引 言

本书主要描述 Windows 资源——初级图形资源、位图、图符和光标。书中对如何创建和加工这些图象及如何在用户程序中使用这些图象进行了一定深度的描述。读者只需具备一定 Windows 编程的基本知识而无需是一个图形编程的好手，便可掌握书中的大部分内容。书中用到的多数材料都是作者通过艰苦编程和历经痛苦的失败而获得的。

作者将 Windows 图形编程比作在一间摆满看不见的家俱的房间中蹒跚行走。第一次走进这个房间，身体就会重重地碰在一张看不见的咖啡桌上。但不久之后，便会发现这些无形的家俱在什么地方，碰伤的次数也会越来越少。为使其他程序员能够顺利地编程，作者尽可能地将 Windows 图形编程中会遇到的细节问题编写成书。这就好象在这些看不见的家俱上喷漆，或至少可比成在上面放置了一层缓冲垫。

书中的例子

本书包含了大量的实例程序。对程序员来讲，实例程序通常是最好的学习方法；即使程序员没有完全读懂某段程序，也能从该程序中获得一些有用的东西，并根据自己的需要对程序进行修改。在修改程序的过程中，程序员会懂得许多复杂的事情。这并不意味着本书中没有很多细微的东西，恰恰相反，这正说明大量复杂的对象是建立在简单的对象基础之上的。

书中的例子都是用 C 和 C++语言编写的。作者之所以使用一部分 C++代码，不是编程必须要用 C++语言，而是由于用 C++语言编程要简单些。在 Windows 环境下，用 C++建立图形对象会给程序员省去许多麻烦。Windows 下与加工图形有关的操作多数都是建立和清理，程序员在程序开始时建立对象，在程序结束时清理对象(最好这样)。而这正是 C++所擅长的，构造函数和析构函数显然能实现这些功能。(作者在此多提一句：若读者没学过 C++语言，那么现在就是开始学习的好机会。C++不仅能增强程序员的编程能力，而且能使程序员增加一种很好的 Windows 编程手段。)

读者如果是一个熟练的 C++程序员，就会发现作者并不是用 C++编制每一个图形显示接口(GDI)函数，而是在书中尽力建立一些高层次的类(如可兼容的 DC 类)。这些类将会为程序员免去许多 Windows 图形编程的麻烦。若读者决定采用其中的某些对象类时，应该意识到在书中后面部分出现的对象类是建立在前面的对象类基础之上的。尽管如此，这些对象类是相对独立的，它们不依赖于任何第三部分对象库，而只需任何版本的 C++编译器便可使用。

本书中的 C 语言范例与 Microsoft C/C++ 7.0 版本的编译器兼容；C++语言范例与 Borland C++ 3.1 版本的编译器兼容。书中的资源范例即可用 Microsoft 资源编译器(rc.exe)编译，也可用 Borland 公司的 Resource Workshop 编译。

本书中的许多 C 语言例子是用 Blue Sky Software 中的 WindowsMaker Professional 创建的。用这种方法对原型应用程序很有帮助，同时也能很快地建立应用程序的外壳。

第 1 章

Windows 图形资源：综述

尽管最近几个月出版了大量有关 Windows 编程的书籍，但这些书几乎都未涉及 Windows 编程的一个重要领域：管理图形资源。本书介绍的内容将一改上面所说的那种情况。读完本书，读者将理解什么是 Windows 图形资源，如何创建它们，如何在用户应用程序中包含和管理图形资源。程序员将能编写更完美的程序，同时，使程序具有更直接的用户界面。

图形资源为什么如此重要？很简单，图形资源是图形用户接口(GUI)中的图形。用户通过显示可加工的图形图象，能马上知道程序在做些什么。这样，就可使用户致力于自己想要完成的任务而不是要完成该任务需要做些什么。为用户提供图形信息有许多不同的办法，本书只介绍最常用的几种方法。

Windows 编程中有一种普通的形象化比喻，即图形按钮。用户单击带有图象的按钮之后，便知道程序将响应一个与图象有关的动作。例如，用户知道单击带有一个指向磁盘的箭头的按钮后程序将会把当前文件存盘。

尽管图形按钮用起来很方便，但编制一个图形按钮需要做大量工作。第 7 章“位图按钮”将介绍如何编制支持容易制作的位图(图形)按钮的应用程序。

本书还介绍了一种对创建图形资源十分有用的东西，ICE/Works。它是市场上出售的产品 ICE/Works Processional 的缩版。这种有用的方法允许程序员加工图形图象。本书将在附录 A 中对其作详细介绍

1.1 位图

Windows 环境下，位图可以说是所有图形资源的心脏。位图之所以称为位图是因为它将计算机的内存片断(位)映射成为一幅图形图象。位图最简单的形式是将字节中的每一位映射成位图中的一个象素。由于位只能有两个值 0 和 1，因此，与之对应的象素也只能有两个状态 on 和 off。对一幅理想的位图来说，一个 1 就是一个 on-象素(黑)，一个 0 就是一个 off-象素(白)。见图 1.1。

