新编 Windows API 参考大全 完整版

书名：新编 Windows API 参考大全
作者：本书编写组
页数：981 页
开数：16 开
字数：2392 千字
出版日期：2000 年 4 月第二次印刷
出版社：电子工业出版社
书号：ISBN 7-5053-5777-8
定价：98.00 元

内容简介

作为 Microsoft 32 位平台的应用程序编程接口，Win32 API 是从事 Windows 应用程序开发所必备的。本书首先对 Win32 API 函数做完整的概述；然后收录五大类函数：窗口管理、图形设备接口、系统服务、国际特性以及网络服务；在附录部分，讲解如何在 Visual Basic 和 Delphi 中对其调用。

本书是从事 Windows 应用程序开发的软件工程师的必备参考手册。

前言

Win32 API 作为 Microsoft 32 位平台（包括：Windows 9x，Windows NT3.1 / 4.0 / 5.0，WindowsCE）的应用程序编程接口，它是构筑所有 32 位 Windows 平台的基石，所有在 Windows 平台上运行的应用程序都可以调用这些函数。

从事 Windows 应用程序开发，离不开对 Win32 API 函数的调用。只有充分理解和利用 API 函数，才能深入到 Windows 的内部，充分挖掘系统提供的强大功能和灵活性。

近年来，随着 Microsoft 32 位平台的版本升级， Win32 API 函数的构成、功能与调用方式都有很大的发展变化，然而，国内很少有相关的新版资料出版。为了满足广大开发人员的迫切需求，我们经过认真收集、整理素材，组织编写了这本与各种 Microsoft 32 位平台最新版本同步的 Win32 API 参考手册。

全书收录了五大类函数：窗口管理、图形设备接口、系统服务、国际特性以及网络服务。所有函数均附有功能说明、参数说明、返回值说明、备注以及引用说明。另外，在本书的第一章，我们对 WiN32 API 函数作了完整的概述；在附录部分，讲解了如何在 Visual Basic 和 Delphi 中对其调用。

由于篇幅较大，涉及技术内容广泛，加之时间仓促，书中难免存在不少错误或疏漏，希望广大读者给与批评指正。

编 者

目录

第一章 Win32 API 概论 1
1.1 为什么使用 Wu32 API 1
1.2 Win32 API 简介 1
第二章 窗口管理函数（Window Control Function） 14
2.1 易用特性函数（Accessibility Features） 14
2.2 按钮函数（Button） 22
2.3 插入标记（^）函数（Caret） 24
2.4 组合框函数（Combo box） 27
2.5 通用对话框函数（Common Dialog Box） 29
2.6 标函数（Cursor） 42
2.7 对话框函数 Dialog Box 47
2.8 编辑控制函数（Edit Control） 64
2.9 图标函数（Icon） 65
2.10 键盘加速器函数（Keyboard Accelerator） 73
2.11 键盘输入函数（Keyboard Input） 75
2.12 列表框函数（List boX） 90
2.13 菜单函数（Menu） 91
2.14 消息和消息总队列函数（Message and Message Queue） 108
2.15 鼠标输入函数（Mouse Input） 120
2.16 多文档接口函数（Multiple Document Interface） 125
2.17 资源函数（Resource） 127
2.18 滚动条函数（Scroll Bar） 137
2.19 窗口函数（Window） 143
2.20 窗口类函数（Window Class） 175
2.21 窗口过程函数（Window Procedure） 183
2.22 窗口属性函数（Window Property） 184
第三章 图形设备接口函数 189
3.1 位图函数（Bltmap） 189
3.2 笔刷函数（Brush） 209
3.3 剪裁函数（Clipping） 214
3.4 颜色函数（Color） 220
3.5 坐标空间及映射函数（Coordinate Space and Transformation） 228
3.6 设备环境函数（Device Context） 238
3.7 填充图形函数（Filled Shape） 258
3.8 字体和文本函数（Font and Text） 263
3.9 ICM 2.0 函数 290
3.10 直线和曲线函数（Line and Curve） 290
3.11 元文件函数（Metafile） 290
3.12 多显示器支持函数（Multple Display Monltors） 290
3.13 绘图和画图函数（Painting and Drawing） 290
3.14 路径函数（Path） 291
3.15 画笔函数（Pen） 291

3.16 打印和打印假脱机函数（Printing and Print Spooler）.....291
3.17 矩形函数（Rectangle）.....291
3.18 区域函数（Region）.....291
第四章 网络服务函数.....292
第五章 国际特性函数.....293
第六章 系统服务函数.....294

第一章 Win32 API 概论

1.1 为什么使用 Wu32 API

在 Windows 程序设计领域处于发展初期时，Windows 程序员可使用的编程工具唯有 API 函数。这些函数在程序员手中犹如“积木块”一样，可搭建出各种界面丰富、功能灵活的应用程序。不过，由于这些函数结构复杂，所以往往难以理解，而且容易误用。

随着软件技术的不断发展，在 Windows 平台上出现了很多优秀的可视化编程环境，程序员可以采用“所见即所得”的编程方式来开发具有精美用户界面和功能的应用程序。这些可视化编程环境操作简便、界面友好，比如：Visual C++，Delphi，Visual Basic 等等。在这些工具中提供了大量的类库和各种控件，它们替代了 API 的神秘功能。事实上，这些类库和控件都是构筑在 Windows API 的基础上的，但它们使用方便，加速了 Windows 应用程序的开发，所以受到程序员的普遍采用。有了这些类库和控件，程序员们便可以把主要精力放在整体功能的设计上，而不必过于关注具体细节。不过，这也导致了非常多的程序员在类库面前“固步自封”，对下层 API 函数的强大功能一无所知。

实际上。程序员要想开发出更灵活、更实用、更具效率的应用程序，必然要涉及到直接使用 API 函数。虽然类库和控件使应用程序的开发容易得多，但它们只提供 Microsoft Windows 的一般功能，对于一些比较复杂和特殊的功能来说，单使用类库和控件是难以实现的，必须直接使用 API 函数来编写。API 函数是构筑整个 Windows 框架的基石，只有充分理解和利用 API 函数，才能深入到 Windows 的内部，充分发挥各种 32 位平台的强大功能和灵活性，才能成功地扩展和突破类库、控件和可视开发环境的限制。

1.2 Win32 API 简介

Win32 API 即为 Microsoft 32 位平台的应用程序编程接口（Application Programming Interface）。所有在 Win32 平台上运行的应用程序都可以调用这些函数。

使用 Win32 API，应用程序可以充分挖掘 Windows 的 32 位操作系统的潜力。 Mircrosoft 的所有 32 位平台都支持统一的 API，包括函数、结构、消息、宏及接口。使用 Win32 API 不但可以开发出在各种平台上都能成功运行的应用程序，而且也可以充分利用每个平台特有的功能和属性。

在具体编程时，程序实现方式的差异依赖于相应平台的底层功能的不同。最显著的差异是某些函数只能在更强大的平台上实现其功能。例如，安全函数只能在 Windows NT 操作系统下使用。另外一些主要差别就是系统限制，比如值的范围约束，或函数可管理的项目个数等等。

标准 Win32 API 函数可以分为以下几类：

窗口管理

窗口通用控制

Shell 特性

图形设备接口

系统服务

国际特性

网络服务

在下面各节中，我们分别介绍这 7 种类型的 API 函数。

1.2.1窗口管理函数

窗口管理函数向应用程序提供了一些创建和管理用户界面的方法。你可以使用窗口管理函数创建和使用窗口来显示输出、提示用户进行输入以及完成其他一些与用户进行交互所需的工作。大多数应用程序都至少要创建一个窗口。

应用程序通过创建窗口类及相应的窗口过程来定义它们所用窗日的外观和行为。窗口类可标识窗口的缺省属性，比如窗口是否接受双击鼠标按钮的操作。或是否带有菜单。窗口过程中包含的代码用于定义窗口的行为，完成所需的任务，以及处理用户的输入。

应用程序可使用 GDI 函数来产生窗口的输出。由于所有的窗口都共享显示屏幕，所以应用程序不接受对整个屏幕的访问。系统管理所有的输出内容，并对它们进行排列和剪裁，使其能够适合相应的窗口。应用程序可以在处理输入消息时，或为了响应系统的需求而在窗口中绘图。当窗口的大小或位置发生变化时，系统通常会向应用程序发送一个消息，要求它对该窗口中原来未显露的区域进行重画。

应用程序以消息的形式接受鼠标和键盘输入。系统将鼠标移动、鼠标按钮操作转换为输入消息，并将这些消息放入该应用程序的消息队列中。系统为每个应用程序都自动提供一个消息队列。应用程序使用消息函数从消息队列中获取消息，并将它们分派给适当的窗口过程进行处理。

应用程序可以直接处理鼠标和键盘输人，也可以让系统使用菜单和键盘加速健将这些低级输入转换成命令消息。你可以使用菜单向用户展现一个命令列表。系统对所有菜单操作所需的动作进行管理，包括让用户选择一个命令，然后再向窗日过程发送一个标识该选择的消息。键盘加速键是应用程序定义的按键操作组合，系统可将其转换为消息。加速键通常对应于菜单中的某个命令，并与该命令产生相同的消息。

应用程序通过在对话框中向用户提示附加信息来响应命令消息。对话框实际是一个临时的窗口，用于显示信息或提示输入。一个对话框通常由一些表示按钮和方框的控制组成，可供用户进行选择或输入信息。对话框中可包括用于输入正文、滚动正文、从列表中选择列表项等操作的控制。对话框管理和处理来自这些控制的输入，使应用程序可使用这些信息，来完成所要求的命令操作。

通过使用“资源”可以共享很多有用的数据，比如位图、图标、字体和字符串等，只需将这些数据作为“资源”添加到应用程序或 DLL 文件中。应用程序通过使用资源函数，找到资源并将它们加载到内存来获取这些数据。

窗口管理函数还提供了其他一些与窗口有关的特性，比如插入标记(Caret)、剪贴板、光标、挂钩(Hook)、图标以及菜单等函数。

窗口管理函数包括以下几类：

易用特性函数（Accessibility Features）

Win32 API 提供的一系列易用特性使得有残疾的人也能很容易的使用计算机，Win32 API 提供了一些函数和结构来控制这些特性。

按钮函数（Button）

Microsoft 提供了对话框和控制来支持应用程序与用户之间的交互通讯。按钮就是一种控制，用户可通过点击按钮来向应用程序提供输入信息。

插入标记函数（Caret）

一个插入标记是位于窗口绘图区中的一个闪动的直线、方块或图标。插入标记通常用于指示文本或图形将插入的位置。Win32 应用程序可以使用插入标记函数来创建一个插入标记，改变它的闪动频率，显示、隐藏插入标记，或重新设置插入标记的位置。

组合框函数（Combo Box）

组合框是由 COMBOBOX 类定义的一种控制，综合了列表框和编辑控制的很多功能。使用组合框函数可以在组合框中显示或获取不同类型的数据。

通用对话框函数（Common Dialog Box）

通用对话框是在通用对话框库中定义的，其功能是用来完成一些通用的任务，比如打开文件、打印文

档等。通用对话框为用户提供了一个统一的用户界面，使用户在不同的应用程序中完成通用任务时的操作都相同，不必每次都学习不同的操作过程。

光标函数（Cursor）

光标是显示屏幕上的一个小图形，其所在的位置由指点设备比如鼠标、光笔或轨迹球等控制。当用户移动鼠标时，系统就会随之移动光标的位置。应用程序使用 Win32 光标函数可以创建、加载、显示、移动、限制和删除光标。

对话框函数（Dialog Box）

对话框是应用程序创建的一个临时窗口，用于获取用户的输入。应用程序通常使用对话框向用户显示一些命令提示信息。一个对话框一般由一个或多个控制（子窗口）组成，这些控制可用来输入文本、选择选项或执行命令动作。

编辑控制函数（Edit Control）

编辑控制是一个矩形窗口，通常用在对话框中，用户可通过键盘向编辑控制中输入和编辑文本。系统对 Unicode 文本（字符采用双字节编码）和 ANSI（字符采用单字节编码）文本都支持。

图标函数（Icon）

图标是一个图片，由一个位图图像组成，并和一个掩码组合构成该图片的透明区域。当提到图标时，可以是下列两种情况：

1）单个图标图像。资源类型为 RT_ICON。

2）一组图标图像，系统或应用程序可从中选择。资源类型为 RT_GROUP_JCON。

应用程序使用图标函数可以创建、显示、删除和复制图标。

键盘加速健函数（Keyboard Accelerator）

键盘加速键（或简称为加速键）是一个按键操作或多个按键操作的组合，可向应用程序发送 WM_COMMAND 或 WM_SYSCOMMAND 消息。

使用键盘加速键函数可以拷贝、创建、加载或删除加速键表，*图标是一个图片，由一个位图图像组成，并和一个掩码组合构成该图片的透明区域。当提到图标时，可以是下列两种情况：

1）单个图标图像。资源类型为 RT_ICON。

2）一组图标图像，系统或应用程序可从中选择。资源类型为 RT_GROUP_JCON。

应用程序使用图标函数可以创建、显示、删除和复制图标。

键盘加速健函数（Keyboard Accelerator）

键盘加速键（或简称为加速键）是一个按键操作或多个按键操作的组合，可向应用程序发送 WM_COMMAND 或 WM_SYSCOMMAND 消息。

使用键盘加速键函数可以拷贝、创建、加载或删除加速键表，还可以将加速键消息转换为命令消息。

键盘输入函数（Keyboard Input）

键盘输入函数提供了接受和处理键盘输入的方法。

列表框函数（List Box）

Microsoft 的 Win32 API 提供了对话框和控制来支持应用程序与用户之间的交互通讯。列表框是一个控制窗口，其中包含一系列选项，可供用户进行选择。使用列表框函数可以在列表框中显示或获取不同类型的数据。

菜单函数（Menu）

菜单函数向 Win32 应用程序提供了一系列创建、管理和使用菜单的方法，包括对菜单条、菜单项。

于菜单等的处理。

消息和消息队列函数（Message and Message Queue）

消息和消息队列函数向 Win32 应用程序提供了一系列使用消息和消息队列的方法，包括对消息进行

传播、发送、获取、转换等操作。

鼠标输入函数（Mouse Input）

鼠标输入函数提供了接受和处理鼠标输入的方法。

多文档接口函数（Multiple Document Interface）

多文档接口(MDI)是应用程序定义用户界面的一种规范，在这种界面下，用户可以同时使用多个文档。

资源函数（Resource）

一个资源是一些二进制数据，可以添加到 Win32 应用程序的可执行文件中。资源既可以是标准的，也可以是自己定义的。标准资源中的数据包括图标、光标、菜单、对话框、位图、增强元文件、字体、加速键表、消息表入口、字符串表入口或版本。应用程序定义的资源（也称为定制的资源）可以包含特

殊应用程序所需的任何数据。

使用资源函数可以添加、删除、拷贝、替换或加载各种资源数据。

滚动条函数（Scroll Bar）

在 Win32 应用程序的窗口中，可以显示比该窗口的显示区更大的数据对象，比如文档或位图。当窗口提供了滚动条时，用户就可以通过拖动滚动条来浏览该数据对象中位于显示区外面的部分。

滚动条包括水平滚动条和垂直滚动条。使用滚动条函数可以创建和管理这两种滚动条。

窗口函数（Window）

在图形化的 Win32 应用程序中，窗口是屏幕上的一个矩形区域，应用程序可在该区域中显示输出结果，并接受用户输入。因此，一个图形化的 Win32 应用程序的首要任务之一就是创建一个窗口。

一个窗口与其他窗口共享显示屏幕，也包括其他应用程序所创建的窗口。一次只能有一个窗口接受用户的输入。用户可以使用鼠标、键盘或其他输人设备与该窗口及拥有该窗口的应用程序进行交互。使用窗口函数可以创建和管理窗口。

窗口类函数（Window Class）

一个窗口类是一个属性的集合，系统将该属性集合用作创建窗口的模板。每个窗口都是某个窗口类的一个成员。使闲置或删除窗口属性。

1.2.2窗口通用控制

系统 Shell 提供了一些控制，使用这些控制可以使窗口具有与众不同的外观。由于这些控制是由 DLL 支持的，是操作系统的一部分，所以它们对所有的应用程序都可用。使用通用控制有助于使应用程序的用户界面与系统 Shell 及其他应用程序保持一致。由于开发一个控制需要花费一定的时间，所以直接使用通用控制也可以节省大量的开发时间。

通用控制是由通用控制库 COMCTL32.DLL 支持的一个控制窗口集。与其他控制一样，一个通用控制也是应用程序的一个子窗口，它与其他窗口联合使用，完成 I/O 操作。通用控制 DLL 包括一个编程接口，应用程序可使用其中的函数创建和管理控制。以及从控制中接受用户输入。

1.2.3Shell 特性

Win32 API 中包含一些接口和函数，应用程序可使用它们来增强系统 Shell 的各方面功能。

一个名字空间是一个符号集合，比如文件和目录名字，或数据库关键字。Shell 使用一个单层结构的名字空间来组织用户关心的所有对象，包括文件、存储设备、打印机及网络资源。名字空间类似于文件系统的目录结构，只不过名字空间中包含的是对象，而不是文件和目录。

快捷键（也称为一个 Shell 连接）是一个数据对象，它包含的信息可用于访问位于 Shell 名字空间的任何位置的其他对象。使用快捷键时，应用程序不必知道对象的当前名字和位置就可以访问该对象。可以通过快捷键访问的对象包括文件、文件夹、磁盘驱动器、打印机及网络资源。

有几种方法可以扩展 Shell。系统使用图标来表示 Shell 名字空间中的文件。缺省情况下，系统对具

有相同文件扩展名的所有文件都显示相同的图标。可以用一个图标句柄来改变某特殊文件的缺省图标。使用上下文相关菜单句柄可以修改一个上下文相关菜单的内容，这也是一种 Shell 扩展。当用户用鼠标右键点击或拖动一个对象时，系统会显示一个上下文相关菜单。该上下文相关菜单中所包含的命令只应用在被点击或拖动的对象上。大多数上下文相关菜单都包含一个 Properties 命令，用于显示所选中项目的属性表。一个属性表由一系列重叠的窗口组成（每个窗口称为一页），用于显示有关某个对象的信息。属性表句柄是一种 Shell 扩展，使用它可以向系统定义的属性表中添加页，或替换控制面板的属性表的某些页。一个拷贝挂钩（Hook）句柄是一种 Shell 扩展，可以允许或拒绝对一个文件对象的移动、拷贝、删除或重命名。

系统 Shell 包含一个快速查看（Quick View）命令，使用户可以直接查看一个文件的内容. 而不必运行创建该文件的应用程序。文件浏览器提供了一个用于查看文件的用户界面。Shell 使用文件扩展名来确定应运行哪个浏览器。你可以为新的文件格式提供文件浏览器，或用具有更强功能的浏览器来替换原来的浏览器。文件浏览器与文件分析器联合使用，后者的功能是对文件名进行分析，以便确定应生成哪种类型文件的 QUck View。你还可以提供其他的文件分析器来支持新的文件类型。

1.2.4图形设备接口

图形设备接口（GDI）提供了一系列的函数和相关的结构，应用程序可以使用它们在显示器、打印机或其他设备上生成图形化的输出结果。使用 GDI 函数可以绘制直线、曲线、闭合图形、路径、文本以及位图图像。所绘制的图形的颜色和风格依赖于所创建的绘图对象，即画笔、笔刷和字体。你可以使用画笔来绘制直线和曲线，使用笔刷来填充闭合图形的内部，使用字体来书写文本。

应用程序通过创建设备环境(DC),可以直接向指定的设备进行输出。设备环境是一个 GDI 管理的结构。其中包含一些有关设备的信息,比如它的操作方式及当前的选择。应用程序可使用设备环境函数来创建 DC。GDI 将返回一个设备环境句柄，在随后的调用中，该句柄用于表示该设备。例如，应用程序可以使用该句柄来获取有关该设备性能的一些信息，诸如它的类型（显示器、打印机或其他设备），它的显示界面的尺寸和分辨率等。

应用程序可以直接向一个物理设备进行输出，比如显示器或打印机；也可以向一个“逻辑”设备进行输出，比如内存设备或元文件。逻辑设备向应用程序所提供的保存输出结果的格式，可以很容易地将其发送到物理设备上。一旦应用程序将输出结果记录到了一个元文件中，那么该元文件就可以被使用任意多次，并且该输出结果可以被发送到任意多个物理设备上。

应用程序可以使用属性函数来设置设备的操作方式和当前的选择。操作方式包括文本和背景颜色，混色方式（也称为二元光栅操作，用于确定画笔或笔刷的颜色与绘图区域现有的颜色如何进行混色），映射方式（用于指定 GDI 如何将应用程序所用的坐标映射到设备坐标系统上）。当前的选择是指绘图时使用哪个绘图对象。

图形设备接口函数包括以下几类：

位图函数（Bltmap）

位图是一个图形对象，可将图像作为文件进行创建、处理（比例缩放、滚动、旋转和绘制）和存储。

位图函数提供了一系列处理位图的方法。

笔刷函数（Brush）

笔刷是一种绘图工具，Win32 应用程序可使用它绘制多边形、椭圆形和路径的内部。绘图应用程序使用笔刷绘制图形；字处理应用程序使用笔刷绘制水线；计算机辅助设计（CAD）应用程序使用笔刷绘制截面视图的内部；电子表格应用程序使用笔刷绘制饼图的扇形和直方图的方条。笔刷函数提供了一系列创建和使用笔刷的方法。

剪裁函数（Clipping）

剪裁是一种处理过程，它将输出到某个区域或路径中的内容限制在应用程序窗口的显示区内。剪裁函数提供了一系列处理剪裁区域的方法。

颜色函数（Color）

颜色是组成 Win32 应用程序所生成的图片和图像的一个重要元素。 Win32 API 提供了一系列管理和使用画笔、笔刷、文本和位图的颜色的函数。

坐标空间及映射函数（Coordinate Space and Transformation）

Win32 应用程序使用坐标空间和映射函数对输出的图形进行比例缩放、旋转、转换、剪裁和反射。

坐标空间是基于笛卡尔坐标系统的一个平面空间。该坐标系统要求有两个垂直相交的、长度相等的坐标轴。共有 4 种坐标空间：现实坐标、页面坐标、设备坐标、物理设备坐标（显示区，或桌面，或打印纸的页面）。映射方式就是改变（“映射”）对象的大小、方向和形状的一种算法。

设备环境函数（Device Context）

设备环境是一个结构，它定义了一系列图形对象及其相关的属性，以及会影响输出结果的绘图方式。这些图形对象包括：画笔（用于画直线），笔刷（用于绘图和填充），位图（用于屏幕的拷贝或滚动），调色板（用于定义可用的颜色集），剪裁区（用于剪裁和其他操作），路径（用于绘图和画图操作）。设备环境函数用于对设备环境进行创建、删除或获取信息。

填充图形函数（Filled Shape）

填充图形是一些几何图形，其轮廓由当前的画笔绘制，内部由当前的笔刷填充。共有 5 种填充图形：椭圆，弦图，饼图，多边形，矩形。填充图形函数用于对填充图形进行操作。

字体和文本函数（Font and Text）

字体用于在视频显示器或其他输出设备上绘制文本。Win32 API 提供了一系列用于安装、选择和查询各种字体的字体和文本函数。

ICM 2.0 函数

Microsoft Windows 98 和 Windows NT 5.0 所使用的颜色管理方案称为 Image Color Management 版本 2.0，或 ICM2.0，由一系列函数组成。

直线和曲线函数（Line and Curve）

直线和曲线用于在光栅设备上绘制输出图形。一条直线是光栅显示器上的一系列高亮像素点（或打印纸上的一系列点），由两个点进行标识：起点和终点。一条规则曲线也是光栅显示器上的一系列高亮像素点（或打印纸上的一系列点），符合某个二次曲线段的周界（或一部分）。不规则曲线则是由不符合二次曲线段的一系列像素点组成。

元文件函数（Metafile）

元文件是一个结构的集合，这些结构是以与设备无关的格式存储图像。设备无关是元文件与位图的差异之一。与位图不同，元文件保证是与设备无关的。不过，元文件有一个缺点：它通常比位图的绘图速度慢。因此，如果一个应用程序要求有较快的绘图速度，而不需要具有设备无关性，则应该用位图代替元文件。

元文件函数提供了一些对元文件进行操作的方法。

多显示器支持函数（Multple Display Monltors）

每个 Windows 工作站所支持的显示器个数是不受限制的。可以用创建邻接区域的方式安排多个显示器。每个显示器的大小和颜色深浅都可以独立设置。

所有的显示器屏幕一起构成了一个虚拟屏幕。桌面窗口覆盖整个虚拟屏幕，而不仅仅是某个显示屏幕。由于现有的应用程序都要求显示器具有一个原点坐标（0，0），所以虚拟屏幕必须在某个显示器上包含原点坐标（0，0），这个显示器就被看作是主显示器。

每个物理显示设备都由一个 HMONITOR 类型的显示器句柄表示。一个显示器在它的整个生存期间具有相同的 HMONITOR 值。

任何显示设备环境（DC）的 Win32 函数所返回的值都是主显示器的 DC。要想获取其他显示器的 DC，可使用 EnumDisplayMonitors 函数。系统对每个显示器调用回调函数，为该显示器传入一个 DC 值。用户可以使用该 DC 在该显示器上绘图。

绘图和画图函数（Painting and Drawing）

绘图和画图函数为应用程序提供了一系列在窗口中绘图的方法，以及如何创建和使用显示设备环境(DC)的方法。

路径函数（Path）

一个路径是指一个或多个被填充、被绘制轮廓或既被填充又被绘制轮廓的图形（或形状）。Win32 应用程序将路径用作很多用途，在绘图和画图应用程序中使用路径。计算机辅助设计（CAD）应用程序用路径来创建唯一剪裁区，绘制不规则形状的轮廓，以及填充不规则形状的内部。路径函数用于创建、改变和绘制路径。

画笔函数（Pen）

画笔是 Win32 应用程序用于绘制直线和曲线的图形工具。画图应用程序使用画笔来画手画线、直线以及曲线。计算机辅助设计（CAD）应用程序用画笔来画可见线、隐藏线、截面线、中心线等等。字处理和桌面出版应用程序用画笔来画边界和水线。电子表格应用程序用画笔来指明图表的趋向，以及勾勒直方图和饼图的轮廓。画笔函数提供了一系列使用画笔的方法。

打印和打印假脱机函数（Printing and Print Spooler）

Microsoft Windows 和 Windows NT 提供了一套完整的函数,使应用程序可以在不同的设备上进行打印，如激光打印机，向量绘图仪，光栅打印机，以及传真机等。

矩形函数（Rectangle）

Win32 应用程序使用矩形来指定显示屏幕上或窗口中的一个矩形区域。矩形函数用于对矩形进行操作。

区域函数（Region）

区域是指一个可被填充、着色、转换和加外框的形状，包括矩形、多边形或椭圆（或这几种形状的组合），用于完成击键测试（测试光标位置）。

区域函数用于对区域进行操作。

1.2.5系统服务

系统服务函数为应用程序提供了访问计算机资源以及底层操作系统特性的手段，比如访问内存、文件系统、设备、进程和线程。应用程序使用系统服务函数来管理和监视它所需要的资源。例如，应用程序可使用内存管理函数来分配和释放内存，使用进程管理和同步函数来启动和调整多个应用程序或在一个应用程序中运行的多个线程的操作。

系统服务函数提供了访问文件、目录以及输入输出（I/O）设备的手段。应用程序使用文件 I/O 函数可以访问保存在指定计算机以及网络计算机上的磁盘和其他存储设备上的文件和目录。这些函数支持各种文件系统，从 FAT 文件系统，CD-ROM 文件系统（CDFS），到 NTFS。

系统访问函数为应用程序提供了一些可以与其他应用程序共享代码或信息的方法。例如，可以将一些有用的过程放到 DLL 中，使它们对所有的应用程序都可用。应用程序只需使用 DLL 函数将动态链接库加载进来并获取各过程的地址，就可以使用这些过程了。通讯函数用于向通讯端口写入数据及从通讯端口读出数据,并控制这些端口的操作方式。有几种内部通讯(IPC)的方法,比如 DDE、管道(Pipe)、邮槽(Mailslot)和文件映射。对于提供安全属性的操作系统来说，应用程序可使用安全函数来访问安全数据，并保护这些数据不会被有意或无意地访问或破坏。

使用系统服务函数可以访问有关系统和其他应用程序的信息。应用程序可用系统信息函数来确定计算机的特别属性，比如是否出现鼠标、显示屏幕上的元素具有多大尺寸。注册和初始化函数用于将应用程序的特殊信息保存到系统文件中，以便于该应用程序的新实例对象，甚至其他应用程序都可以获取和使用这些信息。

应用程序使用系统服务函数可以处理执行过程中的的一些特殊情况，比如错误处理、事件日志、异常处理。还有一些属性可用于调试和提高性能。例如，使用调试函数可对其他进程的执行过程进行单步控制，

而性能监视函数则可对某个进程的执行路径进行跟踪。

系统服务函数还提供了一些特性，可用于创建其他类型的应用程序，比如控制台应用程序和服务。

系统服务函数包括以下几类：

访问控制函数（Access Control）

Microsoft Windows NT 所提供的安全功能对 Win32 应用程序是自动使用的。在系统中运行的每个应用程序都受由 Windows NT 的特殊配置所提供的安全功能所影响。Windows NT 是支持 Win32 安全功能的唯一平台。

Windows NT 的安全功能对大多数 Win32 函数的影响都是最小的，不需要安全功能的 Win32 应用程序不必合并任何特殊代码。不过，你可使用 Windows NT 的安全属性向 Win32 应用程序提供一些服务。

访问控制函数提供了一系列控制访问 Win32 对象（比如文件）、管理函数（比如设置系统时间或审核运行动作的函数）的 Windows NT 安全模型。

原子函数（Atom）

原子表格是一个系统定义的表格，用于保存字符串和相应的标识符。应用程序将一个字符串放到原子表格中，并接受一个 16 位的整数（称为一个原于），用于访问该字符串。放到原子表格中的字符串被称为原子名字。

原子函数提供了一系列对原子进行添加、删除、初始化等的操作。

客户服务器访问控制函数（Client/Server Access Control）

客户/服务器访问控制函数包括三类：

用于模拟客户机。

用于检查和设置私有对象上的安全描述符。

用于生成安全时间日志中的审核消息。

剪贴板函数（Clipboard）

剪贴板是由一系列函数和消息组成，Win32 应用程序可使用它来传输数据。由于所有的应用程序都可以访问剪贴板，所以数据可以很容易地在应用程序之间或一个应用程序内部进行传输。

通讯函数（Communication）

通讯资源是一个物理或逻辑设备，用于提供双向的异步数据流。例如，串行端口、并行端口、传真机以及调制解调器都是通讯资源。对于每个通讯资源都有一个服务供应程序（包含一个库或驱动程序），使应用程序可以访问该资源。通讯函数是通讯设备所使用的函数。

控制台函数（Console）

Microsoft Windows 和 Windows NT 提供了控制台函数，用于管理字符模式的应用程序（这种应用程序未提供自己的图形用户界面）的输入和输出（I/O）

数据解压库函数（Data Decompression Library）

数据解压库函数在 LZEXPAND.DLL 中声明，用于对压缩的文件进行解压。

调试函数（Debugging）

调试器是一个应用程序，开发人员可使用它来检查和改正编程错误。Win32 API 的调试函数为用户提供了一系列的调试手段。

设备输入和输出函数（Device Input and OutPut）

Win32 应用程序使用设备输入和输出控制与设备驱动程序进行通讯。被访问的设备由设备句柄标识；而设备驱动程序要完成的动作则由控制代码来指定。

动态数据交换函数（Dynamic Data Exchange）

Win32 API 为不能使用“动态数据交换管理库（DDEML）”的应用程序提供了一系列实现动态数据交换的函数。

动态数据交换管理函数（Dynamic Data Exchange Management）

动态数据交换（DDE）是一种内部通讯方式，即使用共享内存在应用程序之间交换数据。应用程序可以

使用 DDE 进行一次性的数据传输，以及数据的即时交换和更新。

动态数据交换管理函数为用户提供了一系列管理动态数据交换的手段。

动态链接库函数（Dynamic-Link Library）

动态连接库（DLL）是由函数和数据组成的一些模块。一个 DLL 是由它的调用模块（.EXE 或.DLL）在运行时加载的。当一个 DLL 被加载后，它就被映射到其调用进程的地址空间中。

DLL 可以定义两种函数：外部的和内部的。外部函数可以被其他模块调用，内部函数只能在声明它的 OLL 内部被调用。尽管 DLL 可以输出数据，但它的数据通常只能由它的函数使用。

DLL 提供了一种使应用程序模块化的方法，这样就可以更容易地更新和重用程序的功能。DLL 也有助于在几个应用程序同时使用相同的功能时减少内存开销，因为虽然每个应用程序都拥有一份数据的备份，但它们可以共享代码。

错误函数（Error）

写得好的应用程序应包括一些能够处理意外错误并可从错误中顺利恢复的代码。当发生错误时，应用程序可能需要用户进行干预，或自己恢复。在一些极端情况下，应用程序对能会将用户从系统中退出或关机。错误函数为用户痰腥 LL 内部被调用。尽管 DLL 可以输出数据，但它的数据通常只能由它的函数使用。

DLL 提供了一种使应用程序模块化的方法，这样就可以更容易地更新和重用程序的功能。DLL 也有助于在几个应用程序同时使用相同的功能时减少内存开销，因为虽然每个应用程序都拥有一份数据的备份，但它们可以共享代码。

错误函数（Error）

写得好的应用程序应包括一些能够处理意外错误并可从错误中顺利恢复的代码。当发生错误时，应用程序可能需要用户进行干预，或自己恢复。在一些极端情况下，应用程序对能会将用户从系统中退出或关机。错误函数为用户提供了一些进行错误处理的方法。

事件日志函数（Event Logging）

很多应用程序都在不同的属性错误日志中记录错误和事件。这些属性错误日志具有不同的格式，并显示不同的用户界面，而且无法将数据合并起来得到一个完整的报告。因此，用户必须要检查各种数据来诊断问题。Windows NT 的事件日志为应用程序（和操作系统）提供了一种标准、集中的方法，来记录重要的软件和硬件事件。事件日志服务将事件从不同的地方保存到一个称为“事件日志”的集合中。Windows NT 还提供了一个事件浏览器和编程接口，用于查看日志和检查日志。事件日志函数提供了一系列编写和检查事件日志的方法。

文件函数（File）

文件是计算机存储信息的基本单位，不同的信息可分别存放在不同的文件中。应用程序可使用文件函数对文件进行输入和输出（I/O）操作。

文件安装库函数（File Installation Library）

Win32 API 包含一个文件安装库，应用程序使用它可以更容易地安装文件，使安装程序能分析当前已安装的文件。

文件映射函数（File Mapping）

文件映射函数用于对文件映射对象进行操作。

文件系统函数（File System）

Win32 应用程序依赖文件系统来保存和获取存储设备上的信息。文件系统提供了应用程序在与存储设备相关的个别卷上创建和访问文件及目录时所需的底层支持。

每个文件系统都由一个或多个驱动程序和所支持的动态链接库（定义文件系统的数据格式和特性）组成。它们确定了文件名的约定、安全性及可恢复性的级别，以及输入输出（I/O）操作的一般性能。文件系统函数用于对文件系统进行操作。

句柄和对象函数（Handle and Object）

对象是一个表示系统资源的数据结构，比如表示一个文件、线程或图像。应用程序不能直接访问对象

所表示的对象数据或系统资源，而是必须使用对象句柄。对象句柄可用于检查和修改系统资源。每个句柄在一个内部维护的表中都有一项。在这些项中包含资源的地址以及标识资源类型的方法。句柄和对象函数用于对句柄和对象进行操作。

Hook 函数

Hook 是系统消息处理机制中的一部分。在系统消息处理机制中，应用程序可安装一个子程序来监视系统中的消息传送情况，并可处理某些类型的消息（在这些消息到达目的窗口过程之前）。Hook 函数用于对 Hook 进行操作。

lmageHlp 函数

lmageHlp 函数由 IMAGEHLP DLL 提供。lmageHlp 函数可用于 PE 格式的图像。PE 图像由一个兼容的 Win32 连接程序提供，比如由 Microsoft Developer Studio 提供。

超大整数操作函数（Large Integer Operations）

Win32 API 提供了一系列超大整邮槽是一种单向的内部处理通讯（IPC）机制。Win32 应用程序可以在邮槽中保存消息，邮槽的所有者可以获取保存在其中的消息。这些消息通常是通过网络发送到一台指定的计算机上，或发送到某个指定域中的所有计算机上。域是一组工作站和服务器，共享一个组名。

可以选择使用命名管道来代替邮槽进行内部处理通讯。命名管道是两个进程交换消息的一种简单方法。而邮槽则是一个进程向多个进程广播消息的一种简单方法。需要考虑的重要一点是邮槽使用邮包，而命名管道则不用。邮槽函数可用于创建邮槽、设置或获取邮槽信息。

内存管理函数（Memory Management）

内存管理函数用于分配和使用内存。

管道函数（Pipe）

管道是一段共享内存，用于进程通讯。创建管道的进程称为管道服务程序。连接管道的进程称为管道客户程序。某个进程向管道中写入信息，然后其他进程从管道中读出信息。管道函数用于创建、管理和使用管道。

电源管理函数（Power Management）

电源管理函数用于对计算机的电源进行管理。

进程和线程函数（Process and Thread）

一个 Win32 应用程序由一个或多个进程组成。在最简单的条件下，一个进程就是一个可执行程序，在该进程的环境中运行一个或多个线程。线程是操作系统分配处理器时间的基本单位。一个线程可以执行进程代码的任何部分，包括正被其他线程执行的部分。一个“纤度”（Fiber）是一个执行单位，必须由应用程序手工调度。“纤度”在调度它的线程环境中运行。

作业对象允许进程组被作为一个单位进行管理。作业对象是可命名、可得到及可共享的对象，用于控制与其相关的进程的属性。在作业对象上完成的操作会影响所有与该作业对象相关的进程。

进程和线程函数包括三类函数：进程和线程函数、作业对象函数、“纤度”函数。

注册函数（Registry）

注册表是一个系统定义的数据库，应用程序和系统构件可使用它来保存和获取配置数据。注册函数用于对注册表进行操作。

字符串处理函数（string Manipulation）

字符串处理函数用于对字符串进行处理。

结构化的异常处理函数（Structured Exception Handling）

异常是在程序执行过程中发生的一种事件，发生异常时需要执行正常的控制流程以外的代码。共有两种异常：硬件异常和软件异常。硬件异常是由 CPU 引发的，可能由于执行了某些指令序列而产生，比如除零操作，或访问一个无效的内存地址。软件异常是由应用程序或操作系统显式地引发。例如，当系统检测出一个无效的参数值时就会引发一个异常。

结构化的异常处理是一种同时处理软件异常和硬件异常的机制。因此，在程序中可用作对硬件和软件

异常一起进行处理。使用结构化的异常处理使用户可以完全控制对异常的处理，为调试器提供支持，并且对所有编程语言和机器都是可用的。

同步函数（Synchronization）

Win32 API 提供了各种方法来调整执行过程中的多个进程。同步函数为线程提供了一系列对资源访问进行同步的机制。

系统信息函数（System Information）

系统信息函数用于修改系统的配置、设置和属性。

系统消息函数（System Message）

系统消息函数用于向一些系统部件发送系统消息，比如应用程序、网络驱动器、系统级设备驱动器等。

系统关机函数（System Shutdown）

应用程序可使用系统关机函数将当前的用户退出系统、关机，或锁定工作站。

磁带备份函数（Tape Backup）

备份应用程序可使用磁带备份函数从磁带中读取数据，向磁带中写入数据，初始化磁带，以及获取磁带或磁带驱动信息。

时间函数（Time）

Microsoft Windows 和 Windows NT 提供了各种日期和时间函数，用于获取和设置系统及个别文件的日期和时间。

使用时间函数可以检查和修改日期及时间。

计时器函数（Timer）

计时器是一个内部例程，它反复地测量一个指定的时间间隔（以毫秒为单位）。

计时器函数用于对计时器进行操作。

工具帮助函数（Tool Help）

由“工具帮助库”所提供的函数可使用户更容易地获取有关当前正在执行的 Win32 应用程序的信息，为用户提供工具帮助服务。

窗口站和桌面函数（Window Station and Desktop）

窗口工作站和桌面函数主要是为 Win32 服务的开发人员提供的，用于对新的窗口工作站和桌面功能进行操作。开发由登录用户使用的典型应用程序的开发人员不必考虑窗口工作站和桌面。

Windows NT 4.0 访问控制函数（Windows NT Access-Control）

Windows NT 4.0 访问控制函数用于对安全描述符和访问控制列表（ACL）进行操作。在更高版本的 Windows NT 中也支持这些函数。

Windows NT 4.0 访问控制函数是 Microsoft Win32 提供的三套访问控制函数之一。

WinTrust 函数

WinTrust 函数用于对指定的主题进行指定确认。

1.2.6国际特性

这些特性有助于用户编写国际化的应用程序。Unicode 字符集使用 16 位的字符值来表示计算过程中所用的字符，比如各种符号，以及很多编程语言。国家语言支持（NLS）函数可帮助用户将应用程序本地化；输入方法编辑器（IME）函数（在 Windows 亚洲版中可用）用于帮助用户输入包含 Unicode 和 DCBS 字符的文本。

国际特性函数包括以下几类：

输入方法编辑器函数（Input Method Editor）

输入方法编辑器(IME)有助于简化用户的文本输入过程(文本中包含 Unicode 字符和双字节字符 DBCS)。

输入方法编辑器函数用于创建和管理 IME 窗口。

国家语言支持函数（National Language Support）

使用国家语言支持函数可以帮助 Win32 应用程序支持世界各地的不同语言，满足不同地区用户的特殊需要。

Unicode 和字符集函数（Unicode and Character Set）

Win32 API 通过 Unicode 和传统字符集可以支持国际上的很多不同的书写语言。Unicode 是一种世界通用的字符编码标准，它使用 16 位的字符值来表示各种字符，包括技术符号和出版所用的特殊字符。传统字符集是指以前所用的字符编码标准，比如 Windows ANSI 字符集，它是使用 8 位的字符值或 8 位值的组合来表示在指定的语言或地理区域中所用的字符。

Unicode 和字符集函数用于对字符集进行操作。

1.2.7网络服务

网络函数允许网络上的不同计算机的应用程序之间进行通讯。

网络函数用于在网络中的各计算机上创建和管理共享资源的连接，例如共享目录和网络打印机。

网络接口包括 Windows 网络函数、Windows 套接字（Socket）、NetBIOS、RAS、SNMP、Net 函数，以及网络 DDE。Windows 95 只支持这些函数中的一部分。

网络服务函数包括以下几类：

DLC 函数（DLC）

数据连接控制(DLC)接口是一个具有特殊目的的、不可路由的协议。它不是用于运行 Windows 和 Windows NT 的计算机之间的通讯，而是为运行 Windows 或 Windows NT 的计算机与 IBM 主机或直接连接到网络上的打印机之间提供了连通性。

网络函数（Net）

对于基于 OS/2 的服务器来说，Microsoft LAN Manager 所支持的 Net 函数提供了很多网络操作系统所需的功能，这些功能在本地操作系统中被忽略了。Windows NT、Windows 95 和 Windows 98 具有很多内置的网络功能，因此，有些原始的 Net 函数就不再支持了。

Windows NT、Windows 95 和 Windows 98 支持多种网络函数。Net 函数集提供了一些其他网络函数来覆盖的附加功能。另外，还可以使用这些函数来监测和管理基于 OS/2 的 LAN Mansger 服务器。

NetBIOS 函数

Win32 应用程序可以使用 Network Basic Input/Output System（NetBIOS）接口与网络中的其他计算机上的应用程序进行通讯。

NetBIOS 接口包括一系列显式命令，由一个被称为网络控制块（NCB）的结构提供。应用程序可以对任何支持 NetBIOS 接口的协议发出 NetBIOS 命令。

网络 DDE 函数（Networking DDE）

一个进程可以使用 Win32 API 提供的网络动态数据交换（DDE）函数与在网络中的不同计算机上运行的进程建立会话。

RAS 服务器管理函数（RAS Server Administration）

在 Windows NT 4.0 上，可使用 RAS 服务器管理函数来实现 RAS 服务器管理功能。Windows 95 不提供 RAS 服务器支持。

远程访问服务函数（Remote Access Service）

使用远程访问服务（RAS）可以使远程用户犹如直接连接到计算机网络上一样地访问一个或多个 RAS 服务器。

远程访问服务函数用于实现远程访问服务功能。

服务函数（Service）

Win32 API 提供了一套完整的服务函数。这些函数应该可以代替 NetService 函数，除非需要控制

LANManager2.x 服务器上的服务。

服务函数用于控制服务。一个服务就是一个应用程序，管理员可以使用服务控制程序接口来控制服务。

Windows 网络函数（Windows Networking）

Windows 提供的 Windows 网络（Wnet）函数使用户可以在应用程序中实现网络功能，而不需使用特殊的网络供应程序或物理的网络实现。原因是 Wnet 函数是网络无关的。

1.3 综述

出于篇幅和普适性考虑，本书将只收录 5 大类函数说明，它们分别是：窗口管理，图形设备接口，系统服务，国际特性，网络服务。

第二章 窗口管理函数（Window Control Function）

2.1 易用特性函数（Accessibility Features）

2.1.1SoundSentryProc

函数功能：该函数是一个库定义的回调函数，当 SOUNDSENTRY 易用特性存在，并且一个基于 win32 的应用程序（或者在窗口内运行的应用程序）通过计算机的内置扬声器发声时，它产生一个控制的可视化消息。

函数原型：LRESULT CALLBACK SOUNDSENTRYProc（DWORD dwMillisec，DWORD fdwEffect）；

参数：

Millisec：指定可视化消息的持续时间，以毫秒为单位。该消息是在一个基于 win32 的应用程序（或运行在窗口的应用程序）发声时显示出来的。

tdwEffect：指定要显示的可视化消息的类型。当前值通常应为 SSWF_CUSTOM。

返回值：如果可视化消息已经或将要正确显示，那么返回值为 TRUE，如果消息异步，并当调用该函数时其状态无效，那么应该返回 TRUE。如果出错使得消息无法显示，那么返回值为 FALSE。若想获得更多错误信息，请调用 GetLastError 函数。

备注：包含 SOUNDSENTRYProc 函数的库必须是一个 32 位的 DLL，并且该 DLL 必须导出名为 SOUNDSENTRYProc 的函数，即供外部调用和连接。SOUNDSENTRYProc 函数只是在应用程序或库调用 SystemParameterslnfo 函数之后才调用。SystemParameterslnfo 函数指定 SPI_SETSOUNDSENTRY 项的值以及 SOUNDSENTRY 结构的地址，在 SOUNDSENTRY 结构中，成员 iWindowsEffect 的值设为 SSWF_CUSTOM。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；库文件：用户自定义。

2.1.2SystemParametersinfo

函数功能：该函数查询或设置系统级参数。该函数也可以在设置参数中更新用户配置文件。

函数原型：BOOL SystemParametersinfo(UINT uiAction,UINT uiParam,PVOID pvParam,UINT fWinlni)；

参数：

uiAction：该参数指定要查询或设置的系统级参数。其取值如下；

SPI_GETACCESSTIMEOUT：检索与可访问特性相关联的超时段的信息，PvParam 参数必须指向某个 ACCESSTIMEOUT 结构以获得信息，并将该结构中的 cbSjze 成员和 ulParam 参数的值设为 sizeof（ACCESSTIMEOUT）。

SPI_GETACTIVEWINDOWTRACKING：用于 Windows 98 和 Windows NT 5.0 及以后的版本。它表示是否打开活动窗口跟踪（激活该窗口时鼠标置为开状态），pvParam 参数必须指向一个 BOOL 型变量（打开时接收值为 TRUE，关闭时为 FALSE）。

SPI_GETACTIVEWNDTRKZORDER；用于 Windows 98 和 Windows NT 5.0 及以后版本。它表示通过活动窗口跟踪开关激活的窗口是否要置于最顶层。pvParam 参数必须指向一个 BOOL 型变量，如果要置于顶层，那么

该变量的值为 TRUE，否则为 FALSE。

SPI_GETACTIVEWNDTRKTIMEOUT：用于 Windows 98 和 Windows NT 5.0 及以后版本。它指示活动窗口跟踪延迟量，单位为毫秒。pvParam 参数必须指向 DWORD 类型变量，以接收时间量。

SPI_GETANIMATION：检索与用户活动有关的动画效果。pvParam 参数必须指向 ANIMATIOINFO 结构以接收信息。并将该结构的 cbSize 成员和 ulParam 参数置为 sizeof（ANIMATIONINFO）。

SPI_GETBEEP：表示警告蜂鸣器是否是打开的。pvParam 参数必须指向一个 BOOL 类型变量，如果蜂鸣器处于打开状态，那么该变量的值为 TRUE，否则为 FALSE。

SpI_GETBORDER：检索决定窗口边界放大宽度的边界放大因子。pvParam 参数必须指向一个整型变量以接收该值。

SPI_GETDEFAULTINPUTLANG：返回用于系统缺省输入语言的键盘布局句柄。pvParam 参数必须指向一个 32 位变量，以接收该值。

SPI_GETCOMBOBOXANIMATION：用于 Windows 98 和 Windows NT 5.0 及以后版本。它表示用于组合柜的动打开效果是否允许。pvParam 参数必须指向一个 BOOL 变量，如果允许，那么变量返回值为 TRUE，否则为 FALSE。

SPI_GETDRAGFULLWINDOWS：确定是否允许拖拉到最大窗口。pvParam 参数必须指向 BOOL 变量，如果允许，返回值为 TRUE，否则为 FALSE。对于 Windows 95 系统，该标志只有在安装了 Windows plusl 才支持。

SPI_GETFASTTASKSWITCH：该标志已不用！以前版本的系统使用该标志来确定是否允许 Alt＋Tab 快速任务切换。对于 Windows 95、Windows 98 和 Windows NT 4.0 版而言，快速任务切换通常是允许的。

SPI_GETFILTERKEYS：检索有关 FILTERKEYS（过滤键）易用特征信息。pvParam 参数必须指向接收信息的 filterkeys 结构，并将该结构中的 cbSze 成员和 ulParam 参数的值设为 sizeof（FILTERKEYS）。

SPI_GETFONTSMOOTHING：表示字体平滑处理特征是否允许。该特征使用字体保真技术，通过在不同灰度级上涂上像素使字体曲线显得更加平滑。参数 pvParam 必须指向 BOOL 类型变量，如果该特征被允许，那么返回值为 TRUE，否则为 FALSE。对于 Windows 95 系统，该标志只有在安装了 Windows plusl 才支持。

SPI_GETFOREGROUNDFLASHCOUNT：用于 Windows 98 和 Windows NT 5.0 及以后版本。它表示在拒绝前台切换申请时系统闪烁任务条按钮的次数。参数 pvParam 必须指向 DWORD 变量，以接收该值。

SPI_GETFOREGROUNDLOCKTIMEOUT：用于 Windows NT 5.0 及以后版本或 Windows 98。它表示在系统禁止应用程序强行将自己进入前台期间的时间量，单位为毫秒. 参数 pvParam 必须指向 DWORD 变量以接收时间值。

SPI_GETGRADIENTCAPTIONS：用于 Windows 98 和 Windows NT 5.0 及以后版本。它表示是否允许有用于窗口标题栏的倾斜效果。参数 pvParam 必须指向 BOOL 变量，其值在允许时为 TRUE，禁止时为 FALSE。

SPL_GETGRIDGRANULARITY：检索桌面大小网格的当前颗粒度值。参数 pVparam 必须指向一个整型变量以接收该值。

SPI_GETHIGHCONTRAST:用于 Windows 95 及更高版本、Windows NT 5.0 及以后版本。检索与 HighContrast 易用特征有关的信息。pvParam 参数必须指向用于接收该信息的 HIGHCONTRAST 结构，该结构中的。cbSize 成员和 ulParam 参数的值应设为 sizeof（NIGHCONTRAST）。

SPI_GETICONMETRICS：检索与图标有关的度量信息。参数 pvParam 必须指向一个 ICONMETRICS 结构以接收信息。该结构中的。cbSize 成员和 ulParam 参数的值应设为 sizeof（ICONMETRICS）。

SPI_GETICONTITLELOGFONT：检索当前图标标题字体的逻辑字体信息。参数 ulParam 规定了 logfont 结构的大小，参数 pvParam 必须指向要填充的 Iogfont 结构。

SPI_GETICONTITLEWRAP：确定是否允许图标标题环绕。pvParam 参数必须指向一个 BOOL 类型变量，该变量的值在允许时为 TRUE，否则为 FALSE。

SPI_GETKEYBOARDDELAY：检索键盘重复击键延迟设置，该值范围从 0（大约 250ms 延迟）到 3（大约 1 秒延迟）。与该范围里每一个值相关的实际延迟时间可能与硬件有关。pvParam 参数必须指向一个整型变量以接收设置值。

SPI_GETKEYBOARDPREF：用于 Windows 95 及以后版本。Windows NT 5.0 及以后版本。它确定用户是否依赖键盘而非鼠标，是否要求应用程序显示键盘接口，以免隐藏。pvParam 参数必须指向一个 BOOL 类型变量，如果用户依赖键盘，那么该变量取值为 TRUE，否则为 FALSE。

SPI_GETKEYBOARDSPEED：检索键盘重复击键速度设置情况，该值范围从 0（大约 30 次/秒）至 31（大约 25 次/秒）。实际的击键速率与硬件有关，而且变动的线性幅度有可能高达 20%。参数 pvParam 必须指向 DWORD 变量以接收设置值。

SPI_GETLISTBOXSMOOTHSCROLLING：用于 Windows 98 和 Windows NT 5.0 及以后版本。表示是否允许有列表栏的平滑滚动效果。pvParam 参数必须指向 BOOL 变量，如果允许，则该值为 TRUE，否则为 FALSE。

SPI_GETLDWPOWERACTIVE：确定是否允许屏幕保护的低电压状态。如果允许，那么指向 BOOL 变量的 pvParam 参数会接收到 TRUE 值，否则为 FALSE。对于 Windows 98,该标志对 16 位和 32 位应用程序都支持。

对于 Windows 95，该标志只支持 16 位应用程序。对于 Windows NT，在 Windows NT 5.0 及以后版本中支持 32 位应用程序，对 16 位应用程序则不支持。

SPI_GETLOWPOWERTIMEOUT：检索用于屏幕保护的低电压状态超时值。pvParam 参数必须指向一个整型变量，以接收该值。对于 Windows 98 该标志支持 16 位和 32 位应用程序。对于 Windows95，该标志只支持 16 位应用程序。对于 Windows NT,该标志支持 Windows NT 5.0 及以后版本上的 32 位应用程序。不支持 16 位应用程序。

SPI_GETMENUDROPALIGNMENT。确定弹出式菜单相对于相应的菜单条项是左对齐，还是右对齐、参数 pvParam 必须指向一个 BOOL 类型变量，如果是左对齐。那么该变量值为 TRUE，否则为 FALSE。SPI_GETMINIMIZEDMETRICS:检索最小化窗口有关的度量数据信息。参数 pvParam 必须指向 MINIMIZEDMETRCS 结构，以接收信息。该结构中的 cbSize 和 ulParam 参数的值应设为 sizeof（MINIMIZEDMETRICS）。

SPI_GETMOUSE：检索鼠标的 2 个阈值和加速特性。pvParam 参数必须指向一个长度为 3 的整型数组，分别存储此值。

SPI_GETMOUSEHOVERHEGHT：用于 Windows NT 4.0 及以后版本或 Windows 98。获得在 TrackMouseEvent 事件中，为产生 WM_MOUSEOVER 消息而鼠标指针必须停留的矩形框的高度,以像素为单位。参数 pvParam 必须指向一个 UINT 变量以接收这个高度值。

SPI_GETMOUSEHOVERTIME：用于 Windows NT 4.0 及以后版本、Windows 98，获得在 TrackMouseEvent 事件中，为产生 WM_MOUSEOVER 消息而鼠标指针必须停留在矩形框内的时间，单位为毫秒。参数 pvParam 必须指向一个 UINT 变量以接收该时间值。

SPI_GETMOUSEHOVERWIDTH：用于 Windows NT 4.0 及以后版本、Windows 98。获得在 TrackMouseEvent 事件中，为产生 WM_MOUSEOVER 消息而鼠标指针必须停留的矩形框的宽度,以像素为单位。参数 pvParam 必须指向一个 UINT 变量以接收这个宽度值。

SPI_GETMOUSEKEYS：检索与 MOUSEKEYS 易用特征有关的信息，pvParam 参数必须指向某个 MOUSEKEYS 结构，以获取信息。应将结构的 cbSize 成员和 ulParam 参数设置为 sizeof（MOUSEKEYS）。

SPI_GETMOUSESPEED：用于 Windows NT 5.0 及以后版本、Windows 98。检索当前鼠标速度。鼠标速度决定了鼠标移动多少距离，鼠标的指针将移动多远。参数 pvParam 指向一个整型变量，该变量接收 1（最慢）至 20（最快）之间的数值。缺省值为们 10。这个值可以由最终用户使用鼠标控制面板应用程序或使用调用了 SPI_SETMOUSESPEED 的应用程序来设置。

SPI_GETMOUSETRAILS：用于 WpvParam 必须指向一个 BOOL 类型变量，如果是左对齐。那么该变量值为 TRUE，否则为 FALSE。

SPI_GETMINIMIZEDMETRICS：检索最小化窗口有关的度量数据信息。参数 pvParam 必须指向 MINIMIZEDMETRCS 结构，以接收信息。该结构中的 cbSize 和 ulParam 参数的值应设为 sizeof（MINIMIZEDMETRICS）。

SPI_GETMOUSE：检索鼠标的 2 个阈值和加速特性。pvParam 参数必须指向一个长度为 3 的整型数组，分别存储此值。

SPI_GETMOUSEHOVERHEGHT：用于 Windows NT 4.0 及以后版本或 Windows 98。获得在 TrackMouseEvent 事件中，为产生 WM_MOUSEOVER 消息而鼠标指针必须停留的矩形框的高度,以像素为单位。参数 pvParam 必须指向一个 UINT 变量以接收这个高度值。

SPI_GETMOUSEHOVERTIME：用于 Windows NT 4.0 及以后版本、Windows 98，获得在 TrackMouseEvent 事件中，为产生 WM_MOUSEOVER 消息而鼠标指针必须停留在矩形框内的时间，单位为毫秒。参数 pvParam 必须指向一个 UINT 变量以接收该时间值。

SPI_GETMOUSEHOVERWIDTH：用于 Windows NT 4.0 及以后版本、Windows 98。获得在 TrackMouseEvent 事件中，为产生 WM_MOUSEOVER 消息而鼠标指针必须停留的矩形框的宽度,以像素为单位。参数 pvParam 必须指向一个 UINT 变量以接收这个宽度值。

SPI_GETMOUSEKEYS：检索与 MOUSEKEYS 易用特征有关的信息，pvParam 参数必须指向某个 MOUSEKEYS 结构,以获取信息。应将结构的 cbSize 成员和 ulParam 参数设置为 sizeof(MOUSEKEYS)。SPI_GETMOUSESPEED：用于 Windows NT 5.0 及以后版本、Windows 98。检索当前鼠标速度。鼠标速度决定了鼠标移动多少距离，鼠标的指针将移动多远。参数 pvParam 指向一个整型变量，该变量接收 1（最慢）至 20（最快）之间的数值。缺省值为们 10。这个值可以由最终用户使用鼠标控制面板应用程序或使用调用了 SPI_SETMOUSESPEED 的应用程序来设置。

SPI_GETMOUSETRAILS：用于 Windows 95 及更高版本。它用来表示是否允许 MouseTrails（鼠标轨迹）。该特征通过简单地显示鼠标轨迹并迅速擦除它们来改善鼠标的可见性。参数 prParam 必须指向一个整型变量来接收该值。如果这个值为 0 或 1，那么表示禁止该特征。如果该值大于 1，则说明该特征被允许，并且该值表示在鼠标轨迹上画出的光标数目。参数 ulParam 不用。

SPI_GETNONCLIENTMETRICS：检索与非最小化窗口的非客户区有关的度量信息。参数 pvParam 必须指向 NONCLIENTMETRICS 结构，以便接收相应值。该结构的。cbSize 成员与 ulParam 参数值应设为 sizeof（NONCLIENTMETRICS)。对于 Windows 98，该标志支持 16 位和 32 位应用程序。对于 Windows 95，该标志只支持 16 位应用程序。对于 Windows NT 该标志在 NT 5.0 及以后版本中支持 32 位应用程序，不支持 16 位应用程序。

SPI_GETPOWEROFFACTIVE：确定是否允许屏幕保护中关电。TRUE 表示允许，FA 参数 pvParam 必须指定 SERIALKEYS 结构来接收信息。该结构中的 cbSize 成员和 ulParam 参数的值要设为 sizeof（SERIALKEYS)。

SPI_GETSHOWSOUNDS：确定 ShowSounds 易用特性标志是开或是关。如果是开，那么用户需要一个应用程序来可视化地表达信息，占则只能以听得见的方式来表达。参数 pvParam 必须指向一个 BOOL 类型变量。该变量在该特征处于开状态时返回 TRUE，否则为 FALSE。使用这个值等同于调用 GetSystemMetrics（SM_SHOWSOUNDS)。后者是推荐使用的调用方式。

SPI_GETSNAPTODEFBUTTON：用于 Windows NT 4.0 及以后版本、Windows 98：确定 Snap-TO-Default-Button（转至缺省按钮）特征是否允许。如果允许，那么鼠标自动移至缺省按钮上，例如对话框的“Ok”或“Apply”按钮。pvParam 参数必须指向 Bool 类型变量，如果该特征被允许，则该变量接收到 TRUE，否则为 FALSE。

SPI_GETSOUNDSENTRY:检索与 SOUNDSENTRY 可访问特征有关的信息。参数 pvParam 必须指向 SOUNDSENTRY 结构以接收信息。该结构中的。cbSize 或员和 ulParam 参数的值要设为 sizeof（SOUNDSENTRY)。

SPI_GETSTICKYKEYS：检索与 StickyKeys 易用特征有关的信息。参数 pvParam 必须指向 STICKYKEYS 结构以获取信息。该结构中的 cbSze 成员及 ulParam 参数的值须设为 sizeof（STICKYKEYS)。

SPI_GETSWITCHTASKDISABLE：用于 Windows NT 5.0、Windows 95 及以后版本，确定是否允许 Alt＋Tab 和 AIt＋Esc 任务切换。参数 pvParam 必须指向 UINT 类型变量，如果禁止任务切换，那么返回值为 1，否则为 0。在缺省情况下，是允许进行任务切换的。

SPI_GETTOGGLEKEYS：检索与 ToggleKeys 易用特性有关的信息。参数 pvParam 必须指向 TOGGLEKEYS 结构以获取信息。该结构中的 cbSize 成员和 ulParam 参数值要设置 sizeof（TOGGLEKEYS)。

SPI_GETWHEELSCROLLLINES：用于 Windows NT 4.0 及以后版本、Windows 98。当前轨迹球转动时，获

取滚动的行数。参数 pvParam 必须指向 UINT 类型变量以接收行数。缺省值是 3。

SPI_GETWINDOWSEXTENSION:在 Windows 95 中指示系统中是否装了 Windows Extension 和 Windows Plus!。

参数 ulParam 应设为 1。而参数 pvParam 则不用。如果安装了 Windows Extenson,那么该函数返回 TRUE,否则为 FALSE。

SPI_GETWORKAREA：检索主显示器的工作区大小。工作区是指屏幕上不被系统任务条或应用程序桌面工具遮盖的部分。参数 pvParam 必须指向 RECT 结构以接收工作区的坐标信息，坐标是用虚拟屏幕坐标来表示的。为了获取非主显示器的工作区信息，请调用 GetMonitorlnfo 函数。参数 ulParam 指定宽度，单位是像素。

SPI_ICONVERTICALSPACING：设置图标单元的高度。参数 ulParam 指定高度，单位是像素。

SPI_LANGDRIVER：未实现。

SPI_SCREENSAVERRUNNING：改名为 SPI_SETSCREENSAVERRUNNING。

Spl_SETACCESSTIMEOUT：设置与可访问特性有关的时间限度值，参数 pvParam 必须指向包含新参数的 ACCESSTIMEOUT 结构，该结构的 cbSize 成员与 ulParam 参数的值要设为 sizeof（ACCESSTMEOUT)。

SPI_SETACTIVEWINDOWTRACKING：用于 Windows NT 5.0 及以后版本、Windows 98。设置活动窗口追踪的开或关，如果把参数 pvParam 设为 TRUE，则表示开。pvParam 参数为 FALSE 时表示关。

SPI_SETACTIVEWNDTRKZORDER：用于 Windows NT 5.0 及以后版本、Windows 98。表示是否把通过活动窗口跟踪而激活的窗口推至顶层。参数 pvParam 设为 TRUE 表示推至顶层，FALSE 则表示不推至顶层。

SPI_SETACTIVEWNDTRKTIMEOUT：用于 Wlindows NT 5.0 及以后版本、Windows 98。设置活动窗口跟踪延迟。

参数 pvParam 设置在用鼠标指针激活窗口前需延迟的时间量，单位为毫秒。

SPI_SETBEEP：将警蜂器打开或关闭。参数 ulParam 指定为 TRUE 时表示打开，为 FALSE 时表示关闭。

SPI_SETBORDER：设置确定窗口缩放边界的边界放大因子。参数 ulParam 用来指定该值。

SPI_SETCOMBOBOXANIMATION：用于 Windows NT 5.0 及以后版本和 Windows 98。允许或禁止组合滑动打开效果。如果设置 pvParam 参数为 TRUE，则表示允许有倾斜效果，如果设为 FALSE 则表示禁止。

SPI_SETCURSORS：重置系统光标。将 ulParam 参数设为 0 并且 pvParam 参数设为 NULL。

SPI_SETDEFAULTINPUTLANG：为系统 Shell（命令行解器）和应用程序设置缺省的输入语言。指定的语言必须是可使用当前系统字符集来显示的。pvParam 参数必须指向 DWORD 变量，该变量包含用于缺省语言的键盘布局句柄。

SpI_SETDESKpATTERN：通过使 Windows 系统从 WIN.INI 文件中 pattern=设置项来设置当前桌面模式。

SPI_SETDESKWALLPAPER：设置桌面壁。pvParam 参数必须指向一个包含位图文件名，并且以 NULL（空）结束的字符串。

SPI_SETDOUBLECLICKTIME：设 ulParam 参数的值为目标双击时间。双击时间是指双击中的第 1 次和第 2 次点击之间的最大时间，单位为毫秒。也可以使用 SetDoubleClickTime 函数来设置双击时间。为获取当前双击时间，请调用 GetDoubleClickTime 函数。

SPI_SETDOUBLECLKHEGHT：将 ulParam 参数的值设为双击矩形区域的高度。双击矩形区域是指双击中的第 2 次点击时鼠标指针必须落在的区域，这样才能记录为双击。

SPI_SETDOUBLECLKWIDTH：将 ulParam 参数的值设为双击矩形区域的宽度。

SPI_SETDRAGFULLWINDOWS：设置是否允许拖至最大窗口。参数 uIParam 指定为 TRUE 时表示为允许，为 FALSE 则不可。对于 Windows 95，该标志只有在安装了 Windows plusl 才支持。

SPI_SETDRAGHEIGHT：设置用于检测拖拉操作起点的矩形区域的高度，单位为像素。参考 GETSYSTEMMETRICS 函数的 nlndex 参数中的 SM_CXDRAG 和 SM_CYDRAG。

SPI_SETDRAGWIDTH：设置用于检测拖拉操作起点的矩形区域的宽度，单位为像素。

SPI_SETFASTTASKSWITCH：该标志已不再使用。以前版本的系统使用此标志来允许或不许进行 Alt＋Tab 快速任务切换。对于 Windows 95、Windows 98 和 Windows NT 4.0，通常都允许进行快速任务切换。参考

SPI_SETSWITCHTASKDISABLE。

SPI_SETFILTERKEYS：设置 FilterKeys 易用特性的参数。参数 pvParam 必须指向包含新参数的 FILTERKEYS 结构，该结构中的 cbSize 成员和参数 ulParam 的值应设为 sizeof（FILTERKEYS）。

SPI_SETFONTSMOOTHING：允许或禁止有字体平滑特性。该特性使用字体保真技术，通过在不同灰度级上涂画像素点来使得字体曲线显得更加平滑，为了允许有该特性，参数 ulParam 应设为 TRUE 值，否则为 FALSE。对于 Windows 95，只有在安装了 Windows plusl 才支持该标志。

SPI_SETFOREGROUNDFLASHCOUNT：用于 Windows 98 和 Windows NT 5.0 及以后版本。设置 SetForegroundWindow 在拒绝前台切换申请时闪烁任务拦按钮的次数。

SPI_SETFOREGROUNDLOCKTIMEOUT：用于 Windows 98 和 Windows NT 5.0 及以后版本。它用来设置在用户输入之后，系统禁止应用程序强行将自己进入前台期间的时间长度，单位为毫秒。参数 pvParam 设置这个新的时间限度值。

SPI_SETGRADIENTCAPTIONS：用于 Windows 98 和 Windows NT 5.0 及以后版本。允许或禁止窗口标题栏有倾斜效果。如果允许则将参数 pvParam 设置为 TRUE，否则设为 FALSE。有关倾斜效果方面更多信息，请参考 GetSysColor 函数。

SPI_SETGRIDGRANULARITY：将桌面缩放时网格的颗粒度值设置为参数 ulParam 中的值。

SPI_SETHANDHELD：内部使用，应用程序不应使用该值。

SPI_SETHIGHCONTRAST：用于 Windows 95 及以后版本、Windows NT 5.0 及以后版本。设置 HighContrast 可访问特性的参数。参数 pvParam 必须指向 HIGHCONTRAST 结构，该结构包含新的参数。该结构中的 cbSize 成员及参数 ulParam 的值设为 sizeof（HIGHCONTRAST）。

SPI_SETICONMETRICS：设置与图标有关的信息。参数 pvParam 必须指向包含新参数的 ICONMETRICS 结构，另外还要将参数 ulParam 和该结构中的 cbSize 成员的值设置为 sizeof（ICONMETRICS）。

SPI_SETICONS：重新加载系统图标。参数 ulParam 的值应设为 0，而 pvParam 参数应设为 NULL。

SPI_SETICONTITLELOGFONT：设置用于图标标题的字体。参数 ulParam 指定为 logfont 结构的大小，而参数 pvParam 必须指向一个 LOGFONT 结构。

SPI_SETICONTITLEWRAP：打开或关闭图标标题折行功能。若想打开折行功能，则把参数 ulParam 设为 TRUE，否则为 FALSE。

SPI_SETKEYBOARDDELAY：设置键盘重复延迟。参数 ulParam 必须指定为 0，1，2 或 3。其中 0 表示设置为最短延迟（大约 250ms）3，表示最大延迟（大约 1 秒）。与每个值对应的实际的延迟时间根据硬件情况有可能有些变化。

