetWindowPos,DeferWindowPos,SetWindowPlacement,DestroyWindow函数可以使系统激活另一个不同的其他窗口。尽管一个应用程序可以在任何时刻激活不同的顶层窗口，为了避免用户混淆，应该只响应用户的动作。应用程序使用GetActiveWindow函数来获取一个活动窗口的句柄。

当激活动作从一个应用程序的顶层窗口到另一个应用程序的顶层窗口时，系统会发送WM\_ACTIVATEAPP消息给它两的应用程序，通知他们做出改变。当激活动作在一个应用程序的不同顶层窗口中改变时，系统会给这两个顶层窗口发送WM\_ACTIVATE消息。

## 禁用窗口

一个窗口可以被禁用。禁用窗口不从用户那里接受任何鼠标和键盘消息，但是它接受其他窗口的消息，其他应用程序的消息以及系统的消息。应用程序通常通过禁用窗口来防止用户使用窗口。举个例子，应用程序可能在对话框中通过禁用一个按钮防止用户选择它。应用程序可以在任何时间禁用窗口；也可以使一个窗口恢复正常输入。

默认情况下，窗口是可以被创建的。应用程序可以指定WS\_DISABLED风格来禁用一个创建的窗口。应用程序可以使用EnableWindow函数使一个窗口可用或不可用。当窗口的可用状态将要改变时系统会发送一个WM\_ENALBE消息给窗口。应用程序可以使用IsWindowEnable函数来判断一个窗口是否可用。

在子窗口被禁用后，系统会将其鼠标输入消息发送给其父窗口。父窗口据此来判断子窗口是否可用。有关更多信息，请看鼠标输入。

同一时刻只有一个窗口能接收键盘输入，这个窗口有键盘焦点。如果应用程序使用EnableWindow函数禁用窗口的键盘焦点，窗口会失去键盘焦点在这种情形下窗口是禁用的。禁用窗口然后设置键盘焦点为空，意味着窗口没有键盘焦点。如果子窗口或其他子代窗口有焦点，在父窗口被禁用后子代窗口会被禁用。有关更多信息，请参见键盘输入。

## 界面窗口

窗口可以是可见的也可以是隐藏的。系统会在屏幕上显示一个可见窗口。它通过不绘制来隐藏一个隐藏窗口。如果一个窗口是可见的，用户可以提供窗口输入并看到窗口的输出。如果窗口是隐藏的，他其实被禁用了。隐藏窗口可以从其他窗口或者系统加工信息，但不能处理用户的输入或者显示输出。应用程序可以在创建窗口时设置窗口的可见状态。最后，应用程序可以改变窗口的可见状态。

用WS\_VISABLE风格设置窗口时窗口是可见的。默认情况下，CreateWindowEX函数创建一个隐藏窗口，除非应用程序特别指定WS\_VISABLE风格。通常，应用程序会在窗口被创建后将其设置为可见的，来对用户隐藏窗口的创建细节。举个例子，应用程序可能会让窗口在其外观改变时隐藏。如果在CreateWindowEX函数中WS\_VISABLE风格被指定了，系统会在窗口被创建之后，显示之前发送WM\_SHOWWINDOW消息给窗口。

应用程序可以通过使用IsWindowVisable函数来决定窗口是否可见。应用程序也可以通过使用ShowWindow,SetWindowPos,DeferWindowPos,或者SetWindowPlacement,或者SetWindowLong函数来显示或者隐藏一个窗口。

当一个主人窗口被最小化后，系统会自动隐藏与他关联的奴隶窗口。同样的，当一个主人窗口恢复后，系统会自动显示与它关联的奴隶窗口。在这两种情况下，系统会自动发送WM\_SHOWWINDOW消息给奴隶窗口在他们被隐藏或者显示之前。偶尔，应用程序可能会需要在没有最小化或者隐藏主人窗口时隐藏奴隶窗口。在这种情况下，应用程序会使用ShowOwnedPopups函数.这个函数会对所有奴隶窗口设置或者隐藏WS\_VISABLE风格，同时，发送WM\_SHOWWINDOW消息给奴隶窗口在他们显示或者隐藏之前。隐藏一个主人窗口不会对奴隶窗口有任何影响。