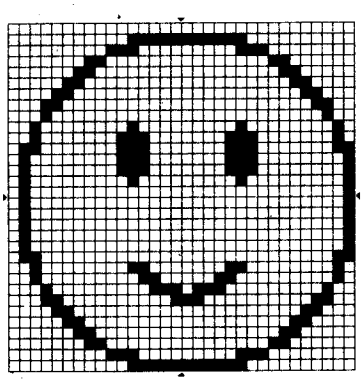

图 1.1 一幅基本的黑白位图

注意，用一位表示一个象素，那程序员所能显示的颜色数目就太少了。若用两位表示一个象素，那么程序员在颜色的选择上就有了一定的灵活性——使用两位象素，象素就可用 4 个值来代表(0,1,2,3)，这样，程序员就会有比一位象素多一倍的颜色供其选择。若用 3 位象素，可选择的颜色又多了一倍，从

4 变成了 8。若用 4 位象素，则程序员就有 16 种颜色供其选择。见图 1.2。

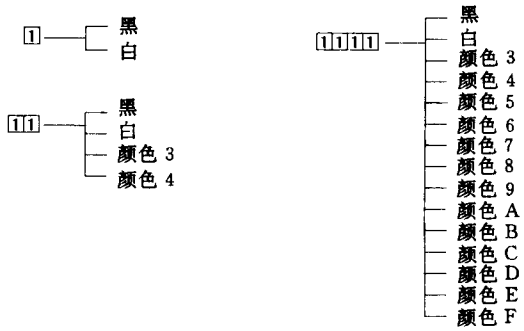

图 1.2　采用 1,2,3,4 位象素可得到的颜色数

代表一个象素的值每增加一位，颜色数目就增加一倍。每个象素占 8 位(1 字节)，就可有 256 种颜色。若每个象素占 16 位(2 字节)，就可有 65 536 种颜色，若每个象素占 24 位(当前 Windows 下最大值)，就可有 1600 万种颜色(精确地讲是 16,777,216)！

1.2　管理颜色

早期的计算机内存较小，而且是珍贵的资源。因此，用大量的内存去显示图象是不现实的。常常用 2 或 4 位颜色显示图象。现在，显示卡通常都有半兆字节(524,288 字节)或 1 兆字节(1,048,576 字节)的显示内存。显存容量的增长意味着现在同时能在屏幕上显示 16,256 或 32,768 种颜色。可显示颜色数目的巨大增长给程序员带来了另一个问题：管理所有这些颜色。

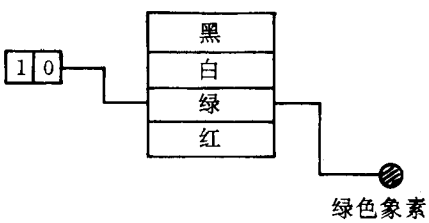

图 1.3　直接颜色映射。2 通常映射为绿色象素

若程序员只采用 4 种颜色，则管理颜色并不成问题，确实，也没什么可管理的。这种情况下，颜色与值是直接对应的，若程序员要用到某种颜色，则直接使用对应的数值即可。颜色与数值的这种直接对应关系。称为直接颜色映射。见图 1.3。

管理 256 种颜色与管理 4 种颜色就不同了，这是由于应用程序不会使用这么多种颜色。也不可能使用一个明显的颜色子集。在一个直接颜色映射分类表中，不论程序员是否要用到某些颜色。都需要知道所有颜色的映射关系。程序员所需的特殊颜色可能分散在整个颜色值范围内；程序员可能需要 2 种小于 5 的颜色，几种 47 周围的颜色和一些接近 200 的颜色。

如果程序员能将所感兴趣的颜色都安排在某一相同的范围内，最理想是从 0 开始线性增大，那么问题就简单多了。但采用直接颜色映射，这是不可能的。然而若采用间接颜色映射，程序员便可精确地作到这一点。

间接颜色映射绕了一个圈子确定颜色。它不是(象直接颜色映射那样)直接确定颜色，而是由应用程序确定代表所需颜色的某一位置。用户应用程序要改变所用的颜色，只需改变代表该颜色的位置中的内容，从而也就改变了所有地方的颜色。

为了更清楚地说明颜色映射，读者不妨想象有一所带有许多邮政信箱的邮局。读者拿着 700 信箱的钥匙去取邮件。读者打开 700 信箱，里面便有一包邮件。这一过程便等效于直接颜色映射。但如果是间接颜色映射，读者在 700 信箱中找不到邮件，而只能找到一把别的信箱的钥匙。拿着这把新钥匙便可取到邮件。

这一过程看上去似乎并不复杂；读者拿着第一把钥匙便能拿到可取到邮件的第二把钥匙。但是，很明显，读者在打开 700 信箱之前，并不知道邮件到底放在哪个信箱里。(读者只知道 700 信箱中没有邮件。)信箱中可能是任何一个信箱的钥匙，也可能什么也没放。有一点非常重要，那就是改变 700 信箱中存放的钥匙，就可改变能取到邮件的信箱。700 信箱实际上就是一个指向邮件的指针。