SPI_SETKEYBOARDPREF：用于 Windows 95 及以后版本、Windows NT 5.0 及以后版本，设置键盘优先序。如果用户依赖键盘而不是鼠标，那么可将参数 ulParam 指定为 TRUE，否则设为 FALSE，并且要求应用程序显示而不隐蔽键盘接口。

SPI_SETKEYBOARDSPEED：设置键盘重击键速度。参数 ulParam 必须指定一个从 0 到 31 的值，其中 0 表示设置成最快速度（大约 30 次/秒），31 表示设置为最低速度（大约 2。5 次/秒），实际的重速率与硬件有关，而且可能变动幅度高达 20%。如果 ulParam 大于 31，那么该参数仍设置为 31。

SPI_SETLANGTOGGLE：为输入语言间切换设置热键集。参数 ulParam 和 pvParam 不用。该值通过读取注册表来设置键盘属性表单中的快捷键。在使用该标志之前必须设置注册表，注册表中的路径是“1”=Alt＋shift，”2”=Ctrl+shift，”3”=none（无）。

SPI_SETLISTBOXSMOOTHSCROLLING：用于 Windows 98 和 Windows NT 5.0 及以后版本。允许或不许列表栏有平滑滚动效果。参数 pvParam 设置为 TRUE 表示允许有平滑滚动效果，为 FALSE 则表示禁止。

SPI_SETLOWPOWERACTIVE：激活或关闭低电压屏幕保护特性。参数 ulParam 设为 1 表示激活，0 表示关闭。参数 pvParam 必须设为 NULL。对于 Windows 98,该标志支持 16 位和 32 位应用程序。对于 Windows 95，该标志只支持 16 位应用程序。对于 Windows NT. 该标志只支持 NT 5.0 及以后版本的 32 位应用程序，不支持 16 位应用程序。

SPI_SETLOWPOWERTIMEOUT：用于设置低电压屏幕保护中的时间值（也称超时值，即在超过某一时间段后自动进行屏幕保护），单位为秒。uIParam 参数用来指定这个新值。参数 pvParam 必须为 NULL。对于 Windows98,该标志支持 16 位和 32 位应用程序。对于 Windows 95,该标志只支持 16 位应用程序。对于 Windows NT 该标志只支持 NT 5.0 及以后版本的 32 位应用程序，不支持 16 位应用程序。

SPI_SETMENUDROPALIGNMENT:设置弹出或菜单的对齐方式。参数 ulParam 指定为 TRUE 时表示是右对齐，FALSE 时为左对齐。

SPI_SETMINIMIZEDMETRICS：设置与最小化窗口有关的数据信息，参数 pvParam 必须指向包含新参数的 MINIMIZEDMETRICS 结构。该结构中的 cbSize 成员与 ulParam 参数的值应设为 sizeof(MINMIZEDMETRICS)。

SPI_SETMOUSE：设置鼠标的两个阀值和加速率。参数 pvParam 必须指向一个长度为 3 的数组，以指定这些值。详细请参考 mouse_event。

SPI_SETMOUSEBUTTONSWAP：调换或恢复鼠标左右按钮的含义，为 FALSE 时表示恢复原来的含义。

SPI_SETMOUSEHOVERHEGHT：用于 Windows 98 和 Windows NT 4.0 及以后版本。设置鼠标指针停留区域的高度,以像素为单位。鼠标指针在此区域停留是为了让 TrackMouseEvent 产生一条 WM_MUOSEHOVER 消息，参数 ulParam 用来设置此高度值。

SPI_SETMOUSEHOVERTIME：用于 Windows 98 和 Windows NT 4.0 及以后版本。设置鼠标指针为了让 TrackMouseEvent 产生 WM_MOUSEHOVER 事件而在停留区域应停留的时间。该标志只有在将调用 dwHoverTime 参数中的 HOVER_DEFAULT 值传送到 TrackMouseEvent 时才使用。参数 ulParam 设置这个新的时间值。

SPI_SETMOUSEHOVERWIDTH：用于 Windows 98 和 Windows NT 4.0 及以后版本。设置鼠标指针停留区域的宽度，以像素为单位。参数 ulParam 设置该新值。

SPI_SETMOUSEKEYS：设置 MouseKeys 易用特性的参数。参数 pvParam 必须指向包含新参数的 MOUSEKEYS 结构。结构中的 cbSize 成员与参数 ulParam 的值应设为 sizeof（MOUSEKEYS）。

SPI_SETMOUSESPEED：用于 Windows NT 5.0 及以后的版本和 Windows 98，设置当前鼠标速度。参数 pvParam 必须指向一个 1（最慢）至 20（最快）之间的整数。缺省值是 10。一般可以使用鼠标控制面板应用程序来设置该值。

SPI_SETMOUSETRAILS：用于 Windows 95 及以后版本：允许或禁止有 MoouseTrails（鼠标轨迹）特性。该特性通过简短地显示鼠标光标轨迹，并迅速地擦除它们来提高鼠标的可见度。禁止该特性可将参数 ulParam 设为 0 或 1，允许时,将 ulParam 设置为一个大于 1 的数值，该值表示轨迹中画出的光标个数。

SPI_SETNONCLIENTMETRICS：设置与非最小化窗口的非客区有关的数据信息，参数 pvParam 必须指向 NONCLIENTMETRICS 结构，该结构包含新的参数。其成员 cbSzie 和参数 ulParam 的值应设为 sizeof（NONCLIENTMETRICS)。

SPI_SETPENWINDOWS；用于 Windows 95 及以后版本：指定是否加载笔窗口，当加载时，参数 ulParam 设为 TRUE，不加载时为 FALSE。参数 pvParam 为 NULL。

SPI_SETPOWEROFFACTIVE：激活或关闭屏幕保护特性参数。ulParam 设为 1 表示激活，0 表示关闭。参数 pvParam 必须为 NULL。对于 Windows 98，该标志支持 16 位和 32 位应用程序。对于 Windows 95，该标志只支持 16 位应用程序。对于 Windows NT，该标志支持 Windows NT 5.0 及以后版本的 32 位应用程序，不支持 16 位应用程序。

SPI_SETPOWEROFFTIMEOUT：设置用于关闭屏幕保护所需的时间值（也称超时值)。参数 ulParam 指定该值。参数 pvParam 必须为 NULL。对于 Windows 98. 该标志支持 16 位和 32 位应用程序。对于 Windows 95，该标志只支持 16 位应用程序。对于 Windows NT,该标志支持 Windows NT 5.0 及以后版本上的 32 位应用程序，不支持 16 位应用程序。

SPI_SETSCREENREADER；用于 Windows 95 及以后版本、Windows NT 5.0 及以后版本，表示屏幕审阅程序是否运行。参数 uiparm 指定为 TRUE 表示运行该程序，FALSE 则不运行。

SPI_SETSCREENSAVERRUNNING：用于 Windows 95 及以后版本，内部使用。应用程序不应该使用此标志 SPI_SETSETSCREENSAVETIMEOUT：参数 ulParam 值为屏幕保护器时间限度值。该值是一个时间量，以秒为单

位，在屏幕保护器激活之前，系统应该一直是空闲的，超过这个值就激活屏幕保护器。

SPI_SETSERIALKEYS：用于 Windows 95 及以后版本：设置 SerialKeys 易用特性的参数。参数 pvParam 必须指向包含新参数的 SERIALKEYS 结构，其成员 cbSize 和参数 ulParam 应设为 sizeof（SERIALKEYS)。

SPI_SETSHOWSOUNDS：将 ShowSounds 易用特性设置为打开或关闭。参数 ulParam 指定为 TRUE 时表示打开，FALSE 表示关闭。

SPI_SETSNAPTODEFBUTTON：用于 Windows NT 4.0 及以后版本、Windows 98。允许或禁止有 snap-to-default-button（跳转至缺省按钮）特性。如果允许，那么鼠标光标会自动移至缺省按钮上，例如对话柜中的 OK 或”apply”按钮。参数 ulParam 设为 TRUE 表示允许该特性，FALSE 表示禁止。

SPI_SETSOUNDSENTRY:设置 SOUNDSENTRY 易用特性的参数。参数 pvParam 必须指向 SOUNDSENTRY 结构，该结构包含新参数，其成员 cbSize 和参数 ulParam 的值应设为 sizeof（SOUNDSENTRY)。

SPI_SETSTICKYKEYS：设置 stickykeys 可访问特性的参数。参数 pvParam 必须指向包含新参数的 stickykeys 结构，其成员 cbSize 和 ulParam 参数的值要设为 sizeof（STICKYKEYS)。

SPI_SETSWITCHTASKDISABLE：用于 Windows NT 5.0 及以后版本，允许或禁止有 Alt＋Tab 和 Alt＋Esc 任务切换特性。参数 ulParam 设为 1 表示允许有该特性，设为 0 则表示禁止。缺省情况下是允许有任务切换特性的。

SPI_SETTOGGLEKEYS：设置 togglekeys 可访问特性的参数，参数 PvParam 必须指向 TOGGLEKEYS 结构，该结构中包含新的参数。其成员 cbSize 和参数 ulParam 的值要设为 sizeof（togglekeys)。

SPI_SETWHEELSCROOLLLINES：用于 Windows 98 和 Windows NT 4.0 及以后版本。设置当鼠标轨迹球转动时

要滚动的行数，滚动的行数是由参数 ulParam 设置的，该行数是在鼠标轨迹球滚动，并且没有使用修改键时的滚动行数。如果该数值为 0，那么不会发生滚动，如果滚动行数比可见到的行数要大，尤其如果是 WHEEL_PAGESCROLL（＃defined sa UINT_MAX)，那么滚动操作应该被解释成在滚动条的下一页或上一页区点击一次。

SPI_SETWORKAREA：设置工作区域大小。工作区是指屏幕上没有被系统任务栏或桌面应用程序桌面工具遮盖的部分。参数 pvParam 是一个指针。指向 RECT 结构，该结构规定新的矩形工作区域，它是以虚拟屏幕坐标来表达的。在多显示器系统中,该函数用来设置包含特定矩形的显示器工作区域。如果 PvParam 为 NULL，那么该函数将主显示器的工作区域设为全屏。

ulParam：与查询或设置的系统参数有关。关于系统级参数的详情，请参考 uiAction 参数。否则在没有指明情况下，必须将该参数指定为 0。

pvParam：与查询或设置的系统参数有关。关于系统级参数的详情，请参考 uiAction 参数。否则在没有指明情况下，必须将该参数指定为 NULL。

fWinlni：如果设置系统参数，则它用来指定是否更新用户配置文件（Profile)。亦或是否要将 WM_SETTINGCHANGE 消息广播给所有顶层窗口,以通知它们新的变化内容。该参数可以是 0 或下列取值之一：

SPIF_UPDATEINIFILE：把新的系统参数的设置内容写入用户配置文件。

SPIF_SENDCHANGED：在更新用户配置文件之后广播 WM_SETTINGCHANGE 消息。

SPI_SENDWININICHANGE 与 SPIF_SENDCHANGE 一样。

返回值：如果函数调用成功，返回值非零：如果函数调用失败，那么返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

备注：该函数一般与应用程序，例如控制面板一起使用。它可以允许用户对 Windows 任意进行定制。

键盘布局名称是从对应于布局的 16 进制语言标识符引生而来的。例如，美国英语（U.S.Englisth）的语言标识符为“ 0×0409”，则主美国英语键盘布局命名为“00000409”其他的键盘布局如 Dvotak 等，命名为”00010409”、”00020409”等，关于此的列表参见 MAKELANGID 宏。

Windows CE 操作系统只支持下列 uiAction 值：

SPI_GETBATTERYIDLETIMEOUT：在 WINDOWS CE 没有因用户操作而挂起之前，干电池电源能坚持给系统

供电的时间量可以使用该标志得到。以秒为单位，如果 pvParam 为 0，那么该标志被忽略。

SPI_GETEXTERNALIDLETIMEOUT：在 Windows CE 没有因用户操作而挂起之前，交流电源能坚持给系统供电时间的时间量可以使用该标志得到。参数 pvParam 指向一个 DWORD 类型变量，以返回时间值，单位为秒。如果 pvParam 为 0，那么该标志被忽略。

SPI_GETMOUSE：检索鼠标的两个阈值和速度。

SPI_GETOEMINFO：返回一个字符串，该字符串包含型号和制造商名称。参数 ulParam 指定为 pvParam 参数中缓冲区的长度，在成功返回时，参数 pvParam 中包含 Unicode 字符集中的字符串。

SPI_GETPLATFORMTYPE：返回一个指定 Windows CE 设备类型的字符串，例如“H/PC”。参数 ulParam 规定 pvParam 参数缓冲区的长度，后者在成功返回时包含一个 Unicode 字符集中的字符串。该字符串允许象 H/PC EXPLORER 一样的应用程序来确定设备类型。

SPI_GETWAKEUPDLETIMEOUT：在用户通知重新激活某个挂起的设备之后，可获取的 Windows CE 延缓响应的时间量。参数 pvParam 指向一个 DWORD 类型变量以返回时间值，单位为秒。如果 pvParam 值为 0 那么该标志被忽略。

SPI_GETWORKAREA：检索工作区大小。工作区是指没有被任务遮盖的屏幕部分。

SPI_SETBATTERYidletimeout：在 Windows CE 没有因用户操作而挂起之前，电池电源能坚持给系统供电的时间量可以使用该标志来设置。只要键盘或触摸屏处在活动状态（有输入），那么 Windows CE 操作系统及电池电源仍将工作。参数 ulParam 指定要设置的时间，单位为秒。如果 ulParam 设置为 0，那么该标志被忽略。

SPI_SETEXTERNALIDLETIMEOUT：在 Windows CE 没有因用户操作而挂起之前，交流电源能坚持给系统供电的时间量可以使用该标志来设置。只要键盘或触摸屏幕处在活动状态，那么 Windows CE 操作系统及 AC 电源仍将工作。参数 ulParam 指定要设置的时间，单位为秒。如果 ulParam 设为 0，那么该标志被忽略。SPL_SETMOUSE：设置鼠标的两个阈值和速度。

SPI_SETWAKEUPIDLETIMEOUT：在用户通知重新激活某个挂起的设备之后，Windows CE 延缓响应的时间长度量可使用该标志来设置。参数 ulParam 指定这个时间量，单位为秒，如果 ulParam 设置为 0，那么该标志被忽略。

SPI_SETWORKAREA 设置工作区大小，工作区是指没有被任务条遮盖的屏幕部分。如果用来获取平台类型或 OEM 信息串的 pvParam 缓冲区太小，那么该函数会调用失败，并出现错误值 ERROR_INSUFFICENT_BUFFER。Windows CE 只支持该函数的 UNICODE 版。Windows CE 不支持参数 fWinlni 的取值为 SPIF_SENDWININICHANGE 的情形。

速查：Windows NT：3.1 及以上版本；Window：95 及以上版本；Windows CE：不支持；头文件：Winuser.h；库文件：user32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.2 按钮函数（Button）

2.2.1CheckDlgButton

函数功能： 该函数改变按钮控制的选中状态。

函数原型： BOOL CheckDlgButton（HWNDhDlg，int nlDButton,UINT uCheck）；

参数：

hDlg：指向含有该按钮的对话框的句柄。

nlDButton：标识要修改的按钮。

uCheck：给定该按钮的选中状态。该参数可取下列值，这些值的含义如下：

BST_CHECKED：设置按钮状态为已选中（checked）。

BST_INDETERMINATE：设置按钮状态变灰，表示不确定状态。只有在该按钮具有 BS_3STATE 或 BS_AUTO3STATE 样式时才能使用该值。

BST_UNCHECKED：设置按钮为未选中状态（unchecked）。

返回值：如果函数执行成功，返回值非零；如果函数失败，则返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

速查；Windows NT：3.1 及以上版本；WindowS：95 及以上版本；Windows CE：不支持：头文件：Winuser.h；库文件：user32.lib。

2.2.2CheckRadioButton

函数功能：该函数给一组单选按钮中的一个指定按钮加上选中标志，并且清除组中其他按钮的选中标志。

函数原型：BOOL CheckRadioButtoh（HWNDhDlg， intnlDFirstButton， intnlDLastBUtton，intnlDCheckButton)；

参数：

hDlg：指向包含单选按钮的对话框的句柄。

nlDFirstButton：指定组中第 1 个单选按钮的标识符。

nlDLastButton：指定组中最后一个单选按组的标识符。

nlDCheckButton：指出要选中的那个单选按钮的标识符。

返回值：如果函数执行成功，返回值非零；如果失败，则返回零。若想获取更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以广版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.2.3IsDlgButtonChecked

函数功能：该函数可以确定某个按钮控制是否有选中标志，或者三态按钮控制是否为灰色的、选中的、或两者都不是。

函数原型：UINT IsDlgButtonChecked（HWND hDlg，Int nlDBUtton)；

参数：

hDlg：指向包含按钮控制的对话框。

nkDButtom：指定按钮控制的整型标识符。

返回值：使用 BS_AUTOCHECKBOX、BS_AUTORADIOBUTTON、BS_AUTO3STATE、BS_CHECKBOX、BS_RADIOBUTION 或 BS_3STATE 样式创建的按钮的返回值可以是如下值之一：

BST_CHECKED：表示按钮被选中。

BST_INDETERMINATE：表示按钮是灰色的，即为不确定状态（只有具有 BS_3STATE 或 BS_AUTO3STATE 样式的按钮才使用该值)。

BST_UNCHECKED：表示该按钮未选中（unckecked)。如果该按钮用其他任何样式，那么返回值为零。

速查：Windows NT：3.1 及以 11 版本；Windows：95 及以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32.lib。

2.3　插入标记（^）函数（Caret）

2.3.1CreateCaret

函数功能;该函数为系统插入标记创建一个新的形状,并且将插入标记的属主关系指定给特定的窗口。插入标记的形状。可以是线、块或位图。

函数原型：BOOL CreateCaret（HWND hWnd，HBIBMAP hBitmap，int nHeight);

参数：

hWnd：指定佣有插入标记的窗口。

hBitmap：标识用于定义插入标记形状的位图。如果该参数为 NULL，那么插入标记是实心的（原色），如果该参数为（HBITMAP）1，那么插入标记是灰色的。如果该参数是位图句柄，那么插入标记就是指定的位图。位图句栖必须已中由 CreateBitmap、CreatDIBitmap 或 LoadBitmap 函数创建。

如果 hBitmap 为位图句栖，那么 CreateCaret 函数将忽略参数 nWidth 和 nHeight，因为该位图定义了自己的宽度和高度。

nWidth：按逻辑单位指定插入标记的宽度，如果该参数为零，那么宽度就设为系统定义的窗口边界宽度。如果 hBitmap 是位图句柄，那么函数 CreateCaret 忽略该参数。

nHeight：按逻辑单位指定插入标记的高度。如果该参数为零，那么高度就设为系统定义的窗口边界高度如果 hBitmap 是位图句柄，那么函数 CreateCaret 忽略该参数。

返回值：如果函数执行成功，返回值为非零；如果函数执行失败，那么返回值为零。若想获取更多错误　信息，请调用 GetLastError 函数。

备注：参数 nWidth 和 nHeight 指定了插入标记的宽度和高度，这些值按逻辑单位表示；按像素表示的真正　宽度和高度与窗口的映射模式有关。

CreateCaret 函数自动清除前一个插入标记的形状，不考虑拥有该插入标记的窗口。新创建的插入标记一直隐藏，直到应用程序调用 ShowCaret 函数使该插入标记可见为止。

系统为每个队列提供一个插入标记。窗口只有在它有键盘焦点（focus）或者它是活动窗口时才创建插入标记。该窗口应在键盘焦点消失或窗口变为不活动之前，清除插入标记。

可以通过使用 GetSystemMetrics 函数，并指定 SM_CXBORDER 和 SM_CYBORDER 值来检索系统窗口边界的宽度或高度。使用窗口边界的宽度或高度可以保证插入标记在高分辨率屏幕上可见。

对于 Windows CE：Windows CE 不支持 hBitmap 参数，并且该参数应设为 NULL。Windows CE 缺省的是实心（Solid）插入标记。

速查:Windows NT:3.1 及以上版本;Windows:95 及以上版本；Windows CE:1.0 及以上版本；头文件：winuser.h；库文件:user32.lib。

2.3.2DestroyCaret

函数功能:该函数清除插入标记的当前形状，从窗口中释放插入标记，并且删除屏幕上的插入标记。如果插入标记的形状是基于位图的，那么 DestroyCaret 不释放该位图。

函数原型:BOOL DestroyCaret（VOLD）

参数:无。

返回值:如果函数执行成功，则返回值非零:如果函数失败，则返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

备注:只有当前任务中的窗口拥有插入标记时 DestoryCaret 才清除插入标记。如果插入标记不为当前

任务中的窗口拥有，那么 DestroyCaret 不执行任何操作，并且返回 FALSE。

系统为每个队列提供一个插入标记。只有当窗口有键盘焦点或窗口是活动的时候，才创建插入记号。窗口应该在键盘焦点消失或窗口变为不活动之前清除插入标记。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.3.3GetCaretBlinkTime

函数功能： 该函数返回一个公用的时间，单位为毫秒。该时间是转化插入标记的像素而需要的时间。用户可以使用控制面板来设置这个值。

函数原型： UINT GetCaretBlinkTime（VOLD）

参数：无。

返回值： 如果该函数执行成功，那么返回值就是闪烁时间，单位为毫秒；如果函数执行失败，那么返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.3.4GetCaretPos

函数功能： 该函数将插入标记的位置（按客户区坐标）信息拷贝到指定的 POINT 结构中。

函数原型： BOOL GetCaretPos（LPPOlNT IpPoint）；

参数：

IpPoint：指向 POINT 结构的指针。该结构接收插入标记的客户坐标信息。

返回值： 如果函数执行成功，那么返回值非零；如果函数执行失败，那么返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

备注： 插入标记位置通常是按包含该插入标记的窗口的客户坐标形式给出的。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows：1.0 及以上版本；头文件：Winuser.h；文件：user32.lib。

2.3.5HideCaret

函数功能： 该函数将屏幕上的插入标记清除。实际上是隐藏插入标记，并不是删除其当前形状或使插入点无效。

函数原型： BOOL HideCaret（HWND hWnd）；

参数：

hWnd：标识有插入标记的窗口。如果该参数为 NULL，那么 HideCaret 函数搜索当前任务，以发现拥有插入标记的窗口。

返回值： 如果函数执行成功，那么返回值为非零：如果函数失败，那么返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

备注： 只有指定的窗口拥有插入标记时，HideCaret 才隐藏插入标记。如果指定的窗口没有插入标记，那么函数 HideCaret 什么也不做，并且返回 FALSE。

隐藏操作是累计的，如果应用程序连续 5 次调用 HideCaret，那么该程序也必须调用 ShowCaret 5 次

才能显示插入标记。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：USer32.lib。

2.3.6SetCaretBlinkTime

函数功能：该函数将插入标记的闪烁时间设置为一个指定的数目，该数目的单位为毫秒，闪烁时间就是转化插入标记像素所需的时间，单位为毫秒。

函数原型：BOOL SetCaretBlinkTime（UINT uMSeconds）；

参数：

uMSeconds：指定新的闪烁时间，时间单位为毫秒。

返回值：如果函数执行成功，那么返回值为非零；如果函数执行失败，那么返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

备注：用户可以使用控制面板来设置闪烁时间。应用程序应遵守用户的设置。只有允许用户设置闪烁时间的应用程序（例如控制面板）才应该使用 SetCaretBlinkTime 函数。

如果改变了闪烁时间，那么后续激活的应用程序会使用修改后的闪烁时间，即使在键盘焦点消失或为非活动时，恢复前一个闪烁时间值，这是由于多线程环境的缘故，在这种环境中，禁止使用一个应用程序与激活另一个应用程序并不同步。这种特性允许系统即使在当前应用程序挂起时也可以激活另一个应用程序。

对于 Windows CE：在 Windows CE 系统中，在一个应用程序中改变插入标记的闪烁时间不会影响后续加载的应用程序中的闪烁时间。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.3.7SetCaretPos

函数功能：该函数将插入标记移动到指定的坐标上。如果拥有该插人标记的窗口是使用 CS_OWNDC 类样式创建的，那么指定的坐标依据与该窗口相关的设备环境的映射模式而定。

函数原型：BOOL SetCaretPos（int X，int Y）；

参数：

X：指定插入标记新的 X 坐标。

Y：指定插入标记新的 Y 坐标。

返回值：如果函数执行成功，那么返回值为非零；如果函数执行失败，那么返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

备注：函数 SetCaretPos 不管插入标记是否隐藏都将移动它。系统为每个队列提供一个插入标记。窗口只能对自己拥有的插入标记进行位置的设置。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本：头文件：Winuser.h；库文件：user32.lib。

2.3.8ShowCaret

函数功能：该函数使插入标记在屏幕的当前位置上可见。当插入标记变为可见时，它自动开始闪烁。

函数原型： BOOL ShowCaret（HWND hWnd）；

参数：

hWnd：标识拥有插入标记的窗口。如果该参数为 NULL，那么 ShowCaret 搜索当前任务以发现有插入标记的窗口。

返回值： 如果函数执行成功，那么返回值为非零；如果函数执行失败，那么返回值为零。若想获取更多错误信息，请调用 GetLastError 函数。

备注： 只有指定的窗口拥有插入标记，并且该插入标记具有形状，没有连续隐藏 2 次或多次时，ShowCaret 才能显示该插入标记。如果这几个条件中的一个或多个没满足，那么 ShowCaret 函数什么也不做，并且返回 FALSE。

速查： Windows NT：3.1 及以上版本 Windows：95 及以上版本 Windows CE：1.0 及以版本；头文件：winuser.h；库文件：user32.lib。

2.4 组合框函数（Combo box）

2.4.1DlgDirListComboBox

函数功能： 该函数用一个目录列表来填充指定的组合框

函数原型： int DlgDirListComboBox（HWND hDlg，LPTSTR lpPathSpec，int nlDComboBox，int nlDStaticPath，UINT

uFiletype）；

参数：

hDlg：包含组合框的对话框句柄。

lpPathSpel：指向一个以 NULL 结束的字符串，格式为

[driver][/][directory/][.][filename]

如果指定的串包括一个驱动器或目录路径，在填充列表之前，DlgDirListComboBox 函数改变当前驱动器和目录。在列表被填充之后，驱动器和目录路径从 lpPathspec 参数标识的串中移出来。

nlDComboBox：指定对话框中组合框的标识符。如果 hlDComboBox 为零，则 DlgDirlistComboBox 函数没有组合框存在或不试图填充它。

hlDStatic Path：指定静态控制的标识符，此静态控制用于显示当前目录。如果 nlDStasticPath 为零，DlgDirListComboBox 函数认为没有这样的控制。

uFiletype：指定被显示的文件的属性。它可以是下列的任意组合：

DDL_ARDIVE：包括档案文件。

DDL_DIRECTORY：包括子目录，子目录名必须用方括号括起来（[]）。

DDL_DRIVES：包括驱动器，驱动器列在格式[-X-]中，其中 X 为驱动器字母名。

DDL_EXCIUSIVE：包括仅带指定属性的文件，缺省时，写保护文件被引出，即使 DDL_READ WRITE 没有被指定。

DDL_HIDBEN：包括隐含文件。DDL_READONLY：包括只读文件。

DDL_READWRITE：包括读写文件且不带有别的属性。DDL_SYSTEM：包括系统文件。

DDL_POSTMSGS：把信息传递到应用信息队列，缺省时，DlgDirlist 函数把信息直接发送给对话框过程。

返回值： 如果函数调用成功，返回值为非零值，否则返回值为零。例如，如果 tpPatbSpec 指定的串不是一个有效路径，函数调用失败。若想获得更多错误信息，请调用 GetLastError 函数。

备注;如果参数 lpszPatbSpec 指向一个零字节串或一个指定驱动器目录,或两者的串,但没有文件名,那么认为文件名为”.”。

Windows NT：如果有目录列表的话，则显示长文件名。

Windows 95：目录列表显示短文件名（为 8.3 形式）。可以用 SHGetFilelnfo 或 GetFullPathName 函数来得到相应的长文件名。

速查：Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;库文件：user32.lib：Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.4.2DlgDirSelectEx

函数功能：该函数从单选列表框中检取当前选择，列表框已经由 DlgDirlist 函数填充，并且选择内容为一个驱动器字母，文件名或目录名。

函数原型：BOOL DlgDirSelectEx（HWND hDlg，LPTSTR lpString，int nCount，int nlDListBox);

参数：

hDlg：包括列表框的对话框句柄。

lpString：指向存放选择路径的一个缓冲区。

ncount：指定由 lpString 指向的缓冲区的字节长度。

nlDListBox：指定对话框中列表框的整型标识符。

返回值：如果当前选择为目录名，返回值为非零值。如果当前选择不是一个目录名，返回值为零，若想

获得更多错误信息，请调用函数 GetLastError 函数。

备往：DlgDirSelectEx 函数把选择复制到由 IpString 参数指向的缓冲区，如果当前选择是一个目录名或驱动器字母，DlgDirSelectEx 则删除方括号（对于驱动器字母，则删去破折号)。这样以便目录名或驱动器字母能插入一个新的路径。如果没有选择，lpString 不改变。DlgDirSelectEx 函数把消息 LB_GETCURSEL 和消息 LB_GETTEXT 发送到列表框，函数禁止从列表框返回多于一个的文件名。列表框不应是复选的列表框，如果是的话，此函数不返回零值且 Ipstring 参数保持不变。

速查：Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;库文件：user32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.4.3DlgDirSelectComboBoxEx

函数功能：该函数从由 DlgDirlistcomboBox 函数填充的组合框中检取当前选择。选择内容为一个驱动器字母、文件名或目录名。

函数原型：BOOL DlgDirSelectComboBox(HWND hDlg,LPTSTR lpString,int nCount,int nlDComboBox);

参数：

hDlg：包括组合框的对话框的句柄。

IpString：指向存放选择路径的缓冲区。

nCount：指定 IpString 参数指向的缓冲区的字节长度。

nlDComboBox：指定控制对话框的组合框的整型标识符。

返回值：如果当前选择为一个目录名，返回值为非零值；否则，返回值为零值。若想获得更多错误信息，请调用 GetLastError 函数。

备注：如果当前选择指定于一个目录名或驱动器字母，DlgDirSelectComboBoxEx 函数则删方括号（对驱动器字母则删除破折号),以便使文件名或驱动器字母能插入新的路径或文件名,如果没有选择,IpString

参数指向的缓冲区的内容没有改变。DlgDirselectComboBoxEx 函数不允许从组合框返回多于一个的文件名。

DlgDirSectectComboBoxEx 把消息 CB_GETCILRSEL 和 CB_GETLBTEXT 发送到组合框。在 Win32API 中，可应用带有三种组合框的 DlgDirSelectComboBoxEx 函数（三种组合框为 CBS_SIMPLE，CBS_DROPDOWN，与 CBS_DROPDOWNLIST）。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5 通用对话框函数（Common Dialog Box）

2.5.1ChooseColor

函数功能： 该函数创建一个能使用户从中选择颜色的通用颜色对话框。

函数原型：BOOL ChooseColor（LPCHOOSECOLOR IpCC）；

参数：

lpCC：指向一个包括初始化对话框信息的 CHOOSECOLOR 结构。当 ChooseColor 函数返回时，此结构含有有关用户颜色选择的信息。

返回值：如果用户点击对话框中的 OK 按钮，返回值为非零值。CHOOSECOLOR 结构中的 rgbResult 成员含有用户选择的颜色的 RGB 颜色值。如果用户取消或关闭 Color 对话框或错误出现，返回值为零。若想获得更多错误信息，请调用 CommDlgExtondedError 函数，此函数的返回值为下列中的一个：

CDERR_FINDRESFAILURE；CDERR_MEMLOCKFAILURE；CDERR_INITIALIZATION；

CDERR_NOHINSTANCE；CDERR_LOCKRESFAILURE；CDERR_NOHOOK

CDERR_LOADRESFAILURE；CDERR_NOTEMPLATE；CDERR_LOADSTRFAlLURE；

CDERR_STRUCTSIZE；CDERR_MEMALLOCFAILURE

备注 Color 对话框不支持彩色调色板，对话框提供的颜色的选择仅限于系统颜色和这些颜色的混合值，可以为对话框提供一个 CCHOOKProc 程序，此挂钩程序能处理发送给对话框的信息。通过建立 CHOOSECOLOR 结构中 Flags 成员的 CC_ENABLEHOOK 标志和指定 IpfnHook 成员中挂钩程序的地址，可使挂钩程序生效。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：commdlg.h；库文件：commdlg32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.2ChooseFont

函数功能：该函数创建一个使用户选择逻辑字体属性的对话框，这些属性包括字体名称、字体风格（如粗体、斜体或正常体）、字号、效果（如强调线，下划线或字体颜色）和手写体（或字符集）。

函数原型：BOOL ChooseFont（LPCHOOSEFONT Ipcf）；

参数：

Ipcf：指向一个含有初始化对话框信息的 CHOOSEFONT 结构。当返回 ChooseFont 函数时，此结构含有用户对字体选择的信息。

返回值：如果用户点击对话框的 OK 按钮，返回值为非零值，CHOOSEFONT 结构中的成员表明用户的选择。如果用户取消或关闭 Font 对话框或出现错误信息，返回值为零。若想获得更多错误信息。请调用 CommDlgExtendedError 函数，其返回值如下：

CDERR_FINDRESFAILURE；CDERR_NOHINSTANCE；CDERR_INITIALIZATION；CDERR_NOHOOK

CDERR_LOCKRESFAILURE；CDERR_NOTEMPLATE；CDERR_LOADRESFAILURE；

CDERR_STRUCTSIZE；CDERR_LOADSTRFAILURE；CDERR_MAXLESSTHANMIN

CDERR_MEMALLOCFAILURE；CDERR_NOFONTS；CDERR_MEMLOCKFAILURE

备注：可以为 Font 对话框提供一个 CFHOOKProc 挂钩程序。此挂钩程序能够处理发送给对话框的信息。

通过建立 CHOOSEFONT 结构中 Flags 成员的 CE ENABLEHOOK 标志和指定 IPfn Hook 成员中挂钩程序的地址可以使挂钩程序有效。

挂钩程序可以把信息 WM_CHOOSEFONT_GETLOGLONT，WM_CHOOSEFONT_SETFLAGS 和

WM_CHOOSEFONT_SETLOGFONT 消息发送给对话框以便得到和创建当前值和对话框的图标。

速查：Windows NT:3.1 及以一上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:Commdlg.h;库文件：comdlg32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.3CommDlgExtendedEorror

函数功能：该函数返回一个对话框错误代码，此代码显示出在执行下列对话框函数时要出现的最近的错误:ChooseColor,GetOpenFileName,ChooseFont,GetSaveFileName,FindText,PrintDlg,ReplaceText,PageSetpDlg。

函数原型：DWORD CommDlgExtendedError（VOID）

参数：无。

返回值：如果最近一次对话框函数调用成功，返回值不确定，如果对话框函数因为用户关闭或取消对话框而返回 FALSE，则返回值为零。否则返回值是非零错误代码。有关更多的信息，参见下列说明部分。

备注：CommDlgExtendedError 函数可以返回公共对话框函数中的一般的错误代码。

另外，也返回某一具体公共对话框的错误代码。由 CommDlgExtendedError 返回的错误代码在 CDERR.H 文件里定义：

下面是 CommDlgExtendedError 的返回错误代码值。

CDERR_DIALOGFAILURE：对话框不能创建。DialogBox 函数对对话框函数的调用失败，例如如果公共对话框的调用指定一个无效的窗口句柄，则此种错误产生。

CDERR_FINDRESFAILURE：公共对话框函数没能找到指定资源。

CDERR_INITIALIZATION：公共对话框函数在初始化过程中失败。当没有足够内存时此错误出现。

CDERR_LOADRESFAILURE：公共对话框函数没能调出指定的资源。

CDERR_LOADSTRFAILURE：公共对话框函数没能调出指定的串。

CDERR_LOCKRESFAILURE：公共对话框函数没能销定指定的资源。

CDERR_EMAILCOLFAILURE；公共对话框函数不能为内部结构分配内存。

CDERR_NOHINSTANCE：在对应的公共对话框初始化结构 Flags 成员中设置 ENABLETEMPLATE 标志，但是在提供相应的事例句柄时出错。

CDERR_NOHOOK：在对应的公共对话框初始化结构 Flags 成员中设置 ENABLEHOOK 标志，但是在提供相应的挂钩程序指针时出错。

CDERR_NOTEMPLATE：在对应的公共对话框初始化结构 Flag 成员中设置 ENABLETEMPLATE 标志，但是在提供相应的模板时出错。

CDERR_REGISTERMSGFAIL：当 RegisterWindowMessage 函数被公共对话框函数调用时，该函数返回错误代码。

CDERR_STRUCTSIZE：对应的公共对话框初始化结构旧 IStructSize 成无效成员。

下列为 Print Dlg 函数的返回值。

PDERR_CREATEICFAILURE：当 PrintDlg 函数想创建一个信息表时出错。

PDERR_DEFAIKTDFFERENT：利用在 DEVNAMES 结构中 wDefault 成员指定的 DN_DEFAULTPRN 标志，可以调

用 PrintDlg 函数。但是被另外一个结构成员描述的打印机与当前缺省的打印机不匹配。(此错误发生在储存 DEVNAMES 结构和用户利用控制面板改变缺省打印机时)。要使用 DEVNAMES 结构所描述的缺省打印机，必须清空 DN_DEFAULTPRN 标志并且要再一次调用 PrintDlg。要使用缺省打印机，必须用 NULL 取代 DEVNAMES 结构俐 DEVMODE 结构，如果此结构存在的话)，并且要再一次调用 PrintDlg 函数。

PDERR_DNDMMISMATCH：OEVMOOE 和 DEVNAMFS 结构中的数据描述了两种不同的打印机。

PDERR_GETDEVMODEFAIL;打印机驱动程序不能初始化一个 DEVMODE 结构(这种错误代码只用于 Windows 3.0 及以上版本的打印机驱动程序)。

PDERR_INITFAILURE：PrintDlg 函数不能初始化，并且没有更多的错误代码来描述此错误。

PDERR_LOADDRVFAILURE：PrinDlg 函数不能为指定的打印机装备设备驱动器。

POERR_NODEFAULTPRN：不存在缺省打印机。

POERR_NODEVKES：未发现打印机驱动程序。

PDERR_PARAEFAILURE：PrintDlg 函数在分析 WIN.INI 文件中的[devces]部分的字符串时出错。

PDERR_PRINTERNOTFOUND；WIN.INI 文件的[device]部分不包含所请求打印机的入口

PDERR_RETDEFFAILURE_PD：RETURNDEFAULT 标志被指定在 PRINTDLG 结构的 Flags 成员中。但 hDevMode 或 hDevNames 成员不是 NULL。

PDERR_SETUPFAILURE：PrintDlg 函数在装载所需要的资源时出错。

下面是 ChooseFont 函数的返回值。

CFERR_MAXLESSTHANMIN：CHOOSEFONT 结构中的 nSizeMax 成员所给定的大小小于 nSizeMin 成员给定的大小。

CFERR_NOFONTS：不存在字体。

下面是 GetOpenFileName 和 GetSaveFileName 函数的返回值。

FNERR_BUFFERTOOSMALL：由 OPENFILENAME 结构的 lpstrFile 成员指向的缓冲区对由用户指定的文件名来说太小。前两种 IpstrFile 缓冲区的字节含有一个指定大小的整型值。用来存放全文件名。

FNERR_INVAIDFILENAME：文件名无效。

FNERR_SUBASSFAILIIRE：由于没有足够内存，在对列表框分类时出错。

下面是 FindText 和 ReplaceText 函数的返回值。

FRERR_BUFFERLENGTHZERO：结构 FINDREPLACE 中的一个成员指向一个无效的缓冲区。

在 Windows CE 中 Windows CE 支持另外的四种返回值，如下：

CDERR_REGISTRYFAILURE　公共对话框函数无法读注册表。

下面的一些返回值只适用于 PrintDlg 函数。

PDERR_NOPORTS：没有注册的端口 PDERR_NOPRINTERS：没有注册的打印机

PDERR_CREATEDCFAILURE：CreateDC 调用失败

Windows CE 不支持 CDERR_LOADSTRFAILURE、CDERR_MEMLOCKFAILURE 或 CDERR_REGISTERMSGFAIL 返回值。Windows CE 也不支持任何 PDERR_错误值。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本：头文件：commdlg.h；库文件：comdlg32.lib。

2.5.4FindText

函数功能：该函数创建一个系统定义的无模式 Find 对话框，为使用户指定一个串来查找文本内的文字。

函数原型：HWND FindText (LPFINDREPLACE lpfr);

参数：

Ipfr：指向一个 FINDEPLACE 结构，此结构包含用来初始对话框的信息。对话框用此结构把用户输入的信息传送到应用程序。有关更多的信息，请参见下面说明部分。

返回值：如果函数调用成功，返回值是对话框的窗口句柄。可以使用窗口句柄与对话框联系或关闭它；如果函数调用失败，返回值为 NULL。若想获得更多的错误信息，请调用 CommDlgExtendedError 函数。其返回值如下：

CDERR_FINDRESFAILURE；CDERR_MEMLOCKFAILURE；CDERR_INITIALIZATION

CDERR_NOHINSTANCE；CDERR_LOCKRESFAILURE；CDERR_NOHOOK

CDERR_LOADRESFAILURE；CDERR_NOTEMPLATE；CDERR_LOADSTRFAILURE

CDERR_STRUCTSIZE；CDERR_MEMALLOCFAILURE；FRERR_BUFFERLENGTHZERO

备注：FindText 函数不执行查找操作，相反，对话框把 FINDMSGSTRING 已登记的信息传送到对话框窗口的窗口函数。当创建对话框时 FINDReplace 结构中的 hwndCwner 成员标识窗口。

调用 Find Text 函数前，必须调用 RegisterWindowMessage 函数以得到 FINDMSGSTRING 信息的标识符，对话框函数在用户点击 FindNext 按钮或对话框被关闭时利用此标识符传送信息。FINDMSGSTRING 信息的 IParam 参数包含一个指向 FINDREPLACE 结构的指针，此结构的 Flags 成员显示开诚信息的事件。

其他成员显示用户的输入。

若想创建对话框，必须利用应用程序的主信息链中的 IsDialogMessage 函数来保证对话框正确处理键盘输入，例如 Tab 和 Esc 键。IsDialogMessage 返回值显示 Find 对话框是否处理信息。

可以为 Find 对话框提供一个挂钩函数 FRHookProc。挂钩函数可处理发送到对话框中的信息。为使挂钩函数生效，可设置 HNDREPLACE 结构的 Flags 成员的 FR_ENABLEHOOK 标志，且指定 IpfnHook 成员中挂钩函数的地址。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：comdlg32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.5GetFileTitle

函数功能：该函数返回由 IpszFile 参数标识的文件名。

函数原型：short GetFTitle（LPCTSTR LPTSTR IpszTitle，WORD cbBuf）；

参数：

IpszFile：指向一个文件名或文件位置的指针。

IpszTitle：指向一个缓冲区，在此缓冲区中函数复制文件名。

chBuf：指定由 IpszTitle 函数指向的缓冲的字节长度。

返回值：如果函数调用成功，返回值为零；如果文件名无效，返回值为一负值。如果由 IpszTitle 参数指向的缓冲区的太小，返回值为一正整数，其值指定了所需缓冲区的大小。所需缓冲区的大小还包括结尾的 NULL 字符。

备注：如果由 IpszFile 参数指向的缓冲区包含下列任何一个成员，GetFileTitle 函数返回一个错误信息值。

一个空字符串；一个含有星号（）、开括号（[）、闭方括号（]）的串、一个以冒号（:）、斜杠（/）或倒斜杠结尾的串；一个长度超过缓冲区长度的串；一个无效字符（如，一个空格或一个不能打印的字母）：为得到文件名所需的缓冲区的大小，用设置为 NULL 的 IpszTitle 和设置为零的 cbBuf 调用函数。函数将返回所需的大小。

GetFileTitle 函数返回一个串，系统应用此串为用户显示文件名。这就意味着如果返回串应用在文件系统函数的调用中不可能准确地标识文件。

如果证 IpszTitle 缓冲区太小，GetFile Title 返回的大小需要含有显示名。在 IpszFile 缓冲区指定的需要的大小和字符之间没有许可的联系。

速查：Windows NT：3.1 及以上版本：Windows：95 及以上版本：Windows CE：不支持：头文件：Commdlg.h；库文件：comdlg32.fib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.6GetOpenFileName

函数功能：该函数创建一个 Open 公共对话框，使用户指定驱动器、目录和文件名、或使用户打开文件。

函数原型： BOOL GetOpenFileName（LPOPENFILENAME Ipofn）；

参数：

Ipofn：指向包含初始化对话框的信息的一个 OPENFILENAME 结构。当 OpenfileName 函数返回时，此结构包含有关用户文件选择的信息。

返回值：如果用户指定了一个文件名，点击 OK 按钮，返回值为非零。由 OPENFILENAME 结构的 IPstrFile 成员指向的缓冲区含有全路径和用户指定的文件名。如果用户取消或关闭 Open 对话框或错误出现，返回值为零。若想获得更多的错误信息，请调用 CommDlgExtendedError 函数。

备注： 从 Windows 95 和 Windows NT 4.0 版开始，缺省的 Open 对话框提供了与 Windows Explorer 相似的用户界面特征。可以为一个浏览器风格的 Open 对话框提供一个 OFNHOOKProc 挂钩函数。设置 OPENFILFNAME 结构的 Flags 成员中 OFN_EXPLORER 和 OFN_ENABLEHOOK 标示和指定 IpfnHook 成员中挂钩函数的地址，可使挂钩函数生效。

Windows 95 和 Windows NT 仍支持旧风格的 Open 对话框以便维持与一个 Windows 3.1 或 Windows NT3.51 用户界面相一致的用户界面。使 OFNHOOKProcOldstyle 挂钩函数生效和保证 OFN_EXPLORER 标志没有被设置，就可以显示旧式的 Open 对话框。

为显示允许用户选择一个目录而不是一个文件的对话框，要调用 SHBrowseForFolder 函数。

Windows CE：并不是每一个 OPENFILENAME 结构的成员都在 Windows CE 中被定义。有关定义成员的更多的信息，请参见 OPENFILENAME 结构的资料主题。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：COmmdlg.h；库文件：comdlg32.lib；Unlcode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.7GetSaveFileName

函数功能： 该函数创建一个 Save 公共对话框，以便用户指定驱动器、目录和文件名。

函数原型： BOOL GetSaveFileName（LPOPENFILENAME Ipofn）；

参数：

Ipofn：指向一个包含初始化对话框信息的 OPENFILENAME 结构。当 GetsaveFileName 函数返回时，此结构含有关于用户文件选择的信息。

返回值： 如果用户指定了一个文件名且点击 OK 按钮，返回值为非零值。由 OPENFILENAME 结构中的归 IpstrFile 成员指向的缓冲区含有全路径和用户指定的文件名。如果用户取消或关闭 Save 对话框或错误出现，返回值为零。若想获得有关更多的错误信息，请调用 CommDlgExtendError 函数，其返回值同 GetOpenFileName 返回值。

速查： Windows NT：3.1 引及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：commdlg.h；库文件：comdlg32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.8PageSetupDlg

函数功能： 该函数创建一个 PageSetup 对话框，此对话框能使用户指定打印页的属性。这些属性包括

纸张大小和来源，送纸方向和页边距。

函数原型： BOOL PageSetupDlg（LPPAGESETUPDLGI ppsd);

参数：

Ippsd：指向一个包含初始化对话框信息的 PAGESETUPDLG 结构。当函数返回时，该结构存放有关用户选择的信息。

返回值： 如果用户点击 OK 钮，返回值为非零值，Ippsp 参数指向的 PAGESETUPDLG 结构中的成员显示用户的选择。如果用户取消或关闭 PageSetup 对话框或错误出现，返回值为零。若想获得更多的错误信息，请调用 CommDlgExtendedError 函数

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：commdlg32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.9PrintDlg

函数功能：该函数显示打印对话框或打印设置对话框。打印对话框使用户指定特殊的打印工作的特点。

打印设置对话框不能应用在新应用程序中，它已经被 PageSetupDlg 函数创建的打印设置公共对话框所替代。

函数原型： BOOL PrintDIg（LPPRINTDLG Ippd);

参数：

Ippd：指向一个含有初始化对话框信息的 PRINTDLG 结构。当 PRINTDLG 函数返回时，此结构含有关用户选择的信息。

返回值：如果用户点击 OK 按钮，返回值为非零值。由 lppd 参数指向的 PRINTDLG 结构中的成员显示用户的选择。如果用户取消或关闭 Print 或 PrinterSetup 对话框或错误出现，返回值为零。若想获得更多的错误信息，请调用 CommDlgError 函数。如果用户取消或关闭对话框，函数返回零值：否则，返回值如下：

CDERR_FINDRESFAILURE PDERR_CRETELCFAILUPE

COERR_INITIALIZATION PDERR_DEFAULTDIFFERENT

CDERR_LOADRESFAILURE PDERR_DNDMMISMATCH

CDERR_LOADSTRFAILURE PDERR_GETDEVMODEFAIL

CKERR_LOCKRESFAILURE PDERR_INITFAILURE

CDERR_MEMALLOCFAILURE PDERR LOADDRVFAILURE

CDERR_MEMLOCKFAILURE PDERR_NODEFAULTPRN

CDERR_NOHINSTANCE PDERR_NODEVICES

CDFRR_NOHOOK PDERR_PARSEFAILURE

CDERR_NOTEMPLATE PDERR_PRINTERNOTFOUND

CDERR_STRUCTSIZE PDERR_RETDEFFAILURE

备注：如果挂钩函数（由 PRINTDLG 结构中的 IpfnrintHOOk 成员或 IpfnSetupHOOk 成员指向的）处理 WM_CTLCOLORDLG 信息，挂钩函数必须返回一个刷子句柄，此刷了用来刷控制背景。

Windows NT 5.0 以及以后的版本：可用 PrintDlgEx 函数来显示一个 Print 属性页，此属性页有一个含有 Print 公共对话框相似的控制的 General 页，其控制与 Print 公共对话框中的控制相似。

Windows CE：PRINTDLG 结构包含 Windows CE 中不同的成员。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：commdlg.h；库文件：comdlg32.lih；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.10 PrintdlgEx

函数功能： 该函数显示一个 Print 属性页。该属性页使用户指定特定的打印工作的特性，一个 Print 属性页有一个控制的 General 页。该控制与 Print 中的对话框相似。属性页也有另外紧随 General 页的指定的应用程序和指定的驱动器特性页。

函数原型： HRESULT PrintDlgEx（LPPRINTDLGEX Ippd)；

参数：

Ippd：指向一个包括初始化属性页信息的 PRINTDLGEX 结构。当 PrintDlgEx 函数返回时，此结构含有关于用户选择的信息。

返回值： 如果函数调用成功，返回值为 S_OK，且 PRINTDLGEX 结构中的 dw ResuhltAction 成员含有下列值：

PD_RESULT_APPLY：用户点击 Apply 按钮，然后点击 Cancel 按钮，这显示出用户想应用在属性页中做的改变，但还不想打印。当 Apply 按钮被点击时，PRINTDLGEX 结构包含用户指定的信息。

PD_RESUCT_CANCEI：用户点击 Cancel 按钮，PRINTDLGEX 结构中的信息未被改变。

PD_RESUCT_PRINT：用户点击 Print 按钮，PRINTDLGEX 结构含有用户指定的信息。

如果函数调用成功,返回值可能是下列 COM 错误代码中的一个,有关更多的信息请参见 ErrorHandling。

E_OUTOFMEMORT：内存不足；E_INVALIDARG：一个或更多的参数无效。

E_POINTER：指针失效；E_HANDLE：句柄失效；E_FAIL：不确定的错误。

备注： 有关更多的信息，请见 Print Propertysheet。

速查： Windows NT：5.0 及以上版本；Windowss：不支持；Windows CE：不支持；头文件：commdlg.h，库文件。comdlg32.lib；Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.11 ReplaceText

函数功能。该函数创建一个系统定义的无模式对话框，此对话框使用户查找或替代一个串，或执行控制和替代操作。

函数原型： HWND Replace Text （LPFINDREPLACE Ipfr)；

参数：

Ipfr：指向一个包含初始化对话框的信息的 FINDREPLACE 结构。对话框应用此结构把用户输入的有关信息传送到应用程序。有关更多的信息，见卜列说明部分。

返回值： 如果函数调用成功，返回值为对话框的窗口句柄，可以用窗口句柄与对话框联系或关闭它。如果函数调用失败，返回值为 NULL。若想获得更多的错误信息，调用 CommDlgExError 函数，其返回值如下：

CDERR_FINDRESFAILURE CDERR_MEMLOCKFAILURE

CDERR_INITIALIZATION COERR_NOHINSTANCE

CDERR_LOADRESFAILURE CDERR_NOHOOK

CDERR_LOADSTRFAILURE CDERR_BITEMPLATE

CDERR_LOCKRESFAILURE CDERR_STRUCTSIZE

CDERR_MEMALLOCFAILURE FRERR_BUFFERLENGTHZERO

备注： PepIACE Text 函数不执行文本替代操作。相反，对话框把 FINDSGSTRING 已登记的信息传送到对话框窗口的窗口函数。当创建对话框时，FINDREPLACE 结构的 hwndowner 成员指定该对话框窗口。

调用 ReplaceText 函数之前，必须调用 RegisterWindowMessage 函数为 FINDSGSTRING 信息登记标识。

当用户击点 Find Next，Replace ALL 按钮时，或当关闭对话框时，对话框函数应用这些标识发送信息。

FINDMSGSTRING 信息中的 IParam 参数含有一个指向 FINDREPLACE 结构的指针。此结构的 Flags 成员表明了形式信息的事件。该结构中其他的成员表明用户的输入信息。

如果创建了 Replace 对话框，必须应用应用程序信息链中的 IsDialogMesssage 函数来保证对话框能正确处理链盘输入信息，例如 Tab 键和 Esc 键。

ISDialogMessage 函数返回值表明 Replace 对话框是否处理信息。

可以为 Replace 对话枢提供一个 FRHookProc 挂钩函数，此挂钩函数能处理发送到对话框中的信息。

为使一个挂钩函数生效,可设置 FINDREPLACE 结构中 Flags 成员的 FR_ENABLEHOOK 标志且指定 IpfnHook 成员中挂钩函数的地址。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：comdlg32.lib：Unicode：在 Windows NT 环境中实现为 Unicode 和 ANSI 两个版本。

2.5.12 CCHookProc

函数功能：该挂钩函数是一个应用程序或库定义的回调函数。ChooseColor 函数与此函数一起使用挂钩函数贮存信息或通告，此信息和通告应用于 Color 公共对话框的缺省对话框函数。

LPCEHOOPROC 类型定义了一个指向此回调函数的指针。CCHOOkProc 是一个应用程序定义的函数名的位置占有者。

函数原型：UINT CALLBACK CCHookProc（HWND hdlg，UINT uiMsg，WPARAM wParam，LPARAM IParam）；

参数：

hdlg：指向信息指定的 Color 对话框的句柄。

uiMag：标识存放的信息。

wParam：指定有关信息号的其他信息。精确意义根据 UiMdsg 参数的值来决定。

IParam：指定有关消息的其他信息。精确意义由 uiMsg 参数的值米决定。如果 uiMsg 多数表明 WM_IMTDIALOG 消息，IParam 是一个指向含有当对话框创建时指定的值的 CHOOSECOLOR 结构的指针。

返回值；如果挂钩函数返回零值，缺省对话框函数处理消息。如果挂钩函数返回非零值，缺省对话框函数忽略消息。

备注当用 ChooseColor 函数创建一个 Color 对话框时，应用程序可以为对话框函数提供 CCHookProc 挂钩函数来处理消息。应用传送到对话创建函数中的 CHOOSECOLOR 结构可使挂钩函数生效，也可指定 IpfnHook 成员中挂钩函数的地址和指定 Flags 成员中 CC_ENABIEHOOK 标志。缺省对话框函数把消息 WM_INITDIALOG 传送到挂钩函数之前先处理此消息。对于其他所有的消息，挂钩函数首先存放消息。然后其返回值决定此缺省对话函数是处理消息或忽略消息。如果挂钩函数处理 WM_CTLCOLORDIG 消息，那么必须返回一个有效的刷子句柄以绘制对话框的背景。总之，如果挂钩函数处理任何一种 WM_CTLCOLOR 消息，它必须返回一个有效的刷了句柄以刷指定的控制板的背景。

不要从挂钩函数中调用 EndDialog 函数。相反地挂钩函数能调用 PostMessage 函数来把带有 IDABORT 值的 WMCOMMAND 消息传送到对话框枢函数中。传送 LDABORT 消息使对话框关闭并使对话框返回值为 FALSE。如果要知道为什么挂钩函数关闭对话框，必须在挂钩函数和应用程序之间提供自身联系机理。

可以对公共对话框的标准控制面板进行子分类。但是公共对话框函数也可以对控制面板子分类、正因如此，在挂钩函数处理消息时必须对控制面板进行子分类。这就保证在对话框函数设置于分类函数之前，子分类函数存放了指定的控制消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：commdlg.h；库文件：用户自定义。

2.5.13 CFHookProc

函数功能：该挂钩函数是一个应用程序定义的或库定义的回调函数，此回调函数与 ChooseFont 函数一起使用。挂钩函数接收用于 Font 公共对话框的缺省对话框函数的消息和通告。

LPCFHOOKPROC 类型定义了一个指向这种回调函数的指针。CFHOOKProc 是一个由应用程序定义的函数名的位置占有者。

函数原型：UINT CALLBACK CFHookProc（HWND hdlg，UINT uiMsg，WPARAM wParam，LPARAM lParam）；

参数：

hdlg：指向消息所指的 Font 对话框窗口的句柄。

uiMsg：指定存放的消息。

wParam：指定有关消息的其他信息。精确的意义根据 uiMsg 参数的值来决定。

lParam：指定有关消息的其他信息。精确的意义根据 uiMSg 参数的值来决定。如果 uiMsg 参数表明 WM_INITDIALOG 消息，则 lParam 是一个指向 CHOOSEFONT 结构的指针，该结构含有创建公共对话框时指定的值。

返回值：如果挂钩函数返回零值，缺省对话框函数处理消息。如果挂钩函数返回一个非零值，缺省对话框函数忽略消息。

备注：用 ChooseFont 函数创建一个 Font 对话框时，可以为对话框函数提供挂钩函数来处理消息。

为使挂钩函数生效，要用传送到对话创建函数的 CHOOSEFONT 结构，指定 IpfnHook 成员中挂钩函数的地址和 Flags 成员中 CF_ENABLEHOOK 标志。

缺省的对话框函数处理 WM_INTDIALOG 消息后，把它传送给挂钩函数。对于其他消息，挂钩函数首先存放此消息。然后挂钩函数的返回值决定缺省对话函数是否处理消息或忽略它。

如果挂钩函数处理 WM_CTLCOLORDLG 信息，它必须返回一个有效的刷子句柄，以刷对话的背景。

总之，如果挂钩函数处理任何一个 WM_CTLCOLOR 消息，那么必须返回一个有效刷子句柄，以绘制指定控制的背景不须从挂钩函数调用 EndDialog 函数。相反，挂钩函数能调用 PostMessage 函数，把带有 IDABORT 值的一个 WM_CDOMMAND 信息传送给对话框函数 Posting IDABORT 关闭对话框，使对话框函数退回 FALSE 值，若要了解为什么挂钩函数关闭对话框，必须提供挂钩函数和应用程序之间的联系机理。

可以对公共对话框的标准控制面板进行子分类。但是公共对话框函数也可以对控制面板进行子分类。正因如此，必须在挂钩函数 WM_INTDIALDG 处理消息时对控制面板子分类。这就保证对话框函数设置子分类函数之前，子分类函数接收指定的控制消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：用户自定义。

2.5.14 FRHookProc

函数功能：该函数是由应用程序定义或库定义的回调函数。它和 FindText 函数或 Replace Text 函数一定使用。挂钩函数接收用于 Find 或 Replace 公共对话框的缺省对话框函数的消息和通告。

LPFRHOOKPROC 类型定义了一个指向此回调函数的指针，FRHookProc 是一个应用程序定义的函数名的位置持有者。数原型：UINT CALLBACK FRHookProc（HWND hdlg，UINT uiMsg WPARAM wParam，LPARAM lParam）；

参数：

hdlg：指向消息所传送到的 Find 或 Replace 公共对话框窗口的句柄。

uiMsg：指定有放的消息。

wParam：指定有关消息的另外的信息，精确的意义要根据 uiMsg 参数的值来定。

lParam：指定有关消息的另外的信息，精确的意义要根据 uiMsg 参数的值来定。如果 uiMsg 参数表明

WM_INItDIALOG 消息。IParam 是一个指向 FINDREPLACE 结构的指针，该结构包含公共对话框创建时指定的值。

返回值：如果挂钩函数返回零，缺省对话框函数处理消息；如果挂钩函数返回非零值，缺省对话框函数忽略消息。

备注：当用 FindText，或 ReplaceText 函数创建一个 Find 或 Replace 公共对话框时，可提供一个处理与公共对话框函数有关的信息和通告的 FRHookProc 挂钩函数。为使挂钩函数生效，应用传送到公共创建函数的 FINDREPLACE 结构且指定 IpfnHook 成员中挂钩函数的地址和指定 Flags 成员中 FR_ENABLEHOOK 标志的地址。缺省对话框函数处理 WM_INTTDIALOG 消息后，把它传递给挂钩函数，对于其他所有消息，挂钩函数首先接收消息，然后挂钩函数的返回值决定缺省对话框函数是处理消息或忽略它。如果挂钩函数处理 WM_CTLCOLORDLG 消息，那么必须返回一个有效的刷子句柄，以便绘制对话框的背景。总之如果挂钩函数处理任何一种 WM_HLOLOR 信息，那么必须返回一个有效的刷子句柄以绘制指定控制的背景。

不须从挂钩函数调用 EndDialog 函数。相反，挂钩函数能调用 PostMessage 函数把带有 IDABORT 值的一个 WM_COMMAND 信息传送给对话框函数 Posting IDABORT 关闭对话框，使对话框函数返回 FALSE 值，若要了解为什么挂钩函数关闭对话框，必须提供挂钩函数和应用程序之间的联系机理。

可以对公共对话框的标准控制面板进行子分类。但是公共对话框函数也可以对控制面板进行子分类。正因如此，必须在挂钩函数函数 WM_INTDIALDG 消息时对控制面板子分类。这就保证对话框函数设置子分类函数之前，子分类函数接收指定的控制消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；windows CE：不支持；头文件：commdlg.h；库文件：用户自定义。

2.5.15 OFNGookProc

函数功能：该挂钩函数是应用程序定义的或库定义的回调函数，此回调函数与 Explorer_Style 和 Save As 公共对话框一起使用。函数贮存从公共对话框发送来的消息或通知 LPOFNHOOKPROC 类型定义一个指向回调函数指针，OFNHOOKProc 是应用程序定义的或库定义的函数名的位置占有者，当创建一个 Open 或 Save As 公共对话框时，如果没有指定 OFN_EXPLORER 标志且需一个挂钩函数。必须用旧式的 OFNHookProcOldSttyle 挂钩函数。这种情况下对话框将显示旧式的用户界面。

函数原型：UINT CALLBACK OFNHookProc（HWND hdlg，UINT uiMsg，WPARAM wParam，LPARAM IParam）；

参数：

hdlg：指向 Open 或 Save as 对话框的子对话框的句柄。用 GetParent 函数可得到指向 Open 或 Save As 对话框窗口的句柄。

uiMsg. 标识存放的消息。

wParam：指定有关消息的其他信息。精确的意义要根据 uiMSg 参数的值来决定。

IParam；指定有关消息的其他信息。精确的意义要根据 uiMsg 参数的值来决定。如果 uiMsg 参数显示了 WM_INITDIALOG 消息，IParam 是一个指向 OPENFILENAME 结构的指针。该结构含有创建对话框时指定的值。

返回值：如果挂钩函数返回零，缺省对话框函数处理消息。如果挂钩函数返回非零值，缺省对话框函数忽略消息。对于 CDN_SHAREVIOLATION 和 CDN_FILEOK 通告消息，挂钩函数应该返回非零值，以表明已经利用 SetWindowLong 函数设置了一个非零 DWL_MSGRESULT 值。

备注：当用 GetOpenFileName 或 GetSaveFileName 函数创建浏览器风格的 Open 或 Save As 公共对话框时，可以提供一个 OFNHookProc 挂钩函数。为使挂钩函数生效，须应用传递到对话创建函数的 OPENFILENAME 结构，且须指定一个指向 PfnHook 成员中的挂钩函数的指针，及指定 Flags 成员中 OFN_ENABLEHOOK 标识。

如果为浏览器风格的公共对话框提供了一个挂钩函数，系统创建了一个缺省对话框的子对话框，挂钩函数为子对话框作为对话函数，此子对话框以 OPENHLENAME 结构指定的模块为基础，或如果没有指定模板，

对话框是一个缺省的子对话框。当缺省对话框函数正在创建 WM_INITDALOG 消息时，创建了子对话框。在子对话框处理它本身的 WM_INITDIALOG 消息时，缺省对话函数删除掉标准控制面板，如果需要，可为子对话框的其他的任何一个控制面板提供空间，然后系统将 CDN_INITDONE 通告消息发送给挂钩函数。

挂钩函数不接收指定给缺省对话框的标准控制面板的消息。可以对控制面板进行子分类，但是如果控制面板使应用程序与公共对话框将来的版本不一致，那么不能进行子分类。但浏览器风格的公共对话框提供了一套消息，此挂钩函数可利用此消息来最小化和控制对话框。它包括了从对话框发送来的一组通知消息，还有可以发送到从对话框检取信息的消息。关于这些消息的全部列表，参见 EXplorer-Style HookProcedures。如果挂钩函数处理 WM_CTLCOLORDLG 信息，它必须返回一个有效的刷子句柄，以刷对话的背景。总之，如果挂钩函数处理任何一个 WM_CTLCOLOR 消息，那么必须返回一个有效刷子句柄，以绘制指定控制背景。不须从挂钩函数调用 EndDialog 函数。相反，挂钩能调用 PostMessage 函数把带有 IDABORT 值的一个 WM_COMMAND 信息，传送给对话框函数 Posting IDABORT 来关闭对话框，使对话框函数返回 FALSE 值。若要了解为什么挂钩函数关闭对话框，必须提供挂钩函数和应用程序之间的联系机理。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：用户自定义。

2.5.16 OFNHookProcOldStyle

函数功能：该挂钩函数是应用程序定义或库定义的回调函数。此回调函数与 Open 和 Save AS 公共话框一起使用。函数接收指定给对话框函数的消息或通告。LPOFNHOOKPROC 类型定义了一个指向这种回调函数的指针。OFNHookProcOldStyle 是应用程序定义的或库定义的函数名的位置占有者。当创建一个 Open 或 Save As 公共对话框时，如果指定 OFN_EXPLORER 标志且需一个挂钩函数，那么必须应用一个 Explorer_style 的 OFNHookProc 挂钩函数。

函数原型：UINT CALLBACK OFNHookProcOldStyle（HWND hdlg，UINT uiMsg WPARAM wParam，LPARAM IParam）；

参数：

hdlg：指向消息指定的 Open 或 Save As 对话框窗口。

uiMsg：标识接收的消息。

wParam：指定有关消息的其他信息。精确的意义要根据 uiMSg 参数的值来决定。

IParam：指定有关消息的其他信息。精确的意义要根据 uiMsg 参数的值来决定。如果 uiMsg 参数显示了 WM_INITDIALOG 消息，IParam 是一个指向 OPENFILENAME 结构的指针。该结构含有创建对话框时指定的值。

返回值：如果挂钩函数返回零，缺省对话框函数处理消息；如果挂钩函数返回非零值，缺省对话框忽略消息。

备注：当用 GetOpenFileName 或 GetSaveFileName 函数创建一个旧式的 Open 或 Save As 对话框时。可以提供一个 OFNHookProcOldStyle 挂钩函数。为使挂钩函数生效，须应用传递到对话创建函数的 OPENHLENAME 结构，且须指定一个指向 IpnHook 成员中的挂钩函数的指针，及指定 Flags 成员中 OFN_ENABLEHOOK 标识。

如果挂钩函数处理 WM_CTLCOLORDLG 消息，那么必须返回一个有效的刷子句柄，以便绘制对话框的背景。总之，如果挂钩函数处理任何一种 WM_CTLCOLOR 信息，那么必须返回一个有效的刷子句柄以绘制指定控制的背景。

不须从挂钩函数调用 EndDialog 函数。相反，挂钩函数能调用 PostMessage 函数，把带有 IDABORT 值的一个 WM_COMMAND 信息传送给对话框函数 PostingIDABORT，并关闭对话框，使对话框函数返回 FALSE 值，若要了解为什么挂钩函数关闭对话框，必须提供挂钩函数和应用程序之间的联系机理。

可以对公共对话框的标准控制面板进行子分类。但是公共对话框函数也可以对控制面板进行子分类。

正因如此，必须在挂钩函数 WM_INTDIALDG 处理消息时对控制面板子分类。这就保证对话框函数设置子分类函数之前，子分类函数接收指定的控制消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：用户自定义。

2.5.17 PagePaintHook

函数功能：该挂钩函数是一个由应用程序或库定义的回调函数，该回调函数与 PageSetup 函数一起使用。该函数接收制作 PageSetup 对话框中样本页绘图的消息。LLPPAGEPAINTHOOK 类型定义了一个指向此回调函数的指针。PagePaintHook 是由应用程序或库定义的函数名的位置持有者。

函数原型：UINT CALLBACK PagePaintHook（HWND hdlg，UINT uiMsg，WPARAM wParam，LPARAM lParam）；

参数：

hdlg：指向 pagesetup 对话枢窗口的句柄。

uiMsg：标识接收的消息。

wparam：指定有关消息的其他信息。精确的意义要根据 uiMsg 参数的值来决定。

lParam：指定有关消息的其他信息。精确的意义要根据 uiMsg 参数的值来决定。

返回值：如果挂钩函数对前三个绘图顺序消息中的任何一个返回值为 TRUE。且前三个消息为 WM_PSD_PAGESETUPDLG，WM_PSD_FULLPAGERECT 或 WM_PSD_MINMARGINRECT），对话框不再发送消息。且直到下一次系统需要重画样本页时才绘出样本页。如果对所有的三个消息。挂钩函数返回 FALSE，对话框发送给图顺序的保留的消息。

如果挂钩函数对绘图顺序中保留消息的任何一个，返回值为 TRUE，那么对话框不绘样本页对应的部分，如果挂钩函数对这些信息中的任何一个返回值为 FALSE，对话框绘样本页部分。

备注 PageSetup 对话框含有一个样本页的构想。此构想展示了用户选择如何影响打印输出的外观。构想由代表选择页或信封类型的长方形组成，点线长方形代表当前页边。部分字母展示了文本在打印出的页中的状况。当用 PageSetupDlg 函数创建 Page Setup Dialog 对话框时，可提供一个 PagePaintHook 挂钩函数来制作样本页的表观。

为使挂钩函数生效，须用传递到创建对话函数的 PAGESETUPDLG 结构，且指定一个指向

IpfnPagePrintHook 成员中挂钩函数的指针和指定 Flags 成员中 PSD_ENABLEPAGEPAINTHOOK 标志。

无论何时，只要对话框将要绘样本页的内容，那么挂钩函数就接受以下列顺序列表的消息：WM_PSD_PAGESETUPDLG：对话框将要绘样本页，挂钩函数利用此消息准备给样本页里的内容。