父窗口可见，与它关联的子窗口也可见。类似的，父窗口隐藏，子窗口也会隐藏。最小化父窗口不会影响子窗口的可见性，子窗口最小化也会影响父窗口，但是WS\_VISABLE风格不会改变。

即使一个窗口有WS\_VISABLE风格，用户也不一定能在屏幕上看见它；其他窗口可能完全覆盖它或者它已经超过了屏幕的边界。此外，可见窗口受支配于与它相关的父窗口的剪裁规则。如果父窗口不可见，则它也不可见。如果父窗口移动到屏幕的边缘，子窗口也随之移动，因为子窗口受制于父窗口的左上角。举个例子，用户可能在移动父窗口时包含子窗口到了屏幕边缘足够远，可能会无法看到子窗口，即使子窗口与父窗口具有WS\_VISABLE风格。

## 最小化、最大化和恢复窗口

最大化窗口是具有WS\_MAXIMAZE风格的窗口。默认情况下，系统放大最大化窗口，他将充满屏幕区，或者在有子窗口的情况下他将充满父窗口。尽管一个窗口可以被设置为与最大化窗口的尺寸相同，但还是会有些许不同。系统自动将窗口的标题栏移动到屏幕的顶部或者父窗口的客户区顶端。同时，系统会禁用窗口的边界以及窗口的标题栏的功能。

最小化窗口是具有WS\_MINIMAZE风格的窗口。默认情况下系统会减少最小化窗口的任务栏按钮，同时移动最小化窗口到任务栏。一个恢复的窗口会恢复到原来的尺寸和位置，在他最小化或者最大化之前。

如果应用程序指定WS\_MAXIMAZE或者WS\_MINIMAZE风格在CreateWindowEX函数中，窗口会先最大化或最小化。创建窗口后，应用程序可以使用CloseWindow函数来最小化窗口。ArrangeIconicWindows函数可以为桌面安排图标，或者为一个父窗口安排一个最小化子窗口。OpenIcon函数将一个最小化窗口恢复到它原来的尺寸和位置。

ShowWindow函数可以最大化，最小化，恢复一个窗口。它也可以设置窗口的可见性和激活状态。SetWindowPlacement函数包含相同的功能，但它可以将一个窗口默认的最大化，最小化，恢复的位置推翻。

IsZoomed和ISIconic函数分别决定是否给窗口最大化和最小化。GetWindowPlacement函数检索一个窗口的最大化，最小和和恢复的位置，也决定窗口的显示状态。

当系统接收到最大化命令或者恢复最小化窗口的时候，他会给窗口发送WM\_QUERYOPEN消息。如果窗口过程返回FALSE，系统会忽视最大化或者恢复命令。

系统自动设置一个最大化窗口的大小和位置为系统定义的默认值。要覆盖这些默认值，应用程序也可以调用SetWindowPlacement函数或者通过处理窗口将要最大化时接收到的WM\_GETMINMAXINFO消息来做。WM\_GETMINMAXINFO函数含有一个指向MINMAXINFO结构体的指针，这个结构体有系统用来设置最大化尺寸和位置的数值。替换这些值可以覆盖默认值。

# 窗口大小和位置

一个窗口的大小和位置表示为一个矩形边界，给出了其相对于屏幕或者父窗口的边界。顶级窗口的坐标是相对于屏幕的左上角，子窗口的坐标是相对于父窗口的左上角。应用程序在创建窗口时指定窗口的初始大小和位置，但它可以在任何时间改变窗口的大小和位置。有关更多信息，请参见形状。

本节包含以下主题：

## 默认的大小和位置

应用程序通过在CreateWindowEX函数中设置CW\_USEDEFAULT可以允许系统计算顶级窗口的初始大小和位置。如果应用程序的坐标设置为CW\_SUEDEFAULT，并且没有创建其他顶级窗口，系统可以相对于屏幕左上角设置新的窗口位置；否则，位置是相对于应用程序最新创建的顶级窗口。如果宽和高设置为CW\_USEDEFAULT，系统会计算新窗口的大小。如果应用程序创建其他顶级窗口，系统设置大小会基于最新创建的顶级窗口。在创建子窗口或者弹出窗口时指定CW\_USEDEFAULT，系统会根据默认的最小化窗口尺寸设置窗口尺寸。