与此相似，间接颜色映射过程中，程序员不指定颜色，而指定指向颜色的指针。程序员通过改变指针，便可在任何时刻改变应用程序所用的颜色。程序员还可根据需要切换几组不同的指针。通常，一组指向颜色的指针称作调色板。Windows 环境下所说的调色板管理器就是指处理调色板和映射颜色的程序。

使用间接颜色映射和调色板的第二个优点就是可使程序具备一定程度的设备无关性。在调色板管理器出现之前，程序员不得不去了解每种不同显示方式下颜色映射的方法以及显示屏幕的解像度(象素总数和颜色总数)。若使用 Windows 调色板管理器，程序员只需简单指出应用程序所需的颜色数，调色板管理器句柄便知道将程序员所需的颜色映射到运行应用程序的计算机硬件上。

1.3 Windows 位图

前面提到过，位图是 Windows 图形资源的核心。根据特性将位图分成：

- 设备相关位图(DDB)
- 设备无关位图(DIB)
- OS2/PM 位图

设备相关位图也就是所说的老式位图，这种位图之所以叫老式位图是由于 Windows 1.0 以后一直延用这种位图。正象它的名称所暗示的那样，它是设备相关的，也就是说位图中位的组织和格式是不能预先确定的。以后，读者将见到这种位图有几类重要的分支。

随着 Windows 3.0 的出现，新出现了设备无关的位图。与设备相关位图相比，这种位图提供一种与显示设备无关的交换图象的方法。

尽管 OS2/Presentation Manager 位图中含有的格式信息少于设备无关的位图，但也可将 OS2/PM 位图看成是由 DIB 位图演变来的。

通常，用户在 Windows 下进行图形操作主要是用第 1 类位图，即设备相关位图。这就是说，比如用户要将一个圆画入设备描述表(第 2 章讨论)，实际上用户是画了一个 DDB。

DIB 最早用于与文件有关的操作。写好一个 DIB 之后，由于 DIB 中含有全部重构位

图所需的信息，因此，所有需要读位图的应用程序都能使用该位图。DIB 在许多时候都很有用，本书对 DIB 将在第 4 章“设备无关位图”中作详细介绍。

OS2/PM 位图用于 OS2/Presentation Manager。由于本书主要介绍的是 Windows，因此，书中除了介绍一下这种位图的读写方法外，略去了其它大部分内容。

还有一点前面尚未提到，即所有位图都是线性的。这就是说位图不是方形的就是矩形的。用户在许多时候要用到非矩形图象，这种图象是 Windwos 的另一种资源，即图符。

1.4 图符

首先，Windows 图符是一种位图的变更形式。一个位图(DIB 或 DDB)中只能含有一幅图象，但图符中却能含有几幅图象。其次，图符中的每一幅图象不是由一组位，而是由两组位构成。第一组位定义的是实际图象(与 DIB 相同)。图符与位图的区别主要在第二组位。图符的第二组位是一个单色位图，它是为上面所说的彩色位图定义的，可以算是彩色位图的透明度。(本书将在第 5 章“图符资源”中介绍透明度实际构成的定义方法。)这就意味着位图无需再是线性的。用第二种图象(通常称之为掩模图象或掩模)，程序员便可定义不规则形状的位图及位图以何种方式掩盖背景。如，程序员要在一个桌面上画一个图符，背景掩模就定义了透过图符桌面看上去是什么样子。

通常用图符表示系统中的程序、文件或折纸机。若用户要从程序管理器或第三个用户桌面上运行一个程序，用户可双击该程序的图符。在编译时，这些图符联接到应用程序中。本书后面将讨论这一点。若没有与应用程序联接在一起的图符，Windows 将使用缺省图符(若存在缺省图符)。

用户可用图符代表自己的应用程序，也可用它做其它事情。本书将在第 5 章中详细讨论图符及图符是如何工作的。

1.5 光标

光标是第三类 Windows 图形资源。光标是与用户系统中的鼠标紧密联系的图象。用户移动鼠标时，光标便在屏幕上移动。

用户调用 LoadCursor()函数便可使用 Windows 提供的光标。另外，用户可以定义自己的光标，并用该函数装入和显示。光标是一种有特殊用途的对象，用户只能用光标代表屏幕上的用户鼠标。尽管光标的使用范围非常有限，但还是可以用光标作一些非常有趣的事情，这一点本书将在第 6 章“光标资源”中讨论。

位图、图符和光标是 Windows 图形资源的三种类型。本书在以后几章讨论这些资源的细节时，将给出许多源代码范例。

本书最后介绍的是 ICE/Works，一个 Windows 图形资源编辑器。这一工具使用户能够创建书中例子所用到的图象。当然，这一图形资源编辑器的功能有限。从市场上购买的图形资源编辑器能够读/写图符、光标和位图，以及. EXE、. DLL 和. DRV 文件。而 ICE/Works 虽然可以读上述所有文件，但只能写图符文件。