WM_PSD_FULLPAGERECT：对话框要绘样本页，此信息指定样本页的长方形边。

WM_PSD_MINMARGINRECT：对话框将要绘样本页，此信息指定空白长方形。

WM_PSD_MARGINRECT：对话框将要绘空白长方形。

WM_PSD_REEKTEXTRECT：对话框将要在空白长方形内绘 Greek 文本。

WM_PSD_ENVSTAMPRECT：对话框将要在信封样本页上绘邮票长方形。此消息仅发向信封。

WM_PSD_YAFULLPAGERECT：对话框将要绘信封样本页的返回地址部分。此消息仅发向信封和其他纸张大小。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：用户自定义。

2.5.18 pageSetupHook

函数功能：该挂钩函数为由应用程序或库定义的回调函数。此回调函数与 PageSetupDlg 函数一起使用，

它接收为 PageSetup 公用对话框设置缺省对话框的消息或通告。LPPAGESETUPHOOK 类型定义了一个指向此回调函数的指针。PageSetupHook 是由应用程序或库定义的函数名的位置持有者。

函数原型：UINT CALLBACK PageSetupHook（HWND hdlg，UINT uiMsg，WPARAM wParam，LPARAM lParam）；

参数：

hdlg：指向消息指定给 Page Setup 对话框窗口的句柄。

uiMsg：标识接收的消息。

wParam：指定有关消息的其他信息。精确的意义要根据 uiMsg 参数的值来决定。

lParam：指定有关消息的其他信息。精确的意义要根据 uiMsg 参数的值来决定。

返回值：如果挂钩函数返回零，则缺省对话框函数处理消息；返回非零值，将忽略消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：用户自定义。

2.5.19 PrintHookProc

函数功能：该挂钩函数是一个由应用程序或库定义的回调函数，此回调函数与 PrintDlg 函数一起使用。

函数接收指定给 Print 公共对话框的缺省对话框函数的消息和通知。LPPWTHOOKPROC 类型定义了一个指向此回调函数的指针。PrintHookProc 为由应用程序或库定义的函数名的位置占有者。

函数原型：UINT CALLBACK PrintHookProc（HWND hdlg，UINT uiMsg，WPARAM wParam，LPARAM lParam）；

参数：

hdlg：指向消息指定给的 Print 公共对话框窗口的句柄。

uiMsg：标识接收的消息。

wParam：指定有关消息的其他信息、精确的意义由 uiMsg 参数的值来决定。

lParam：指定有关消息的其他信息、精确的意义由 uiMsg 参数的值来决定。

返回值：如果挂钩函数返回值为零，缺省对话框函数处理消息；返回非零值，将忽略消息。

备注：当用 PrintDlg 函数创建一个 Print 公共对话框时，可以提供一个 PrintHookProc 挂钩函数来处理指定给对话框函数的消息或通知。

为使挂钩函数生效，要用传递到创建对话框函数的 PRINTDLG 结构，且要指定 IpfnPrintHook 成员中挂钩函数的地址和指定 Flags 成员中 PD_ENABLEPRINTHOOK 标志。

缺省对话框函数把消息 WM_InITDIALOG 传送到挂钩函数之前先处理此消息。对于其他所有的消息，挂钩函数首先存放消息。然后其返回值决定此缺省对话函数是处理消息或忽略消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：commdlg.h；库文件：用户自定义。

2.5.20 SetupHookProc

函数功能：该挂钩函数是一个由应用程序或库定义的回调函数，该回调函数与 PrintDlg 函数一起使用。其他同函数 19。

函数原型：UINT CALLBACK SetupHookProc（HWND hdlg，UINT uiMsg，WPARAM wParam，LPARAM lParam）；

参数：

hdlg：指向消息指定给的 Print 公共对话框窗口的句柄。

UiMsg：标识接收的消息。

wParam：指定有关消息的其他信息、精确的意义由 uiMSg 参数的值来决定。

lParam：指定有关消息的其他信息、精确的意义由 uiMsg 参数的值来决定。

返回值：如果挂钩函数返回值为零，缺省对话框函数处理消息；返回非零值，将忽略消息。

备注；PrintSetup 对话框函数已经被 PageSetup 对话框所取代。这需要用新的应用程序来使用。但是，为了一致性，PrintDlg 函数仍支持 Print Setup 对话框的显示。可以提供一个 SetupHookProc 函数给 PrintSetup 对话框，从便处理指定给对话框函数的消息和通知。

为使挂钩函数生效，要用传递到对话创建函数的 PRINTDLG 结构，且指定 IpfnSetupHook 成员中挂钩函数的地址和指定 Flags 成员中 PD_ENABLESETUPHOOK 标志。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：commdlg.h；库文件：用户自定义。

2.6 标函数（Cursor）

2.6.1ClipCursor

函数功能：该函数把光标限制在屏幕上的一个矩形区域内，如果调用 SetCursor 或用鼠标设置的一个随后的光标位置在该矩形区域的外面，则系统自动调整该位置以保持光标在矩形区域之内。

函数原型：BOOL CliepCursor（CONST RECT* lpRect）；

参数：

IpRect：指向 RECT 结构的指针，该结构包含限制矩形区域左上角和右下角的屏幕坐标，如果该指针为 NULL（空），则光标可以在屏幕的任何区域移动。

返回值：如果成功，返回值非零；如果失败，返回值为零。若想获得更多错误信息，请调用 GetLastError。

备注：光标是一个共享资源，如果一个应用控制了光标，在将控制转向另一个应用之前，必须要使用 ClipCursor 来释放光标，该调用过程必须具有对窗口的 WINSTA_WRITEATTRIBUTES 访问权。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件；user32.lib。

2.6.2CopyCursor

函数功能：该函数复制一光标。

函数原型：HCURSOR CopyCursor（HCURSOR pcur）；

参数：

pcur：被复制光标的句柄

返回值；如果成功，返回值是复制光标的句柄；如果失败，返回值为 NULL（空）。若想获得更多错误信息，请调用 GetLastError 函数。

备注；CopyCursor 函数能使一个应用程序或一个动态连接库（OLL）得到一个属于另一模块的光标形状的句柄。如果另外一个模块被释放，则该应用程序仍然可以使用该光标形状。

在关闭之前，一个应用程序必须调用 DestroyCursor 函数来释放任何与该光标有关的系统资源。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib

2.6.3CreateCursor

函数功能：该函数创建一个指定大小、位模式和热点的光标。

函数原型： HCURSOR CreateCursor（HINSTANCE htnst，int xHotSpot；int yHotSpot；int nWidth；iut nHeight，CONST VOID *pvANDPlane，CONST VOID *pvXORPlane）；

参数：

hlnst：创建光标的应用程序的当前事例句柄。

xHotSpot：指定光标热点的水平位置。

yHotSpot：指定光标热点的垂直位置。

nWidth：以像素为单位指定光标的宽度。

nHeight：以像素为单位指定光标的高度。

pvANDPlane：指向一个字节数组的指针，该数组包含光标 AND 掩码的位值，就象设备相关的单色位图一样。

pvXORPlane：指向一个字节数组的指针，该数组包含光标 XOR 掩码的位值，就象设备相关的单色位图一样。

返回值： 如果成功，返回光标的值；如果失败，返回值为 NULL（空），若想获得更多错误信息，调用 GetLastError 函数。

备注： nWidth 和 nHeight 参数必须指定一个当前显示驱动支持的宽度和高度，因为系统不能创建一个其他尺寸的光标，为了确定显示驱动所支持的宽度和高度，请使用 GetSystemMetrics 函数，指定 SM_CXCUROR 或 SM_CYCURSOR 值。

在一个应用程序关闭之前，必须调用 DestroyCursor 函数来释放与光标有关的任何系统资源。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.6.4DestroyCursor

函数功能： 该函数销毁一个光标并释放它占用的任何内存，不要使用该函数去消毁一个共享光标。

函数原型： 8OOL DestroyCUrsor（HCURSOR hCursor）；

参数：

hCursor：要销毁的光标的句柄，该光标必须不在使用中。

返回值： 如果成功，返回非零；如果失败，返回值为零，若想获得更多错误信息，请调用 GetLastError 函数。

备注： DestroyCursor 函数销毁一个非共享的光标；不要用它销毁一个共享光标。一个共享光标只要调用它的模块仍在内存中，则该共享光标还是有效的，下面的函数可得到一个共享光标：LoadCursor；LoadCursorFronFile；Loadlmage（如果使用 LR-SHARED 标志）；

Copylmage（如果使用 LR_COPYRETURORG 标志并且 hlmags 参数是一个共享光标）。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.6.5GetClipCursor

函数功能： 该函数检取一个矩形区域的屏幕坐标，光标被限制在该矩形区域之内。

函数原型： BOOL GetClipCursor（LPRECT IpRect）；

参数：

IpRect：一个 RECT 结构的指针；接收限制矩形的屏幕坐标。如果该光标没有被限制在一个矩形区域内，则该 RECT 结构接收屏幕的尺寸。

返回值：如果成功，返回非零；如果失败，返回值为零，若想获得更多错误信息，请调用 GetLastError 函数。

备注：该光标是一个共享光标，如果一个应用程序使用 ClipCursor 函数来限制该光标，那么在它放弃控制转向另一个应用之前必须使用 ClipCursor 来释放该光标，该调用过程必须具有对窗口站的 WINSTA_READATTRIBUTES 访问权限。

速查：Windows NT：3.1 及以上下版本；Windows：95 及以上版本；Windows CE：不支持；头文件：Winusre.h；库文件：user32。lib。

2.6.6GetCursor

函数功能：该函数检取当前光标的句柄。

函数原型：HCURSOR GetCursor（VOID）；

参数：无。

返回值：返回值是当前光标的句柄，如果没有光标，返回值为 NULL。

速查：Windows NT：3.1 及以上版本；Wjhdows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.6.7GetCursorpos

函数功能：该函数检取光标的位置，以屏幕坐标表示。

函数原型：BOOL GetCursorPos（LPPOlNT IpPoint）；

参数：

IpPint：POINT 结构指针，该结构接收光标的屏幕坐标。

返回值：如果成功，返回值非零；如果失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：光标的位置通常以屏幕坐标的形式给出，它并不受包含该光标的窗口的映射模式的影响。该调用过程必须具有对窗口站的 WINSTA_READATTRIBUTES 访问权限。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.6.8LoadCursorFromFile

函数功能：该函数根据一个文件中所含的数据创建光标。该文件由它的名字所指定或由一个系统光标鉴别器指定，该函数返回一个新建光标的句柄，文件所包含的光标数据可以是光标格式（CUR）或运动光标格式（.ANI）。

函数原型：HCURSOR LoadCursorFromFile（LPCTSTR IpFileName）；

参数：

IpFileName：指明创建光标所用的文件数据资源，文件中的数据格式必须是 COR 或 ANI，如果 IpFileName 的高位字为非零，则 IpFileName 就是指向一个字符串的指针，该字符串是包含光标数据的文件的名字。

如果 IpFileName 的高位字为零，低位字是系统光标标识符，则该函数在 WIN.INI 文件中搜索与系统光标名字有关的[Cursors]条目，下面是系统光标的名字和标识符的清单：

“AppStarting”：OCR_APPSTARTING；“Arrow”：OCR_NORMAL “Crosshair”：OCR_CROSS；“Hand”：

WindowsNT5.0 and laterOCR_HAND；“Help”：OCR_HELP；“IBeam”：OCR_IBEAM；“Icon”：OCR_ICON；“No”：OCR_NO；“Size”：OCR_SIZE；

“SizeAll”：OCR_SIZEALL；“SizeNESW”：OCR_SIZENESW；“SizeNS”：OCR_SIZENS；“SizeNWSE”：OCR_SIZENESW；“SizeWE”：OCR_SIZEWE；“UpArrow”：OCR_UP；“Wait”：OCR_WAIT

返回值：如果成功，返回值是新建光标的句柄；如果失败，返回值为空（NULL）。若想获得更多错误信息，请调用 GetLastError 函数。GetLastError 也许会返回如下的值：ERROR_FILE_NOT_FOUND，没有找到指定的文件。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 中实现 Unicode 和 ANSI 两个版本。

2.6.9SetCursor

函数功能：该函数确定光标的形状。

函数原型：HCURSOR SetCursor（HCURSOR hCursor）；

参数：

hCursor：光标的句柄，该光标由 CreateCursor 函数载入。如果该参数为 NULL，则该光标从屏幕上移开。在 Windows95 中该光标的宽和高是 GetSysfemMefirics 函数的返回值 SM_CXCURSOR 和观_CYCURSOR，并且光标的位深必须和显示器的位深相匹配，或者光标是单色的。

返回值：如果有前一个光标，则返回值是前光标的句柄；如果没有前光标，则返回值是 NUL。

备注：仅当新光标与前光标不同时，才设置该光标，不然的话，该函数立即返回。该光标是一个共享资源。一个窗口仅当光标在其客户区域，或者它正在捕捉鼠标输入时，它才设置光标的形状。在一个没有鼠标的系统中，该窗口在光标离开它的客户区域或它要把控制权交给其他窗口之前，它会恢复以前的光标。

如果应用程序必须在窗口中设置光标，必须确保指定窗口类的类光标被设为 NULL，如果类光标不是 NULL，则每次移动鼠标时，系统都要恢复类光标。

如果内部的光标显不计数值小于零，则光标不在屏幕上显示。当一个应用程序使用 ShowCursor 函数隐藏光标的次数多于显示光标的次数时，则会发生这种情况。

Windows CE：当一个目标平台不支持鼠标光标时，使用 cursor 光标组件，该光标组件仅支持等待光标，设置等待光标，使用如下的代码：SetCursor（LoadCursor（NULL，IDC_WAIT））；当一个目标平台不支持鼠标光标时，使用 mcursor 光标组件。该组件不支持彩色光标。

速查：Window NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.6.10 SetCursorPos

函数功能：该函数把光标移到屏幕的指定位置。如果新位置不在由 ClipCursor 函数设置的屏幕矩形区域之内，则系统自动调整坐标，使得光标在矩形之内。

函数原型：BOOL SetCursorPOS（int X，int Y）；

参数：

X：指定光标的新的 X 坐标，以屏幕坐标表示。

Y：指定光标的新的 Y 坐标，以屏幕坐标表示。

返回值：如果成功，返回非零值；如果失败，返回值是零，若想获得更多错误信息，请调用 GetLastError 函数。

备注：该光标是共享资源，仅当该光标在一个窗口的客户区域内时它才能移动该光标。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.6.11 SetSystemCursor

函数功能： 该函数使一个应用程序定制系统光标。它用 hcur 规定的光标内容代替 id 定义的系统光标内容，接着销毁 hour。

函数原型： BOOL SetSystemCursor（HCURSOR hour，DWORD id）；

参数：

hcur：光标的句柄，该函数 hcur 标识的光标的内容代替 id 定义的系统光标内容。系统通过调用 DestroyCursor 函数销毁 hour。因此 hour 不能是由 LoadCursor 函数载入的光标。要指定一个从资源载入的光标，先用 CopyCursor 函数复制该光标，然后把该副本传送给 SetSystemCursor 函数。

Id：指定由 hour 的内容替换系统光标。

下面是一系列的系统光标标识符：

OCR_APPSTARTING：标准箭头和小的沙漏；OCR_NORAAC：标准箭头

OCR_CROSS：交叉十字线光标：OCR_HAND：手的形状（WindowsNT5.0 和以后版本）

OCR_HELP：箭头和向东标记；OCR_IBEAM：I 形梁；OCR_NO：斜的圆

OCR_SIZEALL：四个方位的箭头分别指向北、南、东、西

OCR_SIZENESEW：双箭头分别指向东北和西南；OCR_SIZENS：双箭头，分别指向北和南

OCR_SIZENWSE：双箭头分别指向西北和东南；OCR_SIZEWE：双箭头分别指向西和东

OCR_UP：垂直箭头：OCR_WAIT：沙漏**返回值：** 如果成功，返回非零值；如果失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.6.12 ShowCursor

函数功能： 该函数显示或隐藏光标。

函数原型： int ShowCursor（BOOL bShow）；

参数：

bShow：确定内部的显示计数器是增加还是减少，如果 bShow 为 TRUE，则显示计数器增加 1，如果 bShow 为 FALSE，则计数器减 1。

返回值： 返回值规定新的显示计数器。

备注：该函数设置了一个内部显示计数器以确定光标是否显示，仅当显示计数器的值大于或等于 0 时，光标才显示，如果安装了鼠标，则显示计数的初始值为 0。如果没有安装鼠标，显示计数是 C1。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.6.13 LoadCursor

函数功能： 该函数从一个与应用事例相关的可执行文件（EXE 文件）中载入指定的光标资源。该函数已被 Loadlmage 函数替代。

函数原型：HCURSOR LoadCursor（HINSTANCE hlnstance，LPCTSTR lpCursorName）；

参数：

hlnstance：标识一个模块事例，它的可执行文件包含要载入的光标。

LpCursorName：指向以 NULL 结束的字符串的指针，该字符串存有等载入的光标资源名。该参数低位字节和高位字节 0 组成资源标识器也可以由低位字为资源标识符和高位字为零组成。也可以用 MADEINTRES_OURCE 宏命令创建该值。

要使用 Win32 预定义的一个光标，应用程序必须把 hlnsfance 参数设为 NULL，并把 IpCursorName 设为如下值之一：

IDC_APPSTARING：标准箭头和小沙漏；IDC_ARROW：标准光标；IDC_CROSS：十字光标。

返回值：如果成功，返回值是新载入的光标的句柄：如果失败，返回值是 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

备注：LoadCursor 函数仅载入没有被载入过的光标资源，否则，它检取已存在的光标资的句柄。仅当 IpCursorName 参数指向一个光标资源时，该函数才返回一个有效的光标句柄。如果 IpCursorName 不是指向光标而是指向了其他类型的资源（如 icon），则该函数返回值不是 NULL，尽管它不是一个有效的光标句柄，该函数为当前显示设备光标搜寻最贴切的光标资源。光标资源可以是彩色或单色的位图。

Windows CE：当目标平台不支持鼠标光标时。使用 cursor 光标组件。该光标组件支持的唯一的光标是等待光标（IDC_WAIT）。使用 LoadCursor 函数与 SetCursor 函数可设值等待光标。SetCursor（LoadCursor（NULL，IDC_WAIT））当目标平台不支持鼠标光标时，使用 cursor 光标组件，该组件以桌面窗口平台同样的方式支持 LoadCursor 函数，唯一不同的是仅支持单色光标。Windows CE 不支持彩色光标。试图载入一个彩色光标，将产生难以预料的结果。返回值是不确定的。

2.7 对话框函数 Dialog Box

2.7.1CreateDialog

函数功能：CreateDialog 宏从一个对话框模板资源创建一个无模式的对话框，CreateDiaog 宏使用 CreateDialogParam 函数。

函数原型：HWND CreateDialog(HINSTANCE hlnstance,LPCTSTR lpTemplate,HWND hWndParent,DLGPROC IpDialogFunc)；

参数：

hlnstance：标识模块事例，该模块的可执行文件含有对话框模板。

pTemplate：标识对话框模板，此参数或是指向一个以结尾的字符串指针，该字符串指定对话框模板名，或是指定对话框模板的资源标识符的一个整型值。如果此参数指定了一个资源标识符，则它的高位字一定为零，且低位字一定含有标识符，一定用 MAKEINTRESOURCE 宏来创建此值。

hWndParent：标识拥有对话框的窗口。

lpDialogFunc：指向对话框应用程序的指针。有关更多的对话框应用程序的指针，参见 DialogProc。

返回值：如果函数调用成功，则返回值为指向对话框的句柄；如果函数调用失败，则返回值为 NULL。若想获得更多的错误信息，可调用 GetLastError 函数。

备注： CreateDialog 函数用 CreateWindowEx 函数来创建对话框。然后 CreateDialog 函数把一个 WM_INITDIALOG 消息（如果模板指定 DS_SETFONT 类型，则加上一个 WM_SETFONT 消息）传送到对话框应用程序。如果模板指定 WS_VISIBLE 风格，则函数显示对话框，最后 CreateDlalog 返回指向对话框的窗口句柄。

CreateDialog 函数返回之后，应用程序通过 Showwindow 函数显示对话框（如果还没有显示）。应用程

序通过利用 DestroyWindow 函数来清除对话框。

Windows 95 和以后版本：系统每个对话框模板可以支持最多达 255 个控制。如果要把多于 255 个控制放入对话框中，必须在 WM_INITDIALOG 消息处理器中创建控制，而不是把它们放入模板中。

Windows CE：IpTempIate 参数指向的对话框模板中，DIGTEMPLATE 结构并不支持所有类型。

速查:Windows NT：3.1 及以上版本:Windows：95 及以上版本：Windows CE：1.0 及以上版本：头文件：Winuser.h；库文件：user32.lib；Unicode：Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.2CreateDialoglndirect

函数功能：该宏在内存中从对话框模板上创建一个无模式对话框。此宏使用 CreateDialoglndirectparam 函数。

函数原型： HWND CreateDialoglndirect（HINSTANCE hlnstance,LPCDLGTEMPLATE IPTemplate,HWNDhWndParent,DLGPROC IPDialogFunc）；

参数：

hlnstance: 标识创建对话框的模块的事例。

IPTemplate: 指向含有一个模板的全局内存对象的指针。CreateDialoglndirect 用此模板创建对话框。对话框模板由描述对话框的标题组成，跟随着标题之后的是描述每一个控制的一个或多个数据块，模板可以用标准格式或扩展格式。

在标准模板中，标题是由 DLGTEMPLATE 结构跟随一个变长数组组成。每个控制的数据是由 DLGTEMPLATE 结构跟随一个变长数组组成。

在扩展模板中，标题用 DLGTEMPLATEEX 格式，且控制定义用 DLGITEMPLATEEX 格式。CreatDialoglndirect 函数返回后，可释放模板，此模板仅用于启动对话框。

hWndParent：标识拥有对话框的窗口。

IpDialogFunc：指向对话框应用程序的指针，有关更多的对话框应用程序的指针，参见 DialogProc。

返回值：如果函数调用成功，则返回值为指向对话框的句柄。如果函数调用失败，则返回值为 NULL。若想获得更多错误信息，可调用 GetLastError 函数。

备注：CreateDialOglndirect 宏使用 CreateWindowEx 函数来创建对话框，然后该函数把一个 WM_INITDIALOG 消息发送到对话框应用程序，如果模板指定 DS_SETFONT 类型，则函数也把一个 WM_SETFONT 消息发送到对话框应用程序。如果模板指定 WS_VISIBLE 类型，则函数显示对话框，最后 CreateDialoglndirect 返回

指向对话框的窗口句柄。

CreateDialoglndirect 函数返回之后，可用 ShowWindow 函数来显示对话框（如果还没有显示）。用 DestroyWindow 函数来清除对话框。

在标准对话框模板中，DLGTEMPLATE 结构和每一个 DLGITEMTEMPLATE 结构必须按 DWORD 边界对齐，遵循 DLGEMTEPLATE 结构而创建的数据数组也一定按 DWORD 边界对齐。模板中其他所有变长数组一定要按 DWORD 边界进行调整。

在扩展对话框模板上 DLGTEMPLATEEX 结构和每一个 DLGITEMTEMPLATEEX 结构必须按 DWORD 边界对齐；遵循 DLGEMTEPLATE 结构而创建的数据数组也一定按 DWORD 边界对齐。模板中其他所有变长数组一定要按 DWORD 边界进行调整。

所有对话框模板的字符串，例如对话框和按钮的标题，一定是 Unicode 字符串。使用

MltiByteToWidechar 函数产生这些 Unicode 字符串可以创建在 Windows 和 Windows NT 两种系统上工作的代码。

Windows 95 和以后版本：系统可支持每个对话框模板最多为 255 个控制。为把多于 255 个控制放入对

话框，可以在 WM_INITDLAIOG 消息处理器中创建控制，而不是把它们放入模板中。

Windows CE：lpTempate 参数指向的对话框模板中，DLGTEMPLATE 结构并不支持所有的类型。

速查:Windows NT：3.1 及以上版本;Windows:95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h;库文件：user32.lib;Unicode: 在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.3CreateDialoglndirectParam

函数功能：该函数从内存中的对话框模板上创建一个无模式对话框，在显示对话框之前，函数把应用程序定义的值作为 WM__INITDIALOG 消息的 IParam 参数传送到对话框过程。应用程序可用此值初始化对话框控制。

函数原型：HWND CreateDialoglndirectParam(HINSTANCE hlnstance，LPCDLGTEMPLAT IPTemPIate，HWNDhWndParent,DLGPROC IpDialogFunc，LPARAM IParamlnlt)；

参数：

hlnstance：标识将创建对话框的模块的事例。

IpTemplate：指向一个含有模板的全局内存对象的指针，CreateDialoglndirectParam 用该模板来创建对话框。

对话框模板由描述对话框的标题组成，跟随着标题之后的是描述每一个控制的一个或多个数据块，模板可以用标准格式或扩展格式。

在标准模板中，标题是由 DLGTEMPLATE 结构跟随一个变长数组组成。每个控制的数据是由 DLGTEMPLATE 结构跟随一个变长数组组成。

在扩展模板中，标题用 DLGTEMPLATEEX 格式，且控制定义用 DLGITEMPLATEEX 格式。

CreateDiloglndirectParam 函数返回后，可释放模板，此模板仅用于启动对话框。

HWndParent：标识拥有对话框的窗口。

IpDialogFunc：指向对话框过程的指针，有关更多的对话框过程的信息，参见 DialogProc。

lParamlnit：指定传递到 WM__INIDIALOG 消息的 IParam 参数中对话框中的值。

返回值：如果函数调用成功，则返回值为指向对话框的句柄：如果函数调用失败，则返回值为 NULL。若想获得更多的错误信息，可调用 GetLastError 函数。

备注：CreatDialoglndirectParam 函数使用 createWindowEx 函数来创建对话框，然后该函数把一个 WM_INTDIALOG 消息发送到对话框应用程序，如果模板指定 DS_SETFONT 类型，则函数也把一个 WM_SETFONT 消息发送到对话框应用程序。如果模板指定 WS__VISBLE 类型，则函数显示对话框，最后 CreatDialogdirectParam 返回指向对话框的窗口句柄。

CreatDialoglndirectParam 函数返回之后，可用 ShowWindow 函数来显示对话框（如果还没有显示）。用 DestroyWindow 函数来清除对话框。

在标准对话框模板中，DLGTEMPLATE 结构和每一个 DLGITEMTEMPLATE 结构必须按 DWORD 边界对齐，遵循 DLGEMTEPLATE 结构而创建的数据数组也一定按 DWORD 边界对齐。模板中其他所有变长数组一定要按 DWORD 边界进行调整。

在扩展对话框模板上 DLGTEMPLATEEX 结构和每一个 DLGITEMTEMPLATEEX 结构必须按 DWORD 边界对齐，遵循 DLGEMTEPLATE 结构而创建的数据数组也一定按 DWORD 边界对齐。模板中其他所有变长数组一定要按 DWORD 边界进行调整。

所有对话框模板的字符串，例如对话框和按钮的标题，一定是 Unicode 字符串。使用

MultiByteTowidechar 函数产生这些 Unicode 字符串可以创建在 Windows 和 Windows NT 两种系统上工作的代码。

Windows 95 和以后版本：系统可支持每个对话框模板最多为 255 个控制。为把多于 255 个控制放入对话框，可以在 WM_INITDLALOG 消息处理器中创建控制，而不是把它们放入模板中。

Windows CE：可视屏幕面积之外的对话框不能自动地被重新定位。

如果用户在对话框有输入焦点的同时按下 Alt＋H，则系统把一个 WM_HELP 消息传送到对话过程，应用程序应该通过显示对话框描述表积极帮助来响应此消息。

DLGTEMPLATE 结构的类型成员不支持下列类型：

DS_SETFONT：在对话框中不能设置字体。

DS_RECURSE：不需要。任何一个子对话框可自动地被看作递归对话框。

DS_CONTROL：不需要。

WS_EX_CONTROLPARENT:所有对话框都被自动假设为控制母体。用 DS_CENTER 类型可得到缺省位置定位。如果没有指定 WS_CHLD. 则假设为 WS_POPUP 类型。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuse.h 库文件:user32.lib;Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.4CreateDialogParam

函数功能：该函数根据对话框模板资源创建一个无模式的对话框。在显示对话框之前，函数把一个应用程序定义的值作为 WM_INITDIALOG 消息 IParam 参数传到对话框过程应用程序可用此值来初始化对话框控制。

函 数 原 型： HWND CreateDialogParam(HINSTANCE hlnstancem,LPCTSTR IpTemplateName， HWND hWndParent，DLGPROCIpDialogFunc, LPARAM dwlniParam);

参数：

hlnstance：标识一个模块的事例,该模块的可执行文件含有对话框模板。

IpTemplateName：标识对话框模板。此参数可以指向一个以 NULL 结尾的字符串的指针,该字符串指定对话框模板名,或是指定对话框模板的资源标识符的一个整型值。如果此参数指定了一个资源标识符，则它的高位字一定为零且低位字一定含有标识符。一定用 MAKENTRESOURDE 宏指令创建此值。

HwndParent：指定拥有对话框的窗口。

IpDialogFunc：指向对话框过程的指针。有关对话框过程的更详细的信息，请参见 DialogProc。

dwlnitParam：指定传递到 WM_INITDIALOG 消息的 IParam 参数中的对话框过程的值。

返回值：如果函数调用成功则返回值为指向对话框的窗口句柄。如果函数调用失败则返回值为 NULL。

若想获得更多错误信息,请调用 GetlastError 函数。

备注：CreateoialogParam 函数用 CreateWindowEx 函数创建对话框。CreateDialogParam 函数然后把一个 WM_INITDIALOG 消息(和一个 WM__SETFONT 消息,如果模板指定 DS SETFONT 类型）传递到对话框过程。如果模板指定 WS_VISIBLE 类型，则函数显示对话框，最后 CreateDialogParam 返回对话框的窗口句柄。

CreateDialogParam 返回之后应用程序用 ShowWindow 显示对话框（如果还没有显示）。应用程序用 DestroyWindoW 函数来清除对话框。

Windows 95 和以后版本：系统可支持每个对话框模板中最多 255 个控制。为把大于 255 个的控制放入对话框,需要在 WM_INITDIALOG 消息处理器中创建控制,而不是把他们放入模板中。

Windows CE：IPTemplateName 参数指向的对话框模板中 DLGTEMPLATE 结构并不支持所有的类型。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件；winuser.h;库文件：use32.lib;Unicode:在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.5DefDlgProc

函数功能： 该函数为属于应用程序定义的对话框类的窗口过程提供缺省的消息处理。

函数原型：LRESULT DefDlgProc（HWND hDlg，UINT Msg，WPARAM wParam, LPARAM Param)；

参数：

hDlg：指定对话框。

Msg：指定消息数目。

wParam：指定消息中特定的其他信息。

Iparam：指定消息中特定的其他信息。

返回值：返回值指定消息处理的结果且依赖于发送的消息。

备注：DefDlgProc 函数为对话枢预定义类的窗口应用程序。此应用程序通过把消息传送到对框应用程序和为对话框应用程序返回的任何一个 FALSE 消息，提供缺省处理而为对话框提供内部的处理程序。为自定义对话框创建自定义应用程序的应用程序，常用 DefDlgProc 而不是 DefWindowProc 函数来执行缺省的消息处理。

应用程序通过用合适的信息来填充一个 WNCLASS 结构和通过用 RegisterClass 函数登记的类创建自定义对话框类。一些应用程序用 GetClasslnfo 函数指定与定义对话框的名来填充此机构。在这种情况下，应用程序在登记之前至少改变 IpszClassNamede 数目。在所有的情况下，对于自定义对话框的 WNDCLASS 的 cbWndExtra 成员一定至少设置为 DLGWINDOWEXTRA。

DefDlgProc 函数一定不要通过一个对话框应用程序来调用，这样做会导致循环执行。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h;库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.6DialogBox

函数功能： 该宏根据对话框模板资源创建一个模态的对话框。DialogBOX 函数直到指定的回调函数通过调用 EndDialog 函数中止模态的对话框才能返回控制。该宏使用 DialogBoxParam 函数。

函数原型： int DialogBox(HINSTANCE hlnstance,LPCTSTRIpTemplate,HWND hWndParent,DLGPROC IpDialogFunc)；

参数：

hlnstance：标识一个模块的事例该模块的可执行文件含有对话框模板。

IpTemplate：标识对话框模板。此参数可以是指向一个以 NULL 结尾的字符串的指针，该字符串指定对话框模板名，或是指定对话框模板的资源标识符中的一个整型值。如果此参数指定了一个资源标识符则它的高位字一定为零，且低位字一定含有标识符。一定用 MAKEINTRESOURDE 宏指令创建此值。

hWndParent：指定拥有对话框的窗口。

IpDialogFunc：指向对话框过程的指针。有关更详细的关于对话框过程的信息,请参见 DialogProc。

返回值：如果函数调用成功，则返回值为在对函数 EndDialog 的调用中的 nResult 参数. 该函数用于中止对话框。如果函数调用失败，则返回值为 C1。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：DialOgBox 宏用 CreateWindowEx 函数创建对话框。DialogBox 函数然后把一个 WM_INITDIALOG 消息(和一个 WM-SETFONT 消息，如果模板指定 DS_SETFONT 类型）传递到对话框过程。不管模板是否指定 WS_VISIBLE 类型，函数显示对话框，并且使拥有该对话框的窗口（也称属主窗口）失效，且为对话框启动它本

身的消息循环来检索和传递消息。

当对话框应用程序调用 EndDialog 函数时，DialogBox 函数清除对话框户止消息循环,使属主窗口生效（如果以前有效），且返回函数 EndDialog 调用中的 nReSUlt 参数。

Windows 95 和以后版本：系统可支持每个对话框模板中最多 255 个控制。为把大于 255 个的控制放入对话框，需要在 WM_INITDIALOG 消息处理器中创建控制，而不是把他们放入模板中。

Windows CE:lpTemplateName 参数指向的对话框模板中 DLGTEMPLATE 结构并不支持所有的类型。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件:user32.lib Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.7DialogBoxlndirect

函数功能：该宏根据内存中的对话框模板资源创建一个模态的对话框。DialogBoxlndirect 宏直到指定的回调函数通过调用 EndDialog 函数中止模态的对话框才能返回。DiaogBoxlndirect 宏使用 DialogBoxParam 函数。**函数原型：** int DialogBoxlndirect(HINSTANCE hlnstance,LPDLGTEMPLATE IpTemplate，HWND hWndParent，DLGPROClpDialogFunc)；

参数：

hlnstance：标识一个模块的事例，该模块创建对话框。

IpTemplate：此参数指向含有一个模板的全局内存对象的指针。DialogBoxlndirect 用此模板创建对话框。对话框模板由描述对话框的标题组成，跟随着标题之后的是描述每一个控制的一个或多个数据块，模板可以用标准格式或扩展格式。

在标准模板中，标题是由 DLGTEMPLATE 结构跟随一个变长数组组成。每个控制的数据是由 DLGTEMPLATE 结构跟随一个变长数组组成。

在扩展模板中，标题用 DLGTEMPLATEEX 格式，且控制定义用 DLGITEMPLATEEX 格式。

HWndParent：指定拥有对话框的窗口。

IpDialogFunc：指向对话框过程的指针。有关更详细的关于对话框过程的信息，请参见 DialogProc。

返回值：如果函数调用成功则返回值为在对函数 EndDialog 的调用中的 nResult 参数,该 EndDialog 函数用于中止对话框。如果函数调用失败，则返回值为 C1。 若想获得更多错误信息请调用 GetLastError 函数。

备注：DialogBoxlndirect 宏使用 CreateWindowEx 函数创建对话枢. 然后把一个 WM_INITDIALOG 消息传递到对话框过程。如果模板指定 DS_SETFONT 类型 DialogBoxlndirect 函数将一个 WM_SETFONT 消息传递到对话框。(不管模板是否指定 WS_VISIBLE 类型)，函数显示对话框使属主窗口失效，且为对话框启动它本身的消息循环来检索和传递消息。

当对话框应用程序调用 EndDialog 函数时，DialogBoxlndirect 函数清除对话框，中止消息循环，使主窗口生效（如果以前有效）且返回 EndDialog 函数调用中的 nResult 参数。

在标准对话框模板中，DLGTEMPLATE 结构和每一个 DLGITEMTEMPLATE 结构必须按 DWORD 边界对齐，遵循DLGEMTEPLATE 结构而创建的数据数组也一定按 DWORD 边界对齐。模板中其他所有变长数组一定要按 DWORD 边界进行调整。

在扩展对话框模板上 DLGTEMPLATEEX 结构和每一个 DLGITEMTEMPLATEEX 结构必须按 DWORD 边界对齐，遵循 DLGEMTEPLATE 结构而创建的数据数组也一定按 DWORD 边界对齐。模板中其他所有变长数组一定要按 DWORD 边界进行调整。

所有对话框模板的字符串，例如对话框和按钮的标题，一定是 Unicode 字符串。使用

MultiByteToWidechar 函数产生这些 Unicode 字符串可以创建在 Windows 和 Windows NT 两种系统上工作的代码。

Windows 95 和以后版本：系统可支持每个对话框模板最多为 255 个控制。为把多于 255 个控制放入对

话框。可以在 WM_INITDLAIOG 消息处理器中创建控制，而不是把它们放入模板中。

Windows CE：IpTemplateName 参数指向的对话框模板中 DLGTEMPLATE 结构并不支持所有的类型。

速查：Windows NT:3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h;库文件：user32.lib Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.8DialogBoxlndirectParam

函数功能；该函数根据内存中对话框模板创建一个模态的对话框。在显示对话框之前，函数把一个应用程序定义的值作为 WM_INITDIALOG 消息的 IParam 参数传到对话框过程,应用程序可用此值来初始化对话框控制。

函数原型:int DialogBoxlndirectParam(HINSTANCE hlnstance，LPCDLGTEMPLATE hDialogTemplate, HWND hWndParent,DLGPROC IpDiaIogFunc，LPARAM dwlnitParam)；

参数：

hlnstance：标识一个模块的事例，该模块创建对话框。

hDialogTemplate:此参数指向含有一个模板的全局内存对象的指针。DialogBoxlndirectParam 用此模板创建对话框。对话框模板由描述对话框的标题组成，跟随着标题之后的是描述每一个控制的一个或多个数据块，模板可以用标准格式或扩展格式。

在标准模板中，标题是由 DLGTEMPLATE 结构跟随一个变长数组组成。每个控制的数据是由 DLGTEMPLATE 结构跟随一个变长数组组成。

在扩展模板中，标题用 DLGTEMPLATEEX 格式，且控制定义用 DLGITEMPLATEEX 格式。

hWndParent：指定拥有对话框的窗口。

IpDirlogFunc：指向对话框过程的指针。有关对话框过程更详细的信息请参见 DialogProc。

dwlnitParam：指定传递到 WM_INITDIALOG 消息的 IParam 参数中的对话框的值。

返回值：如果函数调用成功，则返回值为在函数 EndDialog 的调用中的 nResult 参数，该 EndDialog 函数用于中止对话框。如果函数调用失败，则返回值为 C1。若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows:95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h;库文件：user32.lib;Unicode:在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.9DialogBoxParam

函数功能：该函数根据对话框模板资源创建一个模态的对话框。在显示对话框之前，函数把一个应用程序定义的值作为 WM_INITDIALOG 消息的 IParam 参数传到对话框过程,应用程序可用此值来初始化对话框控制。

函数原型：int DialogBoxParam（HINSTANCE hlnstance，LPCTSTR IpTemplateName,HWND hWndParent，DLGPROCIPDialogFunc,LPARAM dwlnitParam)；

参数：

hlnstance：标识一个模块的事例，该模块的可执行文件含有对话框模板。

IpTemplateName：标识对话框模板。此参数可以指向一个以 NULL 结尾的字符串的指针,该字符串指定对话框模扳名，或是指定对话框模板的资源标识符的一个整型值。如果此参数指定了一个资源标识符，则它的高位字一定为零，且低位字一定含有标识符。一定用 MAKEINTRESOURDE 宏指令创建此值。

hWndParent：指定拥有对话框的窗口。

IpDirlogFunc：指向对话框过程的指针。有关更详细的关于对话框过程的信息，请参见 DialogProc。

dwlnitaram：指定传递到 WM_INITDIALOG 消息的 IParam 参数中的对话框过程的值。

返回值：如果函数调用成功则返回值为在对函数 EndDialog 的调用中的 nResult 参数，该 EndDialog 函数用于中止对话框。如果函数调用失败，则返回值为 C1。若想获得错误信息，请调用 GetLastError 函数。

备注：DialogBoxParam 函数用 CreateWindowEx 函数创建对话框。然后把一个 WM_INITDIALOGG 消息传递到对话框过程。如果模板指定 DS_SETFONT 类型，DialogBoxParam 函数把一个 WM_SETFONT 消息传递到对话框过程。(不管模板是否指定 WS_VISIBLE 类型）,函数显示对话框使拥有窗口失效,且为对话框启动它本身的消息循环来检取和传递消息。

当对话框应用程序调用 EndDialog 函数时，DialogBoxParam 函数清除对话框中止消息循环；使拥有窗口生效（如果以前有效），且返回函数 EndDialog 的调用中的 nReSUlt 参数。

Windows 95 和以后版本：系统可支持每个对话框模板最多为 255 个控制。为把多于 255 个控制放入对话框，可以在 WM_INITDLALOG 消息处理器中创建控制，而不是把它们放入模板中。

Windows CE：IPTemplateName 参数指向的对话框模板中 DLGTEMPLATE 结构并不支持所有的类型。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h;库文件：USer32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.10 DialogProc

函数功能：该函数为一个应用程序定义可与 DialogBOX 函数一起使用的回调函数。它处理发送到一个模态的或无模式对话框的消息。DLGPROC 类型定义了一个指向此回调函数的指针。DialogProc 函数是应用程序定义函数名的一个占位符。

函数原型：BOOL CALLBACK DialogProc(HWND hwndDlg,UINT UMsg,WPARAM wParam,LPARAM IParam);

参数：

hwndDlg：指定对话框。

uMsg：指定消息。

wParam：指定消息特定的其他信息。

Iparam：指定消息特定的其他信息。

返回值：除了对 MM_INITDIALOG 消息的响应之外如果函数处理消息，则对话框应用程序应该返回非零值.

如果函数不处理消息，则对话框应用程序应该返回零值。再响应 WM_INITDIALOG 消息时，如果函数调用 SetFocus 设置对话框中控制中的一个焦点,则对话框应用程序应该返回零值，否则对话框应用程序应该返回非零值在,这种情况下系统对能够有焦点的对话框中的第一个控制设置焦点。

备注：只要为对话框使用对话框类时才应该使用对话框应用程序。这是缺省的类，并且在对话框模板中没有指定明显的类时才使用。尽管对话框应用程序同 Windows 应用程序类似，但它不能调用 DefWindowProc

函数来处理不需要的消息。不需要的消息通过对话框窗口应用程序内部处理。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winser.h；库文件：用户定义。

2.7.11 EndDialog

函数功能：该函数清除一个模态对话框,并使系统中止对对话框的任何处理。

函数原型：BOOL EndDialog(HWND hDlg,int nResult);

参数：

hDlg：表示要被清除的对话框窗口。

NResult：指定从创建对话框函数返回到应用程序的值。

返回值：如果函数调用成功，则返回值为非零值；如果函数调用失败则返回值为零。若想获得错误信息请调用 GetLastError 函数。

备注：由 DialogBox，DialogBoxParam、DialogBoxlndirect 和 DialogBoxlndirectParam 函数创建的对话框一定要用 EndDialog 函数来清除。应用程序从对话框应用程序内部调用 EndDialog 函数，该函数不能为其他目的而供使用。

对话框应用程序可以在任何时间调用 EndDialog 函数；甚至在 WM_INITDIALOG 消息处理过程中。如果应用程序在 WM_INTDIALOG 消息处理过程中调用该函数，则对话框在显示和输入焦点被设置之前对话框被清除。

EndDialog 函数并不立即清除对话框。而是设置一个标志，并且允许对话框应用程序把控制权返回系统。系统在试图从应用程序队列检索下一个消息之前检测标志。如果已经设置了标志则系统中止消息循环，清除对话框,且用 nResUlt 中的值作为从创建对话框的函数中返回的值。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.7.12 GetDialogBaseUnits

函数功能：该函数返回系统的对话基本单位，该基本单位为系统字体字符的平均宽度和高度。对于使用系统字体的对话框，可以用这些值在对话框模板之间转换，比如在对话框模板和像素之间。对于不使用系统字体的对话框，从对话框模板单位到像素的转换要根据对话框使用的字体而定。对于对话框的其中一种类型用 MaPDialogRect 函数很容易地来执行转换,MaPDialogRect 考虑字体且正确的把一个长方形模板单位转换

为此像素。

函数原型：LONG GetDialogBaseUnits（VOID）;

参数：无。

返回值：返回值为一个 32 位的含有对话基本单位的值。返回值的低位字含有水平对话框基本单位，且高低位字含有垂直对话框基本单位。

备注：GetDialogBaseUnits 函数返回的水平基本单位同系统字体中字符以像素为单位的平均宽度相等；垂直基本单位同系统字体中字符的以像素为单位的平均高度相等；

对于一个没有使用系统字体的对话框基本单位相等于对话框字体字符以像素为单位的平均宽度和平均高度。可以用 GetTextMetrics 和 GetTextExtentPoint32 函数为一个选择的字体来计算这些值。但是,如果计算结果与那些通过系统执行的值不同，那么可以用 MapDialogRect 函数避免可能发生的错误。

每一个水平基本单位同四个水平对话框模板单位相等；每一个垂直基本单位同八个垂直对话框模板单位相等。所以用下列公式来把对话框模板单位转换为像素：

PiselX=（temptateunitX★baseunitX）/4;PiseIY=（templateunitY★baseunitY）/8

同样地. 用下列公式来把像素转换为对话框模板单位：

templateunitX=(pixelX★4)/baseunitX;templateunitY=(plxelY★8)/baseunitY

Windows CE：Windows CE 不支持此函数的任何一个扩展的错误值。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows：1.0 及以上版本；头文件：Winuser.h;库文件：user32.lib

2.7.13 GetDlgCtrllD

函数功能： 该函数返回指定控制的标识符。

函数原型： Int GetDlgCtrllD (HWND hwndCtl);

参数：

hwndCtl:标识控制的句柄。

返回值： 如果函数调用成功，则返回值为控制的标识符，如果函数调用失败,则返回值为零。例如,hwndCtl 参数的一个无效的值将导致函数失败。若想获得更多错误信息，请调用 GetLastError 函数。

备注： GetDlgCtrllD 函数接收子窗口句柄和对话框中的控制句柄。当应用程序调用 CreateWindow 或 CreateWindowEx 函数，通过把标识符的值设置为 hmenu 参数来创建窗口时，应用程序为了窗口句柄设置标识符。

如果 hwndCtrl 标识一个顶层窗口尽管 GetDlgCtrllD 可以返回一个值,但顶层窗口不能有标识符且这样的返回值从不生效。windows CE：标识符仅对子窗口有效。标识符叫以通过把作为 hwndCtrl 参数中的标识符传递到 CreatewindowEx 函数来设置，它也可以通过调用带有设置为 GWL_ID 的 nlndex 参数的 SetwindowLong 和 GetWindowLong 函数来设置和检索。

速查： Windows NT:3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h;库文件：user32.lib。

2.7.14 GetDigltem

函数功能： 该函数检索指定的对话框中的控制句柄。

函数原型： HWND GetDlgltem(HWND hDlg,int nlDDlgltem);

参数：

hDlg:标识含有控制的对话框。

nlDDlgltem:指定将被检索的控制标识符。

返回值： 如果函数调用成功则返回值为给定控制的窗口句柄。如果函数调用失败，则返回值为 NULL，表示为一个无效的对话框句柄或一个不存在的控制。若想获得更多错误信息，请调用 GetLastError 函数。

备注： 可以通过使用任何父子窗口对来使用 GetDlgltem 函数，而不仅只是对话框。只要 hDlg 参数指定一个父窗口，且子窗口有一个独立的标识符（象 CreateWindow 中 hMenu 参数指定的或创建子窗口的 CreateWindowEx 指定的那样），GetDlgltem 就会返回一个有效的句柄到子窗口。

Windows CE：GetDlgltem 函数只为对话框中的直接于控制工作，它不通过嵌套的对话框来搜寻。

速查： Windows NT：3.1 及以上版本；Windows:95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h;库文件：user32.lib。

2.7.15 GetDlgltemlnt

函数功能： 该函数把对话框中指定控制的文本转变为一个整型值。

函数原型： UINT GetDlgltemlnt(HWND hDlg,int nlDDlgltem,BOOL ★IpTranslated,BOOL bSigned);

参数：

hDlg: 指向含有利益控制的对话框的句柄。

nlDDlgltem: 指定文本将被转变的控制的对话项目标识符。

IpTranslated:指向一个 Boolean 变量的指针。该变量保存函数成功/失败的值。TRUE 表示成功，FALSE

表示失败。此参数为可选的。它可以为 NULL。在这种情况下，函数不返回关于成功和失败的信息。

bSigned:指定函数足否在开始时为一个最小的消息检测文本,且如果发现一个消息整型值,则返回它。TRUE 指定应该这样做，FALSE 指定不应该这样做。

返回值：如果函数调用成功则由 IPTranslated 指向的变量被设置为 TRUE，且返回值为控制文本的转变值。

如果函数调用失败则由 IPTranslated 指向的变量被设置为 FALSE，且返回值为零。注意因为零为一个可能转变的值，返回的零值不能通过它自身来表示失败。如果 IpTranslated 为 NuLL，则表示函数没有返回关于成功和失败的信息。如果 bSigned 参数为 TRUE，指定将被检取的值为一个符号整型值，则把返回值设置为一个整型类型。有关详细的错误信息，请调用 GetLastError。

备注：GetDlgltemlnt 函数通过发送 WM_GETTEXT 控制消息来检索给定控制的文本。函数通过去除任何一个文本开头的额外空间来转变要检索的文本，然后转换为数值数据。当函数达到文本的末尾或遇到一个非数值的字符时，则函数停止转变。

如果 bSigned 参数为 TRUE，则 GetDlgltemlnt 函数检测文本开头的符号“C 且把文本转变为符号整型数值。否则，函数创建一个非符号整型数值。

如果转变的值大于 IN_TMAX（对于有符号数）或 UINT_MAX（对于无符号数），则 GetDlgltemlnt 函数返回零值。

Windows CE：对于大于 48 个字符的文本字符串不能被转变。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本：头文件：Winuser.h;库文件：user32.lib

2.7.16 GetDlgltemText

函数功能：该函数获取对话框中与控制有关的文本或标题。

函数原型：UINT GetDlgltemText(HWND hDlg,int nlDDlltem,LPTSTR IpString int nMazCount);

参数：

hDlg：指向含有控制的对话框的句柄。

nlDDlgltem:指定标题或文本将被检索的控制的标识符。

IpString:指向获取标题或文本的缓冲器的指针。

nMaxCount：指定被复制到 lpString 参数指向的缓冲区的字符串的最大长度。如果字符串的字符最大长度超过范围，则该字符串被截断。

返回值：如果函数调用成功，则返回值表示被复制缓冲区的字符串的长度，不包括以 NULL 结尾的字符串。如果函数调用失败,则返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注: GetDlgltemText 函数把一个 WM_GETTEXT 消息发送到控制。

速查:Windows NT: 3.1 及以上版本；Windows：95 及以上版本；Windows CE:1.0 及以上版本：头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.17 GetNextDlgGroupltem

函数功能：该函数检索控制组的第一个控制的句柄，该控制组跟随对话框中指定的控制。

函数原型：HWND GetNextDlgGroupltem（HWND hDlg，HWND hctl，BOOL bPrevious）;

参数：

hDlg：标识正在被搜寻的对话框。

hCtl：指定用来作为搜寻开始点的控制。如果此参数为空，函数将以最后一个点为搜寻开始点。

BPrevious:指定参数如何搜寻,如果此参数为 TRUE,则函数寻找以前的控制组中的控制。如果为 FALSE,则函数寻找控制组中的下一个控制。

返回值： 如果 GetNextDlgGroupltem 函数调用成功，则返回值为控制组中以前的（或下一个）控制。如果函数调用失败，则返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：GetNextDlgGroupltem 函数按照对话框模板中被创建的顺序（或相反的顺序）寻找控制。控制组的第一个控制一定有 WS_GROUP 类型；所有其他的控制组的控制一定被顺序创建且一定没有 WS_GROUP 类型。

当寻找以前的控制时，函数返回第一个位置上可视的、且不失效的控制。如果由 hCtl 给定的控制有 WS_GROUP 类型，则函数暂时反向寻找具有 WS_GROUP 类型的第一个控制，然后重新回到原来的方向进行寻找，返回可视的、且不失效的第一个控制，如果没有发现控制，则返回 hWndCtrl。

当寻找下一个控制时，函数返回第一个位置上的可视控制，且没有 WS_GROUP 类型。如果遇到一个有 WS_GROUP 类型的控制，则函数反向寻找具有 WS_GROUP 类型的第一个控制，且如果此控制为可视的、且没有失效，则返回此控制。否则，函数重新回到原来方向的寻找，返回可视的、且不失效的第一个控制。如果没有发现控制。则返回 hCtl。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.7.18 GetNexTDlgTabltem

函数功能：该函数检索有 WS_GROUP 类型的第一个控制的句柄，该 WS_GROUP 类型控制跟随指定的控制。

函数原型： HWND GetNextDlgTabltem(HWND hDlg,HWND hCtl,BOOL bPrevious);

参数：

hDlg:标识将被搜寻的对话框。

hCtl: 指定用来作为搜寻开始点的控制。如果此参数为 NULL，则函数用对话框中上一个（或下一个）控制作为搜寻开始点。

bPrevious:指定函数怎样寻找对话框。如果此参数为 TRUE，则函数寻找上一个对话框中的控制。如果为 FALSE，则函数寻找下一个对话框中的控制。

返回值： 如果 GetNextDlgTabitem 函数调用成功，则返回值为有 WS_GROUP 类型的上一个（或下一个）控制的窗口句柄。如果函数调用失败，则返回值为 NuLL。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： GetNextDlgTabltem 函数按照对话框模板中被创建的顺序（或相反的顺序）寻找控制。函数返回第一个位置上可视的、且不失效的控制，该控制具有 WS_GROUP 类型。如果不存在此控制，则函数返回 hCtl。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件:user32.lib。

2.7.19 IsDialogMessage

函数功能： 该函数决定一个消息是否指定给指定的对话框，如果是，则处理消息。

函数原型： BOOL IsDialogMessage(HWND hDlg,LPMSG IpMsg);

参数:

hDlg: 标识对话框。

IpMsg: 指向一个含有将被检测的消息的 MSG 结构。

返回值：如果消息被处理，则返回值为非零值；如果消息没有被处理，则返回值为零。

备注：尽管 IsDialogMessage 函数是为无模态对话框而扩展的，但可以用含有控制的任何一个窗回来使用它。当 IsDialogMessage 处理一个消息时,它检测键盘信息并把它们转变成对响应对话框的选择命令。例如当按下 tab 时选择下一个控制或控制组，当按下 down 时选择控制组的下一个控制。

因为 IsDialogMessage 函数要执行消息所有必要的转变和传送，IsDialogMessage 函数处理的消息不必传送给 TranslateMessage 或 DispatchMessage 函数处理。

IsDialogMessage 函数把 WM_GETDLGETDLGCODE 发送到对话枢应用程序，决定应该处理哪个键。

lsDialogMessage 函数把 DMa_ETDEFID 和 DM_SETDEFID 消息发送到窗口。这些消息在 WINUSERH 头文件中定义为 WM_USER 和 WM_USER＋1 所以就有与应用程序定义的有一样值的消息发生冲突。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：usgr32.lib：Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.20 MapDialogRect

函数功能：该函数把指定的对话框单位映射成屏幕单位（像素）。函数 MapDialogRect 用变换坐标替换指定的 RECT 结构中的坐标，这就使得该结构可以用来创建对话框或定位对话框内的控制。

函数原型：BOOL MapDialogRect(HWND hDlg,LPRECT IpRect);

参数：

hDlg:标识对话框。MapDialogRect 函数只接收对话框创建函数中一个创建的对话框的句柄，对于其他窗口的句柄失效。

IpRect：指向一个含有将被转变的对话框坐标的 RECT 结构。

返回值：如果消息被处理，则返回值为非零值。如果消息没有被处理则返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：MapDialogRect 函数假定 RECT 结构内的起始坐标代表对话框单位。为把这些坐标从对话框单位转变为像素，函数检索对话框的当前水平和垂直基本单位，然后应用下列公式：

left=（left*baseunitX）/4;right=（right*baseunitX）/4

top=（top* baseunitY）/ 8; bottom=（bottom* baseunitY）/ 8

在很多情况下，对话框的基本单位同用 GetDialogBaseUnits 函数检索到的单位一样。如果对话板模板有 DS_SETFONT 类型，那么基本单位为模扳给定的字体中的字符的平均宽度和高度，单位为像素。

速查：Windows NT：3.1 及以上版本：Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h;库文件：USer32.lib。

2.7.21 MessageBox

函数功能：该函数创建、显示、和操作一个消息框。消息框含有应用程序定义的消息和标题，加上预定义图标与 Push（下按）按钮的任何组合。

函数原型：int MessageBox(HWND hWnd,LPCTSTR IpCaption,UINT UType);

参数：

hWnd:标识将被创建的消息框的拥有窗口。如果此参数为 NULL，则消息框没有拥有窗口。

IpText：指向一个以 NULL 结尾的、含有将被显示的消息的字符串的指针。

IpCaption：指向一个以 NULL 结尾的、用于对话框标题的字符串的指针。

uType：指定一个决定对话框的内容和行为的位标志集。此参数可以为下列标志组中标志的组合。

指定下列标志中的一个来显示消息框中的按钮，标志的含义如下。

MB_ABORTRETRYIGNORE：消息框含有三个按钮：Abort，Retry 和 Ignore。

MB_OK：消息框含有一个按钮：OK。这是缺省值。

MB_OKCANCEL：消息框含有两个按钮：OK 和 Cancel。

MB_RETRYCANCEL：消息框含有两个按钮：Retry 和 Cancel。

MB_YESNO：消息框含有两个按钮：Yes 和 No。

MB_YESNOCANCEL：消息框含有三个按钮：Yes，No 和 Cancel。

指定下列标志中的一个来显示消息框中的图标：标志的含义如下。

MB_ICONEXCLAMATION:

MB_ICONWARNING：一个惊叹号出现在消息框。

MB_ICONINFORMATION:

MB_ICONASTERISK：一个圆圈中小写字母 i 组成的图标出现在消息框。

MB_ICONOUESTION:一个问题标记图标出现在消息框。

MB_ICONSTOP:

MB_ICONERROR:

MM_ICONHAND：一个停止消息图标出现在消息框。

指定下列标志中的一个来显不缺省的按钮：标志的含义如下。

MB_DEFBUTTON1：第一个按钮为缺省按钮。如果 MB_DEFBUTTON2，MB_DEFBUTTON3，MB_DEFBUTTON4 没有被指定，则 MB_DEFBUTTON1 为缺省值。

MB_DEFSUTTON2；第二个按钮为缺省按钮。

MB_DEFBUTTON3：第三个按钮为缺省按钮。

MB_DEFBUTTON4：第四个按钮为缺省按钮。

指定下列标志中的一个来显示对话框的形态：标志的含义如卜。

MB_APPLMODAL：在 hwnd 参数标识的窗口中继续工作以前，用户一定响应消息框。但是，用户可以移动到其他线程的窗口且在这些窗口中工作。根据应用程序中窗口的层次机构，用户则以移动到线程内的其他窗口。所有母消息框的子窗口自动地失效，但是弹出窗口不是这样。如果既没有指定 MB_SYSTEMMODAL 也没有指定 MB_TASKMOOAL，则 MB_APPLMODAL 为缺省的。

MB_SYSTEMMODAL：除了消息框有 WB_EX_TOPMOST 类型，MB_APPLMODAL 和 WS_EX_TOPMOST 一样。用系统模态消息框来改变各种各样的用户，主要的损坏错误需要立即注意（例如，内存溢出）。如果不是那些与 hwnd 联系的窗口，此标志对用户对窗口的相互联系没有影响。

MB_TASKMODAL：如果参数 hwnd 为 NULL，除了所有属于当前线程高层次的窗口足失效的，MB_TASKMODALL 和 MB_ApPLMODAL 一样。当调用应用程序或库没有一个可以得到的窗口句柄时，使用此标志。但仍需要阻止到调用应用程序甲其他窗口的输入而不是搁置其他线程。

另外，可以指定下列标志。

MB_DEFAULT_DESKTOP_ONLy：接收输入的当前桌面一定是一个缺省桌面。否则，函数调用失败。缺省桌面是一个在用户已经纪录且以后应用程序在此上面运行的桌面。

MB_HELP：把一个 Help 按钮增加到消息框。选择 Help 按钮或按 F1 产生一个 Help 事件。

MB_RIGHT：文本为右调整。

MB_RTLREADING：用在 Hebrew 和 Arabic 系统中从右到左的顺序显示消息和大写文本。

MB_SETFOREGROUND：消息框变为前景窗口。在内部系统为消息个调用 SetForegrundWindow 函数。

MB_TOPMOSI：消息框用 WS_EX_TOPMOST 窗口类型来创建 MB_SERVICE_NOTIFICATION。

Windows NT：调用程序是一个通知事件的用户的服务程序。函数在当前活动桌面上显示一个消息框，即使没有用户登记到计算机。

如果设置了此参数，则 hwnd 参数一定为 NULL。所以消息框可以出现在一个桌面上而不是桌面响应参数 hwnd。

对于 Windows NT 4.0，MB_SERVICE_NOTIFICATION 的值已经改变。对于旧的和新的值，请参见 WINUSER。

Windows NT 4.0 通过把旧值映射到 MessageBox 和 MessageBoxEx 执行中的新值，为先存在的服务程序提供逆兼容。此映射只为有了版本数目的可执行程序而做。

为了建立一个用 MB_SERVICE_NOTIFICATION 的服务器，且可以在 Windows NT 3.X 和 Window NT 4.0 上执行，可有两种选择。在连接时间，指定一个版本数目小于 4.0 的版本，或在连接时间，指定一个 4.0 版本。在运行时间，用函数 GetVersionEx 来检测系统版本，然后在 Windows NT 3.X 上用 MB_SERVICE_NOTIFICATION_NT 3.x 来运行和在 Windows NT 4.0 上用 MB_SERVICE_NOTIFICAION 来运行。MB_SERVCE_NOTIFICATION_NT3.x（WindowNT）此值响应于为 WindowNT3.51 的 MB_SERVICE_NOTIFICAION 定义的值。

返回值：如果没有足够的内存来创建消息框，则返回值为零。如果函数调用成功，则返回值为下列对话框返回的菜单项目值中的一个：

IDABORT：Abort 按钮被选中。IDCANCEL：Cancel 按钮被选中。IDIGNORE：Ignore 按钮被选中。

IDNO：NO 按钮被选中。IDOK：OK 按钮被选中。IDRETRY：RETRY 按钮被选中。

IDYES：YES 按钮被选中。

如果一个消息框有一个 Cancel 按钮，且如果 Esc 键被按下或 Cancel 键被选择，则函数返回 IDCANCEL 值。如果消息框没有 Cancel 按钮，则按 Esc 键没有作用。

备注：当创建一个系统模态消息框来表示系统在内存的低端中时，由 lpTeXt 和 lpCaption 参数指向的字符串不应该从一个资源文件中取出，因为试图装载此资源可能导致失败。

当一个应用程序调用 MessageBox，且为 uType 参数指定 MB_ICONHAND 和 MB_SYSTEMMODAL 标志时，系统不管可用内存为多少，直接显示结果消息框。当这些标志被指定，系统把消息框文本的长度局限于三行。系统不能自动截断要填到消息框的行，但是消息字符串一定含有回车换行，以在合适的位置换行。

如果在对话框出现的的同时创建了消息框，则可使用对话框的句柄作为 hwnd 参数，hwnd 参数不应该标识一个子窗口，例如对话框中的一个控制。

Windows 95：系统可以支持最大达 16364 个窗口句柄。

Windows CE:Windows CE 不支持 uType 参数的下列值：

MB_STSTEMMODAL；MB_TASKMODAL；MB_HELP；MB_RTLREADING;MB_DEFAULT_DESKTOP_ONLY；

MB_SERVICE_NOTIFICATION；MB_USERICON。

不支持下列**返回值：**IDCLOSE；IDHELP。

速查：Windows:3.1 及以上版本:Windows:95 及以上版本;Windows:1.0 及以上版本;头文件:Winuser.h;库文件：USer32.lib；URicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.22 MessageBoxEx

函数功能：该函数创建、显示、和操作一个消息框。消息框含有一个应用程序定义的消息和标题，加上预定义目标与 pus（下推）按钮的任何组合。wLanguageld 参数指定哪一个语言资源集被用于预定义的下推按钮。有关 MessageBoxEX 函数其他参数的完整描述，请参见 MessageBoxEx 函数。

函数原型：int MessageBoxEXHWND hwnd，LPCTSTR IPText, LPCTSTR IpCaPtion, UINT UTye, WORD wLanguageld)；

参数：

hWnd：标识将被创建的消息框的拥有窗口。如果此参数为 NULL，则消息框没有拥有窗口。

IpCaption：指向一个以 NULL 结尾的、含有将被显示的消息的字符串的指针。

IpszTitle：向一个以 NULL 结尾的结、用于对话框标题的字符串的指针。如果此参数为 NULL，则用缺省的标题 Error。

uType:指定一个决定对话框的内容和行为的位标志集。标志的意义参见 MessageBox\uType。

备注:当创建一个系统模态消息框来表示系统在在内存的低端中时，由 IPText 和 IpCaPtion 参数指向的字符串不应该从一个资源文件中取出，因为试图装载此资源可能导致失败。

当一个应用程序调用 MessageBox，且为 uType 参数指定 MB_ICONHAND 和 MB_SYSTEMMODAL 标志时，系统不管可用内存为多少，直接显示结果消息框。当这些标志被指定，系统把消息框文本的长度局限于三行。系统不能自动截断要填到消息框的行，但是消息字符串一定含有回车换行，以在合适的位置换行。

如果在对话框出现的的同时创建了消息框，则可使用对话框的句柄作为 hwnd 参数，hwnd 参数不应该标识一个子窗口，例如对话框中的一个控制。

Windows 95：系统可以支持最大达 16364 个窗口句柄。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h;库文件： user32.lib; Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.23 SendDlgltemMessage

函数功能：该函数把一个消息发送给指定的对话框中的控制。

函数原型：LONG SendDlgltemMessage(HWND hDlg,int nlDDlgltem,UINT Msg,WPARAM wParam,LPARAM IParam);

参数：

hDlg:指定含有控制的对话框。

nlDDigltem:指定接收消息的控制的标识符。

Msg: 指定将被发送的消息。

wParam:指定消息特定的其他信息。

IParam:指定消息特定的其他信息。

返回值：返回值指定消息处理的结果，且依赖于发送的消息。

备注：SendDlgltemMessage 函数直到消息已经被处理时才返回。

使用 SendDlgltemMessage 函数同从一个指定的控制中检索句柄和调用 SendMessagge 函数一样。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.24 SetDlgltemlnt

函数功能：该函数把对话框中控制文本设置为用一个指定整型值的字符串表示。

函数原型：BOOL SetDlgltemlnt(HWND hDlg,int nLDDlgltem,UINT uValue,BOOL bSigned）；

参数：

hDlg:指定含有控制的对话框。

nlDDlgltem:指定将被改变的控制。

uValue:指定用于产生项目文本的整型值。

bsigned:指定 uValue 参数为有符号的还是为无符号的。如果此参数为 TRUE，则 uValuee 为有符号的。如果此参数为 TRUE 且 uValue 小于零，则最小符号被放在第一个数据之前。如果此参数为 FALSE，则 uValue 为无符号的。

返回值：如果函数调用成功则返回值为非零值。如果函数调用失败，则返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：为设置新文本，此函数把一个 WM_SETTEXT 消息发送到指定的控制。

Windows CE：bSigned 参数不被使用。所有被传送到 nValue 参数的值都假定为有符号的。

速查： Windows NT：3.1 及以上版本;Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件:winuser.h;库文件：user32.lib。

2.7.25 SetDlgltemText

函数功能： 该函数设置对话框中控制的文本和标题。

函数原型： BOOL SetDlgltemText(HWND hDlg,int nlDDlgltem,LPCTSTR IpString);

参数：

hDlg: 指定含有控制的对话框。

nlDDlgltem:标识带有将被设置的标题和文本的控制。

IpString:指向一个以 NULL 结尾的字符串指针，该字符串指针包含了将被复制到控制的文本。

返回值： 如果函数调用成功，则返回值为非零值。如果函数调用失败，则返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： SetDlgltem 函数把一个 WM_SETTEXT 消息发送到指定的控制。

速查： Windows NT：3.1 及以上版本：Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.7.26 MessageBoxlndirect

函数功能： 该函数创建、显示和操作一个消息框。消息框含有应用程序定义的消息文本和标题，任何位图和预定义的 push（下推）按钮的任意组合。

函数原型： int MessageBoxlndirect（LPMSGBoXPARAMs IpMsgBoxParams）;

参数：

IpMsgBoxParams: 指向一个含有用于显示消息框信息的 MSGBOXPARAMS 结构。

返回值： 如果没有足够内存来创建消息框，则返回值为零。如果函数调用成功，则返回值为下列对话框返回的菜单项目值中的一个：IDABORT:Abort 按钮被选中；IDCANCEL：Cancel 按钮被选中；IDGNORE:lgnorem 按钮被选中；IDNO：NO 按钮被选中；IDOK：OK 按钮被选中；IDRETRY：RETRY 按钮被选中；IDYES：YES 按钮被选中。

如果一个消息框有一个 Cancel 按钮,且当 Esc 键被按或 Cancel 键被选择时,则函数返回 IDCANCEL 值。如果消息框没有 Cancel 按钮，则按 Esc 键没有作用。

备注:当创建一个系统模态消息框来表示系统在在内存的低端时，由 IpText 和 lpCaption 参数指向的字符串不应该从一个资源文件中取出，因为试图装载此资源可能导致失败。

当一个应用程序调用 MessageBox，且为 uType 参数指定 MB_ICONHAND 和 MB_SYSTEMMODAL 标志时，系统不管可用内存为多少，直接显示结果消息框。当这些标志被指定，系统被把消息框文本的长度局限于三行。系统不能自动截断要填到消息框的行，但是消息字符串一定含有回车换行，以在合适的位置换行。

如果在对话框出现的的同时创建了消息框，则可使用对话框的句柄作为 hwnd 参数，hwnd 参数不应该标识一个子窗口，例如对话框中的一个控制。

Windows 95：系统可以支持最大达 16364 个窗口句柄。

速查： Wbindows NT：4.0 及以上版本;Windows：95 及以下版本;Windows CE：不支持;头文件：winuser.h;库文件：User32.lib：Unicoda：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.8 编辑控制函数（Edit Control）