## 可调大小

系统会为WS\_THICKFRAME风格的窗口维护一个最大化和最小化栈；窗口具有这种风格将有标准边界。最小跟踪尺寸是你可以通过拖动窗口边界的产生的最小窗口。同样的，最大跟踪尺寸是你可以通过拖动窗口边界的产生的最大窗口。

系统的最小化和最大化跟踪尺寸被设置为系统创建窗口时定义的默认值。应用程序可以发现默认值和改变它通过发送WM\_GETMINMANINFO消息。有关更多信息，请参见大小和位置信息。

## 系统命令

有窗口菜单的应用程序可以通过发送系统命令改变窗口的大小和位置。当用户从窗口菜单中选择命令会使系统命令生成。应用程序可以通过发送WM\_SYSCOMMAND消息来模拟用户操作。下面的系统命令会影响一个窗口的大小和位置。

|  |  |
| --- | --- |
| 命令 | 描述 |
| SC\_CLOSE | 关系窗口，这个命令将WM\_CLOSE发送给窗口，窗口执行任何步骤需要先清理和销毁自己 |
| SC\_MAXIMIZE | 最大化窗口 |
| SC\_MINIMIZE | 最小化窗口 |
| SC\_MOVE | 移动窗口 |
| SC\_RESTORE | 一个最大化，最小化窗口恢复到以前的位置和大小 |
| SC\_SIZE | 开始一个大小命令，使用鼠标或键盘来改变窗口的大小 |

## 大小和位置函数

创建一个窗口后，应用程序可以通过调用不同的函数来改变窗口的大小和位置，包括SetWindowPlacement,MoveWindow,SetWindowPos和DeferWindowPos函数。SetWindowPlacement函数可以设置窗口的最大最小尺寸，恢复大小和位置，并显示状态。MoveWindow和SetWindowPos函数类似，可以设置一个应用程序窗口的大小和位置。SetWindowPos函数包括一些标志，影响窗口的显示状态；MoveWindow不包含这些标志。使用BeginDeferWindowPos,DeferWindowPos和EndDeferWindowPos函数可以设置一组窗口的位置，包括大小，位置，在Z序中的位置和显示状态。

应用程序可以通过GetWindowRect函数来获取窗口边框的坐标，GetWindowRect函数用窗口左上角和右下角的坐标填补矩形结构。坐标是相对于屏幕左上角的，即使是子窗口。ScreenToClient函数与MapWindowPoints函数映射子窗口相对于父窗口的屏幕边框坐标。

GetClientRect函数检索窗口客户区的坐标，GetClientRect函数用客户区的左上角和右下角填充矩形结构，但坐标是相对于客户区本身的。这意味着左上角的坐标总是（0,0），右下角的坐标总是客户区的宽和高。

CascadeWindows函数可以使窗口在桌面上瀑布似的落下或者使子窗口在父窗口中瀑布似的落下。TileWindows函数可以在桌面上铺设窗口或者使子窗口在指定的父窗口中。

## 大小和位置的消息

系统会发送WM\_GetMinMaxInfo函数给大小和位置将要改变的窗口。举个例子，当用户点击鼠标移动或者改变窗口菜单，单击改变窗口边界或者标题栏时消息会被发送；当应用程序调用SetWindowPos移动或者改变窗口时也会发送消息。WM\_GETMINMAXINFO包含一个指向MINMAXINFO结构体的指针，这个结构体包含窗口默认的最大尺寸和位置，以及默认的最小和最大跟踪尺寸。应用程序可以通过处理WM\_GETMINMAXINFO或者给MINMAXINFO设置合适的成员来覆盖默认值。一个窗口在收到WM\_MINMAXINFO消息时必须有WS\_THICKFRAME或者WS\_CAPTION风格。在窗口创建过程中或者当它被移动，被改变尺寸时，拥有WS\_THICKFRAME风格的窗口都会收到这个消息。