附录 A 简单介绍了 ICE/Works。尽管如此，最好的学习过程就是使用它的过程。

第 2 章

显示描述表

研究位图(和由位图派生出的对象)前,应先研究显示描述表(DC)。一个 DC 定义了可对位图进行的操作。显示描述表对位图来讲,就象一幅油画框对一片油画布。尽管读者可在画布上自由绘画,但画框限制了画布的形状。与此类似,尽管用户在位图上绘制(创建)图象,但与位图联系在一起的显示描述表已先定义好了位图的许多特性和绘制结果。

显示描述表定义了许多特性,这些特性将影响把位图传递给显示描述表的方式。这就引出了有关显示描述表的一个重要特征:有两类不同的显示描述表,即内存 DC 和屏幕 DC。

屏幕 DC 和它的名称一样,是与物理显示屏(通常为监示器)相联系的显示描述表。这类 DC 定义了将位图提供给屏幕时,位图的真实输出特性。

比较而言,内存 DC 不直接与输出设备挂钩,而是只模仿某一屏幕 DC 的行为。内存 DC 的目的是在屏幕外的私有区域创建和加工位图,然后再将其复制到屏幕(或磁盘,或其它什么地方)。特别是当图象大且复杂时,使用某些 GDI 函数绘制创建的图象将花费较长的时间,而 BitBlt()函数相对快一些。(BitBlt()函数将位图从一个地方复制到另一个地方,本书将在第 3 章"设备相关位图"中对其详细讨论。)

本章将讨论下列题目:

- 显示描述表基础:屏幕 DC
- 显示描述表分类
- 使用 DC:一个简单的例子
- 内存或可兼容 DC

2.1 显示描述表基础:屏幕 DC

所有与 DC 有关的操作都使用或返回一个 HDC 或一个显示描述表句柄。HDC 是 Windows 中最基本的东西,所有 GDI 函数在处理图形时都要用到它。

2.1.1 获取一个 DC

最容易获取的 DC 是窗口的 DC。这里窗口是指应用程序的主窗口,也可以是对话框窗口或子窗口。调用 GetDC()函数可获取窗口的 DC,该函数返回一个显示描述表句柄(HDC)。下段程序示出 GetDC()函数的用法:

```
HDC myDC;        //copy of the DC handle that you can use
...
//Get the handle to the DC of this window
myDC=GetDC(hWnd);
```

其中，hWnd 是窗口句柄，该窗口是用户希望获得其 DC 的窗口。

关于 GetDC()函数，有两个重要方面需要说明一下。首先，GetDC< >返回一个有效的显示描述表。其次，GetDC()返回的 DC 句柄是显示描述表句柄，而不是显示描述表本身。这就意味着要加工显示描述表，程序员要将该句柄传递给加工显示描述表的函数。HDC 就象一个标签，该标签精确地指向用户感兴趣的 Windows DC。(若读者掌握了这一概念，便可完成许多错综复杂的操作。)

2.1.2 释放一个 DC

用户使用完一个 HDC 后，应将其送回系统。这是毫无选择的。显示描述表句柄是 Windows 中十分珍贵的资源，因为系统只有 5 个显示描述表句柄入口。因此，若用户使用完 HDC 后未将其送还系统，很明显——用户将收到 BOOM! 消息，Windows 将停止运转且尖叫报警。实际上，使用完 HDC 后送还 HDC 失败这样的灾难性错误并不难发现——若应用程序能做 5 个图形操作而系统尚未锁死，就可以肯定用户在应用程序的某处未将显示描述表句柄返还给系统。

用户使用完 HDC 后，调用 ReleaseDC()并传递 DC 所属的窗口句柄和 DC 自身的句柄给该函数便可交还该 HDC，如下所示：

```
HDC       myDC;          // DC handle for us to play with

// Get a handle to the DC of this window

myDC = GetDC ( hWnd );

// Do graphics things here...

// Give the DC back now that you're done.

ReleaseDC ( hWnd, myDC );
```

2.1.3 获取一个 DC——另一种方法

调用 GetDC()/ReleaseDC()函数是获取和释放某一窗口的 DC 句柄的一种方法。用户同样可用 BeginPaint()和 EndPaint()函数获取和释放 HDC。两者的区别是：GetDC()/ReleaseDC()函数可在任何时候调用(无论这样做是否有意义)，而 BeginPaint()和 EndPaint()只能在响应 WM_PAINT 消息时使用。

BeginPaint()函数不止简单返回一个 DC 句柄，它还为窗口设定裁剪域并返回裁剪域的矩形边界。(注意，裁剪域矩形边界并不是裁剪域本身——它是完全包含裁剪域的最小矩形区域。)这就比只获取窗口 DC 要慢得多。若未收到 WM_PAINT 消息就调用 BeginPaint()，函数返回的 DC 和裁剪域将无效。