2.8.1EditWordBreakproc

函数功能： 该函数是由应用程序定义的回调函数，该函数与 EM_SETWORDBREAKPROC 信号一起使用，一个多行编辑控制每当必须中断文本行时都调用 EditwordBreakProc 函数。EditwordBreakProc 函数定义了一个指向此回调函数的指针，EditwordBreakProc 是一个应用程序定义的函数名的占位符。

函数原型： int CALLBACK EditWordBreakkPrOC（LPTSTR lpCh，int ichCurrent，int CCh int code)；

参数；

Ipch:指向编辑控制文本的指针。

ichCurrent:指定一个文本缓冲区中字符位置的索引，该文本表示函数应该从这点开始检查字的中断。

cch:指定编辑控制文本中字符的数目。

code：指定回调函数要采取的措施，此参数可以是下列值之一。

WB_CLASSIFY：检索指定位置的字符的字中断标志和字符类，此值是为与超文本编辑控制一起使用。

WB_ISDECIMITER：检查在指定位置的字符是否是分隔符。

WB_LEFT：在指定位置的左边，找到字的开头。

WB_LEFTBREAK：在指定位置的左边，找到字的结束分隔符，此值是为与超文本编辑控制一起使用。

WB_MOVEWORDLEFT：在指定位置的左边，找到字的开头，此值用于 CTRL＋RIGHT 使用时，此值是为与超文本编辑控制一起使用。

WB_MOVEWDRDRIGHT：在指定位置的右边，找到字的开头，此值用于 CTRL＋ROGHT 使用时，此值是为与超文本编辑控制一起使用。

WB_RIGHT：在指定位置的右边，找到字的开头。（对于右对齐编辑控制很有用）

WB_RIGHTBREAK：在指定位置的右边找到字结束分隔符，（这对右对齐编辑控制很有用）此值是为与超文本编辑控制一起使用。

返回值：如果代码参数指定 WB_ISDELIMITER,且如果指定位置的字符为分隔符,则返回值为非零(TRUE)，否则返回值为零。如果代码参数指定 WB_CLASSIFY，返回值为指定位置的字符类和字符字中断标志，否则，返回值为指向文本缓冲区的开头的索引。

备注： 一个回车操作跟着一个换行符，一定被回调函数看作为一个单一的字，紧跟着一个换行符的两个回车操作也一定被看作为单一字。

一个应用程序必须通过 EM_SETWORDBREAKPROC 消息中指定回调函数的地址来安装回调函数。

对于超文本编辑控制，也可用 EM_SETWORDBREAKPROCEX 消息来取代带有 EditWordBreadProcEx 回调函数的缺省扩展字中断程序，该函数还提供了关于文本的其他信息，如字符集。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：用户自定义。Unicode：定义为 Unicode 和 ANSI 两种原型。

2.9 图标函数（Icon）

2.9.1Copylcon

函数功能： 该函数从另外的模块向当前模块复制指定的图标。

函数原型： HICON Copylcon（HICON hlcon);

参数：

hlcom:被复制图标的句柄。

返回值： 如果函数成功，返回值是图标副本的句柄；如果函数失效，返回值是 NULL。想获得更多的错误信息，请调用 GetLastError 函数。

备注： COPylcoon 函数使应用程序或动态链接库（DLL）能获得另一个模块自己拥有图标的句柄，如果这个模块被释放，应用程序图标将仍然能使用这个图标。

速查： Windows NT：3.1 及以上版本；Windows:95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.9.2Createlcon

函数功能： 该函数按指定的大小、彩色、位创建图标。

函数原型： HCON Createlcon（HINSTANCE hlnstance,int nWidth,int nHeight,BYTE cPlanes,BYTE cBitsPixe,CONSTBYTE*IpbANDbits,CONST BYET*IpbXORbits);

参数：

hlnstance:创建图标模块事例的句柄。

nWidth:指定目标宽度（像素）。

nHeight:指定图标高度（像素）。

cPlnes:指定图标位掩码异或的位面数。

cBitsPixel:指定图标位掩码异或中每像素的位数。

IpbAnDbits:某字节数组的指针，这个数组包含一个图标位与的位值。这个位掩码描述一个单色位图。

IpbXORbits:某字节数组的指针，这个数组包含一个目标位异或的位值。这个位掩码描述一个单色位图或设备相关颜色位图。

返回值： 如果函数成功返回值是图标的句柄。如果函数失效，返回值是 NULL。想获得更多的错误信息，请调用 GetLastError 函数。

备注： nWidth 和 nHeight 参数受当前显示驱动程序指定的宽度和高度限制，因为系统不能创建其他大小的图标。使用 GetSystemMetric 函数确定显示驱动程序支持的宽度和高度。

Createlcon 应用下列位与和位异或的真值表：

OR	XOR	显示
0	0	黑色
0	1	白色
1	0	直接显示屏
1	1	反转显示屏

速查： windows NT:3.1 及以上版本；Windows:95N 以上版本：Windows CE:不支持；头文件：winuser.h；库文件：user32.lib。

2.9.3CreatelconFromResource

函数功能： 该函数通过描述图标的资源位创建图标或光标。

函数原型：HICON CreatelconFuomResource（PBYTE presbits,DWORD dwResSize,BOOL flcon,DWORD dwVer);

DresDits:包含图标或光标资源位缓冲区的指针典型应用,可通过调用LOOKuplconldFromDirectory 或LoadResource 函数载入这些位（在Windows95 也可调用LookuplconldFromDirectorvEx)。

DwResSize:以字节为单位指定由DresbitS 参数指定的位集合的大小。

dwVer:指定由presbits 参数指定的资源位的图标或光标格式的的版本号。参数可能是下列的值：

格式	dwVer
Winsows2.x	0×00020000
Winsows3.x	0×00030000

所有基于 Microsoft Win32 的应用程序使用Windows 3.x格式的图标和光标。

返回值：如果函数成功返回值是图标或光标的句柄；如果函数失效，返回值是NULL。想获得更多的错误信息，请调用GetLastError 函数。

备注：CreatelconFromResource，CreateteonFromResourceEx，Createlconlndlrect，Getlconlnfo，LookuPlcondFromDirectory,LookuplconldFromDirectoryEx函数允许外壳应用程序和图标浏览器在整个系统中检查和使用资源。

速查：Windows NT：3.1 及以上版本;Windows：95 及以上版本;Windows CE:不支持;头文件：winuser.h;库文件：user32.lib。

2.9.4CreatelconFromResourceEx

函数功能： 该函数通过描述图标的资源位创建图标或光标。

函数原型：HICON CreatelconFromResourceEx(PBYTE pblconBits,DWORD cblconBits,BOOL flcon,DWORd dwVersion,intcxDesired，int cyDesired，UINT uFlags);

参数：

pblconBits：包含图标或光标资源位缓冲区的指针。典型的应用，可通过调用LookuPlconldFromDirectoryEx 或LoadResource 函数载入这些位。

cblconBits:以字节为单位指定由pblconBits 参数指定的位集合的大小。

flcon: 指定创建图标还是光标，如果参数为TRUE，创建图标；如果参数为FALSE，创建光标。

dwVersion:指定由pblcpmBots 参数指定的资源位的图标或光标格式的版本号。参数可能是下列的值：

Windows2.x 0x00020000； Windows3.x 0x00030000

所有基于Win32 的应用程序使用Windows3.x格式的图标和光标。

cxDesired：指定图标或光标的期望宽度（以像素为单位)。如果参数是零，函数使用 SM_CXICON 或SM_CXCURSOR 系统公制值设置宽度。

cyDesired:指定图标或光标的期望高度（以像素为单位)。如果参数是零，函数使用 SM_CYICON 或SM_CYCURSOR 系统公制值设置高度。

VFlags:指定的组合值：

LR_DEFAULTCOLOR：使用缺省颜色格式；

LR_MONOCHROME：创建单色图标或光标

返回值：如果函数成功，返回值是图标或光标的句柄：如果函数失效，返回值是NULL。想获得更多的错误信息，请调用GetLastError 函数。

备 注 ： CreatelconFromResourceEx ， CreatelconFromResource ， Createlconlndirect ， Getlconlnfo,LookuplcontoFromDirectoryEx 函数允许外壳应用程序和图标浏览器在整个系统中检查和使用资源。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：Winuser.h；库文件：user32.lib。

2.9.5Create;cpm;mdorect ZIWe

函数功能： 该函数从 ICONINFO 结构创建目标或光标。

函数原型： HICON Createlconlndirect（PICONINFO）piconirfo）；

参数：

piconinfo:函数用以创建图标或光标的 ICONINFO 结构指针。

返回值： 如果函数成功，返回值是所创建图标或光标的句柄。如果函数失效，返回值是 NULL。想获得更多的错误信息；请调用 GetLastError 函数。

备注：系统在创建图标或光标之前复制 ICONINFO 结构中的位图。因为系统可能在设备上下文中临时选用该位图，ICONINFO 结构的 hbmMask 和 hbmColor 成员不要被选入设备上下文。应用程序必须继续管理原始位图，不再需要时，删除它们。当你完成使用图标，用 Destroylcon 函数清除它。

Windows CE：图标成员不支持光标，当用该成员时，总是将 ICONINFO 结构的 flcon 域设为真值。

当使用目标光标成员时，你可使用 Createlconlndirect 函数创建图标或单色光标。WindowsCE 不支援彩色光标。

速查：Wingows NT：3.1 及以上版本;Windows：95 及以上版本；Windows CE:1.0 及以上版本；头文件：winuser.h;库文件：user32.lib。

2.9.6Destroylcon

函数功能： 该函数清除图标和释放任何被图标占用的存储空间。

函数原型： BOOL Destroylcon（HICON hlcon）；

参数：

hlcon：是要清除留标的句柄。该图标应处于未被使用状态。

返回值： 如果函数成功，返回值是非零：如果函数失效，返回值是零。想获得更多的错误信息，请调用 GetLastError 函数。

备注:只有利用 Createlconlndirect 函数创建的图标和光标才能调用 Destroylcon 函数，不要使用该函数清除一个共享图标。只要调入它的模块存在于存储器中，共享图标就一直有效。下列函数可获取共享图标：

Loadlcon;Loadlmage（如果你使用 LR_共享标记）；copylmage（如果你使用 LR_COPYRETURNORG 而且 hlmage 参数为共享目标）。

Windows CE：Destroylcon 函数可以通过图标句柄调用，这些图标句柄来自于 Createlconlndirect,ExtractlconEx,Loadlmage 或 Loadlco 函数。在调用 Destroylcon 函数之后这些图标句柄变为无效。

速查： Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:1.0 及以上版本；头文件：winuser.h;库文件：user32.lib。

2.9.7Drawlcon

函数功能： 该函数在限定的设备上下文窗口的客户区域绘制图标。

函数原型： BOOL Drawlcon（HDC hDC，int X，nit Y HICON hlcon ）；

参数：

hDC:窗口设备上下文的句柄。

X：指定图标左上角的逻辑 X 坐标。

Y：指定图标左上角的逻辑 y 坐标。

hlcon:被绘制图标的句柄。图标资源必须已经通过 Loadlcon 或 Loadlmage 函数被装载过。

返回值:如果函数成功，返回值是非零；如果函数失效，返回值是零。想获得更多的错误信息，请调用 GetLastError 函数。

备注： Drawlcon 函数将目标的左上角置于由 X 和 Y 参数指定的位置，该位置受当前设备上下文的映射方式支配。

Windows CE：Drawlcon 函数被当做宏执行，定义为 DrawlconEx（hdc，x，y， hicon，0，0，0，NULL，DI_NORMAL）。

速查： Windows NT:3.1 及以上版本;Windows：95 及以上版本;Windows CE:1.0 及以上版本；头文件：winuser.h;库文件：user32.lib。

2.9.8DrawlconEx

函数功能： 该函数在限定的设备上下文窗口的客户区域绘制图标，执行限定的光栅操作，并按特定要求伸长或压缩图标或光标。

函数原型： BOOL DrawlconEX（HDC hdc，int xLeft，int yTOp，HICON hlcon，int cxWidth，int cyWidth UINT isteplfAniCur,HBRUSH hbrFlickerFreeDraw,UINT diFlags）;

参数：

hdc：窗口设备上下文的句柄。

xLeft：指定目标或光标左上角的逻辑 x 坐标。

yTop：指定图标或光标左上角的逻辑 y 坐标。

hlcon:被绘制图标的句柄，该参数可标识一个激活的光标。图标或光标资源必须已经通过 Loadlmage 函数被装载过。这参数能识别激活的光标。

cxWidth:指定图标或光标的逻辑宽度。如果其值为零且 diFlags 参数是 DI_DEFAULTSIZE；函数使用 SM_CXICON 或_CXCURSOR 系统公制值设置宽度；如果其值为零且不使用 DI_DEFAULTSIZE，函数使用资源实际宽度。

cyWidth:指定图标或光标的逻辑高度。如果其值为零且 diFlags 参数是 DI_DEFAULTSIZE，函数使用 SM_CYICON 或 SM_CYCURSOR 系统公制值设置高度；如果其值为零且不使用 DI_DEFAULTSIZE，函数使用资源实际高 isteplfAniCur:如果 hlcon 标识一个动态光标，参数指定要绘制的帧索引；如果 hlcon 不标识一个动态光标，该参数被忽略。

hbrFlickerFreeDraw:系统用做闪烁·自由绘图的刷子句柄。如果 hbrFlickerFreeDraw 是有效的刷子句柄,系统利用背景颜色刷于创建一个反屏位图将图标或光标绘制到位图中，并将位图复制到由 hdc。标识的设备上下文中。

diFlags：指定绘图的标记，参数可为下列的值：

DI_COMPAT：系统采用缺省图像而不是用户定义的图像绘制图标和光标。

DI_DEFAULTSIZE：如果 CXWidth 和 CyWidth 参数被设为零，采用系统指定的图标和光标的的公制宽度

和高　度绘制图标和光标;如果标记未被指定且 cxWidth 和 cyWith 参数设为零标,函数使用资源实际大小。

DI_IMAGE：用图像绘制目标或光标。DI_MASK：用屏蔽绘图图标或光标。

DL_NORMAL：DI_IMAGE 与 DI_MASKR 的组合。

返回值：如果函数成功，返回值是非零；如果函数失效返回值是零。想获得更多的错误信息请调用 GetLastError 函数。

备注：DrawlconEx。函数将图标的左上角置于由 xLeft 和 yTop 参数指定的位置，该位置受当前设备上下文的映射方式支配。

Windows CE：下列的参数设置必须被使用：

djFlags 必须是 DI_NORMAL，DI_IMAGE，或 DI_MASK（不支持 DI_COMPAT and DI_DEFAULTSIZE)。

cxWidth 和 cyHeight 必须是零或图标的原始尺寸。hbrFlickerFreeDraw 必须是 NULL。

iSteplfAniCur 必须是零，动态图标不被支援。Windows CE 不支援：伸长的和压缩：也就是说该图标分辨率对给定的 HICON 是固定的，不支持国标大小的再调整。

映射方式：用 DrawlconEx 绘制光标时，hlcon 参数不能标识动态光标。

速查：Windows NT：31 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.9.9ExtractAssociatedlcon

函数功能；该函数返回存在于文件中的索引图标或存在于相关联可执行文件中的图标句柄。

函数原型：HICON ExtractAssociatedlcon（HINSTANCE hlnst，LPTSTR lplconPath)；

参数：

hlnst：指定调用函数的应用程序的事例。

lplconPath：包含图标文件的全称路径和文件名的指针，函数从文件或者与该文件相关联的可执行文件中抽取图标句柄。如果目标句柄是从可执行文件获得，函数将可执行文件的全称路径和文件名存储到 lplconPath 指定的字符串中。

Ipilcon：指定要被获取的目标句柄索引的字指针；如果图标句柄从可执行文件获得，函数将图标的标识符存储于 Ipilcon 指定的字中。

返回值：如果函数成功，返回值是有效的图标句柄。如果图标来自相关的可执行文件，函数将其全称路径和文件名存储到 IplconPath 指定的字符串中，并将图标识符存储于 Ipilcon 指定的字中。如果函数失效返回值是 NULL。

备注： EXtr8CtAssociatedlcon 函数首先按 IplconPath 指定的文件查找索引图标，如果函数不能从该文件获得图标句柄，且该文件有关联的可执行文件，它便从可执行文件中搜索图标。可执行文件的关联基于文件的扩展名，存储于每用户的注册表中，并可被资源管理器中的关联命令所定义。

速查：Windows NT:3.5 及以上版本；Windows:95 及以上版本;Windows CE:不支持:头文件:shellaoi.h;库文件：shell32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.9.10 Extractlcon

函数功能：该函数从限定的可执行文件，动态链接库（DLL)；或者图标文件中恢复图标句柄. 为恢复大或小的图标句柄数组，使用 ExtractlconEx 函数。

函数原型：HICON EXtractlcon（HINSTANCE hlnst，LPCTSTR lpszExeFileName，UINT nlconlndex)；

参数：

hlnst：调用函数的应用程序的事例句柄。

IpszExeFileName:代表可执行文件，DLL，或者图标文件的文件名的空结束字符串指针。

nlconlndex:指定要恢复图标基于零的变址。例如，如果值是 0,函数返回限定的文件中第一个图标的句柄，如值是 0 函数返回限定文件中图标的总数；如果文件是可执行文件或 DLL 返回值为 RT_GROUP_ICON 资源的数目:如果文件是一个.ICO 文件，返回值是 1；在 Windows95，WindowsNT4.0 和更高版本中，如果值为不等于向-1 的负数，函数返回限定文件图标句柄，该文件的资源标识符等于 nlconlndex 绝对值。例如，使用-3 来获取资源标识符为 3 的图标。为获取资源标识符为 1 的图标，可采用 ExtractlconEx 函数。

返回值：返回值是图标句柄。如果限定的文件不是可执行文件，DLL，或者图标文件返回是 1；如果发现在文件中没有图标，返回值是 NULL。

备注：必须调用 Destroyclon 函数来清除由 Extractlcon 函数返回的图标句柄。

速查：Windows NT：3.1 及以上版本；Windows:95 及以上版本；Windows CE:不支持;头文件：shellapi.h;库文件：shell32.lib;Unicode:在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.9.11 ExtractlconEx

函数功能：该函数从限定的可执行文件；动态链接库（DLL），或者图标文件中生成图标句柄数组。

函数原型：UINT EXtractlcohEX（LPCTSTR IpSZFile，int nlconlndex,HICON FAR*PhiCORLarge，HICON FAR’*phiconSmall,UINT nlcons)；

参数：

IpszFile:定义可获取图标的可执行文件，DLL,或者图标文件的名字的空结束字符串指针。

nlconlndex:指定抽取第一个图标基于零的变址；例如，如果该值是零；函数在限定的文件中抽取第一图标；如该值是 C1 且 phlconLarge 和 phiconSmall 参数均为 NULL，函数返回限定文件中图标的总数；如果文件是可执行文件或 DLL;返回值是 RT_GROUP_ICON 资源的数目;如果文件是一个 ICO 文件,返回值是 1;在 Windows95，WindowsNT4.0,和更高版本中，如果值为负数且 phlconLarge 和 phiconSmall 均不为 NULL，函数从获取图标开始，该图标的资源标识符等于 nlconlndex 绝对值。例如，使用-3 来获取资源标识符为 3 的图标。

phiconLarge:指向图标句柄数组的指针，它可接收从文件获取的大图标的句柄。如果该参数是 NULL 没有从文件抽取大图标。

phiconSmall：指向图标句柄数组的指针，它可接收从文件获取的小图标的句柄。如果该参数是 NULL，没有从文件抽取小图标。

nlcons:指定要从文件中抽取图标的数目。

返回值：如果 nlconlndex 参数是-1，PhiconLarge 和 PhiconSmall 参数是 NULL，返回值是包含在指定文件中的图标数目；否则，返回值是成功地从文件中获取图标的数目。

备注：必须调用 Destroylcon 函数来清除由 ExtractlconEx 函数返回的图标。为恢复大小图标尺寸,可使用 SM_CXICON,SM_CYICON,SM_CXSMICON,SM_CYSMICON 标记来调用 GetSystemMetrics 函数。

Windows CE：nlconlndex 参数必须是零或 CN（N 是指定的资源标识符)；nlcons 参数必须是 1。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：shellspi.h；库文件：shell32.lib；Unicodg：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.9.12 Getlconlnfo

函数功能：该函数恢复限定的图标或光标的信息。

函数原型：BOOL Getlconlnfo（HICON hlcon，PICONINFO piconinfo)；

参数：

nlcon:图标或光标的句柄；为恢复标准图标或光标信息。

Piconinfo:指向 ICONINFO 结构的指针。函数填充结构的成员。

返回值：如果函数成功，返回值是非零且函数填充限定的 ICONINFO 结构的成员。如果函数失效；返回值是零。想获得更多的错误信息，请调用 GetLastError 函数。

备注：Getlconlnfo 为 ICONINFO 的成员 hbmMask 和 hbmColor 创建位图，调用应用程序必须管理这些位图和并在不再需要时删除它们。

速查:Windows NT:3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32.lib。

2.9.13 LookuplconldFromDirectory

函数功能：该函数通过图标或光标数据搜索最适合当前显示设备的图标或光标。

函数原型：int LookuplconldFromiDirectory（PBYTE presbits,BOOL flcon)；

参数：

presbits: 指向图标或光标图录数据的指针。因为该函数不验证资源数据，它导致通用的保护（GP）错误或返回一个未定义的值（如果 presbitS 未指向有效的资源数据)。

flcon:指定是寻求图标或是光标。如果该参数是真，函数搜索图标；如果参数是假，函数搜索光标。

返回值：如果函数成功，返回值最适合当前显示设备的图标或光标的整数型资源标识符。如果函数失效，返回值是零。想获得更多的错误信息，请调用 GetLastError 函数。

备注:在一些设备相关和设备独立的格式中，RT_GROUP_ICON 类型的资源文件（RT_GROUP_CURSOR 指示光标）包含图标（或光标）数据。LookuPIconldFromDirectory 函数搜索最适合当前显示设备的图标（或光标）的资源文件并返回它的整型标识符。FindResource 和 FindResourceEx 函数利用该标识符使用 MAKEINTRESOURCE 宏在模块中查找资源。

图标所在目录是从一个具有 RT_PROUP_ICON 资源类型的资源文件装入（或者光标的

RT_GROUP_CURSOR 类型)，并且一特定图标的整型资源名称被载入。LookuPIconldFromDirectory 函数返回一个代表图标资源名的整型标识符，该图标与当前显示器件最匹配。

Loadlcon，LOadCursor，LOadlmsge 函数使用这个函数搜索最适合当前显示设备的图标或光标的特定的资源数据。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.9.14 LookuplconldFrom

函数功能：该函数通过图标或光标数据搜索最适合当前显示设备的图标或光标。

函数原型：int LookuPlconldFromDirectoryEx（PBYTE presbits，BOOL flcon，int cxDesired，int cyDesired，UINT Flags)；

参数：

presbits:指向图标或光标目录数据的指针。因为该函数不验证资源数据，它导致通用的保护（GP）错误或返回一个未定义的值（如果 presbits 未指向有效的资源数据)。

flcon: 指定是寻求图标或是光标。如果该参数是真；函数搜索图标；如果参数是假，函数搜索光标。

cxDesired:指定图标的期望宽度（以像素为单位）。如果该参数是零，函数使用 SM_CXICON 或 SM_CXCURSOR 系统公制值。

cyDesired:指定图标的期望宽度（以像素为单位）。如果该参数是零，函数使用 SM_CYICON 或

SM_CYCURSOR 系统公制值。

Flags:指定下列值的组合：

LR_DEFAULTCOLOR：使用缺省颜色格式；LR_MONOCHROME：创建单色图标或光标。

返回值：如果函数成功，返回值最适合当前显示设备的图标或光标的整数型资源标识符。如果函数失效,返回值是零。想获得更多的错误信息，请调用 GetLastError 函数。

备注：在一些设备相关和设备独立的格式中，RT_GROUP_ICON 类型的资源文件（RT_GROUP_CURSOR 指示光标）包含目标（或光标）数据。LookuPlconldFromDirectoryEx 函数搜索最适合当前显示设备的图标（或光标）的资源文件并返回它的整型标识符。FindResourceEx 函数利用该标识符使用 MAKENTRESOURCE 宏在模块中查找资源。

图标所在目录是从一个具有 RT_GROUP_ICCON 资源类型的资源文件装入（或者光标的

RT_GROUP_CURSOR 类型），并且一特定图标的整型资源名称被载入。LookuPlconldFromDirectoryEx 函数返回一个代表图标资源名的整型标识符，该图标与当前显示器件最匹配。

Loadcon,LOadlmage，LOadcUrSor 函数使用这个函数搜索最适合当前显示设备的图标或光标的特定的资源数据。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.9.15 Loadlcon

函数功能：该函数从与应用事例关联的可执行文件（EXE）中装载限定的图标资源。

函数原型：HICON Loadlcon（HINSTANCE hlnstance,LPCTSTR lplconName）；

参数：

hlnstance:模块事例句柄，该模块的可执行文件中包含被装载的图标。当标准的图标是被装载时，该参数必须是 NULL。

IplconName:包含被装载图标资源名称的空结束的字符串指针。作为选择，该参数可在字的低位包含资源标识符而字的高位为 0，使用 MAKEINTRESOURCE 宏产生该值。

为了使用某一的预先确定的图标，将 hlnstance 参数设为 NULL 且 IplconName 参数为下列值之一：

IDI_APPLICATION：缺省应用程序图标；IDI_ASTERISK：与 IDI_INFORMATION 相同。

IDI——ERROR：手·形状图标：IDI_EXCLAMATION:与 IDI_WARNING 相同。

IDI_HAND:与 IDI_ERROR 相同；IDI_INFORMATION:星号图标。

IDI_QUESTION:问号图标;IDI_WARNING:感叹号图标。

IDI_WINLOGO:Windows Logo 语言图标。

返回值：如果函数成功，返回值是新装载图标的句柄。如果函数失效，返回值是 NULL。想获得更多的错误信息，请调用 GetLastError 函数。

备注：仅当图标资源还没有被装载时才能用 Loadlcon 装载它，否则，它得到已存在资源的句柄。该函数搜索最适合当前显示器的图标的图标资源。图标资源可能是彩色或单色位图。

Loadlcon 只能装载大小符合 SM_CXICON 和 SM_CYICON 系统公制值的目标。使用 Loadmage 函数装载其他大小的图标。Windows CE：预先确定的图标（IDI*）不被支持。在 Windows CE 1.0 中，图标必须是每像素二位（.ic2）的图标或单色图标。

速查：Windows NT：3.1 及以上版本；Windows；95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.10 键盘加速器函数（Keyboard Accelerator）

2.10.1 CopyAcceleratorTable

函数功能：拷贝加速键表。拷贝加速键表函数拷贝指定的加速键表。此函数用于获得与一加速键表句柄相对应的加速键表数据，或用于确定加速键表数据的大小。

函数原型：int CopyAcceleratorTable(HACCEL hAccelSrc,LPACCEL IpAcceIDst,int cAccelEntries)；

参数：

hAccelSrc:欲拷贝的加速键表的句柄。

IpAccelDst:指向 ACCEL 结构数组的指针，该结构数组中存在着将要拷贝的加速键表信息。

cAccelEntries:指定由 IpAcceIDst 参数指向的欲拷贝到缓冲区的 ACCEL 结构的个数。

返回值：如果 IpAccelDst 为空，则返回值给出初始加速键表入口的个数。否则，给出己拷贝的加速键表的入口个数。

速查：WindOWS NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：Winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.10.2 CreateAcceleratorTable

函数功能：创建加速键表。该函数创建一个加速键表。

函数原型：HACCEL CreateAcceleratorTable（LPACCEL lpaccl，int cEntries)；

参数：

Ipaccl:指向描述加速键表的 ACCEL 结构数组的指针。

cEntires:指定数组中 ACCEL 结构的个数。

返回值：如果函数调用成功，则返回值为所创建的加速键表的句柄；否则，返回值为空。若想获得更多的错误信息，请调用 GetLastError。

备 注：关闭应用程序之前，必须调用 DestroyAcceeleratorTable 函数撤消所有由 CreatedAccelerstorTable 函数创建的加速键表。

速查：Windows NT：3.1u 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本：头文件：winuser.h;库文件：USer32.lib; Unicode:在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.10.3 DestroyAcceleratorTable

函数功能：撤消加速键表。该函数撤消一个加速键表。在关闭应用程序之前，必须使用该函数撤消所有由 DestroyAcceleratorTable 函数创建的加速键表。

函数原型：BOOL DestroyAcceleratorTable（HACCEL hAccel)；

参数：

hAccdel：将被撤消的加速键表的句柄。该句柄必须己通过调用 DestroyAcceleratorTable 函数而创建。

返回值：若函数调用成功，则返回非零值，若函数调用失败，则返回值为零。若要获得更多的错误信息，可以调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Windows.h;库文件：user32.lib。

2.10.4 LoadAccelerators

函数功能：调入加速键表。该函数调入指定的加速键表。

函数原型：HACCEL LoadAccelerators（HINSTANCE hlnstance，LPCTSTR lpTableName）;

参数：

hlnstance:模块的一个事例的句柄，该模块的可执行文件中包含将要调入的加速键表。

IpTableName:指向一个以空结尾的字符串的指针，该字符串包含了即将调入的加速键表的名字。另一种可选的方案是，该参数可以在加速键表资源的低位字中指定资源标识符，而高位字中全零。MADEINTRESOURCE 宏可被用于创建该值。

返回值：若函数调用成功，则返回非零值。若函数调用失败，则返回值为零。若要获得更多的错误信息，可以调用 GetLastError 函数。

备注：若加速键表尚未装入，该函数可从指定的可执行文件中将它装入。从资源中装入的加速键表，在程序结束时可自动释放。Windows CE：资源不被拷贝到 RAM 中，因而不能被修改。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h;库文件：user32.lib；Uncode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.10.5 TranslateAccelerator

函数功能：翻译加速键表。该函数处理菜单命令中的加速键。该函数将一个 WM－DEYDOUN 或 WM-SYSDEYDOWN 消息翻译或一个 WM-COMMAND 或 WM-SYSCOMMAND 消息（如果在给定的加速键表中有该键的入口），然后将 WM-COMMAND 或 WM－SYSCOMMAND 消息直接送到相应的窗口处理过程。

TranslateAccelerator 直到窗口过程处理完消息后才返回。

函数原型：int TranslateAccelerator（HWND hWnd,HACCEL hAccTable，LPMSG IpMsg）;

参数：

hWnd:窗口句柄，该窗口的消息将被翻译。

hAccTable:加速键表句柄。加速键表必须由 LoadAccelerators 函数调用装入或由 CreateAccd_eratorTable 函数调用创建。

LpMsg:MSG 结构指针，MSG 结构中包含了从使用 GetMessage 或 PeekMessage 函数调用线程消息队列中得到的消息内容。

返回值：若函数调用成功，则返回非零值；若函数调用失败，则返回值为零。若要获得更多的错误信息，可调用 GetLastError 函数。

备注：为了将该函数发送的消息与菜单或控制发送的消息区别开来，使 WM_COMMAND 或 WM_SYSCOMMAND 消息的 wParam 参数的高位字值为 1。用于从窗口菜单中选择菜单项的加速键组合被翻译成 WM－SYSCOMMAND 消息：所有其他的加速键组合被翻译成 WM-COMMAND。若 TransLateAccelerator 返回非零值且消息已被翻译，应用程序就不能调用 TranslateMessage 函数对消息再做处理。每个加速键不一定都对应于菜单命令。若加速键命令对应于菜单项，则 WM－INITMEMU 和 WM－INITMENUPOPUP 消息将被发送到应用程序，就好像用户正试图显示该菜单。然而，如下的任一条件成立时，这些消息将不被发送：

窗口被禁止，菜单项被禁止。

加速键组合无相应的窗口菜单项且窗口已被最小化。鼠标抓取有效。有关鼠标抓取消息，参看 SetCapture 函数。若指定的窗口为活动窗口且窗口无键盘焦点（当窗口最小化时一般是这种情况），

TranslatMssage 翻译 WM－SYSDEYUP 和 WM-SYSKEYDOWN 消息而不是 WM－DEYUP 和 WM－DEYDOWN 消息。

当按下相应于某菜单项的加速键，而包含该菜单的窗口又已被最小化时，TranslateMessage 不发送 WM－COMMAND 消息。但是，若按下与窗口菜单或某单项的任一项均不对应的加速键时，TranslateMessage 将发送一 WM-COMMAND 消息，即使窗口已被最小化。

Windows CE：所有的加速键消息被翻译成 WM－COMMAND 消息；Windows CE 不支持 WM－SYSCOMMAND 消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本：Windows CE：1.0 及以上版本；头文件：windows.h;库文件：user32.lib; Unicode：在 Windows NT 实现为 Unicode 和 ANSI 两种版本。

2.11 键盘输入函数（Keyboard Input）

2.11.1 ActivateKeyboadLayout

函数功能：激活键盘布局。该函数 Windows NT 和 Windows 95 中的实现有很大不同。本参考页中首先给出了完整的 Windows NT 的实现，下来又给出了 Windows 95 版本的实现，以便大家更好地了解二者的区别。

在 Windows NT 中 ActivateKeyboadLayout 函数激活一种不同的键盘布局，同时在整个系统中而不仅仅是调用该函数的进程中将该键盘布局设为活动的。

函数原型：HKL ActivateKeyboardLayout（ HKL hkl,UINT Flags);

参数：

hkl：将被激活的键盘布局的句柄。该布局必须先调用 LeadKeyboadLayout 函数装入，该参数必须是键盘分局的句柄，或是如下的值中的一种：

HKL_NEXT：在系统保持的，已装入的布局的循环链表中，选择下一布局。

HKL_PREV：在系统保持的，已装入的布局的循环链表中，选择前一布局。

Flags:定义键盘布局如何被激活。该参数可取如下的一些值：

LFREORDER：若该位被设置，则已装入的键盘布局的循环链路表将被重新排序。若该位没有设置，则循环链路表的顺序不变。例如，若用户激活了英语键盘布局，同时依序装入了法语、德语、西班牙语键盘布局，然后通过设置 KLF_REORDE 位激活德语键盘布局，则会产生如下顺序：德语、英语、法语、西牙语键盘布局。若激活德语键盘布局时未设置 KLF_REORDER 位，则产生如下的键盘布局的键盘布局序列：德语、西班牙语、英语、法语。若装入的键盘布局少于三种，则该标志域的值不起作用。

KLF_SETFORPROCESS：在 Windows NT 5.0 以上版本中使用。该参数用于整个进程中激活指定的键盘布，并向当前进程的所有线程发送 WM_INPUTLANGCHANGE 消息。

KLF_UNLOADPREVLOUS：卸载先前活动的键盘布局。

返回值：如果函数调用成功，返回值为前一键盘布局的句柄。否则，返回值为零。若要获得更多多错误信息，可调用 GetLastError 函数。

备注：在任一时刻可以装入多种键盘布局，但一次仅能激活一种布局。装入多种键盘布局使得可以快速地在多种布局之间切换。Windows 95 ACtivateKeyboadLayout 函数为当前线程设置输入语言。该函数接受一个键盘布局句柄，该句柄标识键盘的一个局部的和物理布局。

速查：Windows NT:3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h;库文件：user32.lib。

2.11.2 EnableWindow

函数功能： 该函数允许 / 禁止指定的窗口或控制接受鼠标和键盘的输入，当输入被禁止时，窗口不响应鼠标和按键的输入，输入允许时，窗口接受所有的输入。

函数原型： BOOL EnableWindow（HWND hWnd，BOOL bEnable）;

参数：

hWnd:被允许/禁止的窗口句柄。

bEnable:定义窗口是被允许，还是被禁止。若该参数为 TRUE，则窗口被允许。若该参数为 FALSE，则窗口被禁止。

返回值： 如果窗口原来是被禁止的，返回值不为零；如果窗口原来不是被禁止的，返回值为零。若想获得更多的错误信息，可调用 GetLastError 函数。

备注： 若窗口的允许状态将发生变化，WM_ENABLE 消息将在 Enblewindow 函数返回前发送出去，若窗口已已被禁止，它所有的子窗口也被禁止，仅管并未向子窗口发送 WM_ENABLE 消息。

窗口被激活前必须处于允许状态。比如，一个应用程序将显示一个无模式对话框并且已使该对话框的主窗口处于禁止状态，则在撤消该对话框之前须使其主窗口处于允许状态。否则，其他窗口将接受并被少活。若子窗口被禁止，则系统决定由哪个窗口接受鼠标消息时将忽略该窗口。

缺省情况下，窗口被创建时被置为允许。若创建一个初始化为禁止状态的窗口，应用程序需要在 GeateWindow 或 GeateWindowEX 函数中定义 WS_DOSABLED 样式。窗口创建后，应用程序可用 EnbleWindow 来允许禁止窗口。

应用程序可利用此函数允许 / 禁止对话框中的某个控制。被禁止的控制既不能接受键盘输入，也不能被用户访问。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h;库文件：user32.lib。

2.11.3 GetActiveWindows

函数功能： 该函数可以获得与调用线程的消息队列相关的活动窗口的窗口句柄。

函数原型： HWND GetActiveWindow（VOID）

参数：无。

返回值： 返回值是与调用线程的消息队列相关的活动窗口的句柄。否则，返回值为 NULL。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winser.h; 库文件：user32.lib。

2.11.4 GetAsyncKeyState

函数功能： 该函数用于确定函数被调用时，相应按键是处于按下状态，还是处于弹起状态；并且按下此键前否调用过 GetAsyncKeystate 函数。

函数原型： SHORT GetAyncKeystate（int vKey）;

参数：

vKey:定义虚拟键码（若有 256 个虚拟键码）。欲获得更多信息，参看 Virtual_Key Codes。在 Windows NT 中可以使用左右键区分常量来定义某些键。欲获得更多信息，参看备注部分。

返回值： 若函数调用成功，返回值给出了自最后一次调用 GetAsyncKeyStat。以来，指定的键是否处

于按下状态，并且确定了该键目前是按下或是被弹起。若最高位被置为 1，则键被接下：若最低位被置为 1，则该键在前次调用 GetAsyncKeystate 以来处于被接下的状态。若另一进程或线程中的窗口拥有键盘焦点，则

返回值为零。

Windows 95：Windows 95 不支持左右键区分常量，若用这些常量调用 GetAsyncKeystate 函数，则返回值为零。

备注：该函数支持鼠标按钮，但是，它检查的不是物理按钮映射到的逻辑按钮的状态，而实际物理按钮的状态。例如，函数调用 GetAsyncKeystate（VK_LBUTTON）总是返回物理的鼠标左按钮的状态，而不管该按钮映射为逻辑上的左按钮，可以调用 GetsystemMetrics（SM_SWAPBUTON）来确定系统当前物理鼠标按钮与逻辑鼠标按钮的对应关系，当鼠标按钮被左右交换后，函数返回 TRUE。

可以使用虚拟键码常数 VD_SHIFT，VK_CONTROL，VK_MENU 作为 vKey 参数的值，这样给出 Shift Ctrl，AIt 键的状态，而不区分是左键还是右键。

Windows NT：可以使用如下的虚拟键码常数作为 vKey 的值来区分左右键的情况：

VK_LSHIFT VK_RSHIFT;VW_LCONTROL VK_RCONTROL；VK_LMENU VK_RMENU

这些可区分的左右键常量仅当调用 GetKeyboardState，SetKeyboardstate，GetAsyncKeyState，GetKeyState 和 MaPViruaIKey 函数时才可用。

Windows CE：GetAsyncKeyState 函数支持左右虚键常量，所以定义按下左键还是右键。这些常数是 VKLSHIFT，VK_RSHIFT，VK_LCONTROL，VK_RCONTROL，VK_LMENU 和 VK_RMENU。

在 WindowsCE 中返回值的最低位是无效的，应当忽略。

GetAsyncKeystate 将返回当前键的状态，即使是另一进程或线程中的窗口拥有键盘焦点。

可以使用 VK_LBUTTON 虚拟键码常量来确定触模屏上笔尖的状态（按下/弹起）。

速查：Windows NT：3.1 及以上版本：Windows：95 及以上版本：Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.11.5 GetFocus

函数功能：该函数获取与调用线程消息队列相关的窗口的句柄，该窗口拥有输入焦点。

函数原型：HWND GetFocus（VOID）

参数：无。

返回值：为拥有键盘输入焦点的窗口句柄，若调用线程的消息队列没有相关的持有键盘输入焦点的窗口，则返回值为 NULL。

备往：尽管 GetFocus 返回 NULL，但可能另一线程的队列与拥有输入焦点的窗口相关。便用 GetForeyroundWindow 函数来获得用户目前工作的窗口。可以使用 AttachThreadlnPut 函数把线程的消息队列与另一线程的窗口关联起来。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.11.6 GetKeyboardLayout

函数功能：该函数可以获得指定线程的活动键盘布局。若 dwLayout 参数为零，将返回活动线程的键盘布局。

函数原型：HKL GetKeyboardLayout（DWORD dwLayout）；

参数：

dwLayout：标识欲查询的线程标识符，当前线程标识符为 0。

返回值：返回值为指定线程的键盘布局句柄。返回值的低位字包含了输入语言的语言标识符，高位字包含了键盘物理布局的句柄。

速查；Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE；不支持；头文件：winuser.h；库文件：user32.lib。

2.11.7 GetKeyboardLayoutList

函数功能：该函数可以获得与系统中输入点的当前集相对应的键盘布局句柄。该函数将句柄拷贝到指定的缓冲区中。

函数原型：UINT GetKeyboardLayoutList（int nBuff，HKL FAR★IpList）

参数：

nBuff：指定缓冲区中可以存放的最大句柄数目。

Iplist：缓冲区指针，缓冲区中存放着键盘布局句柄数组。

返回值：若函数调用成功，则返回值为拷贝到缓冲区的键盘布局句柄的数目，或者，若 nBuff 为 0，则运回值为接受所有当前键盘布局的缓冲区中的大小（以数组成员为单位）。若函数调用失败，返回值为 0。若想获得更多错误信息，可调用 GetLastError 函数。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.11.8 GetKeyboardLayoutName

函数功能：该函数可以获得活动键盘布局的名字。

函数原型：BOOL GetKeyboardLayoutName（LPTSTR pwszKLID）；

函数：

pwszKLID：缓冲区指针，缓冲区中用于接收至少有 KL_NAMELENGTH 个字符的键盘布局的名字（包含空结束符在内）。该参数值将是提供给 LoadKeyboardLayout 函数的字符串的一个副本，除非发生键盘布局替换。

函数值：若函数调用成功，则返回非 0 值。若函数调用失败，则返回值为 0。若要获得更多错误信息，可调用 GetLastError 函数。

备注：Windows NT：GetKeyboardLayoutName 接收系统的活动键盘布局的名字。

Windows 95：GetKeyboardLayoutName 接受调用线程的活动键盘布局的名字。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.11.9 GetKeyboardState

函数功能：该函数将 256 个虚拟键的状态拷贝到指定的缓冲区中。

函数原型：BOOL GetKeyboardState（PBYTE IpKeyState）；

参数：

IpKeyState：指向一个 256 字节的数组，数组用于接收每个虚拟键的状态。

返回值：若函数调用成功，则返回 0 值。若函数调用不成功，则返回值为 0。若要获得更多的错误信

息，可以调用 GetLastError 函数。

备注： 应用程序可以调用该函数来检取所有虚拟键的当前状态。当键盘消息被从该线程的消息队列中移去时，虚拟键的状态发生改变。当键盘消息被发送到该线程的消息队列中，或者，当键盘消息被发送到其他线程的消息队列或被从其他线程的消息队列中检取到时，虚拟键的状态不发生改变。（例外：通过 AttachThreadlnput 连接的线程共享同一键盘状态。）

当函数返回时，由 lpKeyState 参数指向的每一个数组成员中都包含了一个虚拟键的状态数据。若最高位被置为 1，则该键处于 down 状态；否则，该键处于 up 状态。若最低位被置为 1，则该键被触发。当一个键被打开时称之为被触发，如 capslock 键。若最低位被置为 0，该键被关闭且不被触发。一个触发键也键盘上的指示灯（如果有的话）在该键被触发时点亮，在不被触发时灭掉。

若要检取单个虚拟键的状态信息，可以调用 GetKeyState 函数。若要检取任一虚拟键的当前状态，而不管相应的键盘消息足否已从消息队列中检取到，可以使用 GetAsyncKevState 函数。

应用程序可以使用虚拟键码常数 VK_SHIFT，VK_CONTROL 和 VK_MENU 作为 IpKeyState 所指向的数组的下标。这样给出的 Shift，Ctrl，Alt 键的状态不区分左右键。应用程序也可以使用如下的虚拟键码常数作为以上键的区分左右键的下标：

VK_LSHIFT， VK_RSHIFT， VK_LCONTROL， VK_RCONTROL； VW_LMENU， VW_RMENU 仅当应用程序调用 GetKeyboardState；SetKeyboardState，GetAsycKeystate，GetKeyState 和 MapVirtualKey 函数时，才可用这些区分左右键的常数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件： USer32.lib。

2.11.10 GetKeyNameText

函数功能： 该函数检取表示键名的字符串。

函数原型： int GetKeyNameText（LONG IParam，LPTSTR IpString，int nSize）；

参数：

IParam：指定被处理的键盘消息（例如 WM_KEYDOWN）的第二个参数。该函数 Lporam 参数的如下部分：

16-23：扫描码；24：扩展标志，用于区别增强型键盘上的某些键：25：“无关”位，调用该函数的应用程序设置此位来表明函数不应区分诸如左右 ctrl 键和 shift 键。

LpString：指向接受键名的缓冲区的指针。

nSize：指定键名的最大字符长度，包括空结束符。（该参数值应与 LpString 参数指定的缓冲区的大小相等）。

返回值： 若函数调用成功，将拷贝一个以空结尾的字符串的指定缓冲区中，且返回值为串的长度（字符数），不计终止的空字符。

若函数调用失败，返回值为 o，若想获得更多的错误信息，可调用 GetLastError 函数。

备注： 键名字符串的格式取决于当前键盘布局，键盘驱动程序持有一张字符串形式的键名表（每个名字长度大于一个字符）并依据当前安装的键盘布局对键名进行翻译。每个字符键的名字是该字符本身，该键的名字被完整的拼写出来。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.11.11 GetKeyState

函数功能： 该函数检取指定虚拟键的状态。该状态指定此键是 UP 状态，DOWN 状态，还是被触发的（开

关每次按下此键时进行切换)。

函数原型： SHORT GetKeyState (int nVirtKey);

函数：

nVrtKey：定义一虚拟键。若要求的虚拟键是字母或数字（A～Z，a～z 或 0～9），nVirtKey 必须被置为相应字符的 ASCII 码值，对于其他的键，nVirtKey 必须是一虚拟键码。若使用非英语键盘布局，则取值在 ASCIIa～z 和 0～9 的虚拟键被用于定义绝大多数的字符键。例如,对于德语键盘格式,值为 ASCII0(0X4F）的虚拟键指的是“0”键，而 VK_OEM_1 指“带变音的 0 键”

返回值： 返回值给出了给定虚拟键的状态，状态如下：

若高序位为 1，则键处于 DOWN 状态，否则为 UP 状态。

若低序位为 1，则键被触发。例如 CAPS LOCK 键，被找开时将被触发。若低序位置为 0，则键被关闭，且不被触发。触发键在键盘上的指示灯，当键被触发时即亮，键不被触发时即灭。

备注： 当给定线程从它的消息队列中读键消息时，该函数返回的键状态发生改变。该状态并不反映与硬件相关的中断级的状态。使用 SetKeyboardState 可获取这一信息。

欲检取所有虚拟键状态信息，可以使用 SetKeyboardState 函数。

应用程序可以使用虚拟键码常数 VK_SHIFT，VK_CONTROL 和 VK_MENU 作为 nVirtKey 参数的值。它给出 shift，ctrl 或 alt 键的值而不区分左右键，应用程序也可以使用如下的虚拟键码常数作 nVirtKey 的值来区分前述键的左键、右键的情形。

VK_LSHIFT，VK_RSHIFT；VK_LCONTROL，VK_RCONTROL；VK_LMENU，VK_RMENU

仅当应用程序调用 GetKeyboardSlate，SetKeyboardState，GetAsyncKeystate；GetKeyState 和 MapVirtualKey 函数时，才可用这些区分左右键的常数。

Windows CE：GetKeyState 函数仅能用于检查如下虚拟键的 DOWN 状态。

VK_LSHIFT，VKRSHIFT，VK_LCONTROL；VK_RCONTROL；VK_LMENU，VK_RMENU

GetKeyState 函数只能用于检查 VK_CAPITAL 虚拟键的触发状态。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.I 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.11.12 IsWindowEnabled

函数功能： 该函数用于判断指定的窗口是否允许接受键盘或鼠标输入。

函数原型： BOOL IsWindowEnabled (HWND hWnd);

参数：

hWnd：被测试的窗口句柄。

返回值： 若窗口允许接受键盘或鼠标输入，则返回非 0 值，若窗口不允许接受键盘或鼠标输入，则返回值为 0。

备注： 子窗口只有在被允许并且可见时才可接受输入。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.11.13 keybd_event

函数功能：该函数合成一次击键事件。系统可使用这种合成的击键事件来产生 WM_KEYUP 或 WM_KEYDOWN 消息，键盘驱动程序的中断处理程序调用 keybd_event 函数。在 Windows NT 中该函数已被使用 Sendlhput 来替代它。

函数原型：VOID keybd_event（BYTE bVk，BYTE bScan，DWORD dwFlags，DWORD dwExtralnfo）；

参数：

bVk：定义一个虚据拟键码。键码值必须在 1～254 之间。

bScan:定义该键的硬件扫描码。

dwFlags:定义函数操作的名个方面的一个标志位集。应用程序可使用如下一些预定义常数的组合设置标志位。

KEYEVENTF_EXETENDEDKEY:若指定该值,则扫描码前一个值为 OXEO(224)的前缀字节。DEYEVENTF_KEYUP：若指定该值，该键将被释放；若未指定该值，该键交被接下。dwExtralnfo：定义与击键相关的附加的 32 位值。

返回值：该函数无返回值。

备注：尽管 keybd_event 传递一个与 OEM 相关的硬件扫描码给系统，但应用程序不能用此扫描码。系统在内部将扫描码转换成虚拟键码，并且在传送给应用程序前清除键码的 UP/down 位。

应用程序可以摸拟 PRINTSCREEN 键的按下来获得一个屏幕快照，并把它存放到剪切板中。若要做到这一点，则要将 keybd_event 的 bVk 参数置为 VK_SNAPSHOT,bScan 参数置为 0（用以获得全屏快照）或 hScan 置为 1（仅获得活动窗口的快照）。

Windows CE：WindowsCE 支持 dwFlags 参数附加的标志位。即使用 KEYEVENTF_SILENT 标志模拟击键，而不产生敲击的声音。Windows CE 不支持 KEYEVENTF_EXTENDEDKEY 标志。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本 ；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.11.14 LoadKeyboardLayout

函数功能：该函数给系统中装入一种新的键盘布局，可以同时装入几种不同的键盘布局，任一时刻仅有一个进程是活动的，装入多个键盘布局使得在多种布局间快速切换。

函数原型：HKLLoadKeyboardLayout（LPCTSTR pwszKLID，UINT Flags）；

参数：

pwszKLID：缓冲区中的存放装入的键盘布局名称，名称是由语言标识符（低位字）和设备标识符（高位字）组成的十六进制值串，例如 U.S.英语对应的语言标识符为 DX0409，则基本的 U.S.英语键盘布局命名为“0000409”。U.S.英语键盘布局的变种（例如 Dvorak 布局）命名为“00010409”,“00020409”等。

Flags：指定如何装入键盘布局，该参数可以是如下的值。

KLF_ACTIVATE：若指定布局尚未装入，该函数为当前线程装入并激活它。

KLF_NOTELLSHELL:当装入新的键盘布局时,禁止一个 ShellProe 过程接收一个 HSHELL_LANGUAGE 代码。

当应用程序依次装入多个键盘布局时，对除最后一个键盘布局外的所有键盘布局使用该值，将会延迟 Shell 的处理直到所有的键盘布局均已被装入。

KLF_RECOROER：将指定键盘布局移动到布局表的头部，使得对于当前线程，该布局的活动的。若不提供 DLF_ACTIVATE 值，则该值记录键盘布局表。

KLF_REPLACE_LANG：Windows NT 4.0 或 Windows 95 以上支持，若新布局与当前布局有同样的语言标识符，那么新布局替代当前布局作为那种语言的键盘布局，若未提供该值，而键盘布局又有同样的标识符，则当前布局不被替换，函数返回 NULL 值。

KLF_SUBSTITUTE_OK：用用户喜欢的键盘布局来替换给定布局，系统初始时设置该标志，并且建议始终设置该标志，仅当在注册 HKEY_CURRENT_USER/Keyboard Layout/Substitate 下定义了一个替代布局时，才发生替换。例如，在名为 00000409 的部分中有一个多于 00010409 的值，则设置该标志装入 U.S.英语键盘布局会导致 Dvorak US.英语键盘布局的装入。系统引导时使用该参数，建议在所有应用程序装入键盘布局时使用该值，以确保用户喜欢的键盘布局被选取。

KLF_SETFORPROCESS：Windows NT 5.0 该位仅法与 KLF_ACTIVATE 一起使用时才有效，为整个进程激活指定键盘布局，且发送 WM_INPUTLANGCHANGE 消息以当前进程的所有线程。典型的 LoadKeyboardLayWut 仅为当前线程激活一个键盘布局。

KLF_UNLOADPREVIOS：WindowsNT5.0，Windows95，Windows98 都不支持，仅当与 KLF_ACTIVATE 一起使用时才有效，仅当装入且激活指定键盘布局成功，先前的布局才能被卸载，建议使用 unLoadKeyboardLayout 函数。

返回值：若函数调用成功，返回与要求的名字匹配的键盘布局句柄。若没有匹配的布局，则返回 NULL。

备注： 应用程序可以通过仅定义语言标识符的串来装入该语言的 IME 向缺省键盘布局。若应用程序想装入 IME 的指定键盘布局，就必须读注册信息以确定传递给 LoadKeyboardLayout 返回的键盘布局句柄来激活。

Windows 95 和 Windows 98：若装载与原先键盘布局使用同种语言的布局，且 KLF_REPLACELANG 标志未被设置，则函数调用失败，仅有一个键盘布局可与给定语言相关联。(对于装载与同一语言相关的多 IME 也是可接受的)。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.11.15 MapVirtualKey

函数功能： 该函数将一虚拟键码翻译（映射）成一扫描码或一字符值，或者将一扫描码翻译成一虚拟键码。

函数原型：UINT MaoVirtualKey（UINT uCode，UINT uMapType)；

参数：

uCode：定义一个键的扫描码或虚拟键码。该值如何解释依赖于 uMapType 参数的值。

uMapType：定义将要执行的翻译。该参数的值依赖于 uCode 参数的值。取值如下：

0：代表 uCodee 是一虚拟键码且被翻译为一扫描码。若一虚拟键码不区分左右，则返回左键的扫描码。若未进行翻译，则函数返回 0。

1：代表 uCode 是一扫描码且被翻译为一虚拟键码，且此虚拟键码不区分左右。若未进行翻译，则函数返回 0。

2：代表 uCode 为一虚拟键码且被翻译为一未被移位的字符值存放于返回值的低序字中。死键（发音符号）则通过设置返回值的最高位来表示。若未进行翻译，则函数返回 0。

3：代表 uCode 为一扫描码且被翻译为区分左右键的一虚拟键码。若未进行翻译，则函数返回 0。

返回值：返回值可以是一扫描码，或一虚拟键码，或一字符值，这完全依赖于不同的 uCode 和 uMapType 的值。若未进行翻译，则函数返回 0。

备注： 应用程序可以使用 MapVirtualKey 将扫描码翻译为虚拟键码常数 VK_SHIFT，VK_CONTROL 和 VK_MENU。反之亦然。这些翻译不区分左右 shift，ctrl，alt 键。应用程序可以通过调用 MapVirtualKey 函数时将 uCode 参数

设置为如下的虚拟键码常数来获得分别相应于上述键的左右键的扫描码：

VK_LSHIFT，VK_RDHIFT；VK_LCONTROL；VK_RCONTROL；VK_LMENU，VK_RMENU

仅当应用程序调用 GetKeyboardState，SetKeyboardStaste，GetAsyncKeystate， GetKeyState 和 MapVirtUalKey 函数

时，才可用这些区分左右键的常数。

Windows CE：Windows CE 仅支持 uMapType 参数取值为 2 的情况，即将虚拟键映射为未被移位的字符。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以 L 版本；头文件：winuser.h；库文件：user32.lib；在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.11.16 MapVlrtualKeyEx

函数功能： 该函数将虚拟键码翻译为扫描码或字符值，或者将扫描码翻译为虚拟键码。该函数使用由给她键盘布局句柄标识的物理键盘和输入语言来翻译这些代码。

函数原型： UINT MapVirtualKeyEx（UINT uCode，UINT uMapType，HKL dwhkl）；

参数：

uCode：定义一个键的虚拟键码或扫描码。该值如何解释依赖于 uMapType 参数的值。

uMapType：定义将要执行的翻译。该参数的值同 MapVirtuaIKe。

dwhkl：翻译给定代码所使用的键盘布局的句柄。该参数值可以是在此之前调用 LoadKeyboardLayout 函数返回的任何键盘布局句柄。

返回值： 返回值可以是一扫描码，或一虚拟键码，或一字符值，这完全依赖于不同的 uCode 和 uMapType 的值。若未进行翻译，则函数返回 0。

返回值： 返回值可以是一扫描码，或一虚拟键码，或一字符值，这完全依赖于不同的 uCode 和 uMapType 的值。若未进行翻译，则函数返回 0。

备注： 应用程序可以使用 MapVirtualKeyEx 将扫描码翻译为虚拟键码常数 VK_SHIT；VK_CONTROL 和 VK_MENU。反之亦然。这些翻译不区分左右 shift，ctrl，alt 键。应用程序可以通过调用 MapVirtualKey 函数时将 uCode 参数设置为如下的虚拟键码常数来获得分别相应于上述键的左右键的扫描码：

Vk_LSHIFT，VK_RDHIFT；VK_LCONTROL，VK_RCONTROL；VK_LMENU，VK_RMENU

仅当应用程序调用 GetKeyboardState，SetKeyboardState，GetAsyncKeystate，GetKeyState，MapVirtuaIKey 和 MapVirtualKeyEx 函数时，才可用这些区分左右键的常数。若要查看完整的虚拟键码表，请参考 Virtual.KeyCodes。

速查： Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.11.17 OemKeyScan

函数功能： 该函数将 0-0x0FF 的 OEM ASCII 代码映射为 OEM 扫描码及其转换状态。该函数通过模拟键盘输入来提供信息，使得一个程序可将 OEM 文本传送到另一程序。

函数原型： DWORD OemKeyScan（WORD wOemChar）；

参数：

wOemChar：定义 OEM 字符的 ASII 值。

返回值： 返回值的低序字中包含给定的 OEM 字符的扫描码，高序字中包含了转换状态，它可能是如下标志位的组合：

1：任一。shift 键被按下；2：任一 ctrl 键被接下；

4：任一 alt 键被按下；8：Hankaku 键被按下；

16：保留（由键盘布局驱动程序定义）；32：保留（由键盘布局驱动程序定义）

若一字符在当前键盘布局下不能通过单击某键产生，则返回值为 OXFFFFFFFF。

备注；该函数对于需要。ctrl+alt 键的字符或者死键不提供翻译。该函数不翻译的字符必须通过使用 alt+键区机制的模拟输入进行拷贝。Numlock 键必须关闭。

该函数对于使用当前键盘布局不能通过一次击键得到的字符不进行翻译，例如需要死键的带音调的字符。该函数不翻译的字符必须通过使用 alt+键区机制的模拟输入进行拷贝。Numlock 键必须关闭。该函数使用 VkKeyScan 函数来实现。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；

库文件：user32.lib。

2.11.18 RegisterHotKey

函数功能： 该函数定义一个系统范围的热键。

函数原型： BOOL RegisterHotKey（HWND hWnd，intid，UINT fsModifiers，UINT vk）；

参数：

hWnd：接收热键产生 WM_HOTKEY 消息的窗口句柄。若该参数 NULL，传递给调用线程的 WM_HOTKEY 消息必须在消息循环中中进行处理。

id：定义热键的标识符。调用线程中的其他热键不能使用同样的标识符。应用功能程序必须定义一个 OX0000-0xBFFF 范围的值。一个共享的动态链接库（DLL）必须定义一个 0xC000-0xFFFF 范围的值伯 GlobalAddAtom 函数返回该范围）。为了避免与其他动态链接库定义的热键冲突，一个 DLL 必须使用 GlobalAddAtom 函数获得热键的标识符。

fsModifoers：定义为了产生 WM_HOTKEY 消息而必须与由 nVirtKey 参数定义的键一起按下的键。该参数可以是如下值的组合：

MOD_ALT：按下的可以是任一 Alt 键。MOD_CONTROL：按下的可以是任一 Ctrl 键。

MOD_SHIFT：按下的可以是任一 Shift 键。

MOD_WIN：按下的可以是任一 Windows 按键。这些键可以用 Microsoft Windows 日志记录下来。

vk：定义热键的虚拟键码。

返回值： 若函数调用成功，返回一个非 0 值。若函数调用失败，则返回值为 0。若要获得更多的错误信息，可以调用 GetLastError 函数。

备注： 当某键被接下时，系统在所有的热键中寻找匹配者。一旦找到一个匹配的热键，系统将把 WM_HOTKEY 消息传递给登记了该热键的线程的消息队列。该消息被传送到队列头部，因此它将在下一轮消息循环中被移去。

该函数不能将热键同其他线程创建的窗口关联起来。

若为一热键定义的击键已被其他热键所定义，则 RegisterHotKey 函数调用失败。

若 hWnd 参数标识的窗口已用与 id 参数定义的相同的标识符登记了一个热键，则参数 fsModifiers 和 vk 的新值将替代这些参数先前定义的值。

Windows CE:Windows CE 2.0 以上版本对于参数 fsModifiers 支持一个附加的标志位。叫做 MOD_KEYUP。

若设置 MOD_KEYUP 位，则当发生键被按下或被弹起的事件时，窗口将发送 WM_HOTKEY 消息。

RegisterHotKey 可以被用来在线程之间登记热键。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.11.19 Sendlnput

函数功能： 该函数用于合成击键、鼠标移动、按钮按下等。

函数原型： UINT SendlnpUt（UINT nlnputs，LPINPUT plnputs，int cbSize）；

参数：

nlnprts：定义 plnputs 指向的结构的数目。

plnputs：指向 INPUT 结构数组的指针。每个结构代表插人到键盘或鼠标输入流中的一个事件。

cbSize：定义 INPUT 结构的大小。若 cbSize 不是 INPUT 结构的大小，则函数调用失败。

返回值： 函数返回被成功地插人键盘或鼠标输入流中的事件的数目。若要获得更多的错误信息，可以

调用 GetlastError 函数。

备注： Sendlnput 函数将 INPUT 结构中的事件顺序地插入键盘或鼠标的输入流中。这些事件与用户插入的（用鼠标或键盘）或调用 keybd_event，mouse_event，或另外的 Sendlnput 插人的键盘或鼠标的输入流不兼容。

Windows CE：Windows CE 不支持 plnputs 参数指向的 INPUT 结构的类型域中的 INPUT_HARDWARE 值。

速查：Windows NT：4.0 及以上版本：Windows：98 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.11.20 SetActiveWindow

函数功能： 该函数激活一个窗口。该窗口必须与调用线程的消息队列相关联。

函数原型： HWND SetActiveWindow（HWND hWnd）；

参数：

hWnd：将被激活的最顶层窗口。

返回值： 若函数调用成功，则返回原先活动窗口的句柄。若函数调用失败，则返回值为 NULL。若要获得更多错误信息，可以调用 GetLastError 函数。

备注： SetActiveWindow 函数激活一个窗口，但当应用程序处于后台时，将不激活指定窗口。当应用程序激活窗口时处于前台，则窗口将被放到前台。

若 hWnd 参数指定的窗口是被调用线程创建的，调用线程的活动窗口的状态将被设置到 hWnd 中。否则，调用线程的活动窗口的状态被设为 NULL。

从另一方面讲，SetForegroundWindow 窗口函数激活一个窗口并将其强制为前台的。当应用程序要显示关键错误或需要用户立即注意的信息时，应只能调用 SetForegroundWindow 函数。

使用 AttachThreadInput 函数可将一个线程的输人处理连接到其他线程。这使得调用 SetActiveWindow 函数的线程可以激活与其他线程的消息队列相关的窗口。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

2.11.21 SetFocus

函数功能： 该函数对指定的窗口设置键盘焦点。该窗口必须与调用线程的消息队列相关。

函数原型： HWND SetFocus（HWND hWnd）；

参数：

hWnd：接收键盘输入的窗口指针。若该参数为 NULL，则击键被忽略。

返回值： 若函数调用成功，则返回原先拥有键盘焦点的窗口句柄。若 hWnd 参数无效或窗口未与调用线程的消息队列相关，则返回值为 NULL。若要获得更多错误信息，可以调用 GetLastError 函数。

备注： SetFocus 函数发送 WM_KILLFOCUS 消息到失去键盘焦点的窗口，并且发送 WM_SETFOCUS 消息到接受键盘焦点的窗口。它也激活接受键盘焦点的窗口或接受键盘焦点的窗口的父窗口。

若一个窗口是活动的，但没有键盘焦点，则任何按键将会产生 WM_SYSCHAR,WM_SYSKEYDOWN 或 WM_SYSKEYUP 消息。若 VK_MENU 键也被按下，则消息的 IParam 参数将设置第 30 位。否则，所产生的消息将不设置此位。

使用 AttachThreadInput 函数，一个线程可将输入处理连接到其他线程。这使得线程可以调用 SetFocus 函数为一个与其他线程的消息队列相关的窗口设置键盘焦点。

Windows CE：不使用 SetFocus 函数为一个与其他线程的消息队列相关的窗口设置键盘焦点。但有一个

例外。若一个线程的窗口是另一线程的子窗口，或这些窗口是具有同一父窗口的兄弟窗口，则与一个线程关联的窗日可以为其他窗口设置焦点，尽管该窗口属于一个不同的线程。在这种情况下，就不必先调用AttachThreadlnpUt 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本：Windows CE：1.0 及以上版；头文件：winuser.h；库文件：user32.lib。

2.11.22 SetKeyboardState

函数功能：该函数拷贝一个存放键盘键状态的 256 字节的数组到调用线程的键盘输入状态表中。该表与GetkeyboardState 和GetkeyState 函数访问的是同一个表。对该表的改变不会影响其他线程的键盘输入。

函数原型：BOOL SetkeyboardState (LPBYTE lpKeyState)；

参数：

lpKeyState：指向一个包含键盘键状态的 256 字节的数组。

返回值：若函数调用成功，则返回值不为 0。若函数调用失败，则返回值为 0。若要获得更多的错误信息，可以调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.11.23 ToAscii

函数功能：该函数将指定的虚拟键码和键盘状态翻译为相应的字符或字符串。该函数使用由给定的键盘布局句柄标识的物理键盘布局和输入语言来翻译代码。

函数原型：int ToAscii(UINT uVirtKey,UINT uScanCode；PBYTE lpKeyState,LPWORD lpChar,UINT uFlags)；

参数：

nVirtkey：指定要翻译的虚拟键码。

uScanCode：定义被翻译键的硬件扫描码。若该键处于 up 状态，则该值的最高位被设置。

LpKeyState：指向包含当前键盘状态的一个 256 字节数组。数组的每个成员包含一个键的状态。若某字节的最高位被设置，则该键处于 down 状态。若最低位被设置，则表明该键被触发。在此函数中，仅有capslock 键的触发位是相关的。NumloCk 和 scroll loCk 键的触发状态将被忽略。

LpChar：指向接受翻译所得字符或字符串的缓冲区。

UFlags：定义一个菜单是否处于激活状态。若一菜单是活动的，则该参数为 1，否则为 0。

返回值：若定义的键为死键，则返回值为负值。否则，返回值应为如下的值：

0：对于当前键盘状态，所定义的虚拟键没有翻译。

1：一个字符被拷贝到缓冲区。

2：两个字符被拷贝到缓冲区。当一个存储在键盘布局中的死键（重音或双音字符）无法与所定义的虚拟键形成一个单字符时，通常会返回该值。

备注：若键盘布局中原先存放了一个死键，则提供给 ToAscii 函数的参数可能不足以翻译虚拟键码。

典型地，ToAscii 函数执行基于虚拟键码的翻译。然而，在某些情况下，uScanCode 参数的第 15 位可能被用来区分一个键的按下状态和释放状态。扫描码用于翻译 Alt+数字键的键组合。

尽管 NUMLOCK 事实影响键盘状态的触发键，TOAscii 将忽略 lpKeyState 的触发设置（VK_NUMLOCK），因为仅 uVirtKey 参数就足以区分光标移动键（VL_HOME，INSERT,等等）和数字键（VK_DECIMAL，VK_NUMPADO_VK_NUMPAD9）。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Wiodows CE：不支持；头文件：Winuser.h；库文件：user32.lib。

2.11.24 ToAsciiCx

函数功能： 该函数将指定的虚拟键码和键盘状态翻译为相应的字符或字符串。该函数使用由给定的键盘布局句柄标识的物理键盘布局和输入语言来翻译代码。

函数原型： int ToAsciiEx(UINT uVirtKey,UINT uScanCode，PBYTE lpKeyState，LPWORD lpChar,UINT uFlags, HKL dwhkl)；

参数：

nVirtkey：指定要翻译的虚拟键码。

uScanCode：定义被翻译键的硬件扫描码。若该键处于 up 状态，则该值的最高位被设置。

IpKeyState：指向包含当前键盘状态的一个 256 字节数组。数组的每个成员包含一个键的状态。若某字节的最高位被设置，则该键处于 down 状态。若最低位被设置，则表明该键被触发。在此函数中，仅有 Capslock 键的触发位是相关的。Numlock 和 Scroll lock 键的触发状态将被忽略。

lpchar：指向接受翻译所得字符或字符串的缓冲区。

uFlags：定义一个菜单是否处于激活状态。若一菜单是活动的，则该参数为 1，否则为 0。

dwnkl：翻译给定代码所使用的键盘布局的句柄。该参数可以是先前 LoadKeyboardLayout 函数返回的键盘布局句柄。

返回值：同上。

速查：Windows NT：4.0 以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winustr.n；库文件：user32.lib。

2.11.25 ToUnicode

函数功能： 该函数将给定的虚拟键码和键盘状态翻译成相应的字符或字符串。

函数原型： int ToUnicode(UINT wVirtKey,UINT wScanCode，PBYTE lpKeyState，LPWSTR pwszBuff，int cchBuff，UINTwFlags)；

参数：

wVirtKey：定义将被翻译的虚拟键码。

wScanCode：定义被翻译键的硬件扫描码。若该值的最高为被置为 1，则该键处于 Up 状态。

lpKeyState：指向一个包含当前键盘状态的 256 字节数组。数组中的每个成员（字节）包含了一个键的状态。若一字节的最高位被置为 1，则该键处于 down 状态。