在窗口的大小，位置，Z序中的位置或者显示状态将要发生改变时，系统会发送WM\_WINDOWPOSCHANGING消息给窗口。这个消息包含一个指向WINDOWPOS结构体的指针，此结构含有窗口新的尺寸，位置，在Z序中的位置和显示状态。通过设置WINDOWPOS的成员，应用程序可以影响窗口的新的尺寸，位置和外观。

当窗口的大小，位置，Z序中的位置或者显示状态发生改变后，系统会发送WM\_WINDOWPOSCHANGED消息给窗口。这个消息包含一个指向WINDOWPOS结构的指针，它可以通知窗口新的大小，位置，Z序中的位置和显示状态。通过WM\_POSCHANGED消息设置WINDOWPOS的成员对窗口没有影响。窗口必须处理WM\_SIZE和WM\_MOVE消息在将WM\_WINDOWPOSCHAGED传递给DeferWindowProc结构体时。否则，系统不会发送WM\_SIZE和WM\_MOVE消息给窗口。

当窗口被创建或者改变的时候系统会发送WM\_NCCALCSIZE消息给窗口。系统通过消息来计算窗口客户区相对于窗口左上角的大小和位置。窗口通常将这个消息发送给默认的窗口过程，然而，此消息有可能对应用程序有用，应用程序可以在窗口的大小改变时定制客户区或者保存客户区的大小。有关更多信息，请参见绘画。

# 窗口动画

在通过使用AnimateWindow消息显示或者隐藏窗口时可以产生特别效果。当窗口在进行滚动，滑动或者淡入淡出这种效果的时候，这全是建立在你调用AnimateWindow消息时指定了标志。

默认情况下，系统使用动画。也就是说，窗口会表现出滚动关闭或者打开。你可以通过使用dw标志来指定窗口是否进行水平，垂直或者斜角滚动。

当你指定AW\_SLIDE标志时，系统会使用滑动特效。也就是说，窗口会划入视野或者滑出视野。你可以使用dw标志来指定窗口水平，垂直，斜角滑动。

当你指定AW\_BLEND标志时，系统会使用alpha混合褪色。

你也可以使用AW\_CENTER标志来让窗口表现出向内崩溃或者向外膨胀。

# 窗口布局和镜像

窗口布局定义了文本和Windows图形设备接口在窗口或者设备上下文中的安排。一些语言，例如英语，法语和德语，要求从左到右布局。其他语言，例如阿拉伯语和希伯来语，要求从右到左布局。窗口布局适用于文本，也会影响窗口的其他GDI元素，包括位图，图标，原点的位置，按钮，级联树控件，以及水平坐标的左右移动。例如，在应用程序中设置了RTL布局后，原点定义在窗口或者设备的右边缘。然而，并非所有的对象会被窗口布局影响。举个例子，对话框，消息框和设备上下文不与窗口关联，比如图元文件和打印机，必须单独处理。随后的专题是这个问题的详细解释。

在阿拉伯或者希伯来版本的窗口中窗口函数允许你指定或者改变窗口布局。注意改变RTL布局不支持拥有CS\_OWNDC风格的窗口或者具有GM\_ADVANCED模式的DC。

默认情况下，窗口布局从左到右。设置为RTL窗口布局的，要使用WS\_EX\_LAYOUTRTL风格来调用CreateWindowEx函数。同样的子窗口也具有与其父窗口类似的布局。通过在CreateWindowEx函数中使用WS\_EX\_NOINHERITLAYOUT可以禁止所有子窗口继承镜像。注意镜像不能被奴隶窗口继承或者在CreateWindowEX函数中的父窗口hWnd成员被设置为NULL的窗口继承。禁止个体窗口继承镜像，通过向GetWindowLong函数或者SetWindowLong函数处理WM\_NCCREATE关闭WS\_EX\_LAYOUTRTL标志。这个处理是除了其他处理之外必须进行的。下面的代码片段显示了这是如何实现的。

|  |
| --- |
| SetWindowLong(hWnd,  GWL\_EXSTYLE,  GetWindowLong(hWnd,GWL\_EXSTYLE)&~WS\_EX\_LAYOUTRTL)) |