与 GetDC()函数相同，用户应释放由 BeginPaint()函数获取的 DC 句柄。释放句柄应调用 EndPaint()函数。

2.2 显示描述表分类

与 Windows 中的许多事情类似，DC 也可分成许多类，开始很简单，然后越来越复杂，本书将介绍一下每一类 DC 在用法上的区别。

2.2.1 公共 DC：可共享的一个 DC

就一个窗口而言，显示描述表可有不同的类型。当用户创建窗口时，若不指明关于 DC 的特殊信息，窗口就会拥有一个公共显示描述表。这是缺省的情况。

窗口框架内的所有事物称为客户区；窗口边框，包括各种窗口位，如系统菜单按钮、标题栏以及最大最小按钮，都是非客户区对象。用户调用 GetWindowDC()函数获取整个区域的 DC 便可在窗口的这些区域内绘图。

公共显示描述表用起来很方便，但有一定的限制。首先，当用户要在窗口客户区绘图时(见图 2.1)，用户需调用 GetDC()或 BeginPaint()函数检取窗口的 DC。其次，用户每次获取 DC 句柄后，DC 的所有属性都被设置成缺省值。(见表 2.1DC 的属性及其缺省值。)若用户修改这些缺省值，用户释放 DC 句柄后，修改便无效了。因此，除非用户认可 DC 的缺省值，否则，用户每次获取公共 DC 句柄后，都要按自己的需要修改属性。

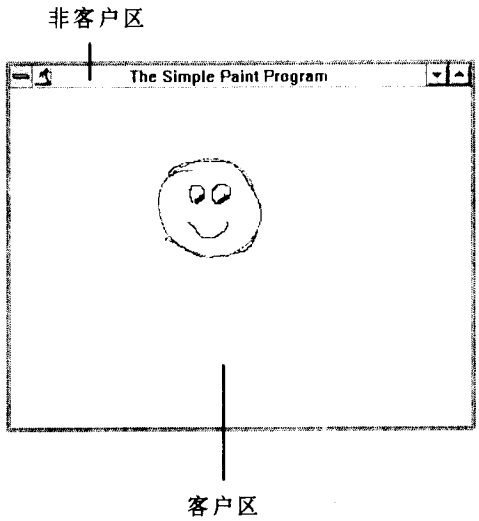

图 2.1 窗口的客户区

2.2.2 类 DC：与其它窗口共享一个 DC

若每次获取DC后用户都要修改缺省属性，那就太麻烦了，要想免去这一麻烦，用户

表 2.1 显示描述表的缺省属性

属 性	缺 省 值
背景颜色	白
背景模式	OPAQUE
位图	非缺省(对 CreateCompatibleDC()可能不适用,该函数同时创建了一个单象素单色位图。详情参见 CreateCompatibleDC()。)
画刷	WHITE_BRUSH
画刷原点	(0,0)
裁剪域	显示面
彩色调色板	DEFAULT_PALETTE
当前画笔位置	(0,0)
绘制方式	R2_COPYPEN
字体	系统字体
字间空间	0
映射方式	MM_TEXT
画笔	BLACK_PEN
多边形填充模式	ALTERNATE
拉伸方式	BLACKONWHITE
正文颜色	BLACK
视口范围	(1,1)
视口原点	(0,0)
窗口范围	(1,1)
窗口原点	(0,0)

可创建一个具有类显示描述表的窗口。要做到这一点,只需在创建窗口时将窗口的类属性指定为 CS_CLASSDC。

类显示描述表是一种半私有的资源,它可由某一特殊类窗口共享。为更清楚地说明这一点,举个例子,比如某些具有相同属性的小控制窗口。

用户在使用类显示描述表前,应先进行检取(也就是说,用 GetDC()和 BeginPaint()函数检取),但使用结束后,不必将其释放。这是与公共 DC 最大的不同点,公共 DC 使用结束后必须将其释放。尽管如此,还常常出现这样一种情况,用户的低层程序可能不能准确知道正在处理那个窗口(特别是按照某些强大的图形操作,如 BitBlt()处理窗口)。因此,用户程序通常假设 DC 使用完后都被释放。对类 DC 和私有 DC(下面将要讨论)来说,不必这样做,但如上文所述,对公共 DC 必须在使用完后将其释放。

> **忠告**:通常,使用完 DC 后,将其交还给系统。

类 DC 和公共 DC 的另一个区别是类 DC 不是从 Windows 的 DC 高速缓存中取出的(这 5 个 DC 在系统内都可得到)。这意味着使用类 DC 使应用程序占用更多的内存。这是因为 Windows 将为用户的类 DC 分配一部分内存(大约 800 字节)。

使用类 DC 的一个最大的优点就是用户修改 DC 的缺省属性后 DC 将不再变化(直到用户再次修改)。这就意味着用户对 DC 作一次修改之后,所有使用该 DC 的窗口都受该