PwszBuff：接受翻译所得 Unicode 字符或字符串的缓冲区指针。

cchBuff：定义 pwszBuff 参数指向的缓冲区中字符串的大小。

wFlags：形成函数执行条件的一个标志域集。若一个菜单处于激活状态，则将第 0 位设置为 1。第 1 位到第 31 位保留。

返回值： 该函数返回一个如下的值：

-1：指定的虚拟键码是死键（重音或双音字符）。即使已敲击了几个字符，且这几个字符已存储在键盘状态中时，也将忽略键盘布局，返回该值。如果可能的话，即使对于 Unicode 键盘布局，该函数也已给出了一个将死键字符写入 pwszBuff 参数定义的缓冲区的间隔形式。例如，函数写入字符 SPACING ACUTE（0x00B4），而不是写入字符 NON_SPACING ACUTE（0x0301）。

0：对于当前键盘状态，所定义的键没有翻译。没有写人任何东西到 pwszBuff 参数定义的缓冲区。

1：一个字符被写入 pwszBuff 参数定义的缓冲区

2：两个或两个以上字符被写入 pwszBuff 参数定义的缓冲区。发生这种情况最常见的原因是由于存放在键盘布局中的一个死键无法由指定的虚拟键码组合成单键字符。

备注：若键盘布局中原先存放了一个死键，则提供给 ToUnicode 函数的参数可能不足以翻译虚拟键码。典型地，ToUnicode 函数只执行基于虚拟键码的翻译。然而，在某些情况下，wScanCode 参数的第 15 位可能被用来区分一个键的按下状态和释放状态。

速查：Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：winuser.h；库文件： user32.lib。

2.11.26 ToUnicodeEx

函数功能：该函数将给定的虚拟键码和键盘状态翻译成相应的字符或字符串。

函数原型：int ToUnicodeEx(UINT wVirtKey，UINT wScanCode，PBYTE lpKeyState，LPWSTR pwszBuff，int cchBuff，UINTwFlags,HKL dwhkl)；

参数：

wVirtKey：定义将被翻译的虚拟键码。

WScanCode：定义被翻译键的硬件扫描码。若该值的最高为被置为 1，则该键处于 uP 状态。

LpKeyState：指向一个包含当前键盘状态的 256 字节数组。数组中的每个成员（字节）包含了一个键的状态。若一字节的最高位被置为 1，则该键处于 down 状态。

pwszBuff:接受翻译所得 Unicode 字符或字符串的缓冲区指针。

cchBuff：定义 pwszBuff 参数指向的缓冲区中字符串的大小。

wFlags：形成函数执行条件的一个标志域集。若一个菜单处于激活状态，则将第 0 位设置为 1。第 1 位到第 31 位保留。

dwhkl：用键盘层翻译指定的代码。

返回值；该函数返回值同 ToUnicode。

速查：Windows NT：4.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：Winuser.h；库文件：user32.lib。

2.11.27 UnloadKeyboardL

函数功能：该函数移去一个键盘布局。

函数原型：BOOL UnloadKeyboardLayout（ HKL hkl)；

参数：

hkl：将被卸载的键盘布局句柄。

返回值：若函数调用成功，返回值不为 0。若函数调用失败，返回值为 0。函数调用失败可有以下几种原因：传递了一个无效的键盘布局句柄。键盘布局已被预先调入。键盘布局正被使用。

若要获得更详尽的错误信息，可以调用 GetLastError 函数。

备注：Windows 95：UnloadKeyboardLayout 不能卸载系统缺省的键盘布局。这样可以确保对于用户敲入的 shell 命令和文件系统使用的名字总有一个合适的字符集可用。

Windows NT：UnloadKeyboardLayout 能够卸载系统缺省的键盘布局。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.11.28 UnreglsterHotKey

函数功能： 该函数释放调用线程先前登记的热键。

函数原型： BOOL UnregisterHotKey（HWND hWnd，int id)；

参数：

hWnd：与被释放的热键相关的窗口句柄。若热键不与窗口相关，则该参数为 NULL。

id：定义被释放的热键的标识符。

返回值： 若函数调用成功，返回值不为 0。若函数调用失败，返回值为 0。若要获得更多的错误信息，可以调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:Winuser.h;库文件：user32.lib。

2.11.29 VkKeyScan

函数功能： 该函数将一个字符翻译成相应的虚拟键码和对于当前键盘的转换状态。该函数已被 VkKeyScanEx 函数所替代。仍然可以使用 VkKeyscan 函数，但是不必再定义键盘布局。

函数原型： SHORT VkKeyScan(TCHARch)；

参数：

ch：定义被翻译成虚拟键码的字符。

返回值： 若函数调用成功，则返回值的低位字节中包含了虚拟键码，高位字节中包含了上挡状态，这些状态可以是如下标志位的组合：

1：按下的可以是任一 Shift 键。2：按下的可以是任一 Ctrl 键。

4：按下的可以是任一 AIt 键。8：按下的是 Hankaku 键。

16：保留（由键盘驱动程序定义)。32：保留（由键盘驱动程序定义)。

若函数不能将传递的字符代码翻译成一个按键，则低位与高位字节将均置为_1。

备注： 对于使用右手 Alt 键作为 Shift 键的键盘布局（例如法语键盘布局)，转换状态由值 6 来表示，因为右手 Alt 键在内部被翻译为 Ctrl+Alt。

数字键盘（VK_NUMPAD0 ——VK_NUMPAD9）的翻译被忽略掉了。该函数仅主键盘部分的字符翻译为相应的击键动作。例如，字符“7”被翻译成 VK_7，而不是 VK_NUMPAD7。

应用程序使用该函数通过发送 WM__KEYUP 和 WM_KEYDOWN 消息来传送字符。

速查：Windows NT:3.7 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.11.30 vkKeyScanEx

函数功能： 该函数将一个字符翻译成相应的虚拟键码和对于当前键盘的上挡状态。该函数使用由给定的键盘布局句柄标识的物理键盘布局和输入语言来翻译字符。

函数原型： SHORT VkKeyScanEx（TCHA ch,HKL dwhkf)；

参数：

ch:定义被翻译成虚拟键码的字符。

Dwhkl：用于翻译字符的键盘布局句柄。该参数值可以是任意先前由 LoadKeyboardLayout 函数返回的键盘布局句柄。

返回值： 同 VkKeyScan。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib； Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.11.31 GetKBCodePage

函数功能： 该函数已过时。可以使用 GetOEMCP 函数检取系统的 OEM 代码页标识符。GetKBCodePage 函数返回当前代码页。

函数原型： UINT GetKBCodePage（VOID）

参数:无。

返回值： 返回值 OEM 代码页标识符，或者若登记值为不可读的，返回值则为缺省值。若要查看 OEM 代码页标识符表，可以参考 GetOEMCP 函数。

2.12 列表框函数（ListboX）

2.12.1 DlgDirList

函数功能： 该函数用与指定的文件名匹配的所有文件的名字填充列表框。

函数原型： int DlgDirList(HWND hDlg,LPTSTR lpPathSpec,int nlDListBox,int nlDStaticPath,UINT uFileType);

参数：

hDlg：包含列表框的对话框句柄。

loPathSpec：指向包含路径名或文件名的以 NULL 结尾的字符串指针。DIgDirList 修改此串，该串必须有足够的长度来保存修改的内容。关于此参数的更详尽的信息，请看备注部分。

nlDListBox：定义一个列表框的标示。如果该参数为 0，DlgDirList 函数认为没有列表框存在，也不试图填充。

nlDStaticPath：定义用于显示当前驱动器和目录的静态控制的标识符。若此参数为 0，DlgDirList 认为不存在这样的控制。

UFileType：定义将要显示的文件名字的属性。该参数必须是一个或多个如下的值：

DDL_ARCHIyE：包含文档文件。DDL_DIRECTORY：包含于目录。于目录名包含在方括号中。

DDL_DRIVES：包含驱动器。驱动器以［一 X」的形式列出，其中 X 是驱动器符。

DDL_EXCLUSIVE:仅包含指定属性的文件。缺省情况下，可读写的文件将被列出，尽管并未指定 DDL_READWRITE 值。DDL_HIDDEN：包含隐含文件。

DDL_READONLy：包含只读文件。DDL_READWRITE:包含没有其他附加属性的可读写文件。

DDL_SYSTEM:包含系统文件。DDL_POSTMSGS：传递消息给应用程序的消息队列。缺省情况下，DlgDirList 直接发送消息给对话框过程。

返回值： 若函数调用成功，则返回值不为 0。若函数调用失败，则返回值为 0。例如，lpPathSpec 定义的串不是一个有效路径时，函数将失败。若想获的错误信息，可以调用 GetLastError 函数。

备注:若对于 lpPathSpec 参数定义了一个 0 长度的串，或者仅定义了一个目录名，而没有文件名，则串被转换为‘.’。

lpPathSgec 参数有如下形式：[drive：][\u]directory[\idirectory]\u][filename]

在这个例子中，drive 是一个驱动器符，directory 是一个有效的驱动器名，filename 是一个有效的

文件名，文件名中必须包含至少一个通配符。

若 lpPathSpec 包含一个驱动器或目录名，或同时包含两者，则在列表框被填充以前，当前的驱动器和目录将被改变为指定的驱动器和目录。nlDStaticPath 参数标识的静态控制也被用新的驱动器或/和目录名来更新。

列表框填充以后,DlgDirList 通过移去路径和文件名的驱动器和/或目录部分来更新 lpPathSpec 参数。

DlgDirList 发送 LB_RESETCONTENT 和 LB_DIR 消息给列表框。

速查：Windows NT: 3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.12.2 DlgDirSelectEx

函数功能：该函数从单选列表框中检取当前选择。假定列表框已被 DlgDirList 函数填充，并且该选择为一个驱动器符，文件名，或目录名。

函数原型:BOOL DlgDirSelectEx(HWND hDlg,LPTSTR lpString,int nCount,int nlDlistBox)；

参数：

hDlg：包含列表框的对话框句柄。

lpString：接受选定的路径的缓冲区指针。

nCount：定义 lpstring 指向的缓冲区中的字符串长度。

NlDListBox：定义对话框中列表框的整数标识符。

返回值：若当前选择为目录名，则返回值不为 0。若当前选择不是目录名，则返回值为 0。若要获得更多的错误信息，可调用 GetLastError 函数。

备注:DlgDirSelectEx 函数把选择拷贝到 lpString 指向的缓冲区。若当前选择为一目录名或驱动器符，DlgDirSelectEx 函数将去掉封闭的方形括号（对于驱动器符；还要去掉连字符），以便于将名字和符号插入到新的路径中。若没有选择，lpString 的值不变。

DlgDirSelectEx 发送 LB_GETCURSEL 和 LB_GETTEXT 消息到列表框。该函数不允许从列表框返回多个文件名。该列表框一定不能是多选列表框。若为多选框，该函数不返回 0 值，且 lpString 的值不变。

速查：Windows nT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件;winuser.h;库文件：user32.lib; Unicode:在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.13 菜单函数（Menu）

2.13.1 CheckMenuRadlol

函数功能：该函数校校一个指定的菜单项并使其成为一个圆按钮项。同时不校核相关组里的其他菜单项并清除这些项的国按钮的类型标志。

函数原型：BOOL CheckMenuRadioltem(HMEN hMENU，UINT idFirst，UINT idLast，UINT uFlags)；

参数：

hMenu：包含一组菜单项的菜单的句柄。

idFirst：菜单组里第一个菜单项的标识符或位置。

idLast：菜单组里最后一个菜单项的标识符或位置。

IdCheck：要选取的菜单项的标识符或位置。

uFlag：指定 idFirst，idLast，idCheck 含义的值。如果此参数为 MF_BYCOMMAND,则其他参数指定菜

单项标识符。如果此参数为 MF_BYPOSITION，则其他参数指定菜单项位置。

返回值： 如果函数调用成功，返回值非零。如果函数调用失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： CheckMenuRadioltem 设置了 MFT_RADIOCHECK 类型标志，并为由 idCheck 指定的项设置 MFS_CHECKED 状态，同时，清除组里所有其他项目的上述两个标志。被选取的项用项目目标表示，而不是用复选标记目标。要得到更多的关于菜单项类型和状态标志的信息，参看 MENUITEMINFO 结构。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件；winuser.h;输入库：user32.lib。

2.13.2 CreateMenu

函数功能： 该函数创建一个菜单。此菜单最初是空的，但可用函数 InserMenultem，AppendMenu，和 lnsertMenu 来填入菜单项。

函数原型： HMENU CreateMenu（VOID）

参数：无。

返回值： 如果函数调用成功，返回值是新创建菜单的句柄。如果函数调用失败，返回值是 NULL。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 与被分配给一个窗日的菜单相联系的资源会被自动释放。如果此菜单未被分配给一个窗口，应用程序必须在关闭之前释放与菜单相连的资源。应用程序通过调用函数 DestroyMenu 来释放菜单资源。

Windows 95 环境下，系统可支持最多 16，364 个菜单句柄。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.13.3 CreatePopupMenu

函数功能： 该函数创建一个下拉式菜单、子菜单或快捷菜单。此案单最初是空的，但可用函数 InserMenultem 来插入或追加菜单项。也可用函数 InsertMenu 来插人菜单项，用 AppendMenu 来追加菜单项。

函数原型： HMENU CreatePopupMenu（VOID）

参数：无。

返回值： 如果函数调用成功，返回值是新创建菜单的句柄。如果函数调用失败，返回值是 NULL。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 一个应用程序可增加新菜单到已存在的菜单上，或者可以调用函数 TrackPopupMenuEx 或 TrackPopupMenu 来显示快捷菜单。

与被分配给一个窗口的菜单相联系的资源会被自动释放。如果此菜单未被分配给一个窗口，应用程序必须在关闭之前释放与菜单相连的资源。应用程序通过调用函数 DestroyMenu 来释放菜单资源。Windows95 环境下，系统可支持最多 16,364 个菜单句柄。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows cE：1.0 及以上版本：头文件：winuser.h；输入库：user32.lib。

2.13.4 DeleteMenu

函数功能： 该函数从指定菜单里删除一个菜单项。如果此菜单项打开了一个菜单或子菜单，则此函数销毁该菜单或子菜单的句柄，并释放该菜单或子菜单使用的存储器。

函数原型： BOOL DelefeMenu（HMENU hMenu，UINT uPosition，UINT uFlags）；

参数：

hMenu：要被的修改菜单的句柄。

UPosition：指定将被删除的菜单项，按参数 uFlagS 确定的含义。

UFlags：确定参数 UPosition 加如何被解释。此参数可取下列值之一：

MF_BYCOMMAND：表示 uPosition 给出菜单项的标识符。如果 MF_BYCOMMAND 和 MF_BYPOSITION 都没被指定，则 MF_BYCOMMAND 为缺省的标志。

MF_BYPOSITION：表示 uPosition 给出菜单项基于零的相对位置。

返回值： 如果函数调用成功，返回值非零；如果函数调用失败，返回值是零。若想获得更多的错误信息， 请调用 GetLastError 函数。

备注： 只要一个菜单被修改，无论它是否被显示在窗口里，应用程序都应调用 DrawMenubar。

速查： Wihdows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：Winuser.h；输入库：user32.lib。

2.13.5 DestroyMenu

函数动能：该函数销毁指定的菜单，并释放此菜单占用的存储器。

函数原型： BOOL DestroyMenu（HMENU hMenu）；

参数：

hMenu：要销毁的菜单的句柄。

返回值： 如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 一个应用程序在关闭之前，必须调用函数 DestroyMenu 来销毁一个没被分配给窗口的菜单。分配给窗口的菜单，当应用程序关闭时，被自动销毁。

速查： Windows NT：3.1 及以上版本;Windows：95 及以上版本；Windows CE：1.0 及以上版本;头文件：winuser.h；输入库：user32.lib。

2.13.6 DrawMenuBar

函数功能： 该函数重画指定菜单的菜单条。如果系统创建窗口以后菜单条被修改，则必须调用此函数来画修改了的菜单条。

函数原型： BOOL DrawMenuBar（HWND hWnd）；

参数：

hWnd：其菜单条需要被重画的窗口的句柄。

返回值： 如果函数调用成功，返回非零值：如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT:及以上版本;Windows:95 及以上版本;Windows:2.0 及以上版本;头文件:winuser.h;输入库：user32.lib。

2.13.7 EnableMenultem

函数功能：该函数使指定的菜单项有效、无效或变灰。

函数原型：BOOL EnableMenutem（HMENU hMenu，UINT ulDEnablttem，UINT uEnable;

参数

hMenu：菜单句柄。

ulDEnableltem：指定将使其有效、无效或变灰的菜单项，按参数 uEnable 确定的含义。此参数可指定菜单条、菜单或子菜单里的菜单项。

uEnable：指定控制参数 uIDEnableltem 如何解释的标志，指示菜单项有效、无效或者变灰。此参数必须是 MF_BYCOMMAND 或 MF_BYPOSITION，MF_ENABLED 和 MF_DISABLE 或 MF_GRAYED 的组合。

MF_BYCOMMAND：表明参数 uIDEnableltem 给出了菜单项的标识符。如果 MF_BYCOMMAND 和 MF_POSITION 都没被指定，则 MF_BYCOMMAND 为缺省标志。

MF_BYPOSITION：表明参数 uIDEnableltem 给出了菜单项的以零为基准的相对位置。

MF_DISABLED：表明菜单项无效，但没变灰，因此不能被选择。

MF_ENABLED：表明菜单项有效，并从变灰的状态恢复，因此可被选择。

MF_GRAYED：表明菜单项无效并且变灰，因此不能被选择。

返回值：返回值指定菜单项的前一个状态（MF_DISABLED，MF_ENABLED 或 MF_GRAYED）。如果此菜单项不存在，则返回值是 OXFFFFFFFF。

备注：一个应用程序必须用 MF_BYPOSITION 来指定正确的菜单句柄。如果菜单条的菜单句柄被指定，顶层菜单项（菜单条上的菜单项）将受到影响。若要根据位置来设置下拉菜单中的菜单项或子菜单的状态，应用程序指定下拉菜单或子菜单的句柄。

当应用程序指定 MF_BYCOMMAND 标志时，系统在由指定菜单句柄标识的菜单里选取那些打开了子菜单的菜单项。因此除非要复制菜单项，指定菜单条的句柄就足够了。

函数 InsertMenu，InsertMenultem，LoadMenulndirect，ModifyMenu 和 SetMenultemlnfo 也可设置菜单项的状态（有效、无效或变灰）。

Windows CE：Windows CE 不支持参数 uEnable 取 MF_DISABLED 标志。

如果没有变灰，菜单项不能无效。要使菜单项无效，用 MF_RAYED 标志。

速查：Windows NT：3.1 及以上版本;Windows：95 的及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.13.8 GetMenu

函数功能：该函数取得分配给指定窗口的菜单的句柄。

函数原型：HMENU GetMenu（HWND hWnd);

参数：

hWnd：其菜单句柄被取得的窗口的句柄。

返回值：返回值是菜单的句柄。如果给定的窗口没有菜单，则返回 NULL。如果窗口是一个子窗口，返回值无定义。

速查：Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;输入库：user32.lib。

2.13.9 GetMenuDefaultltem

函数功能： 该函数确定指定菜单上的缺省项。

函数原型： UINT GetMenuDefaultltem (HMENY hMenu, UINT fByPos, UINT gmdiFlags);

参数：

hMenu：获取缺省项的菜单的句柄。

fByPos：用于确定是取得菜单项的标识符还是位置的值。如果此参数值为 FALSE，返回标识符，否则返回位置。

gmdiFlags：指定函数如何查找菜单项。此参数可取灵或多个下列值：

GMDI_GOINTOPOPUPS：如果缺省项打开了子菜单，此函数在相应的子菜单里递归查找。如果子菜单没有缺省项，返回值表示打开了子菜单的项。缺省情况下，函数返回指定菜单的第一个缺省项，不管它是否打开了一个子菜单。

GMDI_USEDISABLED：指定函数返回一个缺省项，即使该项无效。缺省情况下，函数跳过无效或变灰的项。

返回值： 如果函数调用成功，返回值是菜单项的标识符或位置；如果函数调用失败，返回值是 C1。若想获得更多的错误信息，请调用 GetLastError 函数。

速查： Windows：4.0 及以上版本；Windows：95 及以上版本：Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.13.10 GetMenultemlD

函数功能： 该函数确定指定菜单里的菜单项个数。

函数原型： int GetMenultemCount (HMENU hMenu);

参数：

hMenu：被检查的菜单的句柄。

返回值： 如果函数调用成功，返回值是菜单里指定的菜单项数；如果函数调用失败，返回值是 C1。若想获得更多的错误信息，请调用函数 GetLastError 函数。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.fib。

2.13.11 GetMenultemID

函数功能： 该函数取得菜单里指定位置处的菜单项的标识符。

函数原型： UINT GetMenultemID (HMENU hMenu,int nPos);

参数：

hMenu：其菜单项标识符将被取得的菜单的句柄。

nPos：指定将取得其标识符的菜单项相对于零的位置。

返回值： 返回值是给定菜单项的标识符。如果菜单项标识符是 NULL 或指定的菜单打开了子菜单，返回值是 OXFFFFFFFF。

速查： Windows NT：3.1 及以上版本：Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输人库：user32.fib。

2.13.12 GetMenuItemInfo

函数功能：该函数取得一个菜单项的信息。

函数原型：BOOL GetMenuItemInfo(HMENU hMenu，UINT uItem，BOOL fByPosition，LPMENUITEMINFOlpmii);

参数：

hMenu：包含指定菜单项的菜单的句柄。

Uitem：将取得其信息的菜单项的标识符或位置。此参数的含义取决于参数 fByPosition 的值。

FbyPosition：此值用于指定参数 Uitem 的含义。如果此参数是 FALSE，则 uitem 表示菜单项的标识符。否则，表示菜单项的位置。

lpmii：指向结构 MENUITEMINFO 的指针；该结构指定要取得的信息并接收菜单项的信息。

返回值：如果函数调用成功，返回非零值：如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：Windows CE 环境下，由参数 lpmii 指向的 MENUITEMINFO 结构的 fMask 成员不能取 MIM_CHECKMARKS 标志。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本;Windows cE：1.0 及以上版本;头文件：winuser.h；输入库：user32.lib Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.13.13 GetMenuItemRect

函数功能：该函数取得指定菜单项的边界矩形。

函数原型：BOOL GetMenuItemRect（HWND hWnd，HMENU hMenu，UINT uItem,LPRECT lprcItem);

参数：

hWnd：含有指定菜单的窗口的句柄。

在 Windows NT 和 Windows 98 环境中，如果此值为 NULL 且 hMenu 代表一个弹出式菜单，此函数将找到菜单窗口。

uItem：菜单项相对于零的位置。

lprcItem：指向结构 RECT 的指针，该结构接收指定菜单项的边界矩形（按屏幕坐标）。

返回值：如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h;输入库：user32.lib。

2.13.14 GetSubMenu

函数功能：该函数取得被指定菜单激活的下拉式菜单或子菜单的句柄。

函数原型:HMENU GetSubMenu（HMENU hMenu，int nPos);

参数：

hMenu：菜单句柄。

nPos：激活下拉式菜单或子菜单的菜单项相对于零的位置。

返回值：如果函数调用成功，返回值是菜单项激活的下拉式菜单或子菜单的句柄。如果菜单项没有激活一个下拉式菜单或子菜单，返回值是 NULL。

速查： Windows NT：3.1 及以上版本；Wiodows：95 及以上版本；Windows CE：1.0 及以上版本;头文件：Winuser.h；输入库：user32.lib。

2.13.15 GetSystemMenu

函数功能： 该函数允许应用程序为复制或修改而访问窗口菜单（系统菜单或控制菜单）。

函数原型： HMENU GetSystemMenu(HWND hWnd,BOOL bRevert)；

参数：

hWvd：拥有窗口菜单拷贝的窗口的句柄。

BPevert：指定将执行的操作。如果此参数为 FALSE，GetSystemMenu 返回当前使用窗口菜单的拷贝的句柄。该拷贝初始时与窗口菜单相同，但可以被修改。

如果此参数为 TRUE，GetSystemMenu 重置窗口菜单到缺省状态。如果存在先前的窗口菜单，将被销毁。

返回值： 如果参数 bRevert 为 FALSE，返回值是窗口菜单的拷贝的句柄：如果参数 bRevert 为 TRUE，返回值是 NULL。

备注： 任何没有用函数 GetSystemMenu 来生成自己的窗口菜单拷贝的窗口将接受标准窗口菜单。

窗口某单最初包含的菜单项有多种标识符值，如 SC_CLOSE，SC_MOVE 和 SC_SIZE。

窗口菜单上的菜单项发送 WM_SYSCOMMAND 消息。

所有预定义的窗口菜单项的标识符数大于 0xF000。如果一个应用程序增加命令到窗口菜单，应该使用小于 0xF000 的标识符数。

系统根据状态自动变灰标准窗口菜单上的菜单项。应用程序通过响应在任何某单显示之前发送的 WM_INITMENU 消息来实现选取和变灰。

Windows CE 环境下，不支持系统菜单，但 GetSyemMenu 以宏的方式实现，以保持和已存在代码的兼容性。可以使用该宏的返回菜单句柄使关闭框无效，与在 Windows 桌面平台上一样。Windows CE 下的返回值没有其他用处。参数 bRevert 无用。

用下面的代码使关闭按钮无效：

EnableMenultem（GetSystemMenu（hwnd，FALSE），SC_CLOSE，MF_BYCOMMAND I MF_GRAYED)；

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输人库：user32.lib。

2.13.16 HlllteMenultem

函数功能： 该函数对菜单条中的菜单项加亮或清除亮度。

函数原型： BOOL HiliteMenultem（HWND hwnd,HMENU hMenu，UINT ultemHilite,UINT uHilite)；

参数：

hWnd：具有菜单的窗口句柄。

hMenu：含有将被加亮的菜单项的菜单条句柄。

UltmHilite：指定将被加亮的菜单项。此参数可以是菜单项的标识符，也可为菜单项在菜单条中的偏移量，其含义由参数 uHilite 的值确定。

yHilite：控制参数 ultemHilite 如何解释的标志，并确定菜单项是否被加亮。此参数必须是 MF_BYCOMMAND 或 MF_BYPOSITION 和 MF_HILITE 或 MF_UNHILITE 的组合。

MF_BYCOMMAND：表示参数 ultemHilite 给出了菜单项的标识符。

MF_BYPOSITION：表示参数 ultemHilite 给出了菜单项相对于零的位置。

MF_HILITE：加亮菜单项。如果此标志未被指定，则清除菜单项的亮度。

MF_UNHILITE：清除菜单项的亮度。

返回值：如果菜单项被设置为指定的加亮状态，返回非零值；如果菜单项未被设置为指定的加亮状态，返回零。

备注：MF_HILITE 和 MF_UNHILITE 标志只能被函数 HiliteMenutem 使用，不能被函数 ModifyMenu 使用。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.13.17 InsertMenultem

函数功能:该函数在菜单的指定位置插入一个新菜单项。

函数原型：BOOL WINAPI InsertMenutem（HMENU hMenu，UINT ultem，BOOL fByPosition，LPMENUITEMINFO lpmii ）；

参数:

hMenu：新菜单项将被插入其中的菜单的句柄。

ultem：在其前面插入新菜单项的菜单项的标识符或位置。此参数的含义取决于参数 fByPosition 的值。

fByFosition：用于确定 ultem 的含义的值。如果此参数为 FALSE，Ultem 表示菜单项的标识符。否则，ultem 表示菜单项的位置。

lpmii：指向结构 MENUITEMINFO 的指针，该结构中包含了新菜单项的信息。

返回值：如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：为了使键盘加速键能控制位图或自己绘制的菜单项，菜单的拥有者必须处理 WM_MENUCHAR 消息。参见自绘制菜单和 WM_MENUCHAR 消息。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib Unicode：在 Windows NT 环境下，以 Unicode 和 ANSI 方式实现。

2.13.18 IsMenu

函数功能：该函数确定一个句柄是否为菜单句柄。

函数原型：BOOL lsMenu（HMENU hMenu）；

参数：

hMenu：被测试的句柄。

返回值：如果 hMenu 是一个菜单句柄，返回非零值。如果 hMenu 不是一个菜单句柄，返回值是零。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：Winuser.h；输入库：user32.lib。

2.13.19 LoadMenu

函数功能：该函数从与应用事例相联系的可执行文件（. EXE）中加载指定的菜单资源。

函数原型：HMENU LoadMenu（HINSTANCE hlnstance，LPCTSTR lpMenuName）；

参数：

hlnstance：含有被加载菜单资源的事例模块的句柄。

LpMenuName：指向含有菜单资源名的以空结束的字符串的指针。同时，此参数可由低位字上的资源标

识符和高位字上的零组成。要创建此值，用 MAKEINTRESOURCE 宏。

返回值：如果函数调用成功，返回值是菜单资源句柄；如果函数调用失败，返回值是 NULL。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：关闭应用程序之前，用函数 DestroyMenu 来销毁菜单并释放加载菜单占用的内存。Windows CE 1.0 不支持层叠式菜单。Windows CE 2.0 及更高版本支持层叠式菜单。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下，以 Unicode 和 ANSI 方式实现。

2.13.20 LoadMenuIndirect

函数功能：该函数加载指定的菜单模板到内存。

函数原型：HMENU LoadMenuIndirect（CONST MENUTEMPLATE”lpMenuTemplate)；

参数：

lpMenuTemplate：指向菜单模板或扩展菜单模板的指针。

一个菜单模板由一个 MENUITEMTEMPLATEHEADER 结构和一个或多个连续的 MENUITEMTEMPLATE 结构组成。一个扩展菜单模板由一个 MENUEX_TEMPLATE_HEADER 结构和一个或多个 MENUEX_TEMPLATE_ITEM 结构组成。

返回值：如果函数调用成功，返回值是菜单句柄；如果函数调用失败，返回值为 NULL。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：此函数的 ANSI 和 Unicode 版本中，在 MENUITEMTEMPLATE 结构中的字符串必须是 Unicode 串。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：User32.lib；Unicode：在 Windows NT 环境下，以 Unicode 和 ANSI 方式实现。

2.13.21 MenuItemFromPo

函数功能：该函数确定指定位置的菜单项（如果存在)。

函数原型：UINT WINAPI MenuItemFromPoint(HWND hWnd,HMENU hMenu,POINT ptScreen)；

参数：

hWnd：含有菜单的窗口的句柄。

在 Windows NT 5.0 和 Windows 98 环境下，如果此值为 NULL 并且参数 hMenu 指定一个弹出式菜单，则此函数将会找到菜单窗口。

hMenu：将被命中测试的菜单项所在的菜单的句柄。

PtScreen：用于指定测试位置的 POINT 结构。如果 hMenu 指定一个菜单条，此参数按窗口坐标，否则按客户坐标定义。

返回值：返回指定位置的菜单项相对于零的位置，或者当指定位置没有菜单项时返回 C1。

速查：Windows NT：4.0 及以上版本：Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.13.22 RemoveMenu

函数功能：该函数从指定菜单删除一个菜单项或分离一个子菜单。如果菜单项打开一个下拉式菜单或子菜单，RemoveMenu 不消毁该菜单或其句柄，允许菜单被重用。在调用此函数前，函数 GetSubMenu 应当取得下拉式菜单或子菜单的句柄。

函数原型： BOOL RemoveMenu（HMENU hMenu，UINT uPosition,UINT uFlgs)；

参数：

hMenu：将被修改的菜单的句柄。

UPosition：指定将被删除的菜单项，其含义由参数 uFlages 决定。

uFlags：指定参数 uPosition 如何解释。此参数必须为下列之一值：

MF_BYCOMMAND：表示 uPositon 给出菜单项的标识符。如果 MF_BYCOMMAND 和 MF_BYPOSITION 都没被指定，则 MF_BYCOMMAND 是缺省标志。

Mu_BYPOSITION：表示 uPositon 给出菜单项相对于零的位置。

返回值： 如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 只要一个菜单被修改，无论它是否在显示窗口里，应用程序都必须调用函数 DrawMenuBar。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.13.23 SetMenu

函数功能： 该函数分配一个新菜单到指定窗口。

函数原型： BOOL SetMenu（HWND hWnd，HMENU hMenu)；

参数：

hWnd：菜单被分配到其中的窗口的句柄。

HMenu：新菜单的句柄。如果菜单参数为 NULL，则窗口的当前菜单被删除。

返回值： 如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLaSError 函数。

备注： 窗口被重画来反映菜单的修改。函数 SetMenu 替换原来的菜单（如果存在），但并不将其销毁。应用程序必须调用函数 DestroyMenu 来销毁菜单。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输人库：user32.lib。

2.13.24 SetMenuDefaultltem

函数功能： 该函数给指定的菜单设置缺省菜单项。

函数原型： BOOL SetMenuDefaultltem（HMENU hMenu，UINT ultem，UINT fByPo)；

参数：

httenu：将为其设置缺省菜单项的菜单的句柄。

Uttrne：新缺省菜单项的标识符或位置，无缺省项时，取值为 C1。此参数的含义由参数 fByPoS 的值决定。

ByPos：用于确定参数 ultem 的值的含义。如果此参数为 FALSE，参数 ultem 表示菜单项的标识符。否则，表示菜单项的位置。

返回值： 如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.13.25 SetMenultemBitm

函数功能： 该函数将指定的位图与一个菜单项相联系。无论该菜单项是否被选取，系统都将适当的位图显示在菜单项旁边。

函数原型： BOOL SetMenultemBitmaPs（HMENU hMenu，UINT uPosition,UINT uFlags，HBITMAP hBitmapUnchecked,HBITMAP hBitmspChecked)；

参数：

hMenu：其菜单项将接受新选取标记位图的菜单的句柄。

uPosition：指定将被修改的菜单项。其含义由参数 uFlags 决定。

UFlags：指定参数 uPosition 将如何解释。此参数必须是下列值之一：

MF_BYCOMMAND：表示参数 uPosition 给出菜单项的标识符。如果 MF_BYCOMMAND 和 MF_POSITION 都没被指定，则 MF_BYCOMMAND 是缺省标志。

MF_BYPOSITION：表示参数 uPosition 给出菜单项相对于零的位置。

hBitmapUnchecked：当菜单项没被选取时显示的位图的句柄。

hBitmapChecked：当菜单项被选取时显示的位图的句柄。

返回值： 如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 如果参数 hBitmapUnchecked 或 hBitmapChecked 的值为 NULL，系统将不为相应选取状态显示任何位图到菜单项旁边。如果两参数值均为 NULL，系统在菜单项被选取时显示缺省的选取标志位图，菜单项未被选取时删除位图。当菜单项被销毁时，位图并没被销毁，需要应用程序来将其销毁。

已选取或未选取的位图应当是单色的。系统将用布尔 AND 运算符组合位图和菜单。这样，位图中白色部分变成透明的，而黑色部分成为菜单项的颜色。如果使用彩色位图，结果会不符合需要。以 CXMENUCHECK 和 CYMENUCHECK 来使用函数 GetSystemMetrics 将取得位图的尺寸。

速查： Windows NT：3.1 及以上版本：Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.13.26 SetMenultemlnfo

函数功能： 该函数改动一个菜单项的信息。

函数原型： BOOL SetMenultemlnfo（HMENU hMenu，UINT uitem，BOOL fByPosition，LPMENUITEMINFO lpmii）；

参数：

hMenu：包含菜单项的菜单的句柄。

ultem：将被修改的菜单项的标识符或位置。此参数的含义由参数 fByPosition 确定。

FByPosition：用于指定参数 ultem 的含义的值。如果此参数值为 FALSE，则参数 ultem 是菜单项的标识符，否则，表示菜单项的位置。

lpmii：指向结构 MENUITEMINFO 的指针。该结构含有菜单项的信息，并且．指定将被修改的菜单项的属性。

返回值： 如果函数调用成功，返回非零值：如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 为了使键盘加速键能控制位图或自己绘制的菜单项，菜单的拥有者必须处理 WM_MENUCHAR 消息。参见自绘制菜单和 WM_MENUCHAR 消息。Windows CE 环境下，只有下列值对参数 lpmii 指向的 MENUITEMINFO 结构中的 fMask 成员有效：MIIM_DATA;MIIM_；MIIM_TYPE;如果 MIIM－TYPE 被指定，结构 MENUITEMINFO 的

fType 成员必须为菜单项的当前类型，也就是说，该类型不能被改变。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下，以 Unicode 和 ANSI 方式实现。

2.13.27 TrackPopupMenu

函数功能：该函数在指定位置显示快捷菜单，并跟踪菜单项的选择。快捷菜单可出现在屏幕上的任何位置。

函数原型：BOOL TrackPopupMenu（HMENU hMenu，UINT uFlags，int x，int y，int nReserved，HWND hWnd，CONST RECT”prcRect）；

参数

hMenu：被显示的快捷菜单的句柄。此句柄可为调用 CreatePopupMenu 创建的新快捷菜单的句柄，也可以为调用 GetSubMenu 取得的与一个已存在菜单项相联系的子菜单的句柄。

uFlags：一种指定功能选项的位标志。用下列标志位之一来确定函数如何水平放置快捷菜单：

TPM_CENTERALLGN：若设置此标志，函数将按参数 x 指定的坐标水平居中放置快捷菜单。

TPM_LEFTALLGN：若设置此标志，函数使快捷菜单的左边界与由参数 X 指定的坐标对齐。

TPM_RIGHTALLGN：若设置此标志，函数使快捷菜单的右边界与由参数 X 指定的坐标对齐。

用下列标志位之一来确定函数如何垂直放置快捷菜单：

TPM_BOTTOMALLGN：若设置此标志，函数使快捷菜单的下边界与由参数 y 指定的坐标对齐。

TPM_TOPALLGN：若设置此标志，函数使快捷菜单的上边界与由参数 y 指定的坐标对齐。

TPM_VCENTERALLGN；若设置此标志，函数将按参数 y 指定的坐标垂直居中放置快捷菜单

用下列标志位之一来确定在菜单没有父窗口的情况下用户的选择：

TPM_NONOTIFY：若设置此标志，当用户单击菜单项时函数不发送通知消息。

TPM_RETURNCMD；若设置此标志；函数将用户所选菜单项的标识符返回到返回值里。

用下列标志位之一来确定在快捷菜单跟踪哪一个鼠标键：

TPM_LEFTBUTTON:若设置此标志，用户只能用鼠标左键选择菜单项。

TPM_RIGHTBUTTON：若设置此标志，用户能用鼠标左、右键选择菜单项。

X：在屏幕坐标下，快捷菜单的水平位置。

Y:在屏幕坐标下，快捷菜单的垂直位置。

NReserved：保留值，必须为零。

HWnd：拥有快捷菜单的窗口的句柄。此窗口接收来自菜单的所有消息。函数返回前，此窗口不接受来自菜单的 WM_COMMAND 消息。

如果在参数 uFlags 里指定了 TPM_NONOTIFY 值，此函数不向 hWnd 标识的窗口发消息。 但必须给 hWnd 里传一个窗口句柄，可以是应用程序里的任一个窗口句柄。

PrcRect：未用。

返回值：如果在参数 uFlags 里指定了 TPM_RETURNCMD 值，则返回值是用户选择的菜单项的标识符。如果用户未作选择就取消了菜单或发生了错误，则退回值是零。如果没在参数 uFlags 里指定 TPM_RETURNCMD 值，若函数调用成功，返回非零值，若函数调用失败，返回零。若想获得更多的错误信息，清调用 GetLastError 函数:

备注：Windows CE 不支持参数 uFlags 取下列值：TPM_NONOTIFY；TPM_LEFTBUTTON；TPM_RIGHTBUTTON。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.13.28 TrackPopupMenuEx

函数功能： 该函数在指定位置显示快捷菜单，并跟踪菜单项的选择。快捷菜单可出现在屏幕上的任何位置。

函数原型：BOOL TrackPopupMenuEX（HMENU hMenu，UINT uFlags，int x,int y，HWND hWnd，LPTPMPARAMS lptpm）;

参数：

hMenu：被显示的快捷菜单的句柄。此句柄可为调用 CreatePoPuPMenu 创建的新快捷菜单的句柄，也可以为调用 GetSubMenu 取得的与一个已存在菜单项相联系的子菜单的句柄。

UFlags：定位或其他选项。此参数可为零或取在函数 CreatePopupMenu 里所列的值，也可取下列之一值：

TPM_HORIZONTAL：在不覆盖排斥矩形就不能在指定位置显示菜单时，系统将先考虑水平对齐的要求。

TPM_VERTICAL：在不覆盖排斥矩形就不能在指定位置显示菜单时，系统将先考虑垂直对齐的要求。排斥矩形是指屏幕上菜单不能覆盖的部分，由 Iptpm 指定。

X：在屏幕坐标下，快捷菜单的水平位置。

Y：在屏幕坐标下，快捷菜单的垂直位置。

hWnd：拥有快捷菜单的窗口的句柄。此窗口接收来自菜单的所有消息。函数返回前，此窗口不接受来自菜单的 WM_COMMAND 消息。

如果在参数 uFlags 里指定了 TPM_NONOTIFY 值，此函数不向 hwnd 标识的窗口发消息。但必须给 hwnd 里传一个窗口句柄，可以是应用程序里的任一个窗口句柄。

lptpm：指向结构 TPMPARAMS 的指针，该结构指定屏幕上菜单不能覆盖的区域。此参数可为 NULL。

返回值：如果在参数 UFlags 里指定了 TPM_RETURNCMD 值，则返回值是用户选择的菜单项的标识符。如果用户未作选择就取消了菜单或发生了错误,则返回值是零。如果没在参数 uFlags 里指定 TPM_RETURNCMD 值，函数调用成功，返回非零值，若函数调用失败，返回零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：Womdpws CE 不支持参数 uFlags 取下列值：

TPM_NONOTIFY; TPM_LEFTTBUTTON;TPM_RIGHTBUTTTON;TPM_HORIZONTAL;TPM_VERTICAL;

参数 lptpm 必须设为 NULL。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.13.29 AppendMenu

函数功能： 该函数在指定的菜单条、下拉式菜单、子菜单或快捷菜单的末尾追加一个新菜单项。此函数可指定菜单项的内容、外观和性能。函数 AppendMenu 已被 lnsertMenuItem 取代。但如果不需要 lnsertMenuItem 的扩展特性，仍可使用 AppendMenu。

函数原型：BOOL AppendMenu（hMenu hMenu，UINT uFlags，UINT uIDNewItem，LPCTSTR lpNewItem）;

参数：

hMenu：将被修改的菜单条、下拉式菜单、子菜单、或快捷菜单的句柄。

UFlags：控制新菜单项的外观和性能的标志。此参数可以是备注里所列值的组合。

UIDNewItem：指定新菜单项的标识符，或者当 uFlags 设置为 MF_POPUP 时，表示下拉式菜单或子菜单的句柄。

LpNewItem:指定新菜单项的内容。此参数的含义取决于参数 uFlags 是否包含 MF_BITMAP, MF_OWNERDRAW

或 MF_STRING 标志，如下所示：

MF_BITMAP：含有位图句柄。MF_STRING：以`\0’结束的字符串的指针。

MF_OWNERDRAW：含有被应用程序应用的 32 位值，可以保留与菜单项有关的附加数据。当菜单被创建或其外观被修改时，此值在消息 WM_MEASURE 或 WM_DRAWITEM 的参数 IParam 指向的结构，成员 itemData 里。

返回值：如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：一旦菜单被修改，无论它是否在显示窗口里，应用程序必须调用函数 DrawMenuBar。

为了使键盘加速键能控制位留或自己绘制的菜单项，菜单的拥有者必须处理 WM_MENUCHAR 消息。

参见自绘制菜单和 WM_MENUCHAR 消息。

下列标志可被设置在参数 uFlags 里：

MF_BITMAP：将一个位图用作菜单项。参数 lpNewltem 里含有该位图的句柄。

MF_CHECKED：在菜单项旁边放置一个选取标记。如果应用程序提供一个选取标记，位图（参见 SetMenultemBitmaps），则将选取标记位图放置在菜单项旁边。

MF_DISABLED：使菜单项无效，使该项不能被选择，但不使菜单项变灰。

MF_ENABLED：使菜单项有效，使该项能被选择，并使其从变灰的状态恢复。

MF_GRAYED：使菜单项无效并变灰，使其不能被选择。

MF_MENUBARBREAK：对菜单条的功能同 MF_MENUBREAK 标志。对下拉式菜单、子菜单或快捷菜单，新列和旧列被垂直线分开。

MF_MENUBREAK：将菜单项放置于新行（对菜单条），或新列（对下拉式菜单、子菜单或快捷菜单）且无分割列。

MF_OWNERDRAW：指定该菜单项为自绘制菜单项。菜单第一次显示前，拥有菜单的窗口接收一个 WM_MEASUREITEM 消息来得到菜单项的宽和高。然后，只要菜单项被修改，都将发送 WM_DRAWITEM 消息给菜单拥有者的窗口程序。

MF_POPUP：指定菜单打开一个下拉式菜单或子菜单。参数 uIDNewltem 下拉式菜单或子菜单的句柄。此标志用来给菜单条、打开一个下拉式菜单或于菜单的菜单项、子菜单或快捷菜单加一个名字。

MF_SEPARATOR：画一条水平区分线。此标志只被下拉式菜单、于菜单或快捷菜单使用。此区分线不能被变灰、无效或加亮。参数 IpNewltem 和 uIDNewltem 无用。

MF_STRING：指定菜单项是一个正文字符串；参数 lpNewltem 指向该字符串。

MF_UNCHECKED：不放置选取标记在菜单项旁边（缺省）。如果应用程序提供一个选取标记位图（参见 SetMenultemBitmaps），则将选取标记位图放置在菜单项旁边。

下列标志组不能被一起使用：

MF_DISABLED，MF_ENABLED 和 MF_GRAYED；MF_BITMAP,MF_STRING 和 MF_OWNERDRAW

MF_MENUBARBREAK 和 MF_MENUBREAK；MF_CHECKED 和 MF_UNCHECKED

Windows CE 环境下，不支持参数 fuFlags 使用下列标志：

MF_BITMAP；MF_DOSABLE；MF_GRAYED

MF_GRAYED 可用来代替 MF_DISABLED 和 MFS_GRAYED。

Windows CE 1.0 不支持层叠式菜单。在使用 Windows CE 1.0 时，不能将一个 MF_POPUP 菜单插入到另一个下拉式菜单中。Window CE 1.0 不支持下列标志：

MF_POPUP；MF_MENUBREAK；MF_MENUBARBREAK

Windows CE 2.0 或更高版本中，支持上述标志，也支持层叠式菜单。

2.13.30 CheckMenultem

函数功能：该函数设置指定菜单项的校核标记属性为选取或不选取。该函数已被函数 SetMenultemlnfo

取代。但若不需要 SetMenultemlnfo 的扩展特性，仍可使用 CheckMenultem。

函数原型： DWORD CheckMenultem（HMENU hMenu，UINT ulDCheckltem，UINT uCheck)；

参数：

hMenu：有关菜单的句柄。

UlDCheckltem：指定要设置其选取标记属性的菜单项。其含义由参数 uCheck 决定。

uCheck：指定控制参数 uIDCheckltem 的含义的标志，并指定菜单项的选取标记属性的状态。此参数可为 MF_BYCOMMAND 或 MF_BYPOSITON 和 MF_CHECKED 或 MF_UNCHECKED 的组合。

MF_BYCOMMAND：表示参数 uIDCheckltem 给出了菜单项的标识符。若标志 MF_BYCOMMMAND 和 MF_BYFOSITION 都没被指定，则 MF_BYCOMMMAND 为缺省值。

MF_CHECKED：设置选取标记属性为选取。

MF_UNCHECKED：设置选取标记属性为未选取。

返回值： 返回值指定菜单项的前一状态（MF_CHECKED 或 MF_UNCHECKED)。如果菜单项不存在，返回值是 OXFFFFFFFF。

备注： 莱单条里的项不能有选取标记。

参数 uIDCheckltem 标识一个打开子菜单的菜单项或命令项。对打开子菜单的菜单项，参数 uIDCheckltem 必须指定菜单项的位置。对命令项，参数 uIDCheckltem 可指定其位置或标识符。

2.13.31 GetMenuCheckMarkDimensions

函数功能： 返回缺省选取标记位图的尺寸。系统在选取的菜单项旁边显示该位图。调用 SetMenultemBitmaPs 为菜单项放置选取标记为图前，应用程序必须调用 GetMenuCheckMarkDimensions 来确定恰当的位图大小。

函数原型： LONG GetMenuCheckMarkDimensions（VOID）

参数：无。

返回值： 返回值指定缺省选取标记位图的高度和宽度（按像素)。高位字包含高度，低位字包含宽度。

2.13.32 1.1GetMenuState

函数功能： 该函数取得与指定菜单项相联系的菜单标志。如果该菜单项打开了一个子菜单，该函数也返回子菜单里的菜单项数。

函数原型： UINT GetMenuState（HMENU hMenu，UINT uld，UINT uFlags)；

参数：

hMenu：含有其菜单项的标志将被提取得的菜单的句柄。

uld：其某单标志将被取得的菜单项，此参数含义由参数 uFlags 决定。

UFlags：用于指定参数 uld 的含义的值。此参数可取下列值之一：

MF_BYCOMMAND:表示参数 uld 给出菜单项的标识符。如果 MF_BYCOMMAND 和 MF_BYPOSITION 都没被指定，则 MF_BYCOMMAND 是缺省值。

MF_BYPOSITION：表示参数 uld 给出菜单项相对于零的位置。

返回值： 如果指定的项不存在，返回值是 OXFFFFFFFF；如果菜单项打开了一个子菜单，则返回值的低位含有与菜单相联系的菜单标志，高位含有子菜单的项数。否则，返回值是菜单标志的掩码（布尔 OR)。

下面列出与菜单项相关的菜单标志。

MF_CHECKED：放置选取标记于菜单项旁边（只用于下拉式菜单、子菜单或快捷菜单)。

MF_DISABLED：使菜单项无效。MF_GRAYED：使菜单项无效并交灰。MF_HILITE：加亮菜单项。

MF_MENUBARBREAK：对下拉式菜单、子菜单和快捷菜单，新列和旧列由垂直线隔开，其余功能同MF_MENUBREAK 标志。

MF_MENUBREAK：将菜单项放于新行（对菜单条）或无分隔列地放于新列（对下拉式菜单、子菜单或快捷菜单）。

MF_SEPARATOR：创建一个水平分隔线（只用于下拉式菜单、子菜单或快捷菜单）。

2.13.33 GetMenuString

函数功能： 该函数将指定菜单项的正文字符串拷贝到指定缓冲区。

函数原型： int GetMenuString(HMENU hMenu,UINT uIDItem,LPTSTR lpString,int nMaxCount,UINT uFlag);

参数：

hWenu：菜单句柄。

uIDItem：指定将被修改的菜单项，其含义由参数 uFlag 决定。

lpString：指向缓冲区的指针，该缓冲区接受以`\0’结束的字符串。如果此参数为 NULL，则函数返回菜单字符串的长度。

nMaxCount：指定将被拷贝的字符串的最大字符数。如果字符串长度比此参数指定的最大值还大，则多余的字符被截去。如果此参数为 0，则函数退回菜单字符串的长度。

uFlag：指定参数 ulDltem 如何被解释。此参数可取下列值之一：

MF_BYCOMMAND：表示参数 uIDltem 给出菜单项的标识符。如果 MF_BYCOMMAND 和 MF_BYPOSITION 都没被指定，则 MF_BYCOMMAND 是缺省值。

MF_BYPOSITION：表示参数 uIDltem 给出菜单项相对于零的位置。

返回值： 如果函数调用成功，返回值是拷贝到缓冲区的字符数，不包括末尾‘\0’结束符：如果函数调用失败，返回值是零。

备注： 参数 nMaxCount 的值必须比正文字符串的长度大一，以容纳末尾的‘\0’结束符。如果参数 nMaxCount 的值为零，函数返回菜单字符串的长度。

2.13.34 InsertMenu

函数功能： 该函数插入一个新菜单项到菜单里，并使菜单里其他项下移。

函数原型： BOOL InsertMenu(HMENU hMenu,UINt uPosition,UINT uFlags,UINT uIDNewltem,LPCTSTR lpNewltem);

参数：

hMenu：将被修改的菜单的句柄。

uPosition：指定新菜单项将被插入其前面的菜单项，其含义由参数 uFlagS 决定。

uFlags：指定控制参数 uPosition 的解释的标志、新菜单项的内容、外观和性能。此参数必须为下列值之一和列于备注里的一个值的组合。

MF_BYCOMMAND：表示 uPosition 给出菜单项的标识符。如果 MF_BYCOMMAND 和 MF_BYPOSITION 都没被指定，则 MF_BYCOMMAND 为缺省的标志。

MF_BYPOSITION：表示 uPosition 给出新菜单项基于零的相对位置。如果 uPosition 为 0xFFFFFFFF 新菜单项追加于菜单的末尾。

uIDNewltem：指定新菜单项的标识符，或者当参数 uFlags 设置为 MF_POPUP 时，指定下拉式菜单或子菜单的句柄。

LpNewltem：指定新菜单项的内容。其含义依赖于参数 UFlags 是否包含标志 MF_BITMAP,MF_OWNERDRAW 或 MF_STRING。如下所示：

MF_BITMAP：含有位图句柄。MF_STRING：以`\0’结束的字符串的指针（缺省）。

MF_OWNERDRAW：含有被应用程序应用的 32 位值，可以保留与菜单项有关的附加数据。当菜单被创建或其外观被修改时,此值在消息 WM_MEASURE 或 WM_DRAWITEM 的参数 IParam 指向的结构中、成员 itemData 里。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：一旦菜单被修改，无论它是否在显示窗口里，应用程序必须调用函数 DrawMenuBar。

下列标志可被设置在参数 uFlagS 里：

MF_BITMAP:将一个位图用作菜单项。参数 IpNewltem 里含有该位图的句柄。

MF_CHECKED:在菜单项旁边放置一个选取标记。如果应用程序提供一个选取标记位图（参见 SetMenultemBitmaps），则将选取标记位图放置在菜单项旁边。

MF_DISABLED：使菜单项无效，使该项不能被选择，但不使菜单项变灰。

MF_ENABLED：使菜单项有效，使该项能被选择，并使其从变灰的状态恢复。

MF_GRAYED：使莱单项无效并变灰，使其不能被选择。

MF_MENUBARBREAK：对菜单条的功能同 MF_MENUBREAK 标志。对下拉式菜单、子菜单或快捷菜单，新列和旧列被垂直线分开。

MF_MENUBREAK：将菜单项放置于新行（对菜单条），或新列（对下拉式菜单、子菜单或快捷菜单）且无分割列。

MF_OWNERDRAW：指定该菜单项为自绘制菜单项。菜单第一次显示前，拥有菜单的窗口接收一个 WM_MEASUREITEM 消息来得到菜单项的宽和高。然后，只要菜单项被修改，都将发送 WM_DRAWITEM 消息给菜单拥有者的窗口程序。

MF_POPUP:指定菜单打开一个下拉式菜单或子菜单。参数 uIDNewltem 下拉式菜单或子菜单的句柄。此标志用来给菜单条、打开一个下拉式菜单或子菜单的菜单项、子菜单或快捷菜单加一个名字。

MF_SEPARATOR：画一条水平区分线。此标志只被下拉式菜单、子菜单或快捷菜单使用。此区分线不能被变灰、无效或加亮。参数 IpNewltem 和 uIDNewltem 无用。

MF_STRING：指定菜单项是一个正文字符串：参数 IpNewltem 指向该字符串。

MF_UNCHECKED:不放置选取标记在菜单项旁边（缺省）。如果应用程序提供一个选取标记位图（参见 SetMenultemBitmaps），则将选取标记位图放置在菜单项旁边。

下列标志组不能被一起使用：

MF_BYCOMMAND 和 MF_BYPOSITION

MF_DISABLED，MF_ENABLED 和 MF_GRAYED

MF_BITMAJP,MF_STRING，MF_OWNERDRAW 和 MF_SEPARATOR

MF_MENUBARBREAK 和 MF_MENUBREAK

MF_CHECKED 和 MF_UNCHECKED

Windows CE 环境下，不支持参数 fuFlags 使用下列标志：

MF_BTMAP；MF_DISABLE

参数项如果没变灰，不能使其无效。要使菜单项无效，用 MF_GRAYED 标志。

Windows CE 1．0 不支持层叠式菜单。在使用 Windows CE 1．0 时，不能将一个 MF_POPUP 菜单插入到另一个下拉式菜单中。

2.13.35 ModifyMenu

函数功能：该参数修改已存在的菜单项，并指定菜单项的内容、外观和性能。

函数原型：BOOL ModifyMenu(HMENU hMnu,UINT uPosition,UINT uFlags,UINT uIDNewltem,LPCTSTR IpNewltem);

参数：

hMnu：将被修改的菜单的句柄。

uPosition：指定将被修改的菜单项，其含义由参数 UFlags 决定。

UFlags：指定控制参数 uPosition 的解释的标志、菜单项的内容、外观和性能。此参数必须为下列值之一和列于备注里的一个值的组合。

MF_BYCOMMAND:表示 uPostion 给出菜单项的标识符。如果 MF_BYCOMMAND 和 MF_BYPOSITION 都没被指定则 MF_BYCOMMAND 为缺省的标志。

MF_BYPOSITION:表示 uPosition 给出菜单项基于零的相对位置。

UIDNewltem：指定被修改菜单项的标识符，或者当参数 uFlags 设置为 MF_POPUP 时，指定下拉式菜单或子菜单的句柄。

lpNewltem：指定被修改菜单项的内容。其含义依赖于参数 UFlags 是否包含标志 MF_BITMAP,MF_OWNERDRAW 或 MF_STRING。如下所示：

MF_BITMAP：含有位图句柄；MF_STRING:以`\0’结束的字符串的指针（缺省）。

MF_OWNERDRAW:含有被应用程序应用的 32 位值，可以保留与菜单项有关的附加数据。当菜单被创建或其外观被修改时,此值在消息 WM_MEASURE 或 WM_DRAWITEM 的参数 lparam 指向的结构中,成员 itemData 里。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：如果函数 ModifyMenu 替换了打开下拉式菜单或子菜单的菜单项，则函数销毁旧的下拉式菜单或子菜单，并释放它们占用的内存。

为了使键盘加速键能控制位图或自绘制的菜单项，菜单的拥有者必须处理 WM_MENUCHAR 消息。参见自绘制菜单和 WM_MENUCHAR 消息。

一旦菜单被修改，无论它是否在显示窗口里，应用程序必须调用函数 DrawMenuBar。要修改已存在菜单项的属性，使用函数 CheckMenultem 和 EnableMenultem 更快。

2.14 消息和消息总队列函数（Message and Message Queue）

2.14.1 BroadcastSystemMessage

函数功能：该函数发送消息给指定的接受者。接受者可以是一个应用程序、安装驱动器、网络驱动器、系统级设备驱动器或这些系统组件的组合。

函数原型：long BroadcastSystemMessage(DWORD dwFIags,LPDWORD IpdwRecipients,UINT UiMessage, WPARAMwParam,LPARAM IParam);

参数：

dwFlags：选项标志。可取下列值的组合：

BSF_FLUSHDISK：接受者处理消息之后清洗磁盘。

BSF_FORCEIFHUNG：继续广播消息，即使超时周期结束或一个接受者已挂起。

BSF_IGNORECURRENTTASK:不发送消息给属于当前任务的窗口。这样,应用程序就不会接收自己的消息。

BSF_NOHANG：强制挂起的应用程序超时。如果一个接受者超时，不再继续广播消息。

BSF_NOTIMEOUTIFNOTHUNG：只要接受者没挂起，一直等待对消息的响应。不会出现超时。

BSF_POSTMESSAGE：寄送消息。不能和 BSF_QUERY 组合使用。

BSF_QUERY：每次发送消息给一个接受者，只有当前接受者返回 TRUE 后，才能发送给下一个接受者。

lpdwRecipients：指向变量的指针，该变量含有和接收消息接受者的信息。此变量可为下列值的组合：

BSM_ALLCOMPONENTS：广播到所有的系统组件。

BSM_ALLDESKTOPS：Windows NT 下，广播到所有的桌面。要求 SE_TCB_NAME 特权。

BSM_APPLICATIONS：广播到应用程序。

BSM_INSTALLABLEDRIVERS：Windows 95 下，广播到安装驱动器。

BSM_INTDRIVER：Windows 95 下，广播到网络驱动器。

BSM_VXDS：Windows 95 下，广播到所有系统级设备驱动器。

当函数返回时，此变量接受上述值的组合，以确定真正接受消息的接受者。如果此参数为 NULL，则将消息广播到所有的组件。

uiMessage：系统消息标识符。

WParam：32 位消息特定值。

IParam：32 位消息特定值。

返回值:如果函数调用成功，返回值是正数。如果函数不能广播消息，返回值是 C1。如果参数 dwFlags 为 BSF_QUERY 且至少一个接受者返回 BROADCAST_QUERY_DENY 给相应的消息，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 如果 BSF_QUERY 没指定，函数发送指定的消息给所有请求的接受者，并忽略这些接受者返回的值。

速查：Windows NT:4.0 及以上版本:Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.2 DispatchMessage

函数功能： 该函数调度一个消息给窗口程序。通常调度从 GetMessage 取得的消息。

函数原型： LONG DispatchMessage（CONST MSG*lpmsg);

参数：

lpmsg：指向含有消息的 MSG 结构的指针。

返回值：返回值是窗口程序返回的值。尽管返回值的含义依赖于被调度的消息，但返回值通常被忽略。

备注： MSG 结构必须包含有效的消息值。如果参数 lpmsg 指向一个 WM_TIMER 消息，并且 WM_TIMER 消息的参数 IParam 不为 NULL，则调用 IPa1ram 指向的函数，而不是调用窗口程序。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.3 GetlnputState

函数功能： 该函数确定在调用线程的消息队列里，是否有鼠标键或键盘消息。

函数原型： BOOL GetlnputState（VOID）

参数：无。

返回值： 如果队列里含有一个或多个新的鼠标键或键盘消息，返回非零值。如果队列里没有新的鼠标键或键盘消息，返回值是零。

速查：Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;输入库：user32.lib。

2.14.4 GetMessage

函数功能： 该函数从调用线程的消息队列里取得一个消息并将其放于指定的结构。此函数可取得与指定窗口联系的消息和由 PostThreadMesssge 寄送的线程消息。此函数接收一定范围的消息值。GetMessage 不接收属于其他线程或应用程序的消息。

函数原型： BOOL GetMessage（LPMSG lpMsg，HWND hWnd，UINT wMsgFilterMin，UINT wMsgFilteMax

参数：

lpMsg：指向 MSG 结构的指针，该结构从线程的消息队列里接收消息信息。

hWnd：取得其消息的窗口的句柄。这是一个有特殊含义的值（NULL）。GetMessage 为任何属于调用线程的窗口检索消息，线程消息通过 PostThreadMessage 寄送给调用线程。

wMsgFilterMin：指定被检索的最小消息值的整数。

wMsgFilterMax：指定被检索的最大消息值的整数。

返回值： 如果函数取得 WM_QUIT 之外的其他消息，返回非零值。如果函数取得 WM_QUIT 消息，返回值是零。如果出现了错误，返回值是_1。例如，当 hWnd 是无效的窗口句柄或 lpMsg 是无效的指针时。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 应用程序通常用返回值来确定是否终止主消息循环并退出程序。

GetMesssge 只接收与参数 hWnd 标识的窗口或子窗口相联系的消息，子窗口由函数 IsChild 决定，消息值的范围由参数 wMsgFilterMin 和 wMsgFilterMax 给出。如果 hWnd 为 NULL，则 GetMessage 接收属于调用线程的窗口的消息，线程消息由函数 PostThreadMessage 寄送给调用线程。GetMessage 不接收属于其他线程或其他线程的窗口的消息，即使 hWnd 为 NULL。由 PostThreadMessage 寄送的线程消息，其消息 hWnd 值为 NULL。如果 wMsgFilterMin 和 wMsgFilterMax 都为零，GetMessage 返回所有可得的消息（即，无范围过滤）。

常数 WM_KEYFIRST 和 WM_KEYAST 可作为过滤值取得与键盘输入相关的所有消息：常数 WM_MOUSEFIRST 和 WM_MOUSELST 可用来接收所有的鼠标消息。如果 wMsgFilterMin 和 wMsgFilterMax 都为零，GetMessage 返回所有可得的消息（即，无范围过滤）。

GetMessage 不从队列里清除 WM.PAINT 消息。该消息将保留在队列里直到处理完毕。

注意，此函数的返回值可非零、零或-1，应避免如下代码出现：

while（GetMessage（IpMsg，hwnd，0，0））…

－1 返回值的可能性表示这样的代码会导致致命的应用程序错误。

速查： Windows NT： 3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本：头文件：winuser.h；输入库：user32.Iib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.5 GetMessageExtralnfo

函数功能： 该函数为当前线程取得附加消息信息。附加消息信息是应用程序或驱动程序定义的与当前线程的消息队列联系的 32 位值。可用 SetMessageExtralnfo 来设置线程的附加消息信息，该消息信息将会保留到下一次调用 GetMessage 或 PeekMessage 之前。

函数原型： LONG GetMessageEXtralnfo（VOID）

参数：无。

返回值： 返回值为附加信息。附加信息是设备特定的。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.14.6 GetMessagePos

函数功能：该函数返回表示屏幕坐标下光标位置的长整数值。此位置表示当上一消息由 GetMessage 取得时鼠标占用的点。

函数原型：DWORD GetMessagePos（VOID）

参数：无。

返回值：返回值给出光标位置的 X，y 坐标。X 坐标在低位整数，y 坐标在高位整数。

备注：如前所述，X 坐标在返回值的低位整数，y 坐标在高位整数（都表示有符号值，因为在多显示器的系统里可取得负值）。如果返回值赋给了一个变量，可用 MAKEPOINT 宏从返回值取得 POINT 结构。也可用 GET_X_LPARAM 或 GET_Y_LPARAM 宏来抽取 X，y 坐标。

要得到光标的当前位置而不是上一个消息发生时的位置，调用函数 GetCursorPos。

要点：不要用 LOWORD 或 HIWORD 宏来抽取鼠标位置的 x，y 坐标，因为在多显示器的系统里将返回不正确的结果。多显示器的系统里可取得负的 x， y 坐标，但 LOWORD 和 HIWORD 将坐标当作无符号量。

Windows CE 下，对那些使用记录笔而不是鼠标的设备，光标位置是指当上一信息由 GetMessage 取得时，记录笔在触屏上的位置。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.14.7 GetMessageTime

函数功能：该函数返回由 GetMessage 从当前线程队列里取得上一消息的消息时间。时间是一个长整数，指定从系统开始到消息创建（即，放入线程消息队列）的占用时间（按毫秒计算）。

函数原型：LONG GetMessageTime（VOID）

参数：无。

返回值：返回值为消息时间。

备注：由 GetMessageTime 返回的值对后面的消息并不一定是增长的，因为当计时器计数超过长整数的最大值时，又从零开始计算。为计算消息间的延迟时间，必须验证第二个消息的时间比第一个消息的时间大，然后用第二个消息的时间减去第一个消息的时间。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h 输入库：user32，lib。

2.14.8 GetQueueStatus

函数功能：该函数返回表示调用线程消息队列里的消息的类型的标志。

函数原型：DWORD GetQueueStatus（UINT flags）；

参数：

flags：表示消息类型的队列状态标志。此参数可为下列值的组合：

QS_ALLEVENTS：输入，WM_TIMER， WM_PAINT，WM_HOTKEY 或寄送的消息在队列里。

QS_ALLINPUT：任何消息在队列里。

QS_ALLPOSTMESSAGE：寄送的消息（而不是其他所列消息）在队列里。

QS_HOTKEY：一条 WM_HOTKEY 消息在队列里。QS_INPUT：输入消息在队列里。

QS_KEY：一条 WM_KEYUP WM_KEYDOWN，WM_SYSKEYUP 或 WM_SYSKEYDOWN 消息在队列里。

QS_MOUSE：WM_MOUSEMOVE 消息或鼠标键消息（WM_BUTTONUP WM_RBUTTONDOWN 等）在消息队列里。

QS_MOUSEBUTTON：鼠标键消息（WM_LBUTTONUP，WM_RBUTTONDOWN 等）在消息队列里。

QS_MOUSEMOVE：WM_MOUSEMOVE 消息在消息队列里。

QS_FAINT：WM_PAINT 消息在消息队列里。

QS_POSTMESSAGE：寄送的消息（而不是其他所列消息）在队列里。

QS_SENDMESSAGE：由其他线程或应用程序发送的消息在消息队列里。

QS_TIEMR：一条 WM_TIEMR 消息在消息队列里。

返回值： 返回值的高位字表示队列里当前消息的类型。低位字表示上次调用 GetQueuestatus，GetMessage 或 FeekMessBge 以来加入队列并仍然在队列里的消息的类型。

备注： QS_标志出现在返回值里并不保证以后调用函数 GetMessage 或 PeekMessage 会返回一个消息。GetMesssge 和 PeekMesssge 执行某些内部过滤会导致消息被内部处理。因此，GetQueueStatus 的返回值只能被看作是否调用 GetMessage 或 PeekMessage 的提示。

QS_ALLPOSTMESSAGE 和 QS_POSTMESSAGE 标志在被清除时不一样。QS_POSTMESSAGE 在调用 GetMessage 或 PeekMessage 时清除，无论是否过滤消息。QS_ALLPOSTMESSAGE 在调用 GetMessage 或 PeekMessage 时清除，不过滤消息（wMsgFilterMin 和 wMsgFilterMax 是零）。这对于多次调用 PeeKMessage 来获得不同范围的消息非常有用。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本：Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.14.9 InSendMessage

函数功能： 该函数决定当前窗口程序是否处理另一个线程调用 SendMesssge（在相同进程或不同进程）发送来的消息。

函数原型： BOOL InsendMessage（VOID）；

参数：无。

返回值： 如果窗口程序处理另一个线程调用 SendMessage 发送来的消息，返回非零值。如果窗口程序不处理另一个线程调用 SendMessage 发送来的消息，返回值是零。

速查： Windows NT：3.1 及以上版本：Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h 输入库：user32.Iib。

2.14.10 InSendMessageEx

函数功能： 函数决定当前窗口程序是否处理另一个线程调用 SendMessage（在相同进程或不同进程）发送来的消息。此函数与 InsendMesssge 相似，但另外提供了如何发送消息的信息。

函数原型： DWORD InsendMessageEx（LPVOID IpReserved）；

参数：

IpReserved：保留值，必须为 NULL。

返回值： 如果消息没被发送，返回值是 ISMEX_NOSEND。否则，返回值是一个或多个下列值：

ISMEX_CALLBACK：消息是用函数 SendMesssgeCallback 发送的。发送此消息的线程没被阻塞。

ISMEX_NOTIFY：消息是用函数 SendNotifyMessags 发送的。发送此消息的线程没被阻塞。

ISMEX_REPLIED：窗口程序处理了消息。发送此消息的线程不再被阻塞。

ISMEX_SEND：消息是用函数 SendMessage 或 SendMessageTimeout 发送的。如果 ISMEX_REPLIED 没设置，发送此消息的线程被阻塞。

速查：Wihdows NT：5.0 及以上版本：Windows：98 及以上版本；Wihdows CE：不支持：头文件：winuser.h；输入库：user32.Iib。