你可以通过调用SetProcessDefaultLayout函数设置默认布局。在这之后创建的所有窗口在被调用时将会调用他们的镜像，但是现存的窗口不受影响。关闭镜像，调用SetProcessDefaultLayout(0)函数。

注意，SetProcessDefaultLayout只反映了DCs镜像的窗口。调用SetLayout可以映射任何设备。有关更多信息，请参见讨论镜像设备上下文与与窗口这个话题。

默认情况下位图和图标 也会在窗口中被反射。然而，也不是所有的都这样。例如，那些文本，商业标志和模拟时钟不会被反射。调用SetLayout函数并将其dwLayout参数设置为

LAYOUT\_BITMAPORIENTATIONPRESERVED可以禁用位图的镜像。调用SetLayout(hdc,0)可以关闭设备环境镜像。

查询当前的默认布局，调用GetProcessDefaultLayout函数。如果成功会返回，pdwDefaultLayout会包含LAYOUTRTL或者0。调用GetLayout可以查看设备上下文的布局，如果成功返回，GetLayout会返回一个DWORD,则表明布局方式是通过LAYOUTRTL或者LAYOUT\_BITMAPORIENTATIONPRESERVED设置的。

创建一个窗口后，你可以通过SetWindowLong函数改变窗口布局。举个例子，当一个阿拉伯或者希伯来用户改变一个现有用户窗口的布局时这是必要的。然而，当改变现有的窗口的布局时，你必须使窗口无效然后刷新现有窗口以确保窗口的所有内容都以相同的方式布局。下面的代码实例，是需要改变窗口布局的：

|  |
| --- |
| //使用ANSI版本的GetWindowLong函数和SetWindowLong由于Unicode  //这些调用不是必须的  lExStyles = GetWindowLongA(hWnd, GWL\_EXSTYLE);  // 检查新的布局是否和当前布局相反  if (!!(pLState -> IsRTLLayout) != !!(lExStyles & WS\_EX\_LAYOUTRTL))  {  // 下面的线将更新窗口布局  lExStyles ^= WS\_EX\_LAYOUTRTL; // 切换布局  SetWindowLongA(hWnd, GWL\_EXSTYLE, lExStyles);  InvalidateRect(hWnd, NULL, TRUE); // 更新客户区布局  } |

在镜像中你需要考虑近和远而不是左和右。如果不这样做，就会出现一些问题。当一个镜像区窗口在客户区坐标和屏幕坐标发生绘图时将会产生错误，这是一个常见的编码实例。举个例子，应用程序使用下面的代码控制窗口位置：

|  |
| --- |
| // 如果应用程序映射窗口不要使用这个  // 得到的窗口坐标在屏幕坐标中  GetWindowRect(hControl, (LPRECT) &rControlRect);  //屏幕坐标映射到窗口客户区坐标  ScreenToClient(hDialog, (LPPOINT) &rControlRect.left);  ScreenToClient(hDialog, (LPPOINT) &rControlRect.right); |

这个导致问题的原因是矩形的左边缘变成了窗口的右边缘，反之亦然。为了避免这个问题，用调用MapWindowPoints代替ScreentoCliet调用：

|  |
| --- |
| // 使用这个镜像  GetWindowRect(hControl, (LPRECT) &rControlRect);  MapWindowPoints(NULL, hDialog, (LPPOINT) &rControlRect, 2) |

这段代码正常是因为平台支持镜像，当窗口客户区被映射时，MapWindowPoints函数会将坐标的左右交换。有关更多信息，请参见MapWindowPoints的附录信息：

另一个常见的做法会导致类似的问题，是因为将导致问题的镜像定位到屏幕坐标而不是使用窗口客户的坐标。举个例子，下面的代码在客户端坐标使用屏幕坐标的差异作为X坐标。

|  |
| --- |
| // 如果LTR布局和客户区映射模式是MM\_TEXT的  // 但对于映射对话框是错误的  RECT rdDialog;  RECT rcControl;  HWND hControl = GetDlgItem(hDlg, IDD\_CONTROL);  GetWindowRect(hDlg, &rcDialog); // 得到矩形屏幕坐标  GetWindowRect(hControl, &rcControl);  MoveWindow(hControl,  rcControl.left - rcDialog.left, // 在客户区坐标中使用X位置  rcControl.top - rcDialog.top,  nWidth,  nHeight,  FALSE); |