变化的影响。但裁剪域和原点是个例外，因为每次检取 DC，Windows 将用户检取 DC 的窗口的当前值重新写入 DC。

2.2.3 私有 DC：用户自己的 DC

比类 DC 更进一步，用户可以给所创建的窗口一个自己的 DC。这意味着用户创建窗口时，将窗口的类属性指定为 CS_OWNDC。这样，用户便可得到一个私有显示描述表。

私有 DC 与类 DC 相似，它不在 Windows 的 DC 高速缓存中，用户每创建一个带有这类属性的窗口，都要用去 800 字节(左右)的内存。私有 DC 将维持用户对其缺省属性的修改——当用户修改了某一属性后，若不再修改，将保持修改后的属性。

私有 DC 与类 DC 的最大区别是类 DC 适用于特殊窗口类中的所有实例，而私有 DC 只适用于某一特定窗口实例。这意味着对有上百个实例的窗口类(如控制按钮)来说，最好不要用私有 DC，但对只有一两个实例的窗口类，最好选用私有 DC。

2.2.4 窗口 DC：最后一类 DC

第四类 DC 与前面讨论的三类 DC 有所不同，它称为窗口 DC。窗口 DC 允许用户访问窗口的所有显示空间而不仅仅是客户区。公共 DC，类 DC 和私有 DC 都被限制在窗口的客户区(或其中的一部分)，而窗口 DC 允许用户访问整个窗口，包括标题栏和窗口边界。

这就意味着两种情况。一方面，用户能轻而易举地在窗口边框或移动栏内绘图(作个假设)。另一方面，用户若不十分清楚自己在做什么，窗口边框或移动栏将画得很难看。这也正是编写 DefWindowsProc()函数的原因之一。用户每次收到 WM_NCPAINT 消息后，真的要不仅重画客户区，还要重画窗口边界、标题栏及其它所有东西吗?本书建议不要这样。

窗口 DC 在做某些特殊效果的操作(如，在标题栏或活动窗口中作弹出操作)时十分有用，但在别处几乎没用。

读者现在可能有些迷惑了，下面，本书将介绍一个简单的例子。

2.3 使用 DC：一个简单的例子

本节提供了一个简单(很简单)的绘图程序。该程序很容易：当用户在 SimPaint 窗口内单击并按住鼠标左边按钮时，在鼠标当前位置画一个黑色象素点。

2.3.1 浏览 SimPaint 程序

清单 2.1 至 2.7 是 SimPaint 的源程序代码。

清单 2.1 SIMPAINT.C—SimPaint 的 C 源程序

```
// SIMPAINT.C
//
// Source code for the simple paint program
//
```

```
// Written by Alex Leavens, for ShadowCat Technologies
#include <WINDOWS.H>
#include "SIMPAINT.H"
#include "SIMPAINT.WMC"

BOOL buttonDown = FALSE; // If true, mouse button down
/**********************************
 *
 * WinMain()
 *
 * Startup function for all Windows programs.
 */
    int PASCAL
WinMain(HANDLE   hInstance,      // Current instance of program
        HANDLE   hPrevInstance,  // Previous instance of program (if any)
        LPSTR    lpCmdLine,      // command line (if applicable)
        int      nCmdShow)       // window display style (open/iconic)
{
   MSG msg; // Message passed to our program
   //--------------------------------
   hInst = hInstance; // Save the current instance
   if (! BLDInitApplication(hInstance,hPrevInstance,&nCmdShow,lpCmdLine))
      return FALSE;
   if (! hPrevInstance)          /* Is there another instance of the task */
      {
      if (! BLDRegisterClass(hInstance))
         return FALSE;          /* Exits if unable to initialize */
      }
   MainhWnd = BLDCreateWindow(hInstance);
   if (! MainhWnd)               /* Check if the window is created */
      return FALSE;
   ShowWindow(MainhWnd, nCmdShow);  /* Show the window */
   UpdateWindow(MainhWnd);          /* Send WM_PAINT message to window */
   BLDInitMainMenu(MainhWnd);       /* Initialize main menu if necessary */
   while (GetMessage(&msg,          /* Message structure */
      0,                            /* Handle of window receiving the message */
      0,                            /* Lowest message to examine */
      0))                           /* Highest message to examine */
      {
      if (BLDKeyTranslation(&msg))
         continue;
      TranslateMessage(&msg);       /* Translates character keys */
      DispatchMessage(&msg);        /* Dispatches message to window */
      }
   BLDExitApplication();            /* Clean up if necessary */
```

```
    return(msg.wParam);                /* Returns the value from PostQuitMessage */
}

/*************************************/
/*        WINDOW PROCEDURE FOR MAIN WINDOW    */
/*************************************/

    long FAR PASCAL
BLDMainWndProc ( HWND        hWnd,
                 unsigned    message,
                 WORD        wParam,
                 LONG        lParam )
{
    HDC     hDC;     // Handle to our device context
    WORD mX;         // Mouse x position
    WORD mY;         // Mouse y position
    //-----------------------
    switch (message)
    {
    case WM_CREATE:              /* window creation */
        /* Send to BLDDefWindowProc in (.WMC) for controls in main window */
        return BLDDefWindowProc(hWnd, message, wParam, lParam);
        break;
    case WM_SETFOCUS:            /* window is notified of focus change */
        /* Send to BLDDefWindowProc in (.WMC) for controls in main window */
        return BLDDefWindowProc(hWnd, message, wParam, lParam);
        break;
    case WM_DESTROY:             /* window being destroyed */
        PostQuitMessage(0);
        return BLDDefWindowProc(hWnd, message, wParam, lParam);
        break;
    case WM_MOUSEMOVE:
    // Only paint if the mouse button is down.
    if (buttonDown != TRUE )
       break;
    /* Get the current mouse coordinates */
    mX = LOWORD(lParam);
    mY = HIWORD(lParam);
      hDC = GetDC ( hWnd );   // Get DC of our window
      SetPixel ( hDC,
         mX,
         mY,
         RGB ( 0, 0, 0 ) );        // Draw a black pixel
    ReleaseDC ( hWnd,
                hDC );
      break;
```

```
case WM_LBUTTONDOWN:
    /* Get the current mouse coordinates */
    mX = LOWORD(lParam);
    mY = HIWORD(lParam);
        buttonDown = TRUE;
        hDC = GetDC ( hWnd );   // Get DC of our window
        SetPixel ( hDC,
          mX,
          mY,
          RGB ( 0, 0, 0 ) );     // Draw a black pixel
    ReleaseDC ( hWnd,
                hDC );
      break;
    case WM_LBUTTONUP:
        buttonDown = FALSE;
        break;
    case WM_COMMAND:          /* command from the main window */
        if (BLDMenuCommand(hWnd, message, wParam, lParam))
          break;                  /* Processed by BLDMenuCommand. */
          /* else default processing by BLDDefWindowProc. */
    default:
        /* Pass on message for default processing */
        return BLDDefWindowProc(hWnd, message, wParam, lParam);
    }
    return FALSE;                 /* Returns FALSE if processed */
}
```