2.14.11 PeekMessage

函数功能：该函数为一个消息检查线程消息队列，并将该消息（如果存在）放于指定的结构。

函数原型：BOOL PeekMessage（LPMSG IpMsg，HWND hWnd，UINT wMSGfilterMin，UINT wMsgFilterMax，UINT wRemoveMsg）；

参数：

lpMsg：接收消息信息的 MSG 结构指针。

hWnd：其消息被检查的窗口的句柄。

wMsgFilterMin：指定被检查的消息范围里的第一个消息。

wMsgFilterMax：指定被检查的消息范围里的最后一个消息。

wRemoveMsg：确定消息如何被处理。此参数可取下列值之一：

PM_NOREMOVE：PeekMessage 处理后，消息不从队列里除掉。

PM_REMOVE：PeekMessage 处理后，消息从队列里除掉。

可将 PM_NOYIELD 随意组合到 PM_NOREMOVE 或 PM_REMOVE。此标志使系统不释放等待调用程序空闲的线程。

缺省地，处理所有类型的消息。若只处理某些消息，指定一个或多个下列值：

PM_QS_INPUT：Windows NT5.0 和 Windows 98：处理鼠标和键盘消息。

PM_QS_PAINT：Windows NT 5.0 和 Windows 98：处理画图消息。

PM_QS_POSTMESSAGE：Windows NT 5.0 和 Windows 98：处理所有被寄送的消息，包括计时器和热键。

PM_QS_SENDMESSAGE：Windows NT 5.0 和 Windows 98：处理所有发送消息。

返回值：如果消息可得到，返回非零值；如果没有消息可得到，返回值是零。

备注：和函数 GetMessage 不一样的是，函数 PeekMesssge 在返回前不等待消息被放到队列里。

PeekMesssge 只得到那些与参数 hWnd 标识的窗口相联系的消息或被 lsChild 确定为其子窗口相联系的消息，并且该消息要在由参数 wMsgFiterMin 和 wMsgFiherMax 确定的范围内。如果 hWnd 为 NULL，则 PeekMessage 接收属于当前调用线程的窗口的消息（PeekMessage 不接收属于其他线程的窗口的消息）。如果 hWnd 为 C1，PeekMessage 只返回 hWnd 值为 NULL 的消息，该消息由函数 PostThreadMessage 寄送。如果 wMsgFilterMin 和 wMsgFilterMax 都为零，GetMessage 返回所有可得的消息（即，无范围过滤）。

常数 WM_KEYFIRST 和 WMKEYLAST 可作为过滤值取得所有键盘消息；常数 WM_MOUSEFIRST 和 WM_MOUSELAST 可用来接收所有的鼠标消息。

PeekMessage 通常不从队列里清除 WM_PANT 消息。该消息将保留在队列里直到处理完毕。但如果 WM_PAINT 消息有一个空更新区，PeekMessage 将从队列里清除 WM_PAINT 消息。

Windows CE：有一个 NULL 更新区的 WM_PAINT 消息不从队列里清除。

速查：Windows NT：3.1 及以上版本；Wihdows：95 及以上版本：Windows CE：1.0 及以上版本；头文件：winuser.h 输入库：user32.lib：Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.12 PostMessage

函数功能：该函数将一个消息放入（寄送）到与指定窗口创建的线程相联系消息队列里，不等待线程处理消息就返回。消息队列里的消息通过调用 GetMessage 和 PeekMessage 取得。

函数原型：BOOL PostMessage（HWND hWnd，UINT Msg，WPARAM wParam，LPARAM lParam）；

参数

hWnd：其窗口程序接收消息的窗口的句柄。可取有特定含义的两个值：

HWND.BROADCAST：消息被寄送到系统的所有顶层窗口，包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口。消息不被寄送到子窗口。

NULL：此函数的操作和调用参数 dwThread 设置为当前线程的标识符 PostThreadMessage 函数一样。

Msg：指定被寄送的消息。

wParam：指定附加的消息特定的信息。

IParam：指定附加的消息特定的信息。

返回值：如果函数调用成功，返回非零值：如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：需要以 HWND_BROADCAST 方式通信的应用程序应当用函数 RegisterwindwosMessage 来获得应用程序间通信的独特的消息。

如果发送一个低于 WM_USER 范围的消息给异步消息函数（PostMessage.SendNotifyMessage，SendMesssgeCallback），消息参数不能包含指针。否则，操作将会失败。函数将再接收线程处理消息之前返回，发送者将在内存被使用之前释放。

速查：Windows NT： 3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.13 PostQuitMessage

函数功能：该函数向系统表明有个线程有终止请求。通常用来响应 WM_DESTROY 消息。

函数原型：VOID PostQuitMessage（int nExitCode）；

参数：

pExitCode：指定应用程序退出代码。此值被用作消息 WM_QUIT 的 wParam 参数。

返回值：无。

备注：PostQuitMessage 寄送一个 WM_oUT 消息给线程的消息队列并立即返回；此函数向系统表明有个线程请求在随后的某一时间终止。

当线程从消息队列里取得 WM_QUIT 消息时，应当退出消息循环并将控制返回给系统。返回给系统的退出值必须是消息 WM_QUIT 的 wParam 参数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32，lib：Uhicode：在 Wihdows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.14 PostThreadMessage

函数功能：该函数将一个消息放入（寄送）到指定线程的消息队列里，不等待线程处理消息就返回。

函数原型：BOOL PostThreadMessage（DWORD idThread，UINT Msg，WPARAM wParam，LPARAM IParam）；

参数

idThread：其消息将被寄送的线程的线程标识符。如果线程没有消息队列，此函数将失败。当线程第一次调用一个 Win 32 USER 或 GDI 函数时，系统创建线程的消息队列。要得到更多的信息，参见备注。

Msg：指定将被寄送的消息的类型。

wParam：指定附加的消息特定信息。

IParam：指定附加的消息特定信息。

返回值：如果函数调用成功，返回非零值。如果函数调用失败，返回值是零。若想获得更多的错误信

息，请调用 GetLastError 函数。如果 idThread 不是一个有效的线程标识符或由 idThread 确定的线程没有消息队

列，GetLastError 返回 ERROR_INVALID_THREAD。

备注：消息将寄送到的线程必须创建消息队列，否则调用 PostThreadMessage 会失败。用下列方法之一来处理这种情况：

调用 PostThreadMessage。如果失败，调用 Sleep，再调用 PostThreadMessage，反复执行，直到 PostThreadMessage 成功。

创建一个事件对象，再创建线程。在调用 PostThreadMessage 之前，用函数 WaitForSingleObject 来等特事件被设置为被告知状态。消息将寄送到的线程调用 PeedMessage（£msg，NULL，WM_USER，WM_USER，PM_NOREMOVE）来强制系统创建消息队列。设置事件，表示线程已准备好接收寄送的消息。

消息将寄送到的线程通过调用 GetMesssge 或 PeekMesssge 来取得消息。返回的 MSG 结构中的 hwnd 成员为 NULL。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本：头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.15 RegisterWindowsMessage

函数功能：该函数定义一个新的窗口消息，该消息确保在系统中是唯一的。返回的消息值可在调用函数 SendMessage 或 PostMessage 时使用。

函数原型：UINT RegisterWindowsMessage（LPCTSTR lpString）；

参数：

lpString：指定将被注册的消息的以‘\0’结束的字符串指针。

返回值:如果消息被成功注册，返回值是在范围 0xC000 到 0xFFFF 的消息标识符；如果函数调用失败，返回值是零。要得到更多的错误信息，调用函数 GetLastError。

备注：RegisterWindowMessage 通常为合作应用程序间的通信注册消息。

如果不同的应用程序注册同样的消息字符串，应用程序返回同样的消息值。消息保持注册，直到会话完成。

当一个以上的应用程序必须处理同一个消息时，必须使用 RegisterWindowMessage。要在窗口类里发送私有消息，应用程序可以使用 UM_USER 到 0X7FFF 范围内的任意整数。（在此范围的消息对窗口类私有，而不是对应用程序私有。如，预定义的控制类如 BUTTON， EDIT LISTBOX 和 COMBOBOX 可用此范围的值。）

速查：Windows NT：3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.16 ReplyMessage

函数功能：该函数用于应答由函数 SendMessage 发送的消息,不返回控制给调用 SendMessage 的函数。

函数原型：BOOLReplyMessage（LRESULTIResult）；

参数：

IResult：指定消息处理的结果。可能的值由所发送的消息确定。

返回值：如果调用线程正处理从其他线程或进程发送的消息，返回非零值。如果调用线程不是正处理从其他线程或进程发送的消息，返回值是零。

备注：调用此函数，接收消息的窗口程序允许调用 SendMessage 的线程继续运行，尽管接收消息的线程已返回控制。调用 ReplyMessage 的线程也继续运行。

如果消息不是通过 SendMessage 发送的，或者消息由同一个线程发送，ReplyMessage 不起作用。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本：Windows CE：不支持：头文件：winuser.h；输入库，user32.lib。

2.14.17 SendAsyncProc

函数功能： 该函数是应用程序定义的回调函数，和 SendMessageCallback 一起使用。系统在将消息传给目标窗口程序后，将消息传给回调函数。类型 SENDASYNCPROC 定义了一个指向此回调函数的指针。SendAsyncProc 是此应用程序定义的函数名的占位符。

函数原型：VOID CALLBACK SendAsyncProc（HWND hwnd，UINT uMsg，DWORD dwData，LRESULT IResult）；

参数：

hwnd：其窗口程序接收消息的窗口的句柄。如果将 SendMessageCallback 的参数 hwnd 设置为 HWND_BROADCAST，系统为每个顶层窗口调用一次 SendAsyncProc。

uMsg：指定消息。

dwData：指定从函数 SendMessageCallback 发送来的应用程序定义的值。

IResult：指定消息处理的结果与消息。

返回值：此回调函数无返回值。

备注：通过传一个 SENDASYNCPROC 指针给函数 SendMessageCallback 来安装一个 SendAsyncProc 应用程序定义的回调函数。

此回调函数仅当调用 SendMessageCallback 的线程调用 GetMessage，PeekMessage 或 WaitMessage 时调用。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.14.18 SendMessage

函数功能： 该函数将指定的消息发送到一个或多个窗口。此函数为指定的窗口调用窗口程序，直到窗口程序处理完消息再返回。而函数 PostMessage 不同，将一个消息寄送到一个线程的消息队列后立即返回。

函数原型：LRESULT SendMessage（HWND hWnd，UINT Msg，WPARAM wParam，LPARAM IParam）；

参数：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统中所有顶层窗口，包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口，但消息不被发送到子窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息指定信息。

IParam：指定附加的消息指定信息。

返回值：返回值指定消息处理的结果，依赖于所发送的消息。

备注：需要用 HWND_BROADCAST 通信的应用程序应当使用函数 RegisterWindowMessage 来为应用程序间的通信取得一个唯一的消息。

如果指定的窗口是由调用线程创建的，则窗口程序立即作为子程序调用。如果指定的窗口是由不同线程创建的，则系统切换到该线程并调用恰当的窗口程序。线程间的消息只有在线程执行消息检索代码时才被处理。发送线程被阻塞直到接收线程处理完消息为止。

Windows CE：Windows CE 不支持 Windows 桌面平台支持的所有消息。使用 SendMesssge 之前，要检查

发送的消息是否被支持。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.19 SendMessageCallback

函数功能： 该函数将指定的消息发送到一个或多个窗口。此函数为指定的窗口调用窗口程序，并立即返回。当窗口程序处理完消息后，系统调用指定的回调函数，将消息处理的结果和一个应用程序定义的值传给回调函数。

函数原型： BOOL SendMessageCallback（HWND hwnd，UINT Msg，WPARAM wParam，LPARAM IParam，SEhDASYNCPROC IpResultCallBack，DWORD dwData）；

参数：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统中所有顶层窗口，包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口，但消息不被发送到子窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息指定信息。

IParam：指定附加的消息指定信息。

IpResultCallBack:指向回收函数的指针,窗曰程序处理完消息后调用该回调函数。参见 SendAsyncProc 可得到合适的回调函数的信息。如果 hwnd 为 HWND_BROADCAST,系统为每个顶层窗口调用一次 SendASyncProc 回调函数。

dwData：一个应用程序定义的值，被传给由参数 IPResultCallBack 指向的回调函数。

返回值： 如果函数调用成功，返回非零值。如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 如果发送一个低于 WM_USER 范围的消息给异步消息函数（PostMessage，SendNotifyMesssge；SendMessageCallback），消息参数不能包含指针。否则，操作将会失败。函数将在接收线程处理消息之前返回，发送者将在内存被使用之前释放。

需要以 HWND_BROADCAST 方式通信的应用程序应当用函数 RegisterWindwosMessage 来获得应用程序间通信的独特的消息。

此回调函数仅当调用 SendMessagecallback 的线程调用 GetMessage，PeekMessage 或 WaitMessage 时调用。

速查：Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.20 SendMessageTImeout

函数功能： 该函数将指定的消息发送到一个或多个窗口。此函数为指定的窗口调用窗口程序，并且，如果指定的窗口属于不同的线程，直到窗口程序处理完消息或指定的超时周期结束函数才返回。如果接收消息的窗口和当前线程属于同一个队列，窗口程序立即调用，超时值无用。

函数原型： LRESULT SendMessageTmeoUt（HWND hwnd，UINT Msg，WPARAM wParam，LPARAM IParam，UINTfuFlags，UIUT uTimeout，LPDWORD lpdwResultult）；

参数：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统

中所有顶层窗口，包括无效或不可见的非自身拥有的窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息指定信息。

IParam：指定附加的消息指定信息。

fuFlags：指定如何发送消息。此参数可为下列值的组合：

SMTO_ABORTIFHUNG：如果接收进程处于“hung”状态，不等待超时周期结束就返回。

SMTO_BLOCK：阻止调用线程处理其他任何请求，直到函数返回。

SMTO_NORMAL：调用线程等待函数返回时，不被阻止处理其他请求。

SMTO_ONTIMEOUTIFNOTHUNG：Windows 95 及更高版本：如果接收线程没被挂起，当超时周期结束时不返回。

uTimeout：为超时周期指定以毫秒为单位的持续时间。如果该消息是一个广播消息，每个窗口可使用全超时周期。例如，如果指定 5 秒的超时周期，有 3 个顶层窗回未能处理消息，可以有最多 15 秒的延迟。

IpdwResult：指定消息处理的结果，依赖于所发送的消息。

返回值：如果函数调用成功，返回非零值。如果函数调用失败，或超时，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。如果 GetLastError 返回零，表明函数超时。如果使用 HWND_BROADCAST，SenddMessaggTimeout 不提供单个窗口超时信息。

速查：Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;输入库：user32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 方式实现。

2.14.21 SendNotifyMessage

函数功能：该函数将指定的消息发送到一个窗口。如果该窗口是由调用线程创建的；此函数为该窗口调用窗口程序，并等待窗口程序处理完消息后再返回。如果该窗口是由不同的线程创建的，此函数将消息传给该窗口程序，并立即返回，不等待窗口程序处理完消息。

函数原型：BOOL SendNotifyMessage（HWND hWnd，UINT Msg，WPARAM wParam，LPARAM IParam）；

参数：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统中所有顶层窗口，包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口，但消息不被发送到子窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息指定信息。

IParam：指定附加的消息指定信息。

返回值：如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：如果发送一个低于 WM_USER 范围的消息给异步消息函数（PostMessage，SendNotifyMessage，SendMesssgeCallback），消息参数不能包含指针。否则，操作将会失败。函数将在接收线程处理消息之前返回，发送者将在内存被使用之前释放。

需要以 HWND_BROADCAST 方式通信的应用程序应当用函数 RegisterWindwosMessage 来获得应用程序间通信的独特的消息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境下以 Uncode 和 ANSI 方式实现。

2.14.22 SendMessageExtralnfo

函数功能： 该函数为当前线程设置附加消息信息。附加消息信息是应用程序或驱动程序定义的与当前线程的消息队列联系的 32 位值。GetMessageExtralnfo 来取得线程的附加消息信息。

函数原型： LPARAM SetMesssgeEXtralnfo（LPARAM IParam）;

参数：

IParam：指定与当前线程联系的 32 位值。

返回值： 返回值为前一个 32 位值。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winser.h；输入库：user32.lib。

2.14.23 TranslateMessage

函数功能： 该函数将虚拟键消息转换为字符消息。字符消息被寄送到调用线程的消息队列里，当下一次线程调用函数 GetMessage 或 PeekMessage 时被读出。

函数原型： BOOL TranslateMessage（CONST MSG*lpMsg）;

参数：

IpMsg：指向含有消息的 MSG 结构的指针，该结构里含有用函数 GetMessage 或 PeekMessage 从调用线程的消息队列里取得的消息信息。

返回值： 如果消息被转换（即，字符消息被寄送到调用线程的消息队列里），返回非零值。如果消息是 WM_kEYDOWN，WM_KEYUP WM_SYSKEYDOWN 或 WM_SYSKEYUP，返回非零值，不考虑转换。如果消息没被转换（即，字符消息没被寄送到调用线程的消息队列里），返回值是零。

备注： 此函数不修改由参数 IpMsg 指向的消息。

WM_KEYDOWN 和 WM_KEYUP 组合产生一个 WM_CHAR 或 WM_DEADCHAR 消息。

WM_SYSKEYDOWN 和 WM_SYSKEYUP 组合产生一个 SYSWM_CHAR 或 WM_SYSDEADCHAR 消息。TtanslateMessage 为那些由键盘驱动器映射为 ASCll 字符的键产生 WM_CHAR 消息。

如果应用程序为其他用途处理虚拟键消息，不应调用 TranslateMessage。例如，如果件 ThranslateAccelerator 返回一个非零值，应用程序不应调用 TranslateMessage。

Windows CE:Windows CE 不支持扫描码或扩展键标志，因此，不支持由 TranslateMessage 产生的 WM_CHAR 消息中的 IKeyData 参数（IParam）取值 16-24。

TranslateMessage 只能用于转换调用 GetMessage 或 PeekMessage 接收的消息。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.14.24 WaltMessage

函数功能： 该函数产生对其他线程的控制，如果一个线程没有其他消息在其消息队列里。此函数中止线程，直到一个新消息被放入该线程的消息队列里，再返回。

函数原型；BOOL WaitMessage（VOID）

参数：无。

返回值： 如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：在线程调用一个函数来检查队列后，如果有未经阅读的输入在消息队列里，WaitMessage 不返回。这是因为 PeekMessage，GetMessage，GetQueueStatus：WaitMessage，MsgWaitForMultipleObjects，MsgWaitForMulitpleObjectEx 等函数检查队列后，改变队列的状态信息这样输入不再被认为是新的。如果连续调用 WaitMessage，将等到指定类型的新输入到达后才返回。已存在的未读过的输入（在上次线程检查队列之前接收的）被忽略。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.14.25 PostAppMessage

函数功能：该函数已过时。提供此函数只是为了与 Windows 的 16 位版本兼容。基于 Win32 的应用程序应该使用函数 PostThreadMessage。

2.14.26 SetMessageQueue

函数功能：该函数已过时。提供此函数只是为了与 Windows 的 16 位版本兼容。此函数对基于 Windows 的 32 位版本不起作用，因为消息队列根据需要动态扩展。

2.15 鼠标输入函数（Mouse Input）

2.15.1 DragDetect

函数功能：该函数捕获并跟踪鼠标的移动直到用户松开左键、按下 Esc。键或者将鼠标移动到围绕指定点的“拖动矩形”之外。拖动矩形的宽和高由函数 GetSystemMetrics 返回的 SM_CXDRAG 或 SM_CYDRAG 确定。

函数原型：BOOL DragDetect（HWND hwnd，POINT pt）；

参数：

hwnd：接受鼠标输入的窗口的句柄。

pt：鼠标在屏幕坐标下的初始位置，此函数根据这个点来确定拖动矩形的坐标。

返回值：如果用户在按着鼠标左键时将鼠标移出了拖动矩形之外，则返回非零值；如果用户按着鼠标左键在拖动内移动鼠标，则返回值是零。

备注：拖动矩形的系统度量是可构造的，允许更大或更小的拖动矩形。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.15.2 GetCapture

函数功能：该函数取得捕获了鼠标的窗口（如果存在）的句柄。在同一时刻，只有一个窗口能捕获鼠标；此时，该窗口接收鼠标的输入，无论光标是否在其范围内。

函数原型：HWND GetCapture（VOID）

参数：无。

返回值： 返回值是与当前线程相关联的捕获窗口的句柄。如果当前线程里没有窗口捕获到鼠标，则返回 NULL。

备注：返回 NULL 并不意味着系统里没有其他进程或线程捕获到鼠标，只表示当前线程没有捕获到鼠标。

速查： Windows NT：4.0 及以上版本；Wiodows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.15.3 GetDoubleCllckTime

函数功能： 该函数取得鼠标的当前双击时间。一次双击是指对鼠标键的两次连击，第一次击键后在指定时间内击第二次。双击时间是指在双击中，第一次击键和第二次击键之间的最大毫秒数。

函数原型： UINT GetDoubleClickTime（VOID）

参数：无。

返回值： 返回是当前双击时间，按毫秒计算。

速查： Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.15.4 GetMouseMovePoints

函数功能： 该函数取得鼠标或画笔。

函数原型：int GetMouseMovePoints(UINT cbSize,LPMOUSEMOVEPOlNT lppt,LPMOUSEMOVEPOINT IpptBuf, int, nBufPoints, DWORD resolution);

参数：

cbSize：结构 MOOSEMOVEPOINT 的大小。

lppt：指向结构 MOUSEMOVEPOINT 的指针，该结构包含了有效的鼠标坐标（屏幕坐标）。也可以包含一个时间标记。

函数 GetMouseMovePoints 在鼠标坐标历史记录中查找一点。如果此函数查到该点，则返回包含提供点在内的在此之前的最后一个 nBufPoints。如果应用程序提供一个时间标记，则函数 GetMouseMovePoints 将用它来区分记录于不同时间的两个相等的点。

应用程序使用从消息 WM_MOOSEMOVE 中接收的鼠标坐标来调用此函数，并把它们转换为屏幕坐标。

lpptBut：将接收点的缓冲区的指针。其大小至少应为 cbsze • nBuffPoints。

nBufPoints：指定将取得的点的个数。

resolution：指定希望的分辨率。此参数可取下列值之一：

GMMP_USE_DISPLAY_POINTS：用显示分辨率取得点。

GMMP_USE_DRIVER_pOINTS：用驱动器分辨率取得点。在 Windows CE 平台下，画笔驱动器的分辨率高于显示分辨率。这样，函数 GetMouseMovePoints 可被那些需要准确分辨率的应用程序使用。（如手写体识别软件或计算机辅助设计软件）。

返回值： 如果函数调用成功，返回值是缓冲区里的点的数目。否则，函数返回 C1。若想获得更多的错误信息，请调用 GetlastError 函数。函数 GetLastError 可能返回下面的错误代码。

GMMF_ERR_POINT_NOT_FOUNQ 由 lPPt 指定的点找不到或不再存在于系统缓冲区中。

备注： 系统至少保留着 64 个鼠标坐标及其时间标记。如果应用程序给 GetMouseMovePoints 提供了一个鼠标坐标，而该坐标存在于系统中的鼠标坐标历史记录中，则函数从历史坐标记录取得指定个数的坐标。也可以提供一个时间标记，用来区分历史记录中相同的点。

函数 GetMouseMonePoints 将返回实际发送给调用线程和其他线程的点。

速查：Windows NT：5.0 及以上版本；Windows：98 及以上版本；Windows CE：2.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.15.5 mouse_event

函数功能：该函数综合鼠标击键和鼠标动作。

函数原型：VOID mouse_event（DWORD dwFlags，DWORD dx，DWORD dwFlags，OWORD dx，DWORD dy， DWORD dwData， DWORD dwExtralnfo)；

参数：

dwFlags：标志位集，指定点击按钮和鼠标动作的多种情况。此参数里的各位可以是下列值的任何合理组合：

MOOSE_EVENTF_ABSOLOTE：表明参数 dX，dy 含有规范化的绝对坐标。如果不设置此位，参数含有相对数据：相对于上次位置的改动位置。此标志可被设置，也可不设置，不管鼠标的类型或与系统相连的类似于鼠标的设备的类型如何。要得到关于相对鼠标动作的信息，参见下面备注部分。

MOOSEEVENTFMOVE：表明发生移动。

MOOSEEVENTF_LEFTDOWN：表明接按下鼠标左键。

MOOSEEVENTF_LEFTUP：表明松开鼠标左键。

MOOSEEVENTF_RIGHTDOWN：表明按下鼠标右键。

MOOSEEVENTF_RIGHTUP：表明松开鼠标右键。

MOOSEEVENTF_MIDDLEDOWN：表明按下鼠标中键。

MOOSEEVENTF_MIDDLEUP：表明松开鼠标中键。

MOOSEEVENTF_WHEEL：在 Windows NT 中如果鼠标有一个轮，表明鼠标轮被移动。移动的数量由 dwData 给出。

dx：指定鼠标沿 x 轴的绝对位置或者从上次鼠标事件产生以来移动的数量，依赖于 MOOSEEVENTF_ABSOLOTE 的设置。给出的绝对数据作为鼠标的实际 X 坐标；给出的相对数据作为移动的 mickeys 数。一个 mickey 表示鼠标移动的数量，表明鼠标已经移动。

dy：指定鼠标沿 y 轴的绝对位置或者从上次鼠标事件产生以来移动的数量，依赖于 MOOSEEVENTF_ABSOLVTE 的设置。给出的绝对数据作为鼠标的实际 y 坐标，给出的相对数据作为移动的 mickeys 数。

dwData：如果 dwFlags 为 MOOSEEVENTF_WHEEL，则 dwData 指定鼠标轮移动的数量。正值表明鼠标轮向前转动，即远离用户的方向；负值表明鼠标轮向后转动，即朝向用户。一个轮击定义为 WHEEL_DELTA，即 120。

如果 dwFlagsS 不是 MOOSEEVENTF_WHEEL，则 dWData 应为零。

dwExtralnfo：指定与鼠标事件相关的附加 32 位值。应用程序调用函数 GetMessgeExtrajnfo 来获得此附加信息。

返回值：无。

备注：如果鼠标被移动，用设置 MOUSEEVENTF_MOVE 来表明，dX 和 dy 保留移动的信息。给出的信息是绝对或相对整数值。

如果指定了 MOWSEEVENTF_ABSOLOTE 值，则 dX 和 dy 含有标准化的绝对坐标，其值在 0 到 65535 之间。事件程序将此坐标映射到显示表面。坐标（0，0）映射到显示表面的左上角，（6553，65535）映射到右下角。

如果没指定 MOWSEEVENTF_ABSOLOTE，dX 和 dy 表示相对于上次鼠标事件产生的位置（即上次报告的位置）的移动。正值表示鼠标向右（或下）移动；负值表示鼠标向左（或上）移动。

鼠标的相对移动服从鼠标速度和加速度等级的设置，一个最终用户用鼠标控制面板应用程序来设置这

些值，应用程序用函数 SystemParameterslnfo 来取得和设置这些值。

在应用加速时系统对指定相对鼠标移动提供了两个测试。如果指定的沿 X 轴 y 轴的距离比第一个鼠标阈值大，并且鼠标的加速等级非零，则操作系统将距离加倍。如果指定的沿 X 轴或 y 轴的距离比第二个鼠标阈值大，并且鼠标的加速等级为 2，则操作系统将从第一个阈测试得来的距离加倍。这样就允许操作系统将指定鼠标沿 X 轴或 y 轴的相对位移加到 4 倍。

一旦应用了加速，系统用期望的鼠标速度换算合成的值。鼠标速度的范围是从 1（最慢）到 20（最快），并代表基于鼠标移动的距离指示符移动的数量。缺省值是 10，表示对鼠标的移动设有附加的修改。

函数 mouse_event 需要用的应用程序用来合成鼠标事件。也被应用程序用来取得鼠标位置和鼠标按键状态之外的鼠标信息。例如，如果输入板制造商想将基于画笔的信息传给自己的应用程序，可以写一个直接与输入板硬件通信的动态键接库（DLL），获得附加的信息，并保存到一个队列中。DLL 然后调用 mouse_event，用标准按键和 x/y 位置数据，并在参数 dwExtralnfo 设置排列的附加信息的指针或索引。当应用程序需要附加信息时，调用 DLL（连同存贮在 dwEXtralnfo 中的指针或索引），则 DLL 返回附加信息。

Windows CE：Windows CE 不支持参数 dwFlags 取 MOOSE EVENTF WHEEL 常数。

速查： Windows NT： 3.1 及以上版本； Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.15.6 ReleaseCapture

函数功能： 该函数从当前线程中的窗口释放鼠标捕获，并恢复通常的鼠标输入处理。捕获鼠标的窗口接收所有的鼠标输入（无论光标的位置在哪里），除非点击鼠标键时，光标热点在另一个线程的窗口中。

函数原型： BOOL ReleaseCapture（VOlD）

参数：无。

返回值： 如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetlastError 函数。

备注： 应用程序在调用函数 SetCaPture 之后调用此函数。

Windows 95：调用 ReleaseCapture 会引起失去鼠标捕获的窗日接收一个 WM_CAPTURECHANGED 消息。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本：Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：User32.lib。

2.15.7 SetCapture

函数功能： 该函数在属于当前线程的指定窗口里设置鼠标捕获。一旦窗口捕获了鼠标，所有鼠标输入都针对该窗口，无论光标是否在窗口的边界内。同一时刻只能有一个窗口捕获鼠标。如果鼠标光标在另一个线程创建的窗口上，只有当鼠标键按下时系统才将鼠标输入指向指定的窗口。

函数原型： HWND SetCapture（HWND hwnd）;

参数：

hWnd：当前线程里要捕获鼠标的窗口句柄。

返回值： 返回值是上次捕获鼠标的窗口句柄。如果不存在那样的句柄，返回值是 NULL。

备注： 只有前台窗口才能捕获鼠标。如果一个后台窗口想捕获鼠标，则该窗口仅为其光标热点在该窗口可见部份的鼠标事件接收消息。另外，即使前台窗口已捕获了鼠标，用户也可点击该窗口，将其调入前台。当一个窗日不再需要所有的鼠标输入时，创建该窗口的线程应当调用函数 ReleaseCapture 来释放鼠标。此函数不能被用来捕获另一进程的鼠标输入。

Windows 95：调用 SetCaptune 会引起失去鼠标捕获的窗口接收一个 WM_CAPTURECHANGED 消息。

速查：头文件：Winuser.h；输入库：user32.lib。

2.15.8 SetDoubleCllckTime

函数功能：该函数为鼠标设置双击时间。

函数原型：BOOL SetDoubleCIckTime（UINT ulnterval);

参数：

ulnterval：指定在双击中第一次和第二次点击之间的毫秒数。如果此参数设置为零则系统使用缺省的双击时间，即 500 毫秒。

返回值：如果函数调用成功，返回非零值。如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：函数 SetDoubleClickTime 为系统中所有的窗口修改双击时间。

速查：Windows NT 3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.15.9 SwapMouseButton

函数功能：该函数反转或恢复鼠标左右键的含义

函数原型：BOOL SwapMouseButton（BOOL fSwap);

参数：

fSwap：指定鼠标键的含义是否被反转或恢复。如果此参数为 TRUE，则左键产生右键消息而右键产生左键消息，如果此参数为 FALSE，则恢复鼠标键的最初含义。

返回值：如果在函数调用之前鼠标键的含义已被反转，则返回非零值。如果鼠标键的含义没反转，返回值是零。

备注：鼠标键交换是为给那些用左手操作鼠标的人提供方便。此函数通常只能由控制板调用。尽管一个应用程序能够自由地调用此函数，但鼠标是一种共享资源，其键的含义反转会影响所有应用程序。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib。

2.15.10 TrackMouseEvent

函数功能：当在指定时间内鼠标指针离开或盘旋在一个窗口上时，此函数寄送消息。

函数原型：BOOL TrackMouseEvent（LPTRACKMOUSEEVENT lpEventTrack);

参数：

lpEventTrack；指向结构 TRACKMOUSEEVENT 的指针。

返回值：如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数。

此函数能寄送如下消息：

WM_MOUSEHOVER：在上次调用 TrackMouseEvent 指定的时间里，鼠标盘旋在窗口的客户区。当此消息产生时，盘旋跟踪停止。如果需要进一步的鼠标盘旋跟踪，应用程序应当再次调用 TrackMouseEvent。

WM_MOUSELEAVE：鼠标离开上次调用 TrackMouseEvent 时指定的窗口客户区。当此消息产生时，所有由

TrackMouseEvent 要求的跟踪都被取消。当鼠标再次进入窗口，并且要求进一步的鼠标盘旋跟踪时，应用程序必须调用 TrackMouseEvent。

备注： 当鼠标指针在指定时间内停留在指定矩形内，就被认为是处于盘旋状态。调用函数 SystemParameterslnfo 并使用 SPI_GETMOUSEAOVERWIDTH，SPI_GETMOUSEHOVERAEIGHT 和 SFI_GETMOOSEAOVERTIME 值来取得矩形的大小和时间。

速查： Windows NT 4.0 及以上版本；Windows 98 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；输入库：user32.lib。

2.16 多文档接口函数（Multiple Document Interface）

2.16.1 CreateMDIWindow

函数功能： 该函数创建一个多文档接口 MDI 的子窗口。

函数原型：HWND GreateMDIWindow(LPTSTR IpClassName,LPTSTR IpWindowName,LPTSTR IpWindowName，DWORD dwStyle，int X，int Y，intn Width，int nHeight，HWND hWndParent，HINSTANCE hlnstance，LPARAM IParam)；

参数：

lpClassName：以“\0”为结尾的字符串指针，该字符串指定 MDI 子窗口的窗口类。该类名必须已通过调用 RegisterClassEx 函数注册过。

lpWindowsName：以“＼0”为结尾的字符串指针，该字符串表示窗口的名字。系统在子窗口的标题条中显示此名字。

dwStyle：规定 MDI 子窗口形式。如果 MDI 客户窗口是以 MDIS-ALLCHILDSTYLES 窗日形式创建的，这个参数可以是在 Createwindow 函数描述中列出的窗口形式的任何组合；否则，这个参数必须取下列值之一或多个组合：

WS_MINIMIZE：创建一个初始状态为极小化的 MDI 子窗口。

WS_MAXIMIZE：创建一个初始状态为极大化的 MDI 子窗口。

WS_HSCROLL：创建一个带有水平流动条的 MDI 子窗口。

WS_VSCROLL：创建一个带有垂直流动条的 MDI 子窗口。

X：指定 MDI 子窗口在客户坐标系中水平位置的初值。如果此参数值为 CW_USEDEFAULT，MDI 子窗口被分配为水平位置的缺省值。

Y：指定 MDI 子窗口在客户坐标系中垂直位置的初值。如果此参数为 CW_USEDEFAULT，MDI 子窗口被分配为垂直位置的缺省值。

nWidth：指定 MDI 子窗口的初始宽度，单位为设备单位。如果此参数值为 CW_USEDEFAULT，MDI 子窗口被分配为缺省宽度。

nHeight：指定 MDI 子窗口的初始高度，单位为设备单位。如果此参数值为 CW_USEDEFAULT，MDI 子窗口被分配为缺省高度。

hWndParent：指向 MDI 客户窗口的句柄，该窗口为新的 MDI 子窗口的父窗口。

hlnstance：指向创建 MDI 子窗口的应用事例的句柄。

lParam：指定一个应用程序定义的值。

返回值： 如果函数调用成功，返回值为所创建窗口的句柄；否则，返回值为 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

备注： 使用 CreateMDIWindow 函数与发送 WM_MDCREATE 消息给一个 MDI 客户窗日相似，区别是函数可以在不同的线程中创建一个 MDI 子窗口，而消息不可以。

Windows 95。系统最多可以支持 16，364 个窗口句柄。

速查：Windows NT：3.1 及以上版本；Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;输入库：user32.lib；Unicoae：在 Windows NT 环境中以 Unicode 和 ANSI 方式实现。

2.16.2 DefFrameProc

函数功能：该函数对任何多文档接口（MDI）框架窗日的窗口过程不处理的窗口消息提供缺省处理。窗口过程不能明确处理的所有窗口消息必须通过 DefFrameProc 函数处理，而不是通过 DefWindowProc 函数。

函数原型：LRESULT DefFrameProc（HWND hWnd，HWND hWnd，HWND hWndMDIClient，UINT uMsg，WPARAM wParam，LPARAM IParam)；

参数：

hWnd：MDI 框架窗口句柄。

hWndMDIClient：MDI 客户窗口句柄。

uMsg：指定要处理的消息。

wParam：指定附加的特定消息信息。

IParam：指定附加的特定消息信息。

返回值：返回值指定，消息处理的结果其值与处理的消息有关。如果 hWndMDIClient 参数为 NULL，返回值与 DefWindowProc 函数的相同。

备注：当应用程序的窗口过程不能处理一个消息时，它把消息传递给 DefWindowProc 函数来处理。MDI 应用程序使用 DefFrameProc 和 DefMDIChildProc 函数代替 DefWindowProc 函数提供缺省消息处理。应用程序传递给 DefMDIChildProc 函数的所有消息（例如非客户消息和 WM_SETTEXT 消息）通常应被传递给 DefFrameFro 函数。DefFramePro 函数也处理下列消息：

WM_COMMAND：激活用户选择的 MDI 子窗口。当用户从 MDI 框架窗口的菜单中选择 MDI 子窗口时，此消息被发送伴随该消息的窗口标识符识别被激活的 MDI 子窗口。

WM_MENUCHAR：当用户按下 Alt＋C（减）组合键时，打开活动 MDI 子窗口的窗口菜单。

WM_SETFOCUS：传递键盘响应给 MDI 客户窗口，客户窗口又依次地把它传递给活动的 MDI 子窗口。

WM_SIZE：重新设备新的框架窗口客户域的 MDI 客户窗口大小。如果框架窗口过程设置了不同大小的 MDI 客户窗口，消息将不传递给 DefWindowProc 函数。

速查：Windows NT:3.1 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件;winuser.h;输入库：user32。lib；Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 方式实现。

2.16.3 DefMDIChildProc

函数功能：该函数对任何多文档接口（MDI）子窗口的窗口过程不能处理的窗口消息提供缺省处理。窗口过程不能处理的窗口消息必须传递给 DefMDIChildProc 函数，而不是 DefWindowProc 函数。

函数原型：LRESULT DDMDIChildProc（HWND hwnd，UINT uMsg，WPARAM wParam，LPARAM IParam）

参数：

hWnd：MDI 子窗口句柄。

wParam：指定要处理的消息。

wParam：指定附加的特定消息信息。

IParam：指定附加的特定消息。

返回值：返回值指定信息处理的结果和其值与处理的消息有关。

备注：DefMDIChifdProc 函数假定由 hwnd 参数所识别的 MDI 子窗口的父窗口是由 MDICLIENT 类生成的。

当应用程序的窗口过程不能处理一个消息时，它把消息传递给 DefWindowProc 函数来处理，MDI 应用程序使用 DefFrameProc 和 DefMDfChildProc 函数来代替 DefWindowProc 函数提供缺省消息处理。应用程序传递给 DefwindowProc 函数的所有消息（例如非客户消息和 WMSET_SETTEXT 消息）通常都应传递给 DefMDIChildProc 函数。另外 DefMDIChildProc 函数也能处理下列消息：

WM_CHILDACTIVATE：当 MDI 子窗口被改变大小，移动或显示时执行激活过程。这个消息必须被传递。

WM_GETMINMAXINFO：根据 MDI 客户窗口的当前大小，计算 MDI 子窗口极大化的尺寸。

WM_MENUCHAR：传递消息给 MDI 框架窗口。

WM_MOVE：重新计算 MDI 客户滚动条，如果存在的话。

WM_SETFOCUS：如果子窗口不是活动的 MDI 子窗口，激活它。

WM_SIZE；执行改变窗口大小所必须的操作，特别是 MDI 子窗口极大化或恢复一个 MDI 子窗口时。如果这个消息没有成功地传递给 DefMDIChildProc 函数，则很可能产生不是需要的结果。

WM_SYSCOMMAND:处理窗口菜单命令:SC.NEXTWINDOW,SC_PREVWINDOW,SC_MOVE,SC_SIZE,和 SC_MAXIMIZE。

速查：Window NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；输入库：user32.lib；Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 方式实现。

2.16.4 TranslateMdISysAccel

函数功能：该函数处理与指定 MDI 客户窗口相联系的多文档接口（MDI）子窗口的菜单命令的加速键响应。该函数转换 WM_KEYUP 和 WM_KEYDOWN 消息为 WM_SYSCOMMAND 消息，并把它的发送给相应 MDI 子窗口。

函数原型：BOOL TranslateMDlSysAccel（HWND hWndClient，LPMSG IpMsg）；

参数：

hWndClient：MDI 客户窗口句柄。

lpMsg：由 GetMessage 或 PeedMessage 函数检索到的消息的指针，这个消息必须为 MSG 结构，并包含来自应用程序消息队列的消息信息。

返回值：如果消息被转换为系统命令，则返回值为非零值。如果消息未转换为系统命令，则返回值为 0。

速查：Windows NT：3.1 及以上版本;Windows:95 及以上版本:Windows CE:不支持;头文件:winuser.h；

输入库：user32.lib。

2.17 资源函数（Resource）

2.17.1 BeginUpdateResource

函数功能：该函数返回一个可被 UpdateResource 函数使用的句柄以便在一个可执行文件中增加、删除或替换资源。

函数原型：HANDLE BeginUpdateResource（LPCTSTR pFileName，BOOL bDeleteExistingResources）；

参数：

pFileName：指向一个表示结束的空字符串指针，它是用来指定用以更新资源的基于 32-位可执行文件的文件名。应用程序必须获得访问这个文件的可写权限，并且此文件在当前状态下不能被执行。如果 pFileName 未被指定完全路径，系统将在当前路径下搜寻此文件。

bDeleteExistingResources：说明是否删除 PFileName 参数指定的现有资源。如果这个参数为 TRUE 则现有的资源将被删除,而更新可执行文件只包括由 UpdateResource 函数增加的资源。如果这个参数为 FALSE，

则更新的可执行文件包括现有的全部资源，除非通过 UpdateResource 特别说明被删除或是替换的。

返回值：如果此函数运行成功，其值将通过使用 UpdateResource 和 EndUpdateResource 函数返回一个句柄。如果被指定的文件不是一个可执行文件，或者可执行文件已被装载，或者文件不存在，或是文件不能被打开写入时，则返回值为空。若想获得更多的错误信息，请调用 GetLastError 函数。

速查:Windows 3.1 以上。头文件：winbase.h；库文件：kernel32.lib，Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.2 CopyImage

函数功能：该函数创建一个新的图像（图标、光标、位图）并复制该指定图像的属性到新的图像。若有必要，该函数将伸展位值以填满新图像所需要的尺寸。

函数原型：HANDLE CopyImage（HANDLE hlmage，UjNT uTyPe，int cxDesired，int cyDesired，UINT fuFlags）

参数：

hImage：指向包含将被复制图像的模型中的一个特例的句柄。

uType：说明被复制图像的类型，此参数将可能是如下值：

IMAGE_BITMAP：表示复制一个位图；IMAGE_CURSOR；表示复制一个光标。

IMAGE_ICON：表示复制一个图标。

cxDesired：用来指定图像所需的像素宽度。

cyDesired：用来指定图像所需的像素高度。

fuFlags：指定下列复合值，其含义具体如下：

LR_COPYDELETEORG：表示创建一个副本后删除原始图像。

LR_COPYRETURNORG；表示创建一个图像的精确副本，而忽略参数 cxDesired 和 cyDesired。

LR_MONOCHROME：表示创建一个新的单色图像。

LR_COPYFROMRESOURCE；表示试图从原始资源文件中再装载图标或光标资源而不是简单的复制当前图像。这使得在含有多种尺寸资源的资源文件中再创建一个不同尺寸的副本时非常有用。若没有这个标志，Copylmage 函数将伸展原始图像到新的尺寸；若此标志被设置，Copylmage 函数将在资源文件中选择使用最接近所期待尺寸的值。

此函数只有在 Loadlcon、LoadCursor 或 Loadlmage 函数中的 hlmage 参数被装载成 LR_SHARED 值时才运

行成功的。

返回值：如果函数运行成功，其值将返回最新创建图像的句柄；如果函数运行失败，其值将返回空。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：当使用完资源后，可以调用下表中列举的函数以释放相关内存。

Resource Release function 资源释放函数：Bitmap DeleteObject 位图：DeleteObject；

Cursor DestroyCursor 光标：DestroyCursorr；lcon Destroylcon 图标： Destroylcon。

当过程终止时，系统将自动删除这些资源。因而，调用相关函数可以节省内存空间且减少过程工作设置所需空间的大小。

速查：Windows NT Windows 95 以上，头文件：winuserh；库文件：user32.lib。

2.17.3 EndUpdateResource

函数功能：该函数终止在可执行文件中的资源更新。

函数原型： BOOL EndUpdateResource（HANDLE hUpdate，BOOL fDiscard）；

参数：

hUpdate：用于资源更新的句柄。此句柄通过 BeginUpdateResource 函数返回。

fDiscard：用来说明是否向可执行文件中写入资源更新内容。如果此参数为 TRUE，则在可执行文件中无变化；如果此参数为 FALSE，则在可执行文件中写入变化。

返回值： 如果函数运行成功，并且通过调用 UpdateResource 函数指定的不断积聚的资源修正内容被写入指定的可执行文件，那么其返回值为非零。如果函数运行失败，其返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

速查： Windows NT 3.1 以上，头文件：winbase.h；库文件：kernel32.lib，Unicode：在 Windows NT 上实现为 Unicode 和 ANSl 两种版本。

2.17.4 EnumResLangProc

函数功能： 该函数是一个用户定义的和 EnumResourceLanguages 函数一起使用的回调函数。它接收资源语言。ENUMRESLANGPROC 类型定义指向该响应函数的指针。EnumResLangProc 是用户定义的函数名称的占位符。

函数原型： BOOL CALLBACK EnumResLangProc（HANDLE hModule，LPCTSTR IPszType， LPCTSTR IpszName，WORDwlDLanguage，LONG IParam）；

参数：

hModule：处理那些包含着被列举术语资源的可执行文件的模块。如果这个参数为空，函数将在模块中列出在建当前过程时所用的资源术语。

lpszType：指向一个以 NULL 为结束符的字符串，字符串指定了那些被列举的语句资源的类型名。作为标准的资源类型，此参数可以为下列值，含义如下：

RT_ACCELERATOR：加速器表；RT_ANICURSOR：动态光标；

RT_ANIICON：动态图标；RT_BITMAP：位图资源；

RT_CURSOR：由硬件支持的光标资源；DI_DIALOG：对话框：

RT_FONT：字体资源；RT_FONTDIR；字体目录资源；

RT_GROUP_CURSOR：与硬件无关的光标资源；

RT_GROUP_ICON：与硬件无关的目标资源；

RT_HTML：HTML 文档；RT_ICON：由硬件支持的图标资源：

RT_MENU：菜单资源；RT_MESSAGETABLE：消息表的入口；

RT_PLUGPLAY：即插即用资源；

RT_RCDATA：应用定义资源（原始数据）：RT_STRING 字符表入口；

RT_VERSION：版本资源：RT_VXD：VXD。

lpszName：指向一个以 NULL 为结束符的字符串，字符串说明了在资源中被列举出术语的名称。

wlDLanguage：指定了在资源中被列举出语句的标识符。EnumResourceLanguages 函数提供了这一值。有关原始的语句标识符和由标识符组成子句标识符的列表可以详看 MAKELANGID。

IParam：指定将应用定义的参数传递给 EnumResourceLanguages 函数，此参数将被用于错误检查。

返回值： 此响应函数返回值为 TRUE 时将继续列举：否则，当返回值为 FALSE 时将停止列举。

注意：应用程序必须通过向 EnumResourceLanguages 函数传递这个函数的地址来注册。

速查： Windows NT 3.1、Windows 95 以上，头文件：winbase.h；库文件：由用户定义。

2.17.5 EnumResNameProc

函数功能：该函数是一个用户定义的和 EnumResourceNames 函数一起使用的回调函数。ENUMRESNAMEPROC 类型定义一个指向该回调函数的指针。EnumResNameProc 是用户定义函数名的占位符。

函数原型：BOOL CALLBACK EnumResNameProc（HANDL hModue.LPCTSTR haszTyPe，LPTSTR IPszName，LONGIParam)；

参数:

hModule：处理包含被列举资源文件名的可执行文件的模块。如果这个参数为 NULL，那么函数将在模块中列举出创建当前过程的资源名称。

lpszType：指向以 NULL 为结束符的字符串，它指定了被列举出的资源类型名称。作为标准类型，这个参数的含义同 EnumResLangProc\lpszType。

lpszName：指向以 NULL 为结束符的字符串，它指定了被列举出的资源名称。

IParam：指定传递给 EnumResourceNames 函数的应用程序定义的参数，此参数用于错误检查。

返回值：此响应函数返回值为 TRUE 时将继续列举；否则，当返回值为 FALSE 时将停止列举。

注意：应用程序必须通过向 EnumResourceNames 函数传递这个函数的地址来注册。

速查：Windows NT 3.1、Windows 95 以上，头文件：winbase.h；库文件：由用户定义。

2.17.6 EnumResourceLanguages

函数功能：该函数为每个指定类型和名称的资源搜索模块，并且向定义的回调函数传递所搜寻到的每种资源语言。

函数原型：BOOL EnumResourceLanguages（HMODULE hModule，LPCTSTR lpType，LPCTSTR IpName，ENUMREShNGPROC lpEnumFunc，LONG IParam)；

参数：

hModule：处理包含被列举语言资源的可执行文件的模块。如果这个参数为 NULL，那么函数将在模块中列举出创建当前过程的语言资源。

lpType：指向以 NULL 为结束符的字符串，它指定了被列举出的语言资源类型。作为标准类型，这个参数的含义同 EnumResLangProc\lpType。

IpName：指向以 NULL 为结束符的字符串，它指定了被列举出的语言资源名称。

IpEnumFunc：指向所需要每个列举出的语言资源的响应函数。如要了解更多的信息请参见 EnumResLangProc。

IParam：指定一个申请定义参数值传递给响应函数，此参数可以用来错误检查。

返回值：若函数运行成功，则返回非零值；若函数运行失败，则返回零值。若想获得更多错误信息，请调用 GetLastError 函数。

注意：EnumResourceLanguages 函数将连续列举出语言资源，直到响应函数返回 False 值或全部的语言资源均被列举完毕。

速查：Windows NT 3.1、 Windows 95 以上，头文件：winbase.h；库文件：kernel32.lib；Unicode：在 Windows 和 Winaows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.7 EnumResourceNames

函数功能：该函数为每个指定类型的资源搜寻模块，并将每个查找到的资源名称传递给回调函数。

函数原型： BOOL EnumResourceNames（HINSTANCE hModue，LPCTSTR lpszType，ENUMRESNAMEPROC lpEnumFunc，LONG IParam）；**参数：**

hModule：处理包含被列举资源名称的可执行文件的模块。如果这个参数为 NULL，那么函数将在模块中列举出创建当前过程的资源名称。

lpszType：指向以 NULL 为结束符的字符串，它指定了被列举出的资源类型名称。作为标准类型，这个参数的含义同 EnumResLangProc\lpszType。

lpEnumFunc：指向所需要每个列举出的资源名称的响应函数。如要了解更多的信息请参见 EnumResNameProc。

IParam：指定一个申请定义参数值传递给响应函数，此参数可以用来错误检查。

返回值：若函数运行成功，则退回非零值；若函数运行失败，则退回零值。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：函数将连续列举出资源名称，直到响应函数返回 False 值或全部的资源名称均被列举完毕。

速查： Windows NT 3.1、 Windows 95 以上，头文件：winbase.h；库文件： kernel32.lib；Unicode：在 Windows 和 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.8 EnumResourceTypes

函数功能： 该函数为资源搜寻模块并且将它找到的每个资源类型传递给用户定义的回调函数。

函数原型：BOOL EnumResourceTypes（HMODULE hModule，ENUMRESTYPEPROC lpEnumFunc，LONG IParam）；

参数：

hModule：处理包含被列举资源类型的可执行文件的模块。如果这个参数为 NULL，那么函数将在模块中列举出创建当前过程的资源类型。

lpEnumFunc：指向所需要每个列举出的资源类型的响应函数。如要了解更多的信息请参见 EhumResNameProc。

IParam：指定申请定义值传递给响应函数。

返回值：若函数运行成功，则返回非零值；若函数运行失败，则返回零值。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：函数将连续列举出资源名称，直到响应函数返回 False 值或全部的资源名称均被列举完毕。

速查：Windows NT 3.1、Windows 95 以上，头文件：winbase.h；库文件：kernel32.lib；Unicode：在 Windows 和 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.9 EnumResTypeProc

函数功能： 该函数是一个用户定义的和 EnumResourceTypes 函数一起使用的回调函数。它接收资源类型。EnumResTypeProc 类型定义一个指向这个回调函数的指针。EnumResTypeProc 是这个用户定义函数名称的占位符。

函数原型：BOOL CALLBACK EnumResTyPeProc（HANDLE hModue，LPTSTR lpszType，LONG IParam）；

参数：

hModule：处理包含被列举资源类型的可执行文件的模块。如果这个参数为 NULL，那么函数将在模块中列举出创建当前过程的资源类型。

IpszType：指向以 NULL 为结束符的字符串，它指定了被列举出的资源类型。作为标准类型，这个参数的含义同 EnumResLangProC＼lpszType。

IParam：指定一个申请定义参数值传递给 EnumResourceTypes 函数，此参数可以用来错误检查。

返回值：若响应函数返回 TRUE 值则继续列举；否则，返回 FALSE 值时将停止列举。

注意：应用程序必须通过将函数的地址传递给 EnumResourceTypes 函数来注册。

速查：Windows NT 3.1、Windows 95 以上，头文件：winbase.h；库文件：由用户定义。

2.17.10 FlndResource

函数功能：该函数确定指定模块中指定类型和名称的资源所在位置。

函数原型：HRSRC FindResource（HMODULE hModule，LPCTSTR lpName，LPCTSTR lpType）；

参数：

hModule：处理包含资源的可执行文件的模块。NULL 值则指定模块句柄指向操作系统通常情况下创建最近过程的相关位图文件。

lpName：指定资源名称。若想了解更多的信息，请参见注意部分。

lpType：指定资源类型。若想了解更多的信息，请参见注意部分。作为标准资源类型。这个参数的含义同 EnumResLangProc\lpType。

返回值：如果函数运行成功，那么返回值为指向被指定资源信息块的句柄。为了获得这些资源，将这个句柄传递给 LoadResource 函数。如果函数运行失败，则返回值为 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

注意：如果参数 lpType 或 lpName 的高字节为 0，那么其低字节中所给定的资源的类型或名称标识说明。另外，这些参数指向以 NULL 为终止符的字符串。字符串的第一个字符是＃，后面的字符表示十进制数来表示源类型或名称的整数标识符。例如。字符串"#258"表示整数标识符 258。

如果用整数标识符替代名称提交资源，用程序将减少所需的内存容量。

当使用完加速器表，位图，光标，图标，或是菜单后，可以通过调用下表所列举的函数释放内存。加速器表：DestroyAcceleratorTable；位图：DeleteObject；光标：DestroyCursor；图标：Destroylcon；菜单 DestroyMenu。

当过程创建资源终止时，系统将自动删除这些资源。然而通过调用适当的函数可以保留内存，减少过程中工作设置所需的空间大小。

应用程序可以使用 FindResource 函数去查找任何种类资源，但是这个函数只有在应用程序并发调用 LoadLibray 和 LockResource 函数来存取二进制资源时才被使用。

如果想立即使用某一资源，应用程序将使用下面详细资源函数列表中某一函数去查找装载所需资源，FormatMessage：装裁且格式化信息表接口：LoadAccclerators：装载加速器表；LoadBltmap：装载位图资源；LoadCursor：装载光标资源；Loadlcon：装载图标资源；

LoadMenu：装载菜单资源；LoadString：装载字符串表资源。

例如，应用程序可以使用 Loadlcon 函数装载某一图标以在屏幕上显示。但是，如果是装载某一图标为了将它的数据复制到另一个应用程序中，那么这个应用程序就应该使用 FindResource 和 LoadResource 函数。

字符串资源存储在由若干部分组成的某区域，每部分有 16 个字符串。每部分的字符串是按统一计数方式排列的有序队形式存储的。

TheLoadstring 函数将从相应的区域中摘取字符串资源。

参数 hModule 不能为 NULL 句柄。

参数 lpName 不支持 ID 为零的资源，即 FindResource（h，o，t）不能按预期工作。

不可能在 IpTyPe 参数中传递 RT_ANICURSOR 或 RT_AN1lCON 的值。

速查：Windows NT3.1、Windows95、Windows CE1.0 以上，头文件：winbase.h；库文件：kerne132.lib；Unicode；在 Windows 和 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.11 FindResourceEx

函数功能：该函数确定指定模块中指定类型、名称及语言的资源所在位置。

函数原型：HRSRC FindResourceEx（HXODULE hModule，LPCTSTR lpType，LPCTSTR lpName，WORD wLanguage）；

参数：

hModule：处理包含资源的可执行文件的模块。如果参数值为 NULL，则函数搜索曾经创建的最近过程的模块。

lpType：指向以 NULL 为结束符的字符串，它指定了被列举出的资源类型名称。若要了解更多的信息，请参见注意部分。作为标准类型，这个参数取值同 EnumResLangProc\lpType。

lpName：指向说明资源文件名称并以 NULL 为结束符的字符串。若要了解更多的信息，请参见注意部分。

WLanguage：指明语言资源。若此参数为 MAKELANGID（LANG_NEUTRAL，SUBLANG_NEUTRAL），则为了指定零一种语言，可以使用宏创建这个参数。更多的信息请参见 MAKELANGID。

返回值：如果函数运行成功，其返回值是一个指定资源信息块的句柄。为了获得资源，要将此句柄传递给 LoadResource 函数。如果函数运行失败，返回值为空。若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT 3.1、Windows 95 以上，头文件：winbase.h；库文件：kernel32.lib； Unicode：在 Windows 和 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.12 LoadImage

函数功能：该函数装载目标，光标，或位图。

函数原型：HANDLE LoadImage（NINSTANCE hinst，LPCTSTR lpszName，UINT uType，int cxDesired，int CyDesired，UINT fuLoad）；

参数：

hinst：处理包含被装载图像模块的特例。若要装载 OEM 图像，则设此参数值为 0。

lpszName：处理图像装载。如果参数 hinst 为非空，而且参数 fuLoad 不包括 LR_LOADFROMFILE 的值时，那么参数 lpszName 是一个指向保留在 hinst 模块中装载的图像资源名称，并以 NULL 为结束符的字符串。

如果参数 hinst 为空，并且 LR_LOADFROMFILE 被指定，那么这个参数低位字一定是被装载的 OEM 图像标识的。OEM 图像标识符是在 WINUSER.H 头文件中定义的，下面列举出前缀的含义：

OBM_ OEM：位图；OIC_OEM 图标；OCR_OEM：光标。

如果参数 fuLoad 包含 LR_LOADFROMFILE 值，那么参数 lpszName 是包含有图像的文件名。

uType：指定被装载图像类型。此参数可以为下列值，其含义如下：

IMAGE_BITMAP：装载位图；IMAGE_CURSOR：装载光标；IMAGE_ICON：装载图标。

cxDesired：指定图标或光标的宽度，以像素为单位。如果此参数为零并且参数 fuLoad 值为 LR_DEFAULTSIZE，那么函数使用 SM_CXICON 或 SM_CXCURSOR 系统公制值设定宽度；如果此参数为零并且值 LR_DEFAULTSIZE 没有被使用，那么函数使用目前的资源宽度。

cyDesired：指定图标或光标的高度，以像素为单位。如果此参数为零并且参数 fuLoad 值为 LR_DEFAULTSIZE，那么函数使用 SM_CXICON 或 SM_CXCURSOR 系统公制值设定高度；如果此参数为零并且值 LR_DEFAULTSIZE 没有被使用，那么函数使用目前的资源高度。

fuLoad：根据下面复合值列表指定函数值，值含义如下：

LR_DEFAULTCOLOR：缺省标志；它不作任何事情。它的含义是“无 LR_MONOCHROME”。

LR_CREATEDIBSECTION：当参数 uType 指定为 IMAGE_BITMAP 时，使得函数返回一个 DIB 部分位图，而

不是一个兼容的位图。这个标志在装载一个位图，而不是映射它的颜色到显示设备时非常有用。

LRDIFAULTSIZE：若 cxDesired 或 cyDesired 未被设为零，使用系统指定的公制值标识光标或图标的宽和高。如果这个参数不被设置且 cxDesired 或 cyDesired 被设为零，函数使用实际资源尺寸。如果资源包含多个图像，则使用第一个图像的大小。

LR_LOADFROMFILE：根据参数 lpszName 的值装载图像。若标记未被给定，lpszName 的值为资源名称。

LW_LOADMAP3DCOLORS：查找图像的颜色表并且按下面相应的 3D 颜色表的灰度进行替换。

颜色替代：Dk Gray RGB（128，128，128）COLOR_3DSHADOW；Gray RGB（192，192，192）COLOR_3DFACELt Gray RGB（223，223，223） COLOR_3DLIGHT LR_LOADTRANSPARENT；找到图像中的一个像素颜色值并且根据颜色表中系统的缺省颜色值替代其相应接口的值。图像中所有使用这种接口的像素的颜色都变为系统的缺省窗体颜色。此至仅用来申请相应的颜色表。

若 fuLoad 包括 LR_LOADTRANSPARENT 和 LR_LOADMAP3DCOLORS 两个值，则 LRLOADTRANSPARENT 优先。但是，颜色表接口由 COLOR_3DFACE 替代，而不是 COLOR_WINDOW。

LR_MONOCHROME：装载黑白图。

LR_SHARED：若图像将被多次装载则共享。如果 LR_SHARED 未被设置，则再向同一个资源第二次调用这个图像是就会再装载以便这个图像且返回不同的句柄。

不要对不同标准尺寸的图像使用 LR_SHARED，装载后可能会有改变，或是从文件中被装载。

Windows 95 和 Windows 98：函数根据缓存中被请求的资源名发现的第一个图像，不管被请求的大小。LR_VGACOLOR：使用 VGA 真彩色。

返回值：如果函数运行成功，返回值是相关资源的数据的句柄。如果函数运行失败，返回值为 NULL。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：当使用完资源后，必须通过调用函数以释放加速器表、位图、光标、图标以及菜单所占的内存资源;加速器表:DesteoyAcceleratorTable;位图:DeleteObject;光标:DestroyCursor;图标:Destroylcon;菜单：DestroyMenu

当过程创建终止时，系统将自动删除这些资源。但是调用相关函数也可以保留内存减少过程的工作设置所占空间。

Windows CE：对 IMAGE_BITMAP 来说，参数 cxDesred 和 cyDesred p 必须为零。Windows CE 不支持图表跳跃或闪烁。

参数 fuLoad 必须为（=LR_DEFAULTCOLOR）。

如果的目标平台不支持鼠标光标，可以指定在参数 cxDesred 和 cyDsired 的 SM_CXCURSOR 和 SM_CYCURSOR 的值，但不能指定参数 uType 中 IMAGE_CURSOR 的值。

如果目标平台支持鼠标光标，可以指定在参数 cxDesired 和 cyDesred 的 SM_CXCURSOR 和 SM_CYCURSOR 的值，也能指定参数 uType 中 IMAGE_CURSOR 的值。

速查：Windows NT 3.1、Windows 95、Windows CE 1.0 以上，头文件：minuser.h：库文件；user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.13 LoadResource

函数功能：该函数装载指定资源到全局存储器。

函数原型； HGLOSAL LoadResouare（HMODULE hModule，HRSRC hReslnfo)；

参数：

hModule：处理包合资源的可执行文件的模块句柄。若 hModule 为 NULL，系统从当前过程中的模块中装载资源。

hReslnfo：将被装载资源的句柄。它必须由函数 FirtdResource 或 FindResourceEx 创建。

返回值：如果函数运行成功，返回值是相关资源的数据的句柄。如果函数运行失败，返回值为 NULL。

若想获得更多的错误信息，请调用 GetLastError 函数。

注意:LoadResource 的返回类型是向后兼容的 HGLOBAL 型,而不是因为函数返回一个全局存储块句柄。不要传递这个句柄给函数 GlobalLock 或 GlobalFree。为了得到更多资源数据信息，请调用 LockResource 函数。

当使用完资源后，必须通过调用函数以释放加速器表、位图、光标、图标以及菜单所占的内存资源：加速器表：DestroyAcceleratorTable；位图：DeleteObject；光标：DestroyCursor；图标：Destroylcon；菜单：DestroyMenu。

当过程创建终止时，系统将自动删除这些资源。但是调用相关函数也可以保留内存减少过程的工作设置所占空间。

Windows CE：参数 hModule 必须为非空。

速查：Windows NT 3.1、Windows 95、Windows CE1.0 以上，头文件：winbase.h；库文件：kernel132.lib。

2.17.14 LockResource

函数功能： 该函数锁定内存中的指定资源。

函数原型： LPVOID LOCkResource（HGLOBAL hResDate）；

参数：

hResDate：被装载的资源的句柄。函数 LosdResource 可以返回这个句柄。

返回值： 如果被装载的资源被所住了，返回值是资源第一个字节的指针；否则为 NULL。

注意：通过使用函数 FindResource 或 FindResourceEx 返回句柄试图所住资源，不再工作。可以返回一个错误的数据和任意数据的指针。

当使用完资源后，必须通过调用函数以释放加速器表、位图、光标、目标以及某单所占的内存资源：加速器表：DestroyAcceleratorTable；位图：DeleteObject；光标：DestroyCursor；图标：Destroylcon；菜单：DestroyMenu。

当过程创建终止时，系统将自动删除这些资源。但是调用相关函数也可以保留内存减少过程的工作设置所占空间。

速查：Windows NT 3.1、Windows 95、Windows CE 1.0 以上，头文件：winbase.h；库文件：kerne132.lib。

2.17.15 SizeofResource

函数功能： 该函数返回指定资源字节数大小。

函数原型： DWORD SizeofResource（HMODULE hModule，HRSRC hReslnfo）；

参数：

hModule：包合资源的可执行文件模块的句柄。

hReslnfo：资源句柄。此句柄必须由函数 FindResource 或 FindResourceEx 来创建。

返回值： 如果函数运行成功，返回值资源的字节数。如果函数运行失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT 3.1、Windows 95、Windows CE 1.0 以上，头文件：winbase.h；库文件：kerne132.lib。

2.17.16 UpdateResource

函数功能： 该函数增加，删除，或替代某可执行文件中的资源。

函数原型：BOOL UPdateResource(HANDLE hUpdate,LPCTSTR lPTyPe,LPCTSTR IPName,WORD wLanguage, LPVOID lgData, DWORD cbData);

参数：

hUpdate：指定更新文件句柄。此句柄由 BeginUpdateResource 函数返回。

lpType：指向说明将被更新的资源类型的字符串，它以 NULL 为终止符。这个参数可以是一个通过宏 MAKENTRESOURCE 传递的整数值，含义参见 EnumResLangProc＼lpType。

lpName：指向说明待被更新的资源名称的字符串，它以 NULL 为终止符。这个参数可以是一个通过宏 MAKEINTRESOURCE 传递的整数值。

wLanguage：指定将被更新资源的语言标识。要了解基本的语言标识符以及由这些标识符组成的字语言标识符的列表，可参见宏 MAKELANGID。

lpData：指向被插入可执行文件的资源数据的指针。如果资源是预定义类型值之一，那么数据必须是有效且适当排列的。注意这是存储在可执行文件中原始的一进制数据，而不是由 Loadlcon，LoadString 或其他装载特殊资源函数提供的数据。所有包含字符串、文本的数据必须是 Unicode 格式；IpData 不能指向 ANSI 数据。

如果 lpData 为 NULL，所指定的资源将从可执行文件中被删除。

cbData：指定 lpData 中的资源数据数据大小，以字节计数。

返回值：如果函数运行成功，返回值为非零；如果函数运行失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：应用程序重复使用 UpdateResource 去改变资源数据。每次 UpdateResource 调用都要占用系统内部的一个增加、删除、替代的列表，而实际上并没有将数据写到可执行文件中。应用程序必须通过使用 EndUpdateResource 函数将每次积累的变化写入可执行文件中。

速查：Windows NT 3.1 以上，头文件：winbase.h；库文件：kerne132.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.17.17 FreeResource

函数功能：该函数已过时。它为 16 位的基于 Win32 的应用程序提供了一个简单的端口。对于 Win32 应用程序（32 位）没有必要释放用 LoadResource 函数装载资源。由 LoadResource 所获得的资源随着被装载模块的卸载自动被释放。但是，为了保留内存减少的程序工作设置所占空间大小，必须通过调用下列表中的函数以释放加速器表、位图、光标、图标以及菜单所占的内存资源。

加速器表：DestroyAcceleratorTable；位图：DeleteObject；光标：DestroyCursor；

图标：Destroylcon；菜单：DestroyMenu。

2.17.18 UnlockResource

函数功能能：该函数已过时。这个函数仅兼容于 16 位的 Windows，对于 32 位的应用程序不必要解锁资源。

2.18 滚动条函数（Scroll Bar）

2.18.1 EnableScrollBar

函数功能：该函数可以激活一个或两个滚动条箭头或是使其失效。

函数原型：BOOL EnableScrollBar（HWND hWnd，UINT WSBflags，UINT wArrows）；

参数：

hWnd：根据参数 wSBflags 的值，处理对雷体或滚动条的处理。

wSBflags：指定滚动条的类型。这个参数可以是下面的值，含义如下：

SB_BOTH：可以将指定窗体的水平及垂直滚动条上的箭头激活或使其失效。此时参数 hWnd 一定指向窗体。

SB_CTL：标识滚动条控制器。此时参数必须指向滚动条控制器。

SB_HORZ：可以将指定窗体的水平滚动条上的箭头激活或使其失效。此时参数 hWnd 一定指向窗体。

SB_VERT：可以将指定窗体的垂直滚动条上的箭头激活或使其失效。此时参数 hWnd 一定指向窗体。

wArrows：指定滚动条上的箭头是否被激活或是无效，并指出哪一个箭头位有效或是无效。这个参数可以是下面的值，其含义如下：

ESB_DISABLE_BOTH：使滚动条上两面的箭头功能无效。

ESB_DISABLE_DOWN：使垂直滚动条上下面的箭头功能无效。

ESB_DISABLE_LEFT：使水平滚动条上左边箭头功能无效。

ESB_DISABLE_LTUP：使水平滚动条上左边箭头或垂直滚动条上面的箭头功能无效。

ESB_DISABLE_RLGHT：使水平滚动条上右边箭头功能无效。

ESB_DISABLE_RTDN：使水平滚动条上右边箭头或垂直滚动条下边的箭头功能无效。

ESB_DISABLE_UP：使垂直滚动条上向下箭头功能无效。

ESB_ENABLE_BOTH：激活滚动条两面的箭头。

返回值：如果被指定的箭头被激活或未被激活，其返回值为非零；如果箭头已经是被请求的状态或是出现错误，那么返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT 3.1、Windows 95 以上，头文件：winuser.h；库文件：user32.lib。