这段代码很好当对话框有从左到右（LTR）布局并且客户区有MM\_TEXT模式时，因为客户区坐标系中X的新位置与在屏幕坐标系中被控制的对话框的左边缘不同。然而，在映射对话框中，左和右是相反的，所以你应该使用如下的MapWindowPoints:

|  |
| --- |
| RECT rcDialog;  RECT rcControl;  HWND hControl - GetDlgItem(hDlg, IDD\_CONTROL);  GetWindowRect(hControl, &rcControl);  // MapWindowPoints可以在映射和无映射窗口中正常使用  MapWindowPoints(NULL, hDlg, (LPPOINT) &rcControl, 2);  // 现在rcControl在客户区坐标中  MoveWindow(hControl, rcControl.left, rcControl.top, nWidth, nHeight, FALSE) |

## 镜像对话框和消息框

对话框和消息框不继承布局，因此必须显式的设置布局。对一个消息框，在MB\_RTLREADING选项中调用MessageBox或者MessageBoxEx.为了将一个对话框布置为从右到左，可以在对话框模板结构中使用扩展风格WS\_EX\_LAYOUTRTL。属性表是对话框的一个特例。每个选项卡都被视为一个单独的对话框，所以你需要在每个你想要映射的选项卡中设置WS\_EX\_LAYOUTRTL风格。

## 镜像设备上下文不与窗口关联

DCs不与窗口关联，如图元文件或者打印机DCs，不要继承布局，所以你必须明确的设置布局。为了改变设备上下文布局，必须使用SetLayout函数。

SetLayout函数很少用于窗口。通常，窗口只有在处理WM\_PAINT消息时才会接收与之关联的DC。有时程序通过GetDC来为一个窗口创建DC。无论哪种方式，DC最初的布局都是通过BeginPaint或者按照窗口的WS\_EX\_LAYOUTRTL标志GetDC.

GetWindowOrgEx、GetWindowExtEx、GetViewportOrgEx、GetViewportExtEx函数的返回值不会被SetLayout函数影响。

当布局是RTL时，GetMapMode将用MM\_ANISOTROPIC代替MM\_TEXT。用MM\_TEXT调用SetMapMode可以正确的运行。只有GetMapMode的返回值会被影响。同样的，当映射模式是MM\_TEXT时调用SetLayout会引起映射模式转为MM\_ANISOTROPIC.

# 窗口的销毁

一般来说，应用程序必须摧毁所有的窗口，通过调用DestroyWindow函数。当窗口被销毁时，系统会隐藏窗口，如果它是可见的，然后删除任何与窗口相关联的数据。这个窗口句柄将无效，不能再被应用程序使用。

应用程序会在窗口创建之后的销毁很多不必要的窗口。举个例子，应用程序通常会销毁一个对话框窗口，只要应用程序有足够的来自用户的输入可以工作。应用程序最终会销毁主窗口。

销毁一个窗口之前，应用程序应该保存或者删除任何与窗口相关的数据。而且它应该释放任何与窗口相关的资源。如果应用程序没有释放资源，系统将做这件事。

销毁一个窗口不会影响创建这个窗口的窗口类。新的窗口仍然可以使用这个类创建，并和现在已有的窗口类一起运作。销毁一个窗口也会销毁它的后代窗口。DestroyWindow函数会发送一个WM\_DESTROY消息给窗口，然后是子窗口，自带窗口。所有窗口的子代窗口也会被销毁。

当用户点击了关闭按钮后，拥有菜单的窗口会收到一个WM\_CLOSE消息。通过处理这个消息，应用程序可以在窗口销毁之前提示用户。如果用户证实这个窗口要被销毁，应用程序可以调用DestroyWindow函数销毁窗口。

如果被销毁的窗口是活动窗口，活跃和焦点都会被转移到另一个窗口。这个窗口将成为下一个活动窗口，也可以用ALT+ESC函数来组合决定。新的活动窗口然后决定哪个窗口接收键盘焦点。