清单 2.2　SIMPAINT. WMC SimPaint 的专用包含文件，文件控制 Windows 消息处理细节

```
/* File  name: SIMPAINT. WMC                                          */
/* "SIMPAINT" Generated by WindowsMAKER Professional                  */
/* Author: Alex Leavens                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*              GLOBAL VARIABLES                                      */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
HANDLE    hInst      = 0;      /* Handle to instance.                */
HWND      MainhWnd   = 0;      /* Handle to main window.             */
HWND      hClient    = 0;      /* Handle to window in client area.   */
FARPROC   lpClient   = 0L;     /* Function for window in client area.*/
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*              PROCESSES KEYBOARD ACCELERATORS                       */
/*              AND MODELESS DIALOG BOX KEY INPUT                     */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
BOOL BLDKeyTranslation(pMsg)
```

```
MSG *pMsg;
    {
    return FALSE; /* No special key input                              */
    }
/*********************************************************************/
/*              CUSTOM MESSAGE PROCESSING FOR MAIN WINDOW            */
/*********************************************************************/
long FAR PASCAL BLDDefWindowProc(hWnd, message, wParam, lParam)
HWND hWnd;              /* window handle                              */
unsigned message;       /* type of message                            */
WORD wParam;            /* additional information                     */
LONG lParam;            /* additional information                     */
    {
    switch (message)
        {
    default:
        /* Pass on message for default processing by Windows          */
        return DefWindowProc(hWnd, message, wParam, lParam);
        }
    return FALSE; /* Returns FALSE if not processed by Windows        */
    }
/*********************************************************************/
/*              PROCESSES ALL MENU ITEM SELECTIONS                   */
/*********************************************************************/
BOOL BLDMenuCommand(hWnd, message, wParam, lParam)
HWND hWnd;              /* Window handle                              */
unsigned message;       /* Type of message                            */
WORD wParam;            /* Additional information                     */
LONG lParam;            /* Additional information                     */
    {
    switch(wParam)
        {
        /* Processing of linked menu items in menu: SIMPAINT          */
        default:
           return FALSE;        /* Not processed by this function.    */
        }
    return TRUE;                /* Processed by this function.        */
    }
/*********************************************************************/
/*     FUNCTIONS FOR INITIALIZATION AND EXIT OF APPLICATION          */
/*********************************************************************/
BOOL BLDInitApplication(hInst,hPrev,pCmdShow,lpCmd)
HANDLE hInst;     /* Handle to application instance.                  */
HANDLE hPrev;     /* Handle to previous instance of application.      */
int *pCmdShow;    /* Pointer to variable that specifies how main window is to be
```

```
                    shown.                                          */
LPSTR lpCmd;        /* Long pointer to the command line.           */
    {
    /* No initialization necessary */
    return TRUE;
    }

BOOL BLDRegisterClass(hInstance) /* Registers the class for the main window */
HANDLE hInstance;
    {
    WNDCLASS WndClass;

    WndClass.style          = 0;
    WndClass.lpfnWndProc    = BLDMainWndProc;
    WndClass.cbClsExtra     = 0;
    WndClass.cbWndExtra     = 0;
    WndClass.hInstance      = hInstance;
    WndClass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    WndClass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    WndClass.hbrBackground  = CreateSolidBrush(GetSysColor(COLOR_WINDOW));
    WndClass.lpszMenuName   = "SIMPAINT";