2.18.2 GetScrolllnfo

函数功能：该函数找到滚动条的参数，包括滚动条位置的最小值、最大值，页面大小，滚动按钮的位置，

函数原型：BOOL GetScrolllnfo（HWND hWnd，int fnBar，LPSCROLLINFO lpsi）；

参数：

hWnd：滚动条控制或有标准滚动条的窗体句柄，由 fnBar 参数确定。

fnBar：指定待找回滚动条参数的类型，此参数可以为如下值，其值含义：

SB_CTL：找回滚动条控制参数。其中参数 hwnd 一定是处理滚动条控制的句柄。

SB_HORZ：找回所指定窗体的标准水平滚动条参数。

SB_VERT：找回所指定窗体的标准垂直滚动条参数。

lpsi：指向 SCROLLINFO 结构。在调用 Getscrolllofo 函数之前，设置 SCROLLINFO 结构中 cbSize 成员以标识结构大小，设置成员 fMask 以说明待找回的滚动条参数。在运行之前，函数复制结构中适当的成员所指定的参数。

成员 fMask 可以是如下值：

SIF_PAGE：复制滚动页码到由 lpsi 指向的 SCROLLINFO 结构的 nPage 成员中。

SIF_POS：复制滚动位置到由 lpsi 指向的 SCROLLINFO 结构的 nPos 成员中。

SIF_RANGE：复制滚动范围到由 lpsi 指向的 SCROLLINFO 结构的 nMin 和 nMax 成员中。

SIF_TRACKPOS：复制当前滚动盒跟踪位置到由 nTrackPos 指向的 SCROLLINFO 结构的 nPage 成员中。

返回值：如果函数找到任何一个值，那么返回值为非零；如果函数没有找到任何值，那么返回值为零；若要得到更多出错信息，请调用 GetLastError 函数。

注意：Getscrolllnfo 函数尽管 WM_HSCROLL 和 WM_VSCROLL 指出了滚动条位置消息，却仅提供了 16 位数据，而函数 SetScrollnfo 和 GetScrollnfo 则提供了 32 位的滚动条数据。因而，当应用程序在处理 WM_HSCROLL 或 WM_VSCROLL 时，要获得 32 位滚动条位置的数据时，则要调用 Getscrolllnfo 函数。

在 WM_HSCROLL 或 WM_VSCROLL 消息中 SB_THUMBTRACK 通告过程中，为了获得 32 位的滚动盒位置，需要调用 GetScrolllnfo 函数以得到结构 SCROLLINFO 成员 fMask 中的 SCROLLINFO 值。函数返回在结构 SCROLLINFO 成员 nTrackPos 中指出的滚动盒跟踪位置的值。这将允许当用户移动滚动盒时能得到其位置。

速查：Windows NT3.51、Windows 95、Windows CE1.0 以上，头文件：winuser.h；库文件：user32.lib。

2.18.3 ScrollDC

函数功能：该函数水平和垂直滚动一个位矩形。

函数原型：BOOL ScrollDC（HDC hDC，int dx，int dy，CONST RECT *IprcScroll，CONST*lprcClip，HRGN hrgnUpdateLPRECT IprcUpdate）；

参数：

hDC：含有要滚动位数的设备描述表句柄。

dx：在设备单元中，指定水平滚动数量。在向左滚动时此参数必须为负。

dy：在设备单元中，指定垂直滚动数量。在向上滚动时此参数必须为负。

lprcScroll：指向包含与滚动矩形的同等之物的 RECT 结构。

lprcClip：指向包含类似于剪下矩形之物的 RECT 结构。只有在剪辑矩形内部的图案才受影响。

hrgnUpdate：处理滚动过程中位覆盖的区域。ScrollDC 定义这个区域，它不一定是一个矩形。

lprcUpdate：指向 RECT 结构，它接收到类似于被限制滚动更新区域矩形之物。这是所需求重画的最大的矩形区域。当函数返回时，不管指定设备内容映射的模式如何，结构中的值将在客户端对应结构中。

返回值：如果函数运行成功，返回值为非零；如果函数运行失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：如果参数 lprcUpdate 为空，系统将不再计算更新矩形。如果参数 hrgnUpdate 和 IprcUpdate 均为空，系统将不再计算更新区域。如果参数 hrgnUpdate 不为空，系统将好像拥有包含一个有效的未被滚动过程覆盖区域句柄（由 ScrollDC 定义滚动过程）。当必须滚动窗体的整个客户区域，使用 ScrollWindowEx 函数。

Windows CE：参数 dx 和 dy 中只有一个可以为非零。

速查：Windows NT 3.1、Windows 95、windows CE1.0 对以上，头文件：winuser.h；库文件：user32.lib。

2.18.4 ScrollWindowEx

函数功能：该函数滚动指定窗体客户区域的目录。

函数原型：int ScrollWindowEx（HWND hWnd，int dx，int dyCONST RECT *prcScroll，CONST RECT *prcCllp，HRGH hrgnUpdate，LPRECT prcUpdate，UINT flags）；

参数：

hWnd：客户区域将被滚动的窗体句柄。

dx：在设备单元中，指定水平滚动数量。在向左滚动时此参数必须为负。

dy：在设备单元中，指定垂直滚动数量。在向上滚动时此参数必须为负。

prcScroll：指向 RECT 结构，它指定了将被滚动的客户区域部分。

prcClip：指向包含了类似于被剪下矩形的 RECT 结构。只有在剪下内部的小块图形才受影响。从矩形外向内部的滚动部分将被着色；而从内向外的滚动部分将不再被着色。

hrgnUpdate：处理已被修改的区域，保存这些由于滚动而无效的区域。此参数可以为空。

prcUpdate：指向 RECT 结构，它接收由于滚动使得矩形无效部分的边界。此参数值可以为空。

flags：指定控制滚动的标志。这个参数可以是下面的值：

SW_ERASE：通过发送 WM_ERASEBKGND 消息给窗体。

SW_INVALIDATE：在滚动后，使得由参数 hrgnUpdate 标识的无效区域被擦除。

SW_SCROLLCHILDREN：动所有由参数 prcScroll 指出交叉重叠矩形的子窗体。子窗体按照 dx 和 dy 规定的像素个数滚动。系统发送消息给所有由 prcScroll 指出交叉重叠矩形的子窗体，即使他们不移动。

SW_SMOOTHSCROLL：Windows NT 5.0 或以上版本中：使用平滑滚动。利用 flags 参数中 HIWORD 部分简要说明所需平滑滚动操作的时间。

返回值：如果函数运行成功，返回值为 SIMPLEREGION（矩形的无效区域），COMPLEXREGION（非矩形的无效区域）或 NULLREGION（没有使无效的区域）如果函数运行成功，返回值为 ERROR。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：如果 SW_INVALIDATE 和 SW_ERASE 标志没有被设定，那么函数 ScrollWindowEx 不能使滚动离开的区域失效。如果其中任意一个标志被设置，ScrollWindowEx 函数就可以使区域无效。这块区域将不再被更新直到应用程序调用 theUpdateWindow 函数，调用 theRedrawWindow 函数（指定 RDW_UPDATENOW 或 RDW_ERASENOW 标志）或是从申请队列中找到 WM_PAINT 消息。

如果窗体拥有 WS_CLIPCHILDREN 类型，那么由 hrgnUpdate 和 prcUpdate 指定的返回区域描述了必须更新的滚动窗体的全部区域，包括所需更新子窗体的任何区域。

若 SW_SCROLLCHILDREN 标志被设置，在子窗体被滚动时，系统将不能完全更新屏幕。位于矩形外边的滚动子窗体部分不被擦除，也不在它的新方向上被重画。为了移动子窗体使之完全不在 prcScroll 指定的矩形条中，可使用 DeferWindowPos 函数。若标志 SW_SCROLLCHILDREN 被设置并且^符号交叉滚动矩形，则光标也重新设置。

所有输入输出均被定义为客户端如果有必要，使用 IptoDP 和 dptoLP 函数转换逻辑相关性。

Windows CE：参数 flags 不支持 SW_SCROLLCHILDREN。参数 dx 和 dy 中只有一个为零。

速查：Windows NT 3.1、Windows 95、Windows CE 1.0 以上，头文件：winuser.h；库文件：user32.lib。

2.18.5 SetScrolllnfo

函数功能：该函数设置滚动条参数，包括滚动位置的最大值和最小值，页面大小，滚动按钮的位置。如被请求，函数也可以重画滚动条。

函数原型：int SetScrohnfo（HWND hWnd；int fnBar，LPSCROLLINFO lpsi，BOOL fRedraw）；

参数：

hWnd：滚动条控制或带标准滚动条的窗体句柄，由 fnBar 参数决定。

fnBar：指定被设定参数的滚动条的类型。这个参数可以是下面值，含义如下：

SB_CTL：设置滚动条控制。而参数 hwnd 必须是滚动条控制的句柄。

SB_HORZ：设置所给定的窗体上标准水平滚动条参数。

SB_VERT：设置所给定的窗体上标准垂直滚动条参数。

IPBI：指向 SCROLLINFO 结构。在调用 SetScrognfo 之前，设置 SCROLLINFO 结构中 cbSize 成员以标识结构大小，设置成员 fMask 以说明待设置的滚动条参数，并且在适当的成员中制定新的参数值。成员 fMask 可以为下面所列复合值，含义如下：

SIF_DfSABLENOSCROLL：如果滚动条的新参数使其为没必要，则使滚动条无效而不再移动它。

SIF_PAGE：设置滚动页码值到由 Ipsi 指向的 SCROLLINFO 结构的 nPage 成员中。

SIF_POS：设置滚动位置值到由 lpsi 指向的 SCROLLINFO 结构的 nPos 成员中。

SIF_RANGE：设置滚动范围值到由 lpsl 指向的 SCROLLINFO 结构的 nMin 和 nMax 成员中。

fRedraw：指定滚动条是否重画以反映滚动条的变化。如果这个参数为 TRUE，滚动条将被重画，否则不被重画。

返回值：返回值是滚动盒的当前位置。

注意：SetScrolllnfo 函数执行任务是检查 SCROLLINFO 结构中由成员 nPage 和 nPos 值的范围。成员 uPage 值必须从 0 到 nMax- nMin＋1，成员 nPos 必须是在 nMin 和 nMax-nMax-max（nPage C1，0）之间的指定值。如果任何一个值超过了这个范围，函数将在指定范围内为它设置一个值。

在 Windows CE 2.0 中，如果在参数 lpsi 中指定一个 NULL 指针，Setscrohnfo 则返回 0，而不返回滚动盒的当前位置。

速查：Windows NT 3.51、Windows 95、Windows CE 1.0 以上，头文件：wnuser.h；库文件：user32.lib。

2.18.6 ShowScrollBar

函数功能：该函数显示或隐藏所指定的滚动条。

函数原型：BOOL ShowScrollBar（HWND hWnd，int wBar，BOOL bShow）；

参数：

hWnd：根据参数 wBar 值，处理滚动条控制或带有标准滚动条窗体。

wBar：指定滚动条是被显示还是隐藏。这个参数讲师下面值之一，具体含义如下：

SB_SOTH：显示或隐藏窗体的标准的水平或垂直滚动条。

SB_CTL：显示或隐藏滚动条控制。参数 hWnd 必须是指向滚动条控制的句柄。

SB_HORZ：显示或隐藏窗体的标准的水平滚动条。

SB_VERT：显示或隐藏官体的标准的垂直滚动条。

bShow：指定滚动条是被显示还是隐藏。此参数为 TRUE，滚动条将被显示，否则被隐藏。

返回值：如果函数运行成功，返回值为非零；如果函数运行失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：当处理滚动条消息时，不能调用这个函数隐藏滚动条。

速查：Windows NT 3.1、Windows 95 以上，头文件：winuserh；库文件：user32.lib。

2.18.7 GetScrollPos

函数功能：该函数获取指定滚动条中滚动按钮的当前位置。当前位置是一个根据当前滚动范围而定的相对值。例如，如果滚动范围是 0 到 100 之间，滚动按钮在中间位置，则其当前位置为 50。该函数提供了向后兼容性，新的应用程序应使用 GetScroiliofo 函数。

函数原型：int GetScrogPos（HWND hWnd；int nBar）；

参数：

hWnd：根据参数 nBar 值，处理滚动条控制或带有标准滚动条窗体。

nBar：指定滚动条将被检查。这个参数可以是下面值，含义如下：

SB_CTL：找回滚动条控制中滚动翻页盒的位置。而参数 hwnd 必须是滚动条控制的句柄。

SB_HORZ：找回窗体上标准水平滚动条中参数滚动翻页盒的位置。

SB_VERT：找回窗体上标准垂直滚动条中参数滚动翻页盒的位置。

返回值：如果函数运行成功，其返回值是滚动翻页盒的当前位置；如果函数运行失败，其返回值是 0。想若想获得更多的错误信息，请调用 GetLastError 函数。

注意：函数 GetScrollPos 可以使应用程序使用 32 位滚动位置。尽管消息 WM_HSCROLL 和 WM_VSCROLL 指出了滚动条位置，但却被限制为 16 位，而函数 SetScrollPos，SetScrollRange，GetScrollPos，和 GetscrollRange 都支持 32 位的滚动条数据。

在 WM_HSCROLL 或 WM_VSCROLL 消息中通告 SB_JHUMBTRACK 时，为了得到滚动条 32 位的位置，请调用 GetScrolllnfo 函数。

在 WM_HSCROLL 或 WM_VSCROLL 消息中通告 SB_THUMBTRACK 时，为了得到 32 位的滚动条，则调用函数 GetScrolllnfo。

速查：Windows3.1、Windows 95 以上，头文件：winuser.h；库文件：user32.lib

2.18.8 GetScrollRange

函数功能：获取指定滚动条中滚动按钮位置的当前最大最小值。

函数原型：BOOL GetscrollRange（HWND hWnd，int nBar，LPINT lpMinPos，LPINT InMaxPos）；

参数：

hWnd：滚动条控制或带标准滚动条窗体的句柄，由 nBar 参数值确定。

nBar：定滚动条哪一个位置将被找回。这个参数可以是下面值，含义如下：

SB_CTL：找回滚动条控制位置。而参数 hWnd 必须是滚动条控制的句柄。

SB_HORZ：找回窗体上标准水平滚动条的位置。

SB_VERT：找回窗体上标准垂直滚动条位置。

lpMinPos：指向所找到最小位置整型变量。

lpMaxPos：指向所找到最小位置整型变量。

返回值：如果函数运行成功，返回值为非零；如果函数运行失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：如果所指定的窗体没有标准的滚动条或者不是滚动条控制，那么 GetscrollRange 函数将复制 0 到参数 lpMinPos 和 lpMaxPos 中。

标准滚动条的缺省范围值是从 0 到 100 之间，滚动条控制的缺省范围为空。

说明滚动条位置的消息 WM_HSCROLL 和 WM_VSCROLL 均为 16 位的数据。但是，因为函数 SetScrolllnfo，SetScrollPos，SetScrollRange；GetScrolllnfo，GetScrollPos，和 GetScrollRange 都支持 32 位的滚动条位置数据，所以有一个解决 16 位 WM_HSCROLL 和 WM_VSCROLL 消息阻碍的途径，请参见函数 GetScrohnfo 的有关技术说明。

速查：Windows NT 3.1、Windows 95 以上，头文件：winuserh；库文件：user32.lib。

2.18.9 ScrollWindow

函数功能：该函数滚动所指定的窗体客户区域内容。函数提供了向后兼容性，新的应用程序应使用 ScrollWindowEX。

函数原型：BOOL ScrollWindow（HWND hWnd，int XAmount，int YAmount，CONST RECT*IpRect， CONST RECT*lpClipRect）；

参数：

hWnd：客户区域将被滚动的窗体句柄。

XAmount：指定水平滚动以设备为单位的数量。如果窗体被滚动模式为 CS_OWNDC 或 CS_CLASSDC，此参数则使用逻辑单位而不使用设备单位。当向左滚动窗体内容时，参数值必须为负。

YAmount：指定垂直滚动设备单位数量。如果窗体被滚动模式为 CS_OWNDC 或 CS_CLASSDC，此参数则使用逻辑单位而不使用设备单位。当向上滚动窗体内容时，参数值必须为负。

lpRect：指向所指定将被滚动的客户区域部分的 RECT 结构。若此参数为 NULL，则整个客户区域均被滚动。

lpClipRect：指向包含类似于剪辑滚动条 RECT 结构。只有剪辑矩形条内部的位受影响。由外向内的滚动矩形内部被着色，而由矩形内向外的滚动将不被着色。

返回值：如果函数运行成功，返回值为非零；如果函数运行失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：如果在被滚动的窗体中由^符，滚动富体将自动隐藏起^符，以防止它被擦掉；当滚动结束后再恢复^符。^符的位置因而被调整过来。

未被 ScrollWindow 覆盖的区域不再被重画，但它组合成窗体的更新区域。应用程序最终最终受到 WM_PAINT 的消息，通知它区域必须被重画。为了在滚动过程的同时重画未覆盖区域，则应在调用 ScrollWindow 函数后马上调用 UpdateWindow 函数。

如果参数 lpRect 为空，则窗体中的任何子窗体的位置由参数 XAmount 和 Yamount 种的数量决定偏移；窗体无效（未着色）的区域也进行偏移。IpRect 为空时 ScrollWindow 则更快。

如果参数 lpRect 不为空，则窗体中的子窗体的位置不改变，窗体中无效（未着色）的区域也不进行偏移。为了防止 lpRect 不为空时更新的问题，则在调用 ScrollWindow 之前先调用 UpdateWindow 函数重窗体。

速查：Windows NT 3.1、Windows 95 以上，头文件：winuser.h；库文件：user32.lib。

2.18.10 SetScrollPos

函数功能：该函数设置所指定滚动条中的滚动按钮的位置，如要求重画滚动条以反映出滚动按钮的新位置。该函数提供了向后兼容性，新的应用程序应使用 SetScrolllnfo 函数。

函数原型：int SetScrollPos（HWN hWnd，int nBar，int nPos，BOOL bRedraw）；

参数：

hWnd：滚动条控制或带有标准滚动条窗体的句柄，由 nBar 参数值确定。

nBar：指定滚动条将被设置。这个参数可以是下面值，含义如下：

SB_CTL：设置滚动条控制中滚动翻页盒的位置。而参数 hwnd 必须是滚动条控制的句柄。

SB_HORZ：设置窗体上标准水平滚动翻页盒的位置。

SB_VERT：设置窗体上标准垂直滚动翻页盒的位置。

nPos：指定滚动翻页盒的新位置。这个位置必须在滚动范围之内。若要了解更多有关滚动范围的信息，请参见 SetScrollRange 函数。

bRedraw：指定滚动条是否被重画以反映出新的滚动翻页盒的位置。如果这个参数为 TRUE，则滚动条将被重画；为 FALSE 则滚动条不被重画。

返回值：如果函数运行成功，其返回值是滚动翻页盒的前一个位置。如果函数运行失败，其返回值是 0。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：如果由于其他并发函数调用，滚动条又被重画，那么设置参数 bRedraw 为负时非常有必要的。

因为说明滚动条位置的消息 WM_HSCROLL 和 WM_VSCROLL 只能为 16 位数据，那些只依赖于说明位置数据消息的应用程序在函数 SetScrollPos 的参数 nPos 中有一个实际最大值。

但是，因为函数 SetScrolllnfo，SetScrollPos，SetScrollRange，GetScrollPos， 和 GetScrollRange

都支持 32 位的滚动条位置数据，所以有一个解决 16 位 WM_HSCROLL 和 WM_VSCROLL 消息阻碍的途径，请参见函数 GetScrolllnfo 的有关技术说明。

速查：Windows NT 3.1、Windows 95、Windows CE 2.0 以上，头文件：winuser.h；库文件：user32.lib。

2.18.11 SetScrollRange

函数功能：该函数设置所指定滚动条位置的最大最小值。

函数原型：BOOL SetScrollRange（HWND hWnd，int nBar，int nMinPos，int nMaxPos，BOOL bRedraw）；

参数：

hWnd：滚动条控制或带有标准滚动条窗体的句柄，由 nBar 参数值确定。

nBar：指定滚动条将被设置。这个参数可以是下面值，含义如下：

SB_CTL：设置滚动条控制的范围。而参数 hwnd 必须是滚动条控制的句柄。

SB_HORZ：设置窗体上标准水平滚动翻页盒的范围。

SB_VERT：设置窗体上标准垂直滚动翻页盒的范围。

nMinPos：指定滚动位置的最小值。

nMaxPos：指定滚动位置的最大值。

bRedraw：指定滚动条是否被重画以反映变化。如果这个参数为 TRUE，滚动条将被重化；如果为 FALSE 则不被重画。

返回值：如果函数运行成功，返回值为非零；如果函数运行失败，返回值为零。若想获得更多的错误信息，请调用 GetLastError 函数。

注意：可以使用 setScrollRange 函数把 nMinPos 和 nMaxPos 设置为一样的值从而隐藏起滚动条。在处理滚动条消息时，应用程序可以不用调用函数 SetScrollRange 来隐藏滚动条。新的应用程序使用函数 ShowScrollBar 来隐藏滚动条。

如果调用函数 SetScrollPos 之后马上调用函数 SetScrollRange，则 SetScrollPos 中的 bRedraw 参数一定为零，以防止滚动条被画两次。

标准滚动条的缺省范围时 0 到 100。滚动条控制的缺省值为 NULL（参数 nMinPos 和 nMaxPos 的值均为零）。两个范围值之间的不同之处在于由参数 nMinPos 和 nMaxPos 指定的值不能超过 MAXLONG 的值。

因为说明滚动条位置的消息 WM_HSCROLL 和 WM_VSCROLL 只能为 16 位数据，那些只依赖于说明位置数据消息的应用程序在函数 SetScrollRange 的参数 nMaxPos 中有一个实际最大值 65，535。但是，因为函数 SetScrolllnfo，SetScrollPos， SetScrollRange，GetScrolllnfo，GetScrollPos，和 GetScrollRange 都支持 32 位的滚动条位置数据，所以有一个解决 16 位 WM_HSCROLL 和 WM_VSCROLL 消息阻碍的途径，请参见函数 GetScrolllnfo 的有关技术说明。

速查：Windows NT 3.1、Windows CE 2.0 以上，头文件：winuser.h；库文件：user32.lib。

2.19 窗口函数（Window）

2.19.1 AdlustWindowRect

函数功能：该函数依据所需客户矩形的大小，计算需要的窗日矩形的大小。计算出的窗口矩形随后可以传递给 CreateWindow 函数，用于创建一个客户区所需大小的窗口。

函数原型：BOOL AdjustWindowRect（LPRECT lpRect ，DWORD dwStyle，BOOL bMENU）；

参数：

lpRect：指向 RECT 结构的指针，该结构包含所需客户区域的左上角和右下角的坐标。函数返回时，该结构容纳所需客户区域的窗口的左上角和右下角的坐标。

dwStyle：指定将被计算尺寸的窗口的窗口风格。

bMenu：指示窗口是否有菜单。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。获取错误信息，参看 GetLastError。

备注：客户矩形是指完全包含一个客户区域的最小矩形；窗日矩形是指完全包含一个窗口的最小矩形，该窗口包含客户区与非客户区。

当一个菜单条下拉出两行或更多行时，AdjustWindowRect 函数不增加额外的空间。

速查：Windows NT：3.1 以上版本：Windows：95 以上版本：Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.19.2 AdjustWindowRectEx

函数功能：该函数依据所需客户矩形大小，计算需要的窗口矩形的大小。计算出的窗口矩形随后可以传送给 CreateWindowEx 函数，用于创建一个客户区所需大小的窗口。

函数原型：BOOL AdjustWindowRectEX(LPRECT lpRect，DWORD dwStyte；BOOL bMenu；DWORD dwExStyle)；

参数：

lpRect：指向 RECT 结构的指针，该结构包含所需客户区域的左上角和右下角的坐标。函数返回时，该结构包含容纳所需客户区域的窗口的左上角和右下角的坐标。

dwStyle：指定将被计算尺寸的窗口的窗口风格。

bMenu：指示窗口是否有菜单。

dwExStyle：指定将被计算尺寸的窗口的扩展窗口风格。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：客户矩形是指完全包含一个客户区域的最小矩形；窗口矩形是指完全包含一个窗口的最小矩形，该窗口包含客户区与非客户区。

当一个菜单条下拉出两行或更多行时，AdjustWindowRect 函数不增加额外的空间。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：1.0 以上版本；头文件：winuser.h；库文件：user32.lib。

2.19.3 AnimateWindow

函数功能：该函数能在显示与隐藏窗口时能产生特殊的效果。有两种类型的动画效果：滚动动画和滑动动画。

函数原型：BOOL AnimateWindow (HWND hWnd, DWORD dwTime, DWORD dwFlags)；

参数：

hWnd：指定产生动画的窗口的句柄。

dwTime：指明动画持续的时间（以微秒计），完成一个动画的标准时间为 200 微秒。

dwFags：指定动画类型。这个参数可以是一个或多个下列标志的组合。标志描述：

AW_SLIDE：使用滑动类型。缺省则为滚动动画类型。当使用 AW_CENTER 标志时，这个标志就被忽略。

AW_ACTIVE：激活窗口。在使用了 AW_HIDE 标志后不要使用这个标志。

AW_BLEND：使用淡出效果。只有当 hWnd 为顶层窗口的时候才可以使用此标志。

AW_HIDE：隐藏窗口，缺省则显示窗口。

AW_CENTER：若使用了 AW_HIDE 标志，则使窗口向内重叠；若未使用 AW_HIDE 标志，则使窗口向外扩展。

AW_HOR_POSITIVE：自左向右显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略。

AW_VER_POSITIVE：自顶向下显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略。

AW_VER_NEGATIVE：自下向上显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。在下列情况下函数将失败：

窗口使用了窗口边界；窗口已经可见仍要显示窗口；窗口已经隐藏仍要隐藏窗口。若想获得更多错误信息，请调用 GetLastError 函数。

备注：可以将 AW_HOR_POSITIVE 或 AW_HOR_NEGTVE 与 AW_VER_POSITVE 或 AW_VER_NEGATIVE 组合来激活一个窗口。

可能需要在该窗口的窗口过程和它的子窗口的窗口过程中处理 WM_PRINT 或 WM_PRINTCLIENT 消息。对话框，控制，及共用控制已处理 WM_PRINTCLIENT 消息，缺省窗口过程也已处理 WM_PRINT 消息。

速查：WIDdOWS NT：5.0 以上版本：Windows：98 以上版本；Windows CE：不支持；头文件：Winuser.h；库文件：user32.lib。

2.19.4 ArrangelconicWindows

函数功能：该函数安排指定父窗口的各个最小化（图标化）子窗口。

函数原型：UNIT ArrangelconicWindows（HWND hWnd）；

参数：

hWnd：父窗口句柄。

返回值：如果函数成功，返回值为一行图标的高度。如果函数失败，返回值为零。若想获得更多错误信息，请调用 callGetLastError 函数。

备注：一个应用程序可以通过使用 ArrangelconicWindows 安排在一个父窗口卫的它自身的最小化的子窗口。这个函数也可以安排桌面图标。使用 GetDesttopWindow 函数获得桌面窗口的句柄。

一个应用程序给多文本接口（MDI）客户窗口发送 WM_MDIICONARRANGE 消息，使客户窗口来安排自身的最小化 MDI 子窗口。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.19.5 BeginDeferWindowPos

函数功能：该函数为一个多窗口位置结构分配内存并且返回该结构的句柄。

函数原型：HDWP BeginDeferWindowPos（int nNumWindows）；

参数：

nNumWindows：指示存储位置信息的初始窗口数目。如有必要，DeferWindowPos 函数可以增加该结构的大小。

返回值：如果函数成功，返回多窗口位置结构。如果分配内存时内存不足，则返回值为 NULL。若想获得更多的错误信息，请调用 GetLastError 函数。

备注：多窗口位置结构是一个内部结构，应用程序不能直接引用。

DeferWindowPos 函数在多窗口位置结构中填充了将被移动的一个或多个窗口的目标位置信息。

EndDeferWindowPos 接收该结构的句柄，并且依据存储在该结构中的信息重定位这些窗口。

如果在多窗口位置结构中的任意一个窗口中设置了 SWP_HIDEWINDOW 和 SWP_SHOWWINDOW 标志，则所有窗口都不能被重定位。

如果系统必须增加在多窗口位置结构中由 nNumWindows 设置的初始窗口数目，但又没有足够的内存分配，则系统的整个窗口重定位顺序失败（BeginDeferWindowsPos，DeferWindowsPos，EndDeferWindowPos）。应用程序通过指定所需最大数目，可以在执行过程中进行早期的检测和处理。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32.lib。

2.19.6 BromgWindowToTop

函数功能：该函数将指定的窗口设置到 Z 序的顶部。如果窗口为顶层窗口，则该窗口被激活；如果窗口为了窗口，则相应的顶级父窗口被激活。

函数原型：BOOL BringWindowToTop（HWND，hWnd）；

参数：

hWnd：设置到 Z 序的顶部的窗口句柄。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：使用 BringWindowToTop 函数显示出被其他窗口部分或全部遮盖的窗口。

调用这个函数类似于调用 SetWindowPos 函数来改变窗口在 Z 序中的位置，但是 BringWindowToTop 函数并不能使一个窗口成为顶层窗口。

如果应用程序不在前台中而想设置在前台中，可以调用 SetForegroundWindow 函数。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h；库文件：user32.lid。

2.19.7 CascadeWindows

函数功能：该函数层叠排列指定父窗口的各指定子窗口。

函数原型：WORD WINAPI CascadeWihdows（HWND hWndParent，UNIT wHow，CONST RECT*lpRect，UNIT cKids， Const HWND FA*lpKids）；

参数：

Parent：父窗口的句柄。如果参数为 NVLL，则假定为桌面窗口。

wHow：指定层叠标志。唯一可用的标志为 MDITILE_SKIPDISABLED，防止被禁止的 MDI 子窗口被层叠排列。

lpBect：指向 RECT 结构的指针，该结构以客户坐标定义矩形区域，并在这个区域中排列窗口、该参数可以为 NULL，这种情况下使用父窗口的客户区域。

cKids：指明由 lpKids 参数指定的数组的成员个数。如果 IpKidS 参数为 NULL，则此参数将被忽略。

lpKids：指向将被排列的子窗口的句柄数组的指针。如果此参数为空，则指定的父窗口（或桌面窗口）的所有子窗口都将被排列。

返回值：如果函数成功，返回值为被排列的窗口数目；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：调用 CascadeWindows 函数使所有最大化窗口恢复到它们原来的大小。

速查： Windows NT：4.0 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；

库文件：User32.1

2.19.8 ChildWindowFromaPoint

函数功能： 该函数确定属于某一父窗口的哪一个子窗口（如果存在的话）包含一个指定的点。

函数原型： HWND ChildWindowFromaPoint（HWND hWndParent.POINT Point）：

Parent：父窗口句柄。

Point：指定一个 POINT 结构，该结构给定了被检查的点的坐标。

返回值： 返回值为包含该点的子窗口的句柄，即使该子窗口是隐藏的或被禁止的。如果该点在父窗口之外，则返回值为 NULL。如果该点在父窗口内，但在任一子窗口外，则返回值为父窗口句柄。

备注： 系统有一个与某一父窗口有联系的所有子窗口的内部列表。列表中的句柄顺序依据这些子窗口的 z 序。如果有多于一个的子窗口包含该点，那么系统返回在列表中包含该点的第一个窗口的句柄。

速查：Windows NT:3.1 以上版本:Windows:95 以上版本;Windows CE;1.0 以上版本;头文件:winuser.h;库文件：user32.lib。

2.19.9 ChildWindowFromaPointEx

函数功能： 该函数确定属于父窗口的哪一个子窗口（如果存在的话）包含着指定的点。该函数可以忽略不可见的、禁止的和透明的子窗口。

函数原型： HWND ChidWindowFromaPointEx（HWND hwndParent,POINT pt，UNIT uFlags）;

参数：

hWndParent:父窗口句柄。

pt：指定一个 POINT 结构，该结构定义了被检查的点的坐标。

uFlags:指明忽略的子窗口的类型。该参数可以是下列参数的组合。

CWP_ALL：不忽略任一子窗口。CWP_SKIPINVISIBLE：忽略不可见的子窗口。

CWP_SKIPDISABLE:忽略禁止的子窗口。CWP_SKIPTRABSPARENT：忽略透明子窗口。

返回值：返回值为包含该点并且满足由 uFlags 定义的规则的第一个子窗口的句柄。如果该点在父窗口内，但在任一满足条件的子窗口外，则返回值为父窗口句柄。如果该点在父窗口之外或函数失败，则返回值为 NULL。

备注： 系统有一个与某一父窗口有联系的所有子窗口的内部列表。列表中的句柄顺序依据这些子窗口的 Z 序。如果有多于一个的子窗口包含该点，那么系统返回在列表中包含该点并且满足由 uFlags 定义的规则的第一个窗口的句柄。

速查：WindowS NT：4.0 以上版本；Windows：95 以上版本；Windows CE:不支持；头文件:Winuser.h；库文件：user32.lib。

2.19.10 CloseWindow

函数功能： 该函数最小化指定的窗口，但并不销毁该窗口。

函数原型： BOOL CloseWindow(HWND hWnd);

参数：

hWnd：将要最小化的窗口的句柄。

返回值： 如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调

用 GetLastError 函数。

备注：窗口尺寸被最小化成一个图标，并移动到屏幕的图标区域。系统显示窗口的图标而不显示窗口，并在图标下显示窗口标题。应用程序必须使用 DestroyWindow 函数销毁窗口。

速查：WindowS NT：3.1 以上版本；Windows：95 以上版本；Windows CE:不支持；头文件:Winuser.h；库文件：user32.lib

2.19.11 Create Window

函数功能：该函数创建一个重叠式窗口、弹出式窗口或子窗口。它指定窗口类，窗口标题，窗口风格，以及窗口的初始位置及大小（可选的）。该函数也指定该窗口的父窗口或所属窗口（如果存在的话），及窗口的菜单。若要使用除 CreateWindow 函数支持的风格外的扩展风格，则使用 CreateWindowEx 函数代替 CreateWindow 函数。

函数原型：HWND CreateWindow(LPCTSTR lpClassName,LPCTSTR lpWindowName,DWORD dwStyle,int x,int y,int nWidth，int nHeight，HWND hWndParent,HMENU hMenu，HANDLE hlnstance，LPVOID lpParam)；

参数：

lpClassName:指向一个空结束的字符串或整型数 atom。如果该参数是一个整型量，它是由此前调用 theGlobalAddAtom 函数产生的全局量。这个小于 0xC000 的 16 位数必须是 lpClassName 参数字的低 16 位，该参数的高位必须是 0。

如果 lpClassName 是一个字符串，它指定了窗口的类名。这个类名可以是任何用函数 RegisterClassEx 注册的类名，或是任何预定义的控制类名。请看说明部分的列表。

LPWindowName:指向一个指定窗口名的空结束的字符串指针。

如果窗口风格指定了标题条，由 lpWindowName 指向的窗口标题将显示在标题条上。当使用 Createwindow 函数来创建控制例如按钮，选择框和静态控制时，可使用 lpWindowName 来指定控制文本。

dwStyle：指定创建窗口的风格。该参数可以是下列窗口风格的组合再加上说明部分的控制风格。风格意义：

WS_BORDER：创建一个单边框的窗口。

WS_CAPTION：创建一个有标题框的窗口（包括 WS_BODER 风格）。

WS_CHIlD：创建一个子窗口。这个风格不能与 WS_POPVP 风格合用。

WS_CHLDWINDOW：与 WS_CHILD 相同。

WS_CLIPCHILDREN:当在父窗口内绘图时，排除子窗口区域。在创建父窗口时使用这个风格。

WS_ClIPBLINGS：排除子窗口之间的相对区域，也就是，当一个特定的窗口接收到 WM_PAINT 消息时，WS_CLIPSIBLINGS 风格将所有层叠窗口排除在绘图之外，只重绘指定的子窗口。如果未指定 WS_CLIPSIBLINGS 风格，并且子窗口是层叠的，则在重绘子窗口的客户区时，就会重绘邻近的子窗口。

WS_DISABLED:创建一个初始状态为禁止的子窗口。一个禁止状态的窗日不能接受来自用户的输人信息。

WS_DLGFRAME:创建一个带对话框边框风格的窗口。这种风格的窗口不能带标题条。

WS_GROUP:指定一组控制的第一个控制。这个控制组由第一个控制和随后定义的控制组成，自第二个控制开始每个控制，具有 WS_GROUP 风格，每个组的第一个控制带有 WS_TABSTOP 风格，从而使用户可以在组间移动。用户随后可以使用光标在组内的控制间改变键盘焦点。

WS_HSCROLL：创建一个有水平滚动条的窗口。

WS_ICONIC：创建一个初始状态为最小化状态的窗口。与 WS_MINIMIZE 风格相同。

WS_MAXIMIZE：创建一个具有最大化按钮的窗口。该风格不能与 WS_EX_CONTEXTHELP 风格同时出现，同时必须指定 WS_SYSMENU 风格。

WS_OVERLAPPED:产生一个层叠的窗口。一个层叠的窗口有一个标题条和一个边框。与 WS_TILED 风格相同。

WS_OVERtAPPEDWINDOW：创建一个具有 WS_OVERLAPPED，WS_CAPTION，WS_SYSMENU WS_THICKFRAME，WS_MINIMIZEBOX，WS_MAXMIZEBOX 风格的层叠窗口，与 WS_TILEDWINDOW 风格相同。

WS_POPUP：创建一个弹出式窗口。该风格不能与 WS_CHLD 风格同时使用。

WS_POPUWINDOW：创建一个具有 WS_BORDER，WS_POPUP,WS_SYSMENU 风格的窗口，WS_CAPTION 和 WS_POPUPWINDOW 必须同时设定才能使窗口某单可见。

WS_SIZEBOX：创建一个可调边框的窗口，与 WS_THICKFRAME 风格相同。

WS_SYSMENU：创建一个在标题条上带有窗口菜单的窗口，必须同时设定 WS_CAPTION 风格。

WS_TABSTOP：创建一个控制，这个控制在用户按下 Tab 键时可以获得键盘焦点。按下 Tab 键后使键盘焦点转移到下一具有 WS_TABSTOP 风格的控制。

WS_THICKFRAME：创建一个具有可调边框的窗口，与 WS_SIZEBOX 风格相同。

WS_TILED：产生一个层叠的窗口。一个层叠的窗口有一个标题和一个边框。与 WS_OVERLAPPED 风格相同。

WS_TILEDWINDOW:创建一个具有 WS_OVERLAPPED，WS_CAPTION，WS_SYSMENU MS_THICKFRAME.

WS_MINIMIZEBOX，WS_MAXMIZEBOX 风格的层叠窗口。与 WS_OVERLAPPEDWINDOW 风格相同。

WS_VISIBLE：创建一个初始状态为可见的窗口。WS_VSCROLL：创建一个有垂直滚动条的窗口。

X：指定窗口的初始水平位置。对一个层叠或弹出式窗口，X 参数是屏幕坐标系的窗口的左上角的初始 X 坐标。对于子富口，x 是子窗口左上角相对父窗口客户区左上角的初始 X 坐标。如果该参数被设为 CW_UCEDEFAULT 则系统为窗口选择缺省的左上角坐标并忽略 Y 参数。CW_USEDEFAULT 只对层叠窗口有效，如果为弹出式窗口或子窗口设定，则 X 和 y 参数被设为零。

Y：指定窗口的初始垂直位置。对一个层叠或弹出式窗日，y 参数是屏幕坐标系的窗口的左上角的初始 y 坐标。对于子窗口，y 是子窗口左上角相对父窗口客户区左上角的初始 y 坐标。对于列表框，y 是列表框客户区左上角相对父窗口客户区左上角的初始 y 坐标。如果层叠窗口是使用 WS_VISIBLE 风格位创建的并且 X 参数被设为 CW_USEDEFAULT，则系统将忽略 y 参数。

nWidth:以设备单元指明窗口的宽度。对于层叠窗口，nWidth 或是屏幕坐标的窗口宽度或是 CW_USEDEFAULT。若 nWidth 是 CW_USEDEFAULT，则系统为窗口选择一个缺省的高度和宽度：缺省宽度为从初始 X 坐标开始到屏幕的右边界，缺省高度为从初始 X 坐标开始到目标区域的顶部。CW_USEDFEAULT 只参层叠窗口有效；如果为弹出式窗口和子窗口设定 CW_USEDEFAULT 标志则 nWidth 和 nHeight 被设为零。

nHelght:以设备单元指明窗口的高度。对于层叠窗口，nHeight 是屏幕坐标的窗口宽度。若 nWidth 被设为 CW_USEDEFAULT，则系统忽略 nHeight 参数。

hWndParent：指向被创建窗口的父窗口或所有者窗口的句柄。若要创建一个子窗口或一个被属窗口，需提供一个有效的窗口句柄。这个参数对弹出式窗日是可选的。Windows NT 5.0；创建一个消息窗口，可以提供 HWND_MESSAGE 或提供一个已存在的消息窗口的句柄。

hMenu：菜单句柄，或依据窗口风格指明一个子窗口标识。对于层叠或弹出式窗口，hMenu 指定窗口使用的菜单：如果使用了菜单类，则 hMenu 可以为 NULL。对于子窗口，hMenu 指定了该子窗口标识（一个整型量），一个对话框使用这个整型值将事件通知父类。应用程序确定子窗口标识，这个值对于相同父窗口的所有子窗口必须是唯一的。

hlnstance：与窗口相关联的模块事例的句柄。

lpParam:指向一个值的指针，该值传递给窗口 WM_CREATE 消息。该值通过在 IParam 参数中的 CREATESTRUCT 结构传递。如果应用程序调用 CreateWindow 创建一个 MDI 客户窗口，则 lpParam 必须指向一个 CLIENTCREATESTRUCT 结构。

返回值：如果函数成功，返回值为新窗口的句柄：如果函数失败，返回值为 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

备注：在返回前，CreateWindow 给窗口过程发送一个 WM_CREATE 消息。对于层叠，弹出式和子窗口，CreateWindow 给窗口发送 WM_CREATE，WM_GETMINMAXINFO 和 WM_NCCREATE 消息。消息 WM_CREATE 的 IParam

参数包含一个指向 CREATESTRUCT 结构的指针。如果指定了 WS_VISIBLE 风格，CreateWindow 向窗口发送所有需要激活和显示窗口的消息。

获取有关任务条是否为创建的窗口显示一个按钮的控制信息，参看 Taskbar 按钮的 Visbility。

以下预定义的控制类可以在 lpClassName 参数中指定。注意在 dwStyle 参数中可以使用的相应的控制风格。

BUTTON 按钮按钮是一个小矩形子窗口，用户可以点击来打开或关闭。按钮控制可以单独使用或包含在组中使用,可以为控制写标签或不写标签。当用户点击按钮控制时按钮的外观有明显的改变。请参看 Button。查看 dwStyle 参数中指定的按钮风格表请参考 Button Style。

COMBOBOX 组合框由一个列表框和一个类似于编辑控制的选择域组成。在使用这个风格控制时，应用程序或者使列表框一直显示或者是作成一个下拉列表。如果列表框可见，则在编辑域中输入字符将使列表框中与字符一致的第一个域高亮。反之，在列表框中选择的项将显示在编辑域中。请参看 Combo Boxes。

查看 dwStyle 参数中指定的组合框风格表请参考 Combo Boxes Style。

EDIT 编辑框一个小的矩形子窗口用户可以使用键盘向其中输入文本。用户可以通过点击或按 Tab 键来选中编辑框控制并且使控制获得焦点。当编辑框中显示一个闪烁的插入记号时，用户可以输入文本。使用鼠标移动光标，选择被替换的字符或设置插入字符的位置或使用回退键删除字符。请参看 Edit。controls。

查看 dwStyle 参数中指定的编辑框风格的表格请参考 Edit Control Style。

LISTBOX 列表框字符串的列表。当应用程序必须显示名称的列表，例如文件名列表等，使用户可以从中选择时就可指定列表框。用户可以通过单击来选择名称。选择时，被选择名高亮，同时传递给父窗口一个通知消息。请参看 LiSt Box Style。查看 dwStyle 参数中指定的列表风格表请参考 List BOX Control Style。

MDICLIENT MDI 客户设计出 MDI 客户窗口。窗口接收控制 MDI 应用程序子窗口的消息。建议使用两种控制风格位：WS_CLIPCHILDREN 和 WS_CHILD。指定了 WS_HSCROLL 和 WS_VSCROLL 风格的 MDI 客户窗口允许用户将 MDI 子窗口滑动进入视窗。请参看 MDI。

RiChEdit 设计一个 Rich Editl.0 版的控制。该控制使用户可以以字符和段落格式测览和编辑文本，并且可以包含嵌入的 COM 对象。请参看 Rich Edit Controls。查看 dwStyle 参数中指定的 RichEdit 风格表请参考 LiSt Box Control Style。

RICHEDIT CLASS 设计一个 Rich Edik2.0 版的控制。该控制使用户可以以字符和段落格式测览和编辑文本，并且可以包含嵌入的 COM 对象。请参看 RichEditControls。查看 dwStyle 参数中指定的 RichEdit 风格表请参考 RichEditControIStyle。

SCROLLBAR 滚动条设计的一个包含着一个滚动盒和两端有方向箭头的矩形。只要用户点击了控制，滚动条就给父窗口发送一个通知消息。如有必要，父窗口负责更新滚动条的位置。请参看 ScrollBars。查看 dwStyle 参数中指定的滚动条风格表请参考 Scroll Bars Style。

STATIC 一个简单的静态文本域，文本盒或矩形用于给控制加标签，组合控制或将控制与其他控制分开。

静态控制不提供输入和也不提供输出。请参看 Static Control Styles。查看 dwStyle 参数中指定的静态文本风格表请参考 Scroll Bars Style。

Windows95：系统可以支持最大 16,364 个窗口句柄。

备注：如果在链接应用程序时指明是 Windows 4.x 版本，除非应用程序的窗口有窗口某单，否则窗口控制没有标题控制。对 Windows3.x 版本没有这种要求。

Windows CE：CreateWindow 是以“宏”方式完成的。它被定义为 CreateWindowEX，并且 dwExStyle 参数被置为长整数 0。不支持菜单条控制，除非被声明为子窗口标志否则 hMenu 参数必须为 NULL。不支持 MDICLIENT 窗口类。dwStyle 参数可以是对话框（Dialogue Box），窗口（Windows），控制（Controls）文件中的窗口风格和控制风格的组合。

下列 dwStyle 标志在窗口中不支持：

WS_CHILDWINDOW WS_ICONC; WS_MAXMIZE WS_MAXIMIZEBOX; WS_MINIMIZE WS_MINIMIZEBOX;

WS_OVERLAPPEDWINDOW WS_POPUPWINDOW; WS_SIZEBOXWS_THICKFRAME WS_TILED WS_TILEDWINDOW

下列 dwStyle 标志在控制和对话框中不支持：

不支持的按钮风格和静态控制风格：

BS_LEFTTEXT SS_BLACKFRAME；BS_MULTILINE SS_GRAYFRAME BS_TEXT SS_METAPICT；BS_USERBUTTON SS_SIMPLE

不支持组合框 SS_WHITERECT 风格。

CBS_OWNERDRAWFIXED SS_BLACKRECT； CBS_OWNWEDRAWVARIABLE SS_GRAYRECT； CBS_SIMPLE8R 旧 HTrIMAGE

不支持列表框控制 SS_WHITEFRAME 风格。

LBS_NODATA

不支持的对话框风格：

LBS_OWNERDRAWFIXED DS_ABSALIGN； LBS_OWNERDRAWVARIABLE DS_CENTERMOUSE； LBS_STANDARD DS_CONTEXTHELP

不支持滚动条的 DS_FIXEDSYS 风格

SBS_BOTTOMALIGN DS_NOFAILCREATE；SBS_RIGHTALIGN DS_NOIDLEMSG；

SBS_SlZEBOXBOTTOMRIHTALIGN DS_SYSMODAL；SBS_SIZEGRIP

可使用 BS_OWNERDRAW 风格来代替 BS_USERBUTTON 风格。

可使用 SS_LEFT 或 SS_LEFTNOWORDWRAP 风格来代替静态控制的 SS_SIMPLE 风格。

不支持 MDICLIENT 窗口类。

所有窗口都隐含 WS_CLIPSIBLINGS 和 WS_CLIPCHILDREN 风格。

Windows CE1.0 版除对话框外不支持被属窗口。如果 hwndParent 参数不为 NULL，则窗口隐含给出 WS_CHILD 风格。Windows CE1.0 不支持菜单条。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：1.0 以上版本；头文件：winuser.h；库文件：user32.lib； Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.19.12 CreateWindowEx

函数功能：该函数创建一个具有扩展风格的重叠式窗口、弹出式窗口或子窗口，其他与 CreateWindow 函数相同。关于创建窗口和其他参数的内容，请参看 CreateWindowEx。

函数原型：HWND CreateWindowEx（DWORD dwExStle，LPCTSTR IpClassName，LPCTSTR lpWindowName，DWORD dwStyle，int x，int y，int nWidth，int nHeight，HWND hWndParent，HMENUhMenu，HANDLE hlnstance，LPVOIDlpParam)；

参数：

dwExStyle：指定窗口的扩展风格。该参数可以是下列值：

WS_EX_ACCEPTFILES：指定以该风格创建的窗口接受一个拖拽文件。

WS_EX_APPWINDOW：当窗口可见时，将一个顶层窗口放置到任务条上。

WS_EX_CLIENTEDGE：指定窗口有一个带阴影的边界。

WS_EX_CONTEXTHELP：在窗口的标题条包含一个问号标志。当用户点击了问号时，鼠标光标变为一个问号的指针、如果点击了一个子窗口，则子窗日接收到 WM_HELP 消息。子窗口应该将这个消息传递给父窗口过程，父窗口再通过 HELP_WM_HELP 命令调用 WinHelp 函数。这个 Help 应用程序显示一个包含子窗口帮助信息的弹出式窗口。 WS_EX_CONTEXTHELP 不能与 WS_MAXIMIZEBOX 和 WS_MINIMIZEBOX 同时使用。

WS_EX_CONTROLPARENT：允许用户使用 Tab 键在窗口的子窗口间搜索。

WS_EX_DLGMODALFRAME：创建一个带双边的窗口；该窗口可以在 dwStyle 中指定 WS_CAPTION 风格来创建一个标题栏。

WS_EX_LEFT：窗口具有左对齐属性，这是缺省设置的。

WS_EX_LEFTSCROLLBAR：如果外壳语言是如 Hebrew，Arabic，或其他支持 reading order alignment 的语言，则标题条（如果存在）则在客户区的左部分。若是其他语言，在该风格被忽略并且不作为错误处理。

WS_EX_LTRREADING：窗口文本以 LEFT 到 RIGHT（自左向右）属性的顺序显示。这是缺省设置的。

WS_EX_MDICHILD：创建一个 MD 子窗口。

WS_EX_NOPATARENTNOTIFY：指明以这个风格创建的窗口在被创建和销毁时不向父窗口发送 WM_PARENTNOTFY 消息。

WS_EX_OVERLAPPED：WS_EX_CLIENTEDGE 和 WS_EX_WINDOWEDGE 的组合。

WS_EX_PALETTEWINDOW：WS_EX_WINDOWEDGE, WS_EX_TOOLWINDOW 和 WS_WX_TOPMOST 风格的组合 WS_EX_RIGHT:窗口具有普通的右对齐属性，这依赖于窗口类。只有在外壳语言是如 Hebrew,Arabic 或其他支持读顺序对齐（reading order alignment）的语言时该风格才有效，否则，忽略该标志并且不作为错误处理。

WS_EX_RIGHTSCROLLBAR：垂直滚动条在窗口的右边界。这是缺省设置的。

WS_EX_RTLREADING：如果外壳语言是如 Hebrew，Arabic，或其他支持读顺序对齐（reading order alignment）的语言，则窗口文本是一自左向右）RIGHT 到 LEFT 顺序的读出顺序。若是其他语言，在该风格被忽略并且不作为错误处理。

WS_EX_STATICEDGE：为不接受用户输入的项创建一个 3 一维边界风格

WS_EX_TOOLWIDOW：创建工具窗口，即窗口是一个游动的工具条。工具窗口的标题条比一般窗口的标题条短，并且窗口标题以小字体显示。工具窗口不在任务栏里显示，当用户按下 alt＋Tab 键时工具窗口不在对话框里显示。如果工具窗口有一个系统菜单，它的图标也不会显示在标题栏里，但是，可以通过点击鼠标右键或 Alt＋Space 来显示菜单。

WS_EX_TOPMOST：指明以该风格创建的窗口应放置在所有非最高层窗口的上面并且停留在其 L，即使窗口未被激活。使用函数 SetWindowPos 来设置和移去这个风格。

WS_EX_TRANSPARENT：指定以这个风格创建的窗口在窗口下的同属窗口已重画时，该窗口才可以重画。

由于其下的同属富日已被重画，该窗口是透明的。

IpClassName:指向一个空结束的字符串或整型数 atom。如果该参数是一个整型量，它是由此前调用 theGlobaIAddAtom 函数产生的全局量。这个小于 0xC000 的 16 位数必须是 IpClassName 参数字的低 16 位，该参数的高位必须是 0。

如果 lpClassName 是一个字符串，它指定了窗口的类名。这个类名可以是任何用函数 RegisterClassEx 注册的类名，或是任何预定义的控制类名。请看说明部分的列表。

lpWindowName:指向一个指定窗口名的空结束的字符串指针。

如果窗口风格指定了标题条，由 lpWindowName 指向的窗口标题将显示在标题条上。当使用 CreateWindow

函数来创建控制例如按钮，选择框和静态控制时，可使用 lpWindowName 来指定控制文本。

dwStyle:指定创建窗口的风格。该参数可以是下列窗口风格的组合再加上说明部分的控制风格。

x：参见 CreateWindow。

y：参见 CreateWindow。

nWidth:CreateWindow。

nHeigth：参见 CreateWindow。

hWndParent：参见 CreateWindow。

hMenu：参见 CreateWindow。

hlnstance：参见 CreateWindow。

lpParam:参见 CreateWindow。

返回值：参见 CreateWindow。

备注:参见 CreateWindow。

速查:Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件:USer32.lib;Unicode:在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.19.13 DeferWindowPos

函数功能:该函数为指定的窗口更新指定的多窗口位置结构,然后函数返回该更新结构的句柄。EndDeferWindowPos 函数使用该结构内的信息同时改变一些窗口的位置和大小。该结构由 BeginDeferWindowPos 函数创建。

函数原型:HWND DeferWindowPos(HDWP hWinPoslnfo,HWND hWnd,HWND hWndlnsertAffer, int x,int y, int cx, int Cy,UNIT uFags);

参数:

hWinPoslnfo:多窗口定位结构的句柄,该结构包含着一个或多个窗口的尺寸和定位信息,可以由函数 BeginDeterWindowPos 返回该结构或是由最近一次调用的 DeferWindowPos 函数返回。

hWnd:窗口的句柄,该窗口的更新信息存储在结构中。

hWndlnsertAfter:被定位窗口的 Z 序的前一窗口的句柄。这个参数必须为窗口句柄或下列值之一:HWND_BOTTOM:将窗口定位在 Z 序的底部。如果 hWnd 参数指定的是一个最顶层窗口,则该窗口将失去顶级位置而被排在所有其他窗口的底部。

HWND_NOTOPMOST:将窗口放置在所有顶层窗口的顶部(即在所有顶层窗口的后面)。如果窗口已经是一个非顶层窗口则此参数不起作用。

HWND_TOP:将窗口放置在 Z 序的顶部。

HWND_TOPMOST:将窗口放置在所有非顶层窗口的顶部、即使未被激活,窗口仍保持顶级位置。如果在 uFlagS 参数中指定了 SWP_NOZORDER 标志则本参数将被忽略。

x:指定窗口左上角的 X 坐标。

y:指定窗口左上角的 y 坐标。

cx:以像素定义窗口的新的宽度。

cy:以像素定义窗口的新的宽度。

uelage:指定下列影响窗口的大小和位置的值的组合:

SWP_DRAWFRAME:在窗口周围画一个边框(该边框定义在窗口类的描述中)。

SWP_FRAMECHANGED:给窗口发送一个 WM_NCCALCSIZE 消息,即使窗口的尺寸不作改变也要发送。如果未指定这个标志,则只有窗口大小 改变时才发送 WM_NCCALCSIZE 消息。

SWP_HIDEWINDOW:隐藏窗口。

SWP_NOACTIVATE:不激活窗口。如果未指定这个标志,则窗口被激活并且根据 hWndlnsertAfter 参数的设置移到或是顶部窗口的顶部或是非顶部窗口的顶部。

SWP_NOMOVE;维持当前位置(忽略 X 和 y 参数)。

SWP_NOOWNERZORDER:不改变所有者窗口在 Z 序中的位置。

SWP_NOREDRAM:不作窗口更新。如果设定了这个标志,则不发生任何窗口刷新的动作。包括不对客户区,非客户区(包括标题条和滚动条),以及由于窗口移动露出的部分父窗口进行刷新。当设定了这个标志时,应用程序一定要明确指出将原窗口清除并且重画窗口的任何部分以及父窗口需要重画的部分。

SWP_NOREPOSITION:同 SWP_NOOWNERZORDER 标志。

SWP_NOSENDCHANGING:防止窗口接受到 WM_WINDOWPOSCHANGING 消息。

SWP_NOSIZE;保持当前大小。(即忽略 CX,Xy 参数)。

SWP_NOZORDER:保持当前 Z 序(忽略 hWndlnsertAfter 参数)。

SWP_SHOWWINDOW:显示窗口。

返回值：返回值指明了被更新的多窗口定位结构。函数返回的句柄可能与传递给函数的句柄不同。这个函数返回的新句柄应在下一次调用时传递给 DeferWindowPos 函数和 EndDeferWindowPos 函数。如果调用函数时系统资源不足，则函数返回 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

备注：如果调用函数 DeferWindowPos 失败，应用程序应放弃窗口定位动作，并且不再调用 EndDeferWindowPos 函数。如果未指定 SWP_NOZORDER，系统将由 hWnd 参数指定的窗口定位于在 hWndlnsertAfer 参数指定的窗口之后的位置。如果 hWndlnsertAfsr 参数为空或为 HWND_TOP，则系统将窗口放置在 Z 序顶端。如果 hWndlnsertAfer 设为 HWND_BOTTOM 则系统将窗口放置在 Z 序的底部。

所有子窗口的坐标都是相对于父窗口客户区的左上角的坐标。

一个窗口可以通过两种方式设为顶部窗口：或是设 hWndlnsertAfter 为 HWND_TOPMOST 并确保未设置 SWP_NOZORDER 标志；或是设置窗口在 Z 序中的位置使其在所有已存在的顶端窗口的顶部。当一个非顶端窗口被设为顶端窗口时，则属于它的窗口均被置为顶端窗口，而其所有者则不变。

如果 SWP_NOACTIVATE 或 SWP_NOZORDER 均未设置（即当应用程序要求在窗口被激活的同时改变其 z 序时），hwdfnsertPos 参数只在下列情况中使用：

在 hWndlnsertAfter 参数中既未设定 HWND_TOPMOST 也未设定 HWND_NoTTOPMOST 标志；由 hWnd 指定的窗口不是激活窗口；

应用程序在将窗口设为活动窗口时应将窗口设置到 Z 序的顶部。应用程序可以不受任何限制地改变被激活窗口在 Z 序中的位置，或在激活一个窗口之后将该窗口移到顶端窗口或非顶端窗口的顶部。

如果一个顶端窗口被重定位到 Z 序的底部(HWND_BOTTOM)或任何非顶端窗口后面时将不再是顶端窗口。

一个非顶端窗口可能拥有一个顶端窗口，反之则不成立。从属的任何窗口（例如一个对话框）都设置为顶层窗口以确保所有的从属窗口都在其所有者之上。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本； Windows CE：不支持；头文件： winuser.h；库文件：user32.lib。

2.19.14 DestroyWindow

函数功能：该函数销毁指定的窗口。该函数发送 WS_DESTROY，WS_NCDESTROY 消息到窗口中以撤消该窗口并且将键盘焦点移开。该函数也销毁窗口菜单，刷新线程消息队列，销毁计时器，删除剪贴板的所有权，并断开剪贴板视窗链接（如果窗口在视窗链接的顶端）。

如果指定的窗口是父窗口或所有者窗口，DestroyWindow 在销毁父窗口或所有者窗口时自动销毁相关的子窗口和从属窗口。该函数首先销毁相关联的子窗口和从属窗口，然后销毁父窗口和所有者窗口。

DestroyWindow 也销毁由 CreateDialog 函数创建的无模式对话框。

函数原型： BOOL DestroyWindow（HWND hWnd）；

参数：

hWnd；将被销毁窗口的句柄。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：一个线程不能用 DestroyWindow 函数销毁由其他线程创建的窗口。

如果被销毁窗口是一个不具有 WS_EX_NOPARENTNOTIFY 风格的子窗口，则其父窗口将接收到一个 WM_PARENTNOTIFY 消息。

Windows CE：DestroyWindow 函数不发送 WM_NCDESTROY 消息。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：1.0 以上版本；头文件：Winuser.h；库文件：user32.lib。

2.19.15 EnableWindow

函数功能：该函数允许或禁止指定的窗口或控制接受鼠标输入或键盘输入。当输入被禁止时窗口不能接收鼠标单击和按键等类输入；当输入允许时，窗口接受所有的输入。

函数原型：BOOL EnableWindow（HWND hWnd，BOOL bEndble)；

参数：

hWnd：允许或禁止的窗口句柄。

bEnable：指定是允许还是禁止窗口。如果这个参数为 TRUE，窗口允许；如果参数为 FALSE，则窗口被禁止。

返回值：如果窗口此前曾被禁止，则返回值为非零；如果窗口此前未被禁止，则返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：如果窗口的使能状态正在改变，则在 EnableWindow 函数返回前发送一个 WM_ENABLE 消息。如果窗口已经被禁止，则它的所有子窗口都被禁止，即使未向它们发送 WM_ENABLE 消息。

在一个窗口被激活前必须是使能的。例如，如果应用程序正在显示一个无模式对话框并且禁止了它的主窗口，则应用程序在销毁对话框之前一定要使能该主窗口。否则，将有另外一个窗口接受键盘焦点并且被激活。如果一个子窗口被禁止，则在系统确定由哪一个窗口接受鼠标消息时该子窗口将被忽略。

当窗口被创建时缺省为使能状态。要创建一个初始被禁止的窗口，应用程序可以在 CreateWindowh 函数和 CreateWindowhEx 中指定 WS_DISABLED 风格。在窗口被创建后，应用程序可以使用 EnaoleWindow 函数来使能和禁止窗口。

应用程序可以使用这个函数来使能或禁止在对话框内的控制。一个被禁止的控制不能接受键盘焦点用户也不能进入该控制。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本； Windows CE：1.0 以上版本；头文件：winuser.h；库文件：user32.lib。

2.19.16 EndDeferWindowPos

函数功能：该函数在一个单一的屏幕刷新周期内同时更新一个或多个窗口的位置和大小。

函数原型：BOOL EndDeferWindowPos（ HWND hWinPoslnfo)；

参数；

hWlnPoslnfo：指向多窗口定位结构的句柄，该结构包含着一个或多个窗口的尺寸和定位信息。这个内部结构由 BeginDeferWindowPos 函数返回或由最近一次调用 DeferWindowPos 函数退回。

返回值：如果函数成功，返回值为非零：如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注；EndDeferWindowPos 函数向内部结构指定的每一个窗口发送 WM_WINDOWPOSCHANGING 和 WM_WINDOWPOSCHANGED 消息。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32.lib。

2.19.17 EnumChildProc

函数功能：该函数是由应用程序定义的，与函数 EnumChildWindows 一起使用的回调函数。它接收子窗口句柄。类型 WNDENUMPROC 定义了一个指向回调函数的指针。EnumChildProc 是一个应用程序定义的函数

名的位置标志符。

函数原型： BOOL CALLBACK EnumChildProc（HWND hwnd，LPARAM ItParam)；

参数：

hwnd：指向在 EnumChildWindows 中指定的父窗口的子窗口句柄。

Lparam：指定在 EnumChildWindows 函数给出的应用程序定义值。

返回值： 为继续枚举，回调函数必须返回 TRUE；为停止枚举，回调函数必须返回 FALSE。

备注： 回调函数可以执行任何要求的任务。

应用程序必须通过将其地址传送给 EnumChildWindows 函数来注册这个回调函数。

速查： Windows NT：3.1 以上版本；Windows：95 以上版； Windows CE：不支持；头文件：winuser.h；库文件：用户自定义。

2.19.18 EnumChildProc

函数功能： 是与函数 EnumChilWindows 一起使用的由应用程序定义的回调函数。它接收子窗口句柄。类型 WNDENUMOROC 定义了指向这个回调函数的指针。EnumChildProc 是一个应用程序定义的函数名的位置标志符。

函数原型： BOOL CALLBACK EnumChildProc（HWND hWnd，LPARAM IParam)；

参数：

hWnd:指向在 EnumChildWlndows 中定义的父窗口的子窗口句柄。

lparam：指定在 EnumchildWindows 中给出的应用程序定义值。

返回值： 为继续列举，回调函数必须返回 TRUE；为停止列举，回调函数必须返回 FALSE。

备注： 回调函数可以执行任何要求的任务。应用程序必须通过将其地址传送给 EnumChildwindows 函数来注册这个回调函数。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持：头文件：winuser.h；库文件：用户自定义。

2.19.19 EnumTreadWindows

函数功能： 该函数枚举所有与一个线程有关的非子窗口。办法是先将句柄传给每一个窗口，随后传送给应用程序定义的回调函数。EnumThreadWindows 函数继续到所有窗口枚举完为止或回调函数返回 FALSE 为止。要枚举一个特定窗口的所有子窗口；使用 EnumChjdwjndows 函数。

函数原型： BOOL EnumThreadWindows{DWORD dwThreadld,WNDENUMROCIpfn,LPARAMIPam}；

参数：

dwThreadld:标识将被列举窗口的线程。

lpfn：指向一个应用程序定义的回调函数指针，请参看 EnumThreadWndProc。

IPararm:指定一个传递给回调函数的应用程序定义值。

返回值： 如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32.Iib。

2.19.20 EnumThreadWndProc

函数功能： 该函数枚举所有与一个线程相关联的非子窗口，办法是先将句柄传送给每一个窗口，随后传送给应用程序定义的回调函数。EnumThreadWindows 函数继续直到所有窗口枚举完为止或回调函数返回 FALSE 为止。要枚举一个特定窗口的所有子窗口，使用 EnumChildWindows 函数。

函数原型： BOOL EnumThreadWindows（DWORD dwThreadld,WNDENUMPROClpfn,LPARAMlparam）;

参数：

dwThreadld：标识将被列举窗口的线程。

lpfn：指向一个应用程序定义的回调函数指针，请参看 EnumThreadWndProc。

lPararm：指定一个传递给回调函数的应用程序定义值。

返回值： 如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 以上版本； Windows：95 以上版本； Windows CE：不支持：头文件：winuser.h；库文件：user32.Iib。

2.19.21 EnumWindows

函数功能： 该函数枚举所有屏幕上的顶层窗口，办法是先将句柄传给每一个窗口，然后再传送给应用程序定义的回调函数。EnumThreadWindows 函数继续到所有顶层窗口枚举完为止或回调函数返回 FALSE 为止**函数原型：** BOOL EnumWindows（WNDENUMPROC lpEnumFunc，LPARAM lParam）;

参数：

lpEnumFunc：指向一个应用程序定义的回调数指针，请参看 EnumWindowsProc。

lPararm：指定一个传递给回调函数的应用程序定义值。

返回值： 如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注： EnumWindows 函数不列举子窗口。

在循环体中调用这个函数比调用 GetWindow 函数更可靠。调用 GetWindow 函数中执行这个任务的应用程序可能会陷入死循环或指向一个已被销毁的窗口的句柄。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：1.0 以上版本；头文件：winuser.h；库文件：user32.lib。

2.19.22 EnumWindowsProc

函数功能：该函数是一个与 EnumWindows 或 EnumDesktopWindows 一起使用的应用程序定义的回调函数。它接收顶层窗口句柄。WNDENUMPROC 定义一个指向这个回调函数的指针。EnumWindowsProc 是应用程序定义函数名的位置标志符。

函数原型： BOOL CALLBACK EnumWindowsProc（HWND hwnd，LPARAM IParam）;

参数：

hwnd：顶层窗口句柄。

lParam：指定在 EnumWIndowsh 或 EnumDesktopWindows 中的应用程序定义值。

返回值： 为继续列表，回调函数必须返回 TRUE；若停止列表，它必须返回 FALSE。

备注：应用程序必须通过传递给 EnumWindows 或 EnumDesktopWindows 应用程序地址来注册这个回调函

数。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本:头文件:winuser.h;库文件：用户自定义。

2.19.23 FindWindow

函数功能：该函数获得一个顶层窗口的句柄，该窗口的类名和窗口名与给定的字符串相匹配。这个函数不查找子窗口。在查找时不区分大小写。

函数型：HWND FindWindow（LPCTSTR IpClassName，LPCTSTR IpWindowName）；

参数：

IpClassName ：指向一个指定了类名的空结束字符串，或一个标识类名字符串的成员的指针。如果该参数为一个成员，则它必须为前次调用 theGlobafAddAtom 函数产生的全局成员。该成员为 16 位，必须位于 IpClassName 的低 16 位，高位必须为 0。

IpWindowName：指向一个指定了窗口名（窗口标题）的空结束字符串。如果该参数为空，则为所有窗口全匹配。

返回值：如果函数成功，返回值为具有指定类名和窗口名的窗口句柄；如果函数失败，返回值为 NULL。

若想获得更多错误信息，请调用 GetLastError 函数。

备注：Windows CE：若类名是一个成员，它必须是从 RegisterClass 返回的成员。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:Winuser.h;库文件：user32.lib； Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.19.24 FindWindowEx

函数功能：该函数获得一个窗口的句柄，该窗口的类名和窗口名与给定的字符串相匹配。这个函数查找子窗口，从排在给定的子窗口后面的下一个子窗口开始。在查找时不区分大小写。

函数原型：HWND FindWindowEx（HWND hwndParent，HWND hwndChildAfter，LPCTSTR lpszClass，LPCTSTR lpszWindow）；

参数；

hwndParent：要查找子窗口的父窗口句柄。

如果 hwnjParent 为 NULL，则函数以桌面窗口为父窗口，查找桌面窗口的所有子窗口。

Windows NT5.0 and later：如果 hwndParent 是 HWND_MESSAGE，函数仅查找所有消息窗口。

hwndChildAfter ：子窗口句柄。查找从在 Z 序中的下一个子窗口开始。子窗口必须为 hwndPareRt 窗口的直接子窗口而非后代窗口。如果 HwndChildAfter 为 NULL，查找从 hwndParent 的第一个子窗口开始。如果 hwndParent 和 hwndChildAfter 同时为 NULL，则函数查找所有的顶层窗口及消息窗口。

lpszClass：指向一个指定了类名的空结束字符串，或一个标识类名字符串的成员的指针。如果该参数为一个成员，则它必须为前次调用 theGlobaIAddAtom 函数产生的全局成员。该成员为 16 位，必须位于 lpClassName 的低 16 位，高位必须为 0。

lpszWindow：指向一个指定了窗口名（窗口标题）的空结束字符串。如果该参数为 NULL，则为所有窗口全匹配。**返回值：**如果函数成功，返回值为具有指定类名和窗口名的窗口句柄。如果函数失败，返回值为 NULL。