    WndClass.lpszClassName  = "SIMPAINT";

    return RegisterClass(&WndClass);
    }

HWND BLDCreateWindow(hInstance)  /* Creates the main window        */
HANDLE hInstance;
    {
    HWND hWnd;                   /* window handle                  */
    int coordinate[4];           /* Coordinates of main window     */

    coordinate[0]=CW_USEDEFAULT;
    coordinate[1]=0;
    coordinate[2]=CW_USEDEFAULT;
    coordinate[3]=0;

    hWnd = CreateWindow("SIMPAINT", /* window class registered earlier */
           "The Simple Paint Program",      /* window caption */
        WS_OVERLAPPED | WS_THICKFRAME | WS_SYSMENU | WS_MINI-
        MIZEBOX | WS_MAXIMIZEBOX,
                                 /* Window style    */
          coordinate[0],         /* X position      */
          coordinate[1],         /* Y position      */
          coordinate[2],         /* Width           */
          coordinate[3],         /* Height          */
          0,                     /* Parent handle   */
          0,                     /* Menu or child ID*/
          hInstance,             /* Instance        */
          (LPSTR)NULL);          /* Additional info */

    return hWnd;
    }
```

```
BOOL BLDInitMainMenu(hWnd)      /* Called just before entering message loop */
HWND hWnd;
        {
        /* No initialization necessary */
        return TRUE;
        }
BOOL BLDExitApplication()               /* Called just before exit of application   */
        {
        /* No processing needed at exit for this design */
        return TRUE;
        }
/*************************************************/
/* ERROR MESSAGE HANDLING (Definitions can be overruled.)          */
/*************************************************/
#ifndef ERRORCAPTION
#define ERRORCAPTION "The Simple Paint Program"
#endif
#ifndef LOADERROR
#define LOADERROR "Cannot load string."
#endif
int BLDDisplayMessage(hWnd,uMsg,pContext,iType)
HWND hWnd;
unsigned uMsg;
char *pContext;
int iType;
    {
    int i, j;
    char Message[200+1];
    if (uMsg)
        {
        if (!LoadString(hInst,uMsg,Message,200))
            {
            MessageBox(hWnd,LOADERROR,ERRORCAPTION,
                        MB_OK | MB_SYSTEMMODAL | MB_ICONHAND);
            return FALSE;
            }
        }
    else
        Message[0]=0;
    if (pContext)
        {
        i = lstrlen(Message);
        j = lstrlen(pContext);
        if (i + j + 1 <= 200)
            {
            lstrcat(Message, " ");
            lstrcat(Message, pContext);
```

```
        }
      }
   return MessageBox(hWnd,Message,ERRORCAPTION,iType);
   }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*          FUNCTIONS FOR DRAWING GRAPHIC BUTTONS                      */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

BOOL BLDDrawIcon(lpDrawItem,pIconName)
LPDRAWITEMSTRUCT lpDrawItem;
char *pIconName;
    {
    HICON hIcon;

    if (!(hIcon = LoadIcon(hInst,pIconName)))
       {
       BLDDisplayMessage(GetActiveWindow(),BLD_CannotLoadIcon,pIconName,
                         MB_OK | MB_ICONASTERISK);
       return FALSE;
       }
    SetMapMode(lpDrawItem->hDC,MM_TEXT);
    return DrawIcon(lpDrawItem->hDC,0,0,hIcon);
    }

BOOL BLDDrawBitmap(lpDrawItem,pBitmapName,bStretch)
LPDRAWITEMSTRUCT lpDrawItem;
char *pBitmapName;
BOOL bStretch;
      {
      HBITMAP hBitmap;
      HDC hMemDC;
      BITMAP Bitmap;
      int iRaster;

      iRaster = GetDeviceCaps(lpDrawItem->hDC,RASTERCAPS);
      if ((iRaster&RC_BITBLT)!=RC_BITBLT)
         return FALSE; /* Device cannot display bitmap */

      if (!(hBitmap = LoadBitmap(hInst,pBitmapName)))
         {
         BLDDisplayMessage(GetActiveWindow(),BLD_CannotLoadBitmap,pBitmapName,
                           MB_OK | MB_ICONASTERISK);
         return FALSE;
         }
      if (!GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&Bitmap))
      {
         DeleteObject(hBitmap);
         return FALSE;
         }
      if (!(hMemDC = CreateCompatibleDC(lpDrawItem->hDC)))
         {
         DeleteObject(hBitmap);
```