若想获得更多错误信息，请调用 GetLastError 函数。

速查 NT：4.0 对以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32. lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.19.25 GetClientRect

函数功能： 该函数获取窗口客户区的坐标。客户区坐标指定客户区的左上角和右下角。由于客户区坐标是相对子窗口客户区的左上角而言的，因此左上角坐标为（0，0）

函数原型： BOOL GetClientRect（HWND hWnd，LPRECT lpRect）；

参数：

GetLastError 函数。

备注： Windows CE：命令条包含在客户区中。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本； Windows CE：1.0 以上版本；头文件：winuser.h；库文件：user32.lib

2.19.26 GetDesktopWindow

函数功能： 该函数返回桌面窗口的句柄。桌面窗口覆盖整个屏幕。桌面窗口是一个要在其上绘制所有的图标和其他窗口的区域。

函数原型： HWND GetDesktopWindow（VOID）

参数：无。

返回值： 函数返回桌面窗口的句柄。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本：Windows CE：不支持；头文件：Winuser.h；库文件：user32.lib。

2.19.27 GetForegroundWindow

函数功能： 该函数返回前台窗口（用户当前工作的窗口）。系统分配给产生前台窗口的线程一个稍高一点的优先级。

函数原型： HWND GetForegroundwindow（VOID）

参数：无。

返回值： 函数返回前台窗回的句柄。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本：Windows CE：1.0 以上版本：头文件：Winuser.h；库文件：user32.lib。

2.19.28 GetLastActivePopup

函数功能： 该函数确定指定窗口中的哪一个弹出式窗口是最近活动的窗口。

函数原型： HWND GetLastActivePopup（HWND hWnd）；

参数：

hWnd:所有者窗口句柄。

返回值： 返回值标识了最近活动的弹出式窗口的句柄。如果满足下列任一条件，则返回值与参数 hWnd 相同：由 hWnd 指定的窗口是最近活动的：由 hWnd 指定的窗口不拥有任何弹出式窗口；由 hWnd 指定的窗口不是顶层窗口或它属于其他窗口。

速查： Windows：3.1 以上版本；Windows：95 以上版本：Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.19.29 GetNextWindow

函数功能： 该函数返回 z 序中的前一个或后一个窗口的句柄。下一窗口在指定窗口的下面，前一窗口则在上面。如果指定的窗口是顶端窗口，该函数返回下一个（或前一个）顶端窗口的句柄。如果指定的窗口是顶层窗口，函数返回下一个（或前一个）顶层窗口的句柄。如果函数是子窗口，则函数搜索下一个或前一个子窗口的句柄。

函数原型： HWND GetNextWindow（HWND hWnd，UNIT wCmd）；

参数：

hWnd：一个窗口的句柄。窗口句柄在 wCmd 参数的基础上获得的相对于这个窗口的句柄。

wCmd:指明窗口返回的是前一窗口的句柄还是后一窗口的句柄。该参数可以是下列两个值之一：

GW HWNONEXT：返回在给定窗口的下面窗口的句柄。

GW_HWNDPREV：返回在给定窗口的上面窗口的句柄。

返回值： 如果函数成功，返回值是前一窗口（或后一窗口）的句柄。如果前后窗口不存在，则返回值为 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

备注： 在设定了 GW_HWNDNEXT 或 GW_GETPREV 标志时，调用该函数与调用 GetWindow 函数相同。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本：Windows CE：不支持：头文件：Winuer.h；库文件：user32.lib。

2.19.30 GetParent

函数功能： 该函数获得一个指定子窗口的父窗口句柄。

函数原型： HWND GetParent（HWND hWnd）；

参数：

hWnd:子窗口句柄，函数要获得该子窗口的父窗口句柄。

返回值： 如果函数成功，返回值为父窗口句柄。如果窗口无父窗口，则函数返回 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

备注： WindowsCE：Windows CE1.0 版本不支持除了对话框之外的所属子窗口。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本:头文件:Winuser.h：库文件：user32.lib。

2.19.31 GetTopWindow

函数功能： 该函数检查与特定父窗口相联的子窗口 z 序，并返回在 z 序顶部的子窗口的句柄。

函数原型： HWND GetTopWindow（HWND hWnd）；

参数：

hWnd:被查序的父窗口的句柄。如果该参数为 NULL，函数返回 Z 序顶部的窗口句柄。

返回值;如果函数成功,返回值为在 Z 序顶部的子窗口句柄。如果指定的窗口无子窗口,返回值为 NULL。若想获得更多错误信息，请调用 GetLastError 函数。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本：Windows CE：不支持；头文件：Winuser.h：库文件；user32.lib。

2.19.32 GetWindow

函数功能： 该函数返回与指定窗口有特定关系（如 Z 序或所有者）的窗口句柄。

函数原型： HWND GetWindow（HWND hWnd，UNIT nCmd）；

参数：

hWnd:窗口句柄。要获得的窗口句柄是依据 nCmd 参数值相对于这个窗口的句柄。

nCmd:说明指定窗口与要获得句柄的窗口之间的关系。该参数值可以是下列之一：

GW_CHILD：如果指定窗口是父窗口，则获得的是在 Z 序顶端的子窗口的句柄，否则为 NULL。函数仅检查指定父窗口的子窗口，不检查继承窗口。

GW_ENABLEDPOUP：（WindowsNT 5.0）返回的句柄标识了属于指定窗口的处于使能状态弹出式窗口（检索使用第一个由 GW_HWNDNEXT 查找到的满足前述条件的窗口）；如果无使能窗口，则获得的句柄与指定窗口相同。

GW_HWNDFIRST：返回的句柄标识了在 Z 序最高端的相同类型的窗口。如果指定窗口是最高端窗口，则该句柄标识了在 Z 序最高端的最高端窗口；如果指定窗口是顶层窗口，则该句柄标识了在 z 序最高端的顶层窗口：如果指定窗口是子窗口，则句柄标识了在 Z 序最高端的同属窗口。

GW_HWNDLAST:返回的句柄标识了在 z 序最低端的相同类型的窗口。如果指定窗口是最高端窗口，则该柄标识了在 z 序最低端的最高端窗口：如果指定窗口是顶层窗口，则该句柄标识了在 z 序最低端的顶层窗口；如果指定窗口是子窗口，则句柄标识了在 Z 序最低端的同属窗口。

GW_HWNDNEXT:返回的句柄标识了在 Z 序中指定窗口下的相同类型的窗口。如果指定窗口是最高端窗口，则该句柄标识了在指定窗口下的最高端窗口：如果指定窗口是顶层窗口，则该句柄标识了在指定窗口下的顶层窗口；如果指定窗口是子窗口，则句柄标识了在指定窗口下的同属窗口。

GW HWNDPREV:返回的句柄标识了在 Z 序中指定窗口上的相同类型的窗口。如果指定窗口是最高端窗口，则该句柄标识了在指定窗口上的最高端窗口；如果指定窗口是顶层窗口，则该句柄标识了在指定窗口上的顶层窗口；如果指定窗口是子窗口，则句柄标识了在指定窗口上的同属窗口。

GW_OWNER：返回的句柄标识了指定窗口的所有者窗口（如果存在）。

返回值： 如果函数成功，返回值为窗口句柄；如果与指定窗口有特定关系的窗口不存在，则返回值为 NULL。

若想获得更多错误信息，请调用 GetLastError 函数。

备注： 在循环体中调用函数 EnumChildWindow 比调用 GetWindow 函数可靠。调用 GetWindow 函数实现该任务的应用程序可能会陷入死循环或退回一个已被销毁的窗口句柄。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件：user32.lib。

2.19.33 GetWindowPlacement

函数功能： 该函数返回指定窗口的显示状态以及被恢复的、最大化的和最小化的窗口位置。

函数原型： BOOL GetWindowPlacement（HWND hWnd，WINDOWPLACEMENT*lpwndpl）；

参数：

hWnd: 窗日句柄。

lpwndpl：指向 WINDOWPLACEMENT 结构的指针，该结构存贮显示状态和位置信息。

在调用 GetWindowPlacement 函数之前，将 WINDOWPLACEMENT 结构的长度设为 sizeof(WIDNOWPLACEMENT)。如果 lpwndpl－＞length 设置不正确则函数 GetWindowPlacement 将失败。

返回值；如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调

用 GetlastError 函数。

备注：由该函数获得的 WINDOWPLACEMENT 结构的 flag 单元总为 0。如果 hWnd 的窗口被最大化，则 showCmd 元为 SHOWMZAXMIZED，如果窗口被最小化，则 showCmd 元为 SHOWMINIMIZED，除此之外为 SHOWNORMN，WINDOWPLACEMENT 长度单元必须置为 sizeOf(WINDOWPLACEMENT)，如果参数设置不正确，函数返回 FALSE。查看设置窗口位置坐标的正确信息，参看 WINDOWPLACEMENT。

速查：Windows Nt：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：Winuser.h；库文件：User32.lib。

2.19.34 GetWindowRect

函数功能：该函数返回指定窗口的边框矩形的尺寸。该尺寸以相对于屏幕坐标左上角的屏幕坐标给出。

函数原型： BOOL GetWindowRect（HWND hWnd，LPRECTlpRect）；

参数：

hWnd:窗口句柄。

lpRect：指向一个 RECT 结构的指针，该结构接收窗口的左上角和右下角的屏幕坐标。

返回值：如果函数成功，返回值为非零：如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:Winuser.h；库文件：User32.lib。

2.19.35 GetWindowText

函数功能： 该函数将指定窗口的标题条文本（如果存在）拷贝到一个缓存区内。如果指定的窗口是一个控制，则拷贝控制的文本。但是，GetWindowTeXt 不能接收在其他应用程序中的控制文本。

函数原型： Int GetWindowText（HWND hWnd，LPTSTR lpString，Int nMaxCount）；

参数：

hWnd:带文本的窗口或控制的句柄。

IpString：指向接收文本的缓冲区的指针。

nMaxCount:指定要保存在缓冲区内的字符的最大个数，其中包含 NULL 字符。如果文本超过界限，它就被截断。

返回值：如果函数成功，返回值是拷贝的字符串的字符个数，不包括中断的空字符；如果窗口无标题栏或文本，或标题栏为空，或窗口或控制的句柄无效，则返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

函数不能返回在其他应用程序中的编辑控制的文本。

备注： 如果目标窗口属于当前进程，GetWindowText 函数给指定的窗口或控制发送 WM_GETTEXT 消息。如果目标窗口属于其他进程，并且有一个窗口标题，则 GetWindowTeXt 返回窗口的标题文本，如果窗口无标题，则函数返回空字符串。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本:Windows CE:1.0 以上版本;头文件:Winuser.h;库文件：user32.lib：Unicode:在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.19.36 GetWindowTextLength

函数功能： 该函数返回指定窗口的标题文本（如果存在）的字符长度。如果指定窗口是一个控制，函数将返回控制内文本的长度。但是 GetWindowTextLength 函数不能返回在其他应用程序中的控制的文本长度。

函数原型： nit GetWindowTextLent（HWND hWnd);

参数：

hWnd：窗口或控制的句柄。

返回值： 如果函数成功，返回值为文本的字符长度。在一定的条件下，返回值可能比实际的文本长度大。请参看说明。如果窗口无文本，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注： 如果目标窗口属于当前进程，GetWindowTextLength 函数给指定的窗口或控制发送 WM_GETTEXT 消息。

在一定的条件下，函数 GetWindowTextLength 的返回值可能比实际的文本长度大。这是由于 ANSI 和 Unlcode 的混和使用以及系统允许 DBCS 字符在文本内存在的原因，但是函数返回值要至少与文本的实际长度相等，因此可以利用这一点指导缓存区的分配。在应用程序既使用 ANSI 函数又使用 Unicode 的普通对话框时就会有缓存分配的问题；同样，当应用程序在一个 Unicode 的窗口过程中使用了 ANSI 的 GetWindowTextLength 函数，或在一个 ANSI 的窗口过程中使用了 Unicode 的 GetWindowTextLength 函数的时候也有缓存分配的问题。查看 ANSI 和 Vnicode 函数，参考 Wind32 函数 prototypes。

要获得文本的实际长度，使用 WM_GETTEXT, LB_GETTEXT 或 CB_GETLBTBTEXT 消息或 GetWindowText 函数。

速查：Windows NT:3.1 以上版本:Windows:95 以上版本:Windows CE:1.0 以上版本;头文件:Winuser.h;库文件：user32.lib;Unicode：在 Windows NT 上实现为山 Unicode 和 ANSI 两种版本。

2.19.37 GetWlndowThreadprocessld

函数功能： 该函数返回创建指定窗口线程的标识和创建窗口的进程的标识符，后一项是可选的。

函数原型；DWORD GetWindowThreadProcessld（HWND hwnd，LPDWORD lpdwProcessld);

参数：

hWnd:窗口句柄。

lpdwProcessld:接收进程标识的 32 位值的地址。如果这个参数不为 NULL，GetWindwThreadProcessld 将进程标识拷贝到这个 32 位值中，否则不拷贝。

返回值： 返回值为创建窗口的线程标识。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件；user32.lib。

2.19.38 IsChild

函数功能： 该函数测试一个窗口是否是指定父窗口的子窗口或后代窗口。如果该父窗口是在父窗口的链表上则子窗口是指定父窗口的直接后代。父窗口链表从原始层叠窗口或弹出窗口一直连到该子窗口。

函数原型： BOOL IsChild（HWND hWndParant，HWND hWnd);

参数：

hWndparant:父窗口句柄。

hWnd：将被测试的窗口句柄。

返回值：如果窗口是指定窗口的子窗口或后代窗口，则退回值为非零。如果窗口不是指定窗口的子窗口或后代窗口，则退回值为零。

速查：Windows NT:3.1 以上版本:Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件：user32.Iib。

2.19.39 IsIconic

函数功能：该函数确定给定窗口是否是最小化（图标化）的窗口。

函数原型：BOOL IsIconic（HWND hWnd);

参数：

hWnd：被测试窗口的句柄。

返回值：如果窗口已图标化，返回值为非零；如果窗口未图标化，返回值为零。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h;库文件：user32.Iib。

2.19.40 IsWindow

函数功能：该函数确定给定的窗口句柄是否识别一个已存在的窗口。

因数原型：BOOL isWindow（HWND hWnd);

参数：

hWnd:被测试窗口的句柄。

返回值：如果窗口句柄标识了一个已存在的窗口，返回值为非零；如果窗口句柄未标识一个已存在窗口，返回值为零。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuserhs;库文件：User32.lib。

2.19.41 IsWindowUnicode

函数功能：该函数确定指定的窗口是否是一个本地 Unicode 窗口。

函数原型： BOOL lswindowUnicode（HWND hwndJ;

参数：

hWnd:被测试窗口的句柄。

返回值：如果窗口是一个本地 Unicode 窗口，返回值为非零；如果窗口不是一个本地 Unicode 窗口，返回值为零，同时说明窗口是一个 ANSI 窗口。

备注；一个窗口的字符集是由函数 RegosterClass 决定的。如果窗口类是以 ANSI 版的 RegisterClass（ RegjsterClassA）注册的，则窗口字符集是 ANSI 的；如果窗口类是以 Unicode 版的 Registerclass（ RegisterClassW）注册的，则窗口字符集是 Unicode。

系统为窗口消息自动作 Unicode 和 ANSI 的双向翻译。例如，如果一个使用 Unicode 字符集的窗口测到一个 ANSI 窗口消息，则系统在调用窗口过程之前先将该消息转换为 Unicode 消息。系统调用 lsWindowUnicode 函数决定是否翻译消息。

速查：Windows NT：3.1 以上版本；Windows： 95 以上版本；Windows CE：不支持；头文件：Winuser.h;

库文件：user32.lib。

2.19.42 IsWindowVlslble

函数功能： 该函数获得给定窗口的可视状态。;

函数原型： BOOL IsWindowVisible (HWND hWnd);

参数;

hWnd：被测试窗口的句柄。

返回值：如果指定的窗口及其父窗口具有 WS_VISIBLE 风格，返回值为非零；如果指定的窗口及其父窗口不具有 WS_VISIBLE 风格，返回值为零。由于返回值表明了窗口是否具有 Ws_VISIBLE 风格，因此，即使该窗口被其他窗口遮盖，函数返回值也为非零。

备注： 窗口的可视状态由 WS_VISIBLE 位指示。当设置了 WS_VISIBLE 位，窗口就可显示，而且只要窗口具有 WS_VISIBLE 风格，任何画在窗口的信息都将被显示。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件：user32.lib。

2.19.43 IsZoomed

函数功能： 该函数确定窗口是否是最大化的窗口。

函数原型： BOOL IsZoomed (HWND hWnd);

参数：

hWnd:被测试窗口的句柄。

返回值： 如果窗口已最大化，则返回值为非零；如果窗口未最大化，则返回值为零。

速查： Windows NT:3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：Winuer.h;库文件：User32.Iib。

2.19.44 MoveWindow

函数功能：该函数改变指定窗口的位置和尺寸。对于顶层窗口，位置和尺寸是相对于屏幕的左上角的：对于子窗口，位置和尺寸是相对于父窗口客户区的左上角坐标的。

函数原型： BOOL MoveWindow (HWND hWnd.int x.int y,int nWidth,int nHeight,BOOL BRePaint);

参数：

hWnd：窗口句柄。

x：指定窗口的新位置的左边界。

Y：指定窗口的新位置的顶部边界。

nWidth：指定窗口的新的宽度。

nHaight：指定窗口的新的高度。

bRepaint:确定窗口是否被刷新。如果该参数为 TRUE,窗口接收一个 WM_PAINT 消息;如果参数为 FALSE，不发生任何刷新动作。它适用于客户区，非客户区（包括标题栏和滚动条），及由于移动子窗口而露出的父窗口的区域。如果参数为 FALSE，应用程序就必须明确地使窗口无效或重画该窗口和需要刷新的父窗口。

返回值： 如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：如果 bRepaint 为 TRUE，系统在窗口移动后立即给窗口过程发送 WM_PAINT 消息(即由 MoveWindow 函数调用 UPdateWindow 函数)。如果 bRepaint 为 FALSE，系统将 WM_PAINT 消息放在该窗口的消息队列中。消息循环只有在派遣完消息队列中的其他消息时才派遣 WM_PAINT 消息。

MoveWindow 给窗口发送 WM_WfNOWPOSCHANGING，WM_WINDOWPOSCHANGED，WM_MOVE，WM_SIZE 和 WM_NCCALCSIZE 消息，

速查：Windows NT:3.1 以上版本:Windows:95 以上版本;Windows CE:1.0 以上版本:头文件:winuser.h;库文件：user32.lib。

2.19.45 Openlcon

函数功能：该函数将一个最小化窗口恢复到原来的位置和尺寸并且激活该窗口。

函数原型：BOOL Openlcon（HWNDhWnd);

参数：

hWnd:被恢复与激活的窗口的句柄。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：Openlcon 向给出的窗口发送 WM_QUERYOPEN 消息。

速查：Windows NT:3.1 以上版本；Windows:95 以上版本；Windows CE：不支持：头文件：winuser.h;库文件：user32.lib。

2.19.46 SetForegroundWindow

函数功能：该函数将创建指定窗口的线程设置到前台，并且激活该窗口。键盘输入转向该窗口，并为用户改各种可视的记号。系统给创建前台窗口的线程分配的权限稍高于其他线程。

函数原型：BOOL SetForegroundWindow（HWND hWnd）

参数：

hWnd：将被激活并被调入前台的窗口句柄。

返回值：如果窗口设入了前台，返回值为非零；如果窗口未被设入前台，返回值为零。

备注：前台窗口是 z 序顶部的窗口，是用户的工作窗口。在一个多任务优先抢占环境中，应让用户控制前台窗口。

Windows NT 5.0：当用户在另一个窗口中工作时，应用程序不能强行设置一个窗口到前台。相反，SetForeground 函数将会激活窗口并且调用 FlashWindowEx 函数通知用户。

Windows CE：拥有窗口的线程不具有优先启动权。

速查：Windows NT;3.1 以上版本;Windows;95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件：user32.lib。

2.19.47 SetParent

函数功能：该函数改变指定子窗口的父窗口。

函数原型：HWND SetPalrent（HWND hWndChild,HWND hWndNewParent);

参数：

hWndChild:子窗口句柄。

hWndNewParent:新的父窗口句柄。如果该参数是 NULL，则桌面窗口就成为新的父窗口。在 WindowsNT5.0 中，如果参数为 HWND_MESSAGE,则子窗口成为消息窗口。

返回值：如果函数成功，返回值为子窗口的原父窗口句柄；如果函数失败，返回值为 NULL。若想获得多错误信息，请调用 GetLastError 函数。

备注：应用程序可以使用 SetParent 函数来设置弹出式窗口，层叠窗口或子窗口的父窗口。新的窗口与窗口必须属于同一应用程序。

如果参数 hWndChild 标识的窗口是可见的，系统将执行适当的重画和刷新动作。

由于兼容的原因，对于将改变父窗口的子窗口，SetParent 函数并不改变该子窗口的 WS_CHILO WS_POPUP 风格。所以，如果 hWndNewParent 参数为 NULL，就应在调用 SetParent 函数之后清空 WS_CHILD 位并且设置为 WS_POPUP 风格。相反的，如果 hWndNewParent 参数不为 NULL 并且在此之前窗口是桌面窗口的子窗口，就应在调用 SetParent 函数之前清空 WS_POPUP 位井设置 WS_CHILD 风格。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：1.0 以上版本：头文件：Winuser.h；库文件：user32.lib。

2.19.48 SetWindowLong

函数功能：该函数改变指定窗口的属性。函数也将在指定偏移地址的一个 32 位值存入窗口的额外窗口存。

函数原型：LONG SetWindowLong（HWN hWnd，int nlndex. LONG dwNewLong)；

参数：

hWnd:窗口句柄，及间接声明的该窗口所属的类。

nlndex:给出了要设置的值的零起点的偏移地址。有效值为从 0 到额外窗口存储空间的字节数一 4。例如

如果指定了 12 位或更多位字节的额外内存，则 32 位值的索引值应为第 3 个 32 位值的索引位 8。设置其他值，要指定下列中的一个值：

GWL_EXSTYLE；设置一个新的扩展窗口风格。GWL_STYLE：设置一个新的窗口风格。

GWL_WNDPROC:为窗口过程设置一个新的地址。

GWL_HINSTANCE：设置一个新的应用程序事例句柄。GWL_ID：为窗口设置一个新的标识。

GWL_USERDATA：设置与窗口有关的 32 位值。每个窗口都有一个对应的 32 位值供创建该窗口的应用程序使用。

当 hWnd 参数标识了一个对话框是可使用下列值的：

DWL_DLGPROC：设置对话框过程的新地址。

DWL_MSGRESULT：设置在对话框过程中处理的消息返回值。

DWL_USER：设置新的额外信息，该信息仅为应用程序所有，例如句柄或指针。

dWNewLong：指定替换值。

返回值：如果函数成功，返回值为给定的 32 位整数的原来的值。如果函数失败，返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

如果给定的 32 位值的原来的值是 0，并且函数成功，则返回值为 0。但是这时函数并不清除最后的错误信息，这就很难判断函数成功与否。这时，就应在调用 SetWindowLong 之前调用 calingsetLastEm 函数清除最后的错误信息。这样，如果函数失败就会返回 0，并且 GetLastError 也返回一个非零值。

备注：如果由 hWnd 参数指定的窗口与调用线程不属于同一进程，将导致 SetWindowLong 函数失败。指定的窗日数据是在缓存中保存的，因此在调用 SetWindowLong 之后再调用 SetWindowPos 函数才 0

SetWindowLong 函数所作的改变生效。

如果使用带 GWL_WNDPROC 索引值的 SetWindowLong 函数替换窗口过程，则该窗日过程必须 WindowProc

回调函数说明部分指定的指导行一致。

如果使用带 DWL_MSGRESULT 索引值的 SetWlndowLong 函数来设置由一个对话框处理的消息的和值，应在此后立即返回 TRUE。否则，如果又调用了其他函数而使对话框过程接收到一个窗口消息，**返回值：** 如果函数成功，返回值为给定的 32 位整数的原来的值。如果函数失败，返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

如果给定的 32 位值的原来的值是 0，并且函数成功，则返回值为 0。但是这时函数并不清除最后的错误信息，这就很难判断函数成功与否。这时，就应在调用 SetWindowLong 之前调用 calingsetLastEm 函数清除最后的错误信息。这样，如果函数失败就会返回 0，并且 GetLastError 也返回一个非零值。

备注： 如果由 hWnd 参数指定的窗口与调用线程不属于同一进程，将导致 SetWindowLong 函数失败。

指定的窗口数据是在缓存中保存的，因此在调用 SetWindowLong 之后再调用 SetWindowPos 函数才 0 SetWindowLong 函数所作的改变生效。

如果使用带 GWL_WNDPROC 索引值的 SetWindowLong 函数替换窗口过程，则该窗日过程必须 WindowProc 回调函数说明部分指定的指导行一致。

如果使用带 DWL_MSGRESULT 索引值的 SetWlndowLong 函数来设置由一个对话框处理的消息的和值，应在此后立即返回 TRUE。否则，如果又调用了其他函数而使对话框过程接收到一个窗口消息，套的窗口可能消息可能改写使用 DWL_MSGRESULT 设定的返回值。

可以使用带 GWL_WNDPROC 索引值的 SetWindowLohg 函数创建一个窗口类的子类，该窗口类是用于创建该窗口的类。一个应用程序可以一个系统类为子类，但是不能以一个其他进程产生的窗口类为于类。

SetWindowLong 函数通过改变与一个特殊的窗口类相联系的窗口过程来创建窗口子类。从而使系统调用新的窗口过程而不是以前定义的窗口过程。应用程序必须通过调用 CallWindowProc 函数向前窗口传递未被新窗口处理的消息。允许应用程序创建一个窗口过程链。可以通过对 RegisterClassEx 函数中使用的 WNDCLASSEX 结构的 chWndExtra 单元指定一个非零值来保留额外窗口内存。

不能通过调用带 GWL_HWNDPARENT 索引值的 SetWindowLong 的函数来改变子窗口的父窗口。应使用 SetParent 函数。

WindowsCE：nlndex 参数必须是 4 个字节的倍数；不支持 Unaligned access。

不支持下列 nlndex 参数值：

GWL_HINSTANCE；GWL_HWNDPARENT;GWL_USERDATA

WindowsCE2.0 版支持在索引参数中的 DWL_DLGPROC 值，但是 WindowsCE1.0 不支持。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本:库文件:user32.lib;

Unicode;在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.19.49 SetWindowPlacement

函数功能： 该函数设置指定窗口的显示状态和恢复，最大化，最小化位置。

函及原型；BOOL SetwlndowPlacement（HWND hWnd，CONST WINDOWPLACEMENT★lpwndpl)；

参数：

hWnd：窗口句柄。

lpwndpl:指向一个 WINDOWPLACEMWNT 结构的指针，该结构给出了新的显示状态和窗口位置。

在调用函数 SetWindowPlacement 之前，将 WINDOWPLACEMWNT 结构的长度单元置为 sizeof（WINDOWPLACEMENT)。如果 lpwndpl－＞length 设置不正确，函数 SetWindowPlacement 将失败。

返回值： 如果函数成功，返回值为非零。如果函数失败，返回值为零。若想获得更多错误信息，请调用 callGetLastErro 函数。

备注： 如果在 WIDNOWPLACEMENT 中指定的信息使窗口完全显示在屏幕之外，系统自动调整坐标以使窗口可见，兼顾屏幕设置和多种监视器配置。

WINDOWPLACEMENT 的长度成员信息设置为 sizeof（WINDOWPLACEMENT），如果设置不正确，函数将返回 FLASE。查看窗口位置坐标的信息，参看 WINDOWPLACEMENT。

速查：Windows NT：3.1 以上版本；Windows；95 以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.19.50 SetWindowPos

函数功能：该函数改变一个子窗口，弹出式窗口式顶层窗口的尺寸，位置和 Z 序。子窗口，弹出式窗口，及顶层窗口根据它们在屏幕上出现的顺序排序、顶层窗口设置的级别最高，并且被设置为 Z 序的第一个窗口。

函数原型：BOOL SetWindowPos（HWN hWnd，HWND hWndlnsertAfter,int X，int Y,int cx，int cy,UNIT. Flags）;

参数：

hWnd:窗口句柄。

hWndlnsertAfter:在 z 序中的位于被置位的窗口前的窗口句柄。该参数必须为一个窗口句柄，或下列值之一：

HWND_BOTTOM：将窗口置于 Z 序的底部。如果参数 hWnd 标识了一个顶层窗口，则窗口失去顶级位置，并且被置在其他窗口的底部。

HWND_DOTTOPMOST：将窗口置于所有非顶层窗口之上（即在所有顶层窗口之后）。如果窗口已经是非顶层窗口则该标志不起作用。

HWND_TOP:将窗口置于 Z 序的顶部。

HWND_TOPMOST:将窗口置于所有非顶层窗口之上。即使窗口未被激活窗口也将保持顶级位置。

查 g 看该参数的使用方法，请看说明部分。

x：以客户坐标指定窗口新位置的左边界。

Y：以客户坐标指定窗口新位置的顶边界。

cx:以像素指定窗口的新的宽度。

cy：以像素指定窗口的新的高度。

uFlags:窗口尺寸和定位的标志。该参数可以是下列值的组合：

SWP_ASNCWINDOWPOS：如果调用进程不拥有窗口，系统会向拥有窗口的线程发出需求。这就防止调用线程在其他线程处理需求的时候发生死锁。

SWP_DEFERERASE：防止产生 WM_SYNCPAINT 消息。

SWP_DRAWFRAME：在窗口周围画一个边框（定义在窗口类描述中）。

SWP_FRAMECHANGED：给窗口发送 WM_NCCALCSIZE 消息，即使窗口尺寸没有改变也会发送该消息。如果未指定这个标志，只有在改变了窗口尺寸时才发送 WM_NCCALCSIZE。

SWP_HIDEWINDOW;隐藏窗口。

SWP_NOACTIVATE：不激活窗口。如果未设置标志，则窗口被激活，并被设置到其他最高级窗口或非最高级组的顶部（根据参数 hWndlnsertAfter 设置）。

SWP_NOCOPYBITS：清除客户区的所有内容。如果未设置该标志，客户区的有效内容被保存并且在窗口尺寸更新和重定位后拷贝回客户区。

SWP_NOMOVE：维持当前位置（忽略 X 和 Y 参数）。

SWP_NOOWNERZORDER：不改变 z 序中的所有者窗口的位置。

SWP_NOREDRAW:不重画改变的内容。如果设置了这个标志，则不发生任何重画动作。适用于客户区和非客户区（包括标题栏和滚动条）和任何由于窗回移动而露出的父窗口的所有部分。如果设置了这个标志，应用程序必须明确地使窗口无效并区重画窗口的任何部分和父窗口需要重画的部分。

SWP_NOREPOSITION：与 SWP_NOOWNERZORDER 标志相同。

SWP_NOSENDCHANGING：防止窗口接收 WM_WINDOWPOSCHANGING 消息。

SWP_NOSIZE：维持当前尺寸（忽略 cx 和 Cy 参数）。

SWP_NOZORDER：维持当前 Z 序（忽略 hWndlnsertAfter 参数）。

SWP_SHOWWINDOW：显示窗口。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误消息，请调用 GetLastError 函数。

备注：如果设置了 SWP_SHOWWINDOW 和 SWP_HIDEWINDOW 标志，则窗口不能被移动和改变大小。如果使用 SetWindowLoog 改变了窗口的某些数据，则必须调用函数 SetWindowPos 来作真正的改变。使用下列的组合标志：SWP_NOMOVEISWP_NOSIZEISWP_FRAMECHANGED。

有两种方法将窗口设为最顶层窗口：一种是将参数 hWndlnsertAfter 设置为 HWND_TOPMOST 并确保没有设置 SWP_NOZORDER 标志；另一种是设置窗口在 Z 序中的位置以使其在其他存在的窗口之上。当一个窗口被置为最顶层窗口时，属于它的所有窗口均为最顶层窗口，而它的所有者的 z 序并不改变。

如果 HWND_TOPMOST 和 HWND_NOTOPMOST 标志均未指定，即应用程序要求窗口在激活的同时改变其在 Z 序中的位置时，在参数 hWndinsertAfter 中指定的值只有在下列条件中才使用：

在 hWndlnsertAfter 参数中没有设定 HWND_NOTOPMOST 和 HWND_TOPMOST 标志。

由 hWnd 参数标识的窗口不是激活窗口。

如果未将一个非激活窗口设定到 z 序的顶端，应用程序不能激活该窗口。应用程序可以无任何限制地改变被激活窗口在 Z 序中的位置，或激活一个窗口并将其移到最高级窗口的顶部或非最高级窗口的顶部。

如果一个顶层窗口被重定位到 z 序的底部（HWND_BOTTOM）或在任何非最高序的窗口之后，该窗口就不再是最顶层窗口。当一个最顶层窗口被置为非最顶级，则它的所有者窗口和所属者窗口均为非最顶层窗口。

一个非最顶端窗口可以拥有一个最顶端窗口，但反之则不可以。任何属于顶层窗口的窗口（例如一个对话框）本身就被置为顶层窗口，以确保所有被属窗口都在它们的所有者之上。

如果应用程序不在前台，但应该位于前台，就应调用 SetForegroundWindow 函数来设置。

Windows CE：如果这是一个可见的顶层窗口，并且未指定 SWP_NOACTIVATE 标志，则这个函数将激活窗口、如果这是当前的激活窗口，并且指定了 SWP_NOACTIVATE 或 SWP_HIDEWINDOW 标志，则激活另外一个可见的顶层窗口。

当在这个函数中的 nFlags 参数里指定了 SWP_FRAMECHANGED 标志时，WindowsCE 重画窗口的整个非客户区，这可能会改变客户区的大小。这也是重新计算客户区的唯一途径，也是通过调用 SetwindowLong 函数改变窗口风格后通常使用的方法。

SetWindowPos 将使 WM_WINDOWPOSCHANGED 消息向窗口发送，在这个消息中传递的标志与传递给函数的相同。这个函数不传递其他消息。

Windows CE 1.0 不支持在 hWndlnsertAber 参数中的 HWND_TOPMOST 和 HWND_NOTOPMOST 常量。

Windows CE1.0 不支持在 fuFags 参数中的 SWP_DRAWFRAME 和 SWP_NOCOPYBITS 标志。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：1.0 以上版本；头文件：winuser.h 库文件：eser32lib。

SetWindowText

函数功能：该函数改变指定窗口的标题栏的文本内容（如果窗口有标题栏）。如果指定窗口是一个控制，则改变控制的文本内容。然而，SetWindowText 函数不改变其他应用程序中的控制的文本内容。

函数原型：BOOL SetWindowText（HWND hwnd，LPCTSTR lpStrjng）；

参数：

hWnd：要改变文本内容的窗口或控制的句柄。

lpString：指向一个空结束的字符串的指针，该字符串将作为窗口或控制的新文本。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：如果目标窗口属于当前进程，SetWindowText 函数会使 WM_SETTEXT 消息发送给指定的窗口或控制。然而，如果控制是以 WS_CAPTION 风格创建的列表框控制，SetWindowText 函数将为控制设置文本，而不是为列表项设置文本。

SetWindowText 函数不扩展 tab 字符（ASCII 代码 0×09），Tab 字符以字符‘}’来显示。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件;winuser.h;库文件：user32.lib；Unicode：Windows NT 上实现为 Unicode 和 ANSI 两种版本。

ShowOwnedPopups

函数功能：该函数显示或隐藏属于指定窗口的所有弹出式窗口。

函数原型：BOOL ShowOwnedPopups（HWND hWnd；BOOL fshow）；

参数：

hWnd：拥有弹出式窗口的窗口句柄，这些弹出式窗口将被显示或隐藏。

fShow：指明弹出式窗口是被显示还是隐藏。如果该参数为 TRUE，则所有隐藏的弹出式窗口均被显示；如果该参数为 FALSE，则所有显示的弹出式窗口均被隐藏。

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：ShowOwnedPopups 函数仅显示由以前调用 ShowOwnedPopups 隐藏的窗口。例如，如果弹出式窗口由调用 ShowWinodw 函数隐藏，则在随后调用 ShowOwnedPopups（将 fShow 参数置为 TRUE）并不能使窗口显示出来。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：Winuser.h；库文件：user32. Lib

2.19.51 ShowWindow

函数功能：该函数设置指定窗口的显示状态。

函数原型：BOOL ShowWindow（HWND hWnd，int nCmdShow）；

参数：

hWnd：窗口句柄。

nCmdShow：指定窗口如何显示。如果发送应用程序的程序提供了 STARTUPINFO 结构，则应用程序第一次调用 ShowWindow 时该参数被忽略。否则，在第一次调用 ShowWindow 函数时，该值应为在函数 WinMain 中 nCmdShow 参数。在随后的调用中，该参数可以为下列值之一：

SW_FORCEMINIMIZE：在 WindowNT5.0 中最小化窗口，即使拥有窗口的线程被挂起也会最小化。在从其他线程最小化窗口时才使用这个参数。

SW_MIOE：隐藏窗口并激活其他窗口。

SW_MAXIMIZE：最大化指定的窗口。

SW_MINIMIZE：最小化指定的窗口并且激活在 Z 序中的下一个顶层窗口。

SW_RESTORE：激活并显示窗口。如果窗口最小化或最大化，则系统将窗口恢复到原来的尺寸和位置。在恢复最小化窗口时，应用程序应该指定这个标志。

SW_SHOW：在窗口原来的位置以原来的尺寸激活和显示窗口。

SW_SHOWDEFAULT：依据在 STARTUPINFO 结构中指定的 SW_FLAG 标志设定显示状态，STARTUPINFO 结构是由启动应用程序的程序传递给 CreateProcess 函数的。

SW_SHOWMAXIMIZED：激活窗口并将其最大化。

SW_SHOWMINIMIZED：激活窗口并将其最小化。

SW_SHOWMINNOACTIVATE：窗口最小化，激活窗口仍然维持激活状态。

SW_SHOWNA：以窗口原来的状态显示窗口。激活窗口仍然维持激活状态。

SW_SHOWNOACTIVATE：以窗口最近一次的大小和状态显示窗口。激活窗口仍然维持激活状态。

SW_SHOWNOMAL：激活并显示一个窗口。如果窗口被最小化或最大化，系统将其恢复到原来的尺寸和大小。应用程序在第一次显示窗口的时候应该指定此标志。

返回值：如果窗口以前可见，则返回值为非零。如果窗口以前被隐藏，则返回值为零。

备注：应用程序第一次调用 ShowWindow 时，应该使用 WinMain 函数的 nCmdshow 参数作为它的 nCmdShow 参数。在随后调用 ShowWindow 函数时，必须使用列表中的一个给定值，而不是由 WinMain 函数的 nCmdSHow 参数指定的值。

正如在 nCmdShow 参数中声明的，如果调用应用程序的程序使用了在 STARTUPINFO 结构中指定的信息来显示窗口，则在第一次调用 ShowWindow 函数时 nCmdShow 参数就被忽略。在这种情况下，ShowWindow 函数使用 STARTUPINFO 结构中的信息来显示窗口。在随后的调用中，应用程序必须调用 ShowWindow 函数（将其中 nCmdShow 参数设为 SW_SHOWDEFAULT）来使用由程序调用该应用程序时提供的启动信息。这个处理在下列情况下发生：

应用程序通过调用带 WS_VISIBLE 标志的函数来创建它们的主窗口函数；

应用程序通过调用清除了 WS_VISIBLE 标志的 CteateWindow 函数来创建主窗口函数，并且随后调用带 SW_SHOW 标志的 ShowWindow 函数来显示窗口；

Windows CE：nCmdShow 参数不支持下列值：

SW_MAXIMINZE；SW_MINIMIZE；SW_RESTORE；SW_SHOWDEFAULT

SW_SHOWMAXIMIZED；SW_SHOWMINIMIZED；SW_SHOWMININOACTIVATE

速查：Windows NT：3.1 以上版本；Windows：95 以上版本：Windows CE：1.0 以上版本；头文件：winuw 库文件：user32.lib。

2.19.52 ShowWindowAsync

函数功能：该函数设置由不同线程产生的窗口的显示状态。

函数原型：BOOL ShowWindowAsync（HWND hWnd，int nCmdshow）；

参数：

hWnd：窗口句柄。

nCmdShow：指定窗口如何显示。查看允许值列表，请查阅 ShowWlndow 函数的说明部分。

返回值：如果函数原来可见，返回值为非零；如果函数原来被隐藏，返回值为零。

备注：这个函数向给定窗口的消息队列发送 show-window 事件。应用程序可以使用这个函数避免在等待一个挂起的应用程序完成处理 show-window 事件时也被挂起。

速查：Windows NT：4.0 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.19.53 TileWindows

函数功能： 该函数并到显示指定父窗口的各指定子窗口。

函数原型；WORD WINAPI TileWindows（HWND hWndParent，UNIT wHow，CONST RECT *IpRect,UNIT cKidS；ConHWND FAR *lpKidS）；参数：

hWndParent： 窗口句柄。如果该参数为 NULL，则假定为桌面窗口。

wHow:指定不参加安排的窗口类型，及是水平并到显示还是或垂直并到显示。该参数可以为下列值之一，可以选择与 MDITILE_SKIPDISABLED 组合以防止停用的的 MDI 的子窗口被并到显示：

MDITILE_HORIZONAL：水平并到显示窗口。MDITILE_VERTICAL：垂直并到显示窗口。

IpRect：指向 RECT 结构的指针，该结构以客户坐标定义矩形区域，在这个区域内排列窗口。如果该参数为 NULL，则使用父窗口的客户区。

cKids：指出在 lpKidS 参数中给出的数组的成员个数。如果 lpKids 为 NULL 则该参数被忽略。

lpKids：指向被排列的子窗口的句柄数组的指针。如果该参数为 NULL，则指定窗口（或桌面窗口）的所有子窗口均被排列。

返回值： 如果函数成功，返回值是被安排的窗口数目；如果函数失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注： 调用 TileWindows 函数使所有最大化的窗口恢复到原来的尺寸。

速查： Windows NT：4.o 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

2.19.54 WindowFromPoint

函数功能： 该函数获得包含指定点的窗口的句柄。

函数原型： HWND WindowFromPoint（POINT Point）；

参数：

Point：指定一个被检测的点的 POINT 结构。

返回值 S：返回值为包含该点的窗口的句柄。如果包含指定点的窗口不存在，返回值为 NULL。如果该点在静态文本控制之上，返回值是在该静态文本控制的下面的窗口。

备注： WindowFromPoint 函数不获取隐藏或禁止的窗口句柄，即使点在该窗口内。应用程序应该使用 ChildWindowFromPoint 函数进行无限制查询。

速查： Windows NT:3.1 以上版本:Windows:95 以上版本;Windows CE:1.0 以上版本:头文件:Winuser.h；库文件：user32.lib。

2.19.55 WinMain

函数功能： 该函数被系统调用，作为一个 32 位应用程序的入口点。

函数原型： int WINAPI WinMain（HINSTANCEE hlnstance，HINSTANCE hPrelnstance，LPSTR lpCmdLine，int nCmdShow）；

参数；

hinstance：应用程序当前事例的句柄。

hPrelnstance：应用程序的前事例的句柄。对于一个 32 的位程序，该参数总为 NULL。

如果需要检测另外一个事例是否已经存在，则使用 CreateMutex 函数创建一个独一无二的名字。即使

互斥名已经存在，CreateMutex 函数也是成功的，但是 GetLastError 函数将返回 ERROR_ALREADY_EXISTS，这就表明在应用程序中有另外一个事例存在，因为它首先创建了互斥名。

lpCmdLine：指向应用程序命令行的空字符串的指针，不包括函数名。获得整个命令行，参看 GetCommandLine。

nCmdShow：指明窗口如何显示。该参数可以是下列值之一：

SW_HIOE：隐藏窗口并且激活另外一个窗口。

SW_MINIMIZE：最小化指定的窗口，并且激活在系统表中的顶层窗口。

SW_RESTORE：激活并显示窗口。如果窗口已经最小化或最大化，系统将以恢复到原来的尺寸和位置显示窗口（与 SW_SHOWNORMAL 相同）。

SW_SHOW：激活一个窗口并以原来的尺寸和位置显示窗口。

SW_SHOWMAXIMIZED：激活窗口并且将其最大化。

SW_SHOWMINIMIZED：激活窗口并将其目标化。

SW_SHOWMINNOACTIVE：将一个窗口显示为图标。激活窗口维持活动状态。

SW_SHOWNA：以窗口的当前状态显示窗口。激活窗口保持活动状态。

SW_SHOWNOACTIVATE：以窗口的最近一次的尺寸和位置显示窗口。激活窗口维持激活状态。

SW_SHOWNORMAL：激活并显示窗口。如果窗口最大化或最小化，系统将其恢复到原来的尺寸和位置（与 SW_RESTORE 相同）。

返回值：如果函数成功，当它接收到一个 WM_QUIT 消息时就中止，函数应该返回在该消息的 wParam 参数的退出值。如果函数在进入消息循环时退出，应该返回零。

备注：WinMain 函数应初始化应用程序，显示主窗口，进入一个消息接收一发送循环，这个循环是应用程序执行的其余部分的顶级控制结构。当接收到一个 WM_QUIT 消息时，程序就中止。这时，WinMain 函数应退出应用程序，并且返回传递给 WM_QUIT 消息的 wParam 参数的值。如果由于调用 PostQuitMessage 函数而接收到 WM_QUIT 消息，wParam 的值是 PostQuiMessage 函数的 nExitCode 的值。请参看“创建一个窗口循环”。

ANSI 应用程序可以使用 WinMain 函数的 lpCmdLine 参数进入命令行字符串(除了程序名之外)。WinMain 不能返回 Unicode 字符串的原因是 IpCmdLine 使用的是 LPSTR 数据类型，而不是 LPTSTR 类型。GetCommandLine 函数可以用于进入命令行的 Unicode 字符串，因为它使用的是 LPTSTR 类型。

Windows CE：Windows CE 不支持下列 nCmdLine 参数值：

SW_MINIMIZE；SW_RESTORE；SW_RESTORE；SW_SHOWMAXMIZED

SW_SHOWMINIMIZED；SW_SHOWMINNOACTIVE

速查：Windows NT：3.1 以上版本；Windows：95 以上版本：Windows CE：1.0 以上版本；头文件：Winbase.h 库文件：用户自定义。

2.19.56 AnyPopup

函数功能：该函数指出一个被属窗口，可见窗口，顶级弹出窗口，或层叠窗日是否在屏幕上存在。这个函数搜索整个屏幕，而不仅仅搜索应用程序的客户区。

函数原型：BOOL AnyPopup（VOID）

参数：无。

返回值：如果一个弹出式窗口存在，返回值为非零，即使该窗口被其他窗口完全覆盖。如果弹出式窗口不存在，返回值为零。

备注：函数不检测无所属关系的弹出式窗口，或无 WS_VISIBLE 设置位的窗口。

速查：Windows NT:3.1 以上版本；Windows：95 以上版本；Windows CE:不支持；头文件：winuser.h；库文件：USer32.lib。

2.19.57 EnumTaskWindows

函数功能： 该函数已经过时，由 EnumThreadwindows 函数取代它。

2.19.58 GetSysModalWindow

函数功能： 该函数已经过时，该函数只是为与 16 位版的窗口程序兼容而提供的。

2.19.59 GetWindowTask

函数功能： 该函数已经过时，该函数只是为与 16 位版的窗口程序兼容而提供的。基于 32 位的应用程序应该使用 GetWindowThreadProcessld 函数。

2.19.60 SetSysModalWindow

函数功能：该函数已经过时，该函数只是为与 16 位版的窗口程序兼容而提供的。新的输入模块不允许系统的模块窗口。

2.20 窗口类函数（Window Class）

2.20.1 GetClasslnfoEx

函数功能： 该函数获得有关窗口类的信息，包括与窗口类相关的小图标的句柄的信息。GetClasslnfo 函数不检索小图标的句柄。

函数原型：BOOL GetClasslnfoEx（HINSTANCE hlnst，LPCTSTR lpszClass，LPWNDCLASSEX lpwcx)；

参数 ：

hlnst：创建类的应用程序的事例的句柄。获得由系统定义的类（如按钮或列表框）的信息，设置该参数为 NULL。

IpszClass ：指向一个包含类名的空结束的字符串的指针。类名必须为事先已注册的类，或是由此前调用 RegisterClaaEx 函数注册的类。或者还可以是一个整型数，如果参数是一个整型数，它必须是由以前调用 GlobaAdd 原子函数创建的全局原子。这个 16 位整型数小于 OXCOOO,必须是 lpszClassS 的低 16 位，其高位字为 0。

Ipwcx：指向接收类信息的 WNDCLASSEX 结构的指针。

返回值：如果函数未发现一个匹配的类，并且成功地拷贝了数据，则返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：35 以上版本；Windows：95 以上版本：Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.20.2 GetClassLong

函数功能： 该函数返回与指定窗口相关的 WNDCLASSEX 结构的指定 32 位值。

函数原型： DWORD GetClassLong（HWND hWnd，int nlndex)；

参数：

hWnd：窗口句柄间接给出的窗口所属的类。

nlndex：指定要恢复的 32 位值。从额外的类存储空间恢复一个 32 位的值，指定的一个大于等于 0 的被恢复值的偏移量。有效值为从 0 开始到额外类存储空间字节数一 4。例如，若指定了 12 位或多于 12 位的额外类存储空间，则应设为第三个 32 位整数的索引位 8。要从 WNDCLASSEX 结构中恢复任何值，需要指定下面值之一：

GCSW 原子：获得一个唯一标识窗口类的原子值，该值与 RegisterClassEx 函数的返回值相同。

GCL_CBWNDEXTRA：获得与类中的每个窗口相关的额外窗口中内存空间的字节大小，进入该存储空间的方法请参看 GetWindwoLong。

GCL_HBRBACKGROUND：获得与类有关的背景刷子的句柄。

GCL_HCURSOR：获得与类有关的光标的句柄。

GCL_HICON：获得与类有关的图标的句柄。

GCL_HICONSM：获得与类有关的小图标的句柄。

GCL_HMOUDLE：获得注册该类的模块的句柄。

GCL_MENUNAME：获得菜单名字符串的地址，该字符串标识了与类有关的菜单资源。

GCL_STYLE：获得窗口类的风格位。

GCL_WNDRPOC：获得与类有关的窗口过程的地址。

返回值： 如果函数成功，返回值是所需的 32 位值；如果函数失败，返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

备注： 通过使用函数 RegisterClassEx 将结构 WNDCLASSEX 中的 cbCIsExtra 单元指定为一个非 0 值来保留额外类的存储空间。

Windows CE：nlndex 参数是一个字节偏移量，但是必须为 4 的倍数。Windows CE 不支持 unaligned access。

nlndex 参数中只可设定为 GCL_HICON 和 GCL_STYLE。

如果使用了 Windows CE 的 lconsurs 组件，该组件支持在适当的目标平台上的鼠标，也可以在 nlndex 中使用 GCL_HCURSOR。

注意支持鼠标的 Windows CE 版本包含 Iconcurs 和 Mcursor 而不是 Icon 和 Cursor 组件。

速查：Wiodows NT:3.1 以上版本;Windows:95 以上版本:Windows CE:1.0 以上版本:头文件:Winuser.h；库文件：USer32.lib：Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.20.3 GetClassName

函数功能： 该函数获得指定窗口所属的类的类名。

函数原型： Int GetClassName（HWND hWnd，LPTSTR IpClassName int nMaxCount)；**参数：**

hWnd：窗口的句柄及间接给出的窗口所属的类。

IpClassName:指向接收窗口类名字符串的缓冲区的指针。

nMaxCount：指定由参数 lpClassName 指示的缓冲区的字节数。如果类名字符串大于缓冲区的长度，则多出的部分被截断。

返回值： 如果函数成功，返回值为拷贝到指定缓冲区的字符个数：如果函数失败，返回值为 0。若想

获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 以上版本：Windows：95 以上版本；Windows CE1.0 以上版本；头文件：winuser.h 库文件：user32.lib; Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.20.4 GetWindowLong

函数功能：该函数获得有关指定窗口的信息，函数也获得在额外窗口内存中指定偏移位地址的 32 位度整型值。

函数原型：LONG GetWindowLong（HWND hWnd，int nlndex)；

参数：

hWnd:窗口句柄及间接给出的窗口所属的窗口类。

nlndex：指定要获得值的大于等于 0 的值的偏移量。有效值的范围从 0 到额外窗口内存空间的字节数一 4 例如，若指定了 12 位或多于 12 位的额外类存储空间，则应设为第三个 32 位整数的索引位 8。要获得任意其他值，指定下列值之一：

GWL_EXSTYLE；获得扩展窗日风格。

GWL_STYLE：获得窗口风格。

GWL_WNDPROC：获得窗口过程的地址，或代表窗口过程的地址的句柄。必须使用 GWL_WNDPROC 函数调用窗口过程。

GWL_HINSTANCE：获得应用事例的句柄。

GWL_HWNDPAAENT：如果父窗口存在，获得父窗口句柄。

GWL_ID:获得窗口标识。

GWL_USERDATA：获得与窗口有关的 32 位值。每一个窗口均有一个由创建该窗口的应用程序使用的 32 位值。

在 hWnd 参数标识了一个对话框时也可用下列值：

DWL_DLGPROC：获得对话框过程的地址，或一个代表对话框过程的地址的句柄。必须使用函数 CallWindowProc 来调用对话框过程。

DWL_MSGRESULT：获得在对话框过程中一个消息处理的返回值。

DWL_USER：获得应用程序私有的额外信息，例如一个句柄或指针。

返回值：如果函数成功，返回值是所需的 32 位值；如果函数失败，返回值是 0。若想获得更多错误信息请调用 GetLastError 函数。

备注：通过使用函数 RegisterClassEx 将结构 WNDCLASSEX 中的 cbWndExtra 单元指定为一个非 0 值来保留额外类的存储空间。

Windows CE：nlndex 参数指定的字节偏移量必须为 4 的倍数。不支持 unaligmned access。

Windows CE：不支持在参数 nlndex 中设定的 GWL_HINSTANCE 和 GWL_HWNDPARENT。

Windows CE1.0 也不支持在 nlndex 参数中的 DWL_DLGPROC 和 GWL_USERDATA。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：1.0 对以上版本；头文件：winuser.h;库文件：user32.lib；在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.20.5 RegisterClassEx

函数功能：该函数为随后在调用 Createwindow 函数和 CreatewindowEx 函数中使用的窗口注册一个窗口类。

函数原型：ATON RegisterClassEX（CONST WNDCLASSEX★Ipwcx)；

参数：

Ipwcx：指向一个 WNDCLASSEX 结构的指针。在传递给这个函数之前，必须在结构内填充适当的类的属性**返回值：**如果函数成功，返回值是唯一识别被注册类的一个原于；如果函数失败，返回值为 0。若想获得更多错误信息，请调用 callGetLastError 函数。

备注：如果使用 RegisterClassEx 来注册窗口类，应用程序通知系统被注册类的窗回的消息使用 ANSI 字符集的文本和字符参数；如果使用 RegisterClassExW 来注册窗口类，应用程序需要系统以 Unicode 来传递消息的文本参数。IsWindowUnicode 函数使应用程序可以查询每一个窗口的字符特征。参看 Win32 API 中的 ANSI 和 Unicode 函数，请查阅 Functiont prototype（函数原型）。

应用程序注册的所有的窗口类在应用程序中止后都为未注册的类。

Windows 95：所有由 DLL 注册的类在 DLL 卸载后均未注册的类。

Windows NT：所有由 DLL 注册的类在 DLL 卸载后仍为已注册的类。

Windows 95：如果 WNDCLASSEX 结构中的 cbWndEXtra 或 cbCIsEXtra 单元包含字节数超过 40 个字节，则 RegisterClassEx 将失败。

速查：Windows NT：4.0 以上版本；Windows：95 以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32.lib; Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.20.6 SetClassLong

函数功能：该函数替换在额外类存储空间的指定偏移地址的 32 位长整型值，或替换指定窗口所属类的 WNDCLASSEX 结构。

函数原型：DWORD SetClassLong（HWND hWnd，int nlndex，LONG dwNewLong)；

参数：

hWnd：窗口句柄及间接给出的窗口所属的类。

nlndex：指定将被替换的 32 位值。在额外类存储空间中设置 32 位值，应指定一个大于或等于 0 的偏移量。

有效值的范围从 0 到额外类的存储空间的字节数一 4；例如，若指定了 12 位或多于 12 位的额外类存储空间，则应设为第三个 32 位整数的索引位 8。要设置 WNDCLASSEX 结构中的任何值，指定下面值之一：

GCL_CBCLSEXTRA：设置与类相关的尺寸的字节大小。设定该值不改变己分配的额外字节数。

GCL_CBWNDEXTRA：设置与类中的每一个窗口相关的尺寸的字节大小。设定该值不改变已分配额外字节数。查看如何进入该内存，参看 SetWindowLOng。

GCL_HERBACKGROUND：替换与类有关的背景刷子的句柄。

GCL_HCURSOR：替换与类有关的光标的句柄。GCL_HICON：替换与类有关的图标的句柄。

GCL_HMODULE：替换注册类的模块的句柄。GCL_STYLE：替换窗口类的风格位。

CGL_MENUNAME ：替换菜单名字符串的地址。该字符串标识与类有关的菜单资源。

GCL_WNDPROC ：替换与窗口类有关的窗口过程的地址。

dwNewLong：指定替换值。

返回值：如果函数成功，返回值是指定的 32 位整数的原来的值；如果未事先设定，返回值为 0。如果函数失败，返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

备注；如果使用 SetClassLong 函数和 GCL_WNDPROC 索引值来替换窗口过程，窗口过程必须与 WindowProccallback 函数中的描述的 guideline 一致。

以带 GCL_WNDPROC 索引值的 SetClassLong 函数创建的一个窗口类的的子类将会影响所有随后以该类创建的窗口。应用程序可以创建一个系统类的子类，但是不能创建由其他进程创建的类的子类。

通过使用 RegisterClassEx 函数将 WNDCLASSEX 结构中的 cbWndExtra 单元指定为一个非零值来保留额外的的类存储空间。

小心使用 SetClassLong 函数。例如，可以通过使用 SetClassLong 来改变类的背景颜色，但是这个改变 Windows CE：nlndex 参数是一个字节偏移量但必须是 4 的倍数。Unaligned 不支持。

不支持在 nlndex 参数中的标准的 CGL_★值，只有一个例外，如果目标设各支持鼠标，则可以在 nlndex 参数中指定 CGL_HCURSOR。

注意支持鼠标的 WindowsCE 版本包含 Iconcurs 和 Mcursor 组件而不是 lcon 和 Cursor 组件。

速查：Windows NT：3.1 以上版本：Windows:95 以上版本；Windows CE：1.0 以上版本；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.20.7 SetWindowLong

函数功能：该函数改变指定窗口的属性. 函数也将指定的一个 32 位值设置在窗口的额外存储空间的指定偏移位置。

函数原型：LONG SetWindowLong（HWND hWnd，int nlndex，LONG dwNewLong)；

参数：

hWnd：窗口句柄及间接给出的窗口所属的类。

nlndex：指定将设定的大于等于 0 的偏移值。有效值的范围从 0 到额外类的存储空间的字节数-4：例如若指定了 12 位或多于 12 位的额外类存储空间，则应设为第三个 32 位整数的索引位 8。要设置其他任何值，可以指定下面值之一：

GWL_EXISTYLE：设定一个新的扩展风格。GWL_STYLE：设定一个新的窗口风格。

GWL_WNDPROC：为窗口过程设定一个新的地址。GWL_ID：设置一个新的窗口标识符。

GWL_HINSTANCE：设置一个新的应用程序事例句柄。

GWL_USERDATA：设置与窗口有关的 32 位值。每一个窗口均有一个由创建该窗口的应用程序使用的 32 位值。

当 hWnd 参数标识了一个对话框时，也可使用下列值：

DWL_DLGPROC：设置对话框过程的新地址。

DWL_MSGRESULT：设置在对话框过程中处理的消息的返回值。

DWL_USER：设置的应用程序私有的新的额外信息，例如一个句柄或指针。

dwNewLong：指定的替换值。

返回值：如果函数成功，返回值是指定的 32 位整数的原来的值。如果函数失败，返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

如果指定 32 位整数的原来的值为 0，并且函数成功，则返回值为 0，但是函数并不清除最后的错误信息，这就很难判断函数是否成功。这时，就应在调用 SetWindowLong 之前调用 callingSetLastError（0）函数来清除最后的错误信息。这样，如果函数失败就会返回 0，并且 GetLastError。也返回一个非零值。

备注；如果由 hWnd 参数指定的窗口与调用线程不属于同一进程，将导致 SetWindowLong 函数失败。

指定的窗口数据是在缓存中保存的，因此在调用 SetWindowLong 之后再调用 SetWindowPos 函数才能使 SetWindowLong 函数所作的改变生效。

如果使用带 GWL_WNDPROC 索引值的 SetWindowLong 函数替换窗口过程，则该窗口过程必须与 WindowProccallback 函数说明部分指定的指导行一致。

如果使用带 DWL_MSGRESULT 索引值的 SetWindowLong 函数来设置由一个对话框过程处理的消息的返回值，应在此后立即返回 TRUE。否则，如果又调用了其他函数而使对话框过程接收到一个窗口消息，则嵌套的窗口消息可能改写使用 DWL_MSGRESULT 设定的返回值。

可以使用带 GWL_WNDPROC 索引值的 SetWindowLong 函数创建一个窗口类的子类，该窗口类是用于创建该窗口的关。一个应用程序可以一个系统美为于类，但是不能以一个其他进程产生的窗口类为子类，SetwindowLong 函数通过改变与一个特殊的窗口类相联系的窗口过程来创建窗口子类，从而使系统调用新

的窗口过程而不是以前定义的窗口过程。应用程序必须通过调用 CallWindowProc 函数向前窗口传递未被新窗口处理的消息，这样作允许应用程序创建一个窗口过程链。

通过使用函数 RegisterClassEx 将结构 WNDCLASSEX 中的 cbWndExtra 单元指定为一个非 0 值来保留新外窗口内存。

不能通过调用带 GWL_HWNDPARENT 索引值的 SetWindowLong 的函数来改变子窗口的父窗口，应使用 SetParent 函数。

Windows CE：nlndex 参数必须是 4 个字节的倍数不支持 unaligned access。

不支持下列 nlndex 参数值：。

GWL_HINSTANCE; GWL_HWNDPARENTGWL;GWL_USERDATA

Windows CE 2.0 版支持在 nlndex 参数中的 DWL_DLGPROC 值，但是 WindowsCE1.0 不支持。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本:头文件:winuser.h;库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

2.20.8 UnRegisterClass

函数功能：该函数删除一个窗口类，清空该类所需的内存。

函数原型：BOOL UnRegisterClass（LPCTSTR IpClassName； HINSTANCE hlnstance)；

参数：

IpClassName：指向一个空结束字符串的指针，或是一个整型原子。如果 IpClassName 是一个字符串，则它指定了窗口类的类名。这个类名必须由此前调用 RegisterClassEx 函数来注册。系统类，如对话框控制，必须被注册。

如果这个参数是一个整型原子，它必须是由此前调用 GlobalAdd 原子函数创建的全局原子。这个 16 位整型数小于 0xC000，必须是 lpszClass 的低 16 位，其高位字必须为 0。

hlnstance：创建类的模块的事例句柄。

返回值：如果函数成功，返回值为非零；如果未发现类或由此类创建的窗口仍然存在，则返回值为 0。

若想获得更多错误信息，请调用 GetLastError 函数。

备注：在调用这个函数之前，应用程序必须销毁由指定类创建的所有窗口。

应用程序注册的所有窗口类在应用程序中止后都为未注册的类。

Windows 95：所有由 OLL 注册的窗口类在 DLL 卸载后均未注册的类。

windows NT：所有由 DLL 注册的类在 DLL 卸载后仍为已注册的类。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件：user32.lib；Unicode：在 Windows NT 上实现为 Uhicode 和 ANSI 两种版本。

2.20.9 GetClasslnfoEx

函数功能：该函数获得有关一个窗口类的信息。GetClasslnfo 函数已经由 GetClasslnfoEx 取代，但是如果不需要获得类的小图标信息. 仍然可以使用该函数。

函 数 原 型： BOOL GetClasslnfoEx （ HINSTANCE hlnstance,LPCTSTR lpClassName, LPWNDCLASS lpWndClass)；

参数：

hlnstance：创建该类的应用程序的事例句柄。要获得由系统定义的类的信息（例如按钮和列表），将该参数设为 NULL。

lpClassName：指向一个包含类名的空结束的字符串的指针。类名必须为事先注册的类，或是由此前调

用 RegisterClassEx 函数注册的类。或者还可以是一个整型数原子，如果参数是一个整型数原子，它必须是由以前调用 GlobalAdd 原子函数创建的全局原子。这个 16 位整型数小于 0xC000，必须是 lpszClass 的低 16 位，高位字必须为 0。

lpWndClass：指向一个接收类信息的一个 WNDCLASS 结构的指针。

返回值：如果函数发现了一个匹配的类并且成功地拷贝了字符串，则返回值为非零；如果函数失败，返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

备注：在 Windows CE 中，如果 lpClassName 是一个原子，则它必须是从函数 RegisterClass 返回的原子。

2.20.10 GetClassWord

函数功能：该函数在窗口类的额外存储空间中的指定偏移地址获取指定窗口所属窗口类的 16 位值。不象 GCW_原子和 GCW_HICONSM，由 16 位 Windows 支持的 GCW_value 已经过时，必须使用函数 GetClassLong 来获得窗口的类值。

函数原型：WORD GetCassWord（HWND hWnd,int nlndex)；

参数:.

hWnd：窗口句柄及间接给出的窗口所属的类。

nlndex：指定要获得的值的大于等于 0 字节的偏移量。有效值为从 0 开始到额外类存储空间字节数一 2。

例如，若指定了 10 位或更多的额外类存储空间，则应设为第 5 个 16 位整数的索引位 8。允许有的另外的有效值：

GCW_原子：恢复一个唯一标识窗口类的原子值，该值与 RegiterClassEx 函数的返回值相同。

GCW_HICONSM：恢复与窗口相关的小图标的句柄。

返回值：如果函数成功，返回值是所需的 16 位值；如果函数失败，返回值是零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：通过使用函数 RegisterClassEx，将结构 WNDCLASSEX 中的 cbClsEXtra 单元设为一个非 0 值来保留额外的类存储空间。

2.20.11 GetWindowWord

函数功能：该函数已经过时。32 位 Windows 程序应使用 GetWindowLong 函数。

2.20.12 RegisterClass

函数功能：该函数注册在随后调用 CreateWindow 函数和 CreateWindowEx 函数中使用的窗口类。RegisterClass 函数已经由函数 RegisterClassEx 函数来代替，但是，如果不需要设置类的小目标则仍然可以使用 RegisterClass 函数。

函数原型：ATON RegisterClass（CONST WNDCLASS ★lpWndClass)；

参数：

lpWndClass：指向一个 WNDCLASS 结构的指针。在将它传递给函数之前，必须在该结构中填充适当的类属性。

返回值：如果函数成功，返回值是唯一标识已注册的类的一个原子；如果函数失败，返回值为 0。若

想获得更多错误信息，请调用 GetLastError 函数。

备注： 如果使用 RegisterClassA 来注册窗口，应用程序通知系统被注册类的窗口的消息使用 ANSI 字符集的文本和字符参数；如果使用 RegisterClassW 来注册窗口，应用程序需要系统以 Unicode 来传递消息的文本参数。lsWindowUnicode 函数使应用程序可以查询每一个窗口的字符特征。参看 Win32API 中的 ANSI 和 Unicode 函数，请查阅 Functiont prototype。

应用程序注册的所有的窗口类在应用程序中止后都为未注册的类。

Windows 95：所有由 DLL 注册的类在 DLL 卸载后均未注册的类。

Windows NT：所有由 DLL 注册的类在 DLL 卸载后仍为已注册的类。

Windows 95：如果 WNDCLASSEX 结构中的 cbWndExtra 或 cbClsExtra 单元包含字节数超过 40 个字节，则 RegisterClassEx 将失败。

Windows CE：由 lpWndClass 参数指向的 WNDCLAS 结构不支持 lpszMenuName 域，因为 WindowsCE 不支持缺省菜单。

除非使用了 WindowsCE 的 lconcurs 组件（这个组件提供了在适当目标平台上的鼠标支持），否则不能使用由 lpWndClass 指向的 WNDCLASS 结构中的 hCursor 域。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本：WindowsCE：1.0 以上版本；头文件：Winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicodee 和 ANSI 两种版本。

2.20.13 SetClassWord

函数功能： 该函数替换指定窗口所属的窗口类的额外存储空间中的指定偏移地址的 16 位值。由 16 位窗口支持的 GCW_值已经过时，必须使用 SetClassLong 函数来设置此前使用 SetClassword 函数的 GCW_值设置的类值。

函数原型： WORD SetClassWord（HWND hWnd，int nlndex，WORD wNewWord）；

参数：

hWnd：窗口的句柄及间接给出的窗口所属的类。

nlndex：指定要获得的值的大于等于 0 字节的偏移量。有效值为从 0 开始到额外类存储空间字节数-2。

例如，若指定了 10 位或更多的额外类存储空间，则应设为第 5 个 16 位整数的索引位 8。

wNewWord：指定替换值。

返回值：如果函数成功，返回值是指定的 16 为整数的原来的值。如果该值未被预先设定，返回值为 0。如果函数失败，返回值为 0。若想获得更多错误信息，请调用 GetLastError 函数。

备注： 通过使用函数 RegisterClassEx 将结构 WNDCLASSEX 中的 cbClsExtra 单元指定为一个非 0 值来保留额外类的存储空间。

2.20.14 SetWindowWord

函数功能： 该函数已经过时。GWW_value 已不再被支持。32 位 Windows 程序应该使用 SetWindowLong 函数。

2.21 窗口过程函数（Window Procedure）

2.21.1 CallWindowProc

函数功能： 该函数 CallWindowProc 将消息信息传送给指定的窗口过程。

函数原型：LRESULT CallWindowProc（WNDPROC lpPrevWndFunc，HWND hWnd. UINT Msg，WPARAM wParam，LPARAMIParam)；

参数：

lpPrevWndFunc：指向前一个窗口过程的指针。如果该值是通过调用 GetWindowLong 函数，并将该函数中的 nlndex 参数设为 GWL_WNDPROC 或 DWL_DLGPROC 而得到的，那么它实际上要么是窗口或者对话框的地址，要么就是代表该地址的句柄。

hWnd：指向接收消息的窗口过程的句柄。

Msg：指定消息类型。

wParam：指定其余的、消息特定的信息。该参数的内容与 Msg 参数值有关。

IParam：指定其余的、消息特定的信息。该参数的内容与 Msg 参数值有关。

返回值： 返回值指定了消息处理结果，它与发送的消息有关。

备注： 使用函数 CallWindowsProc 可进行窗口子分类。通常来说，同一类的所有窗口共享一个窗口过程。子类是一个窗口或者相同类的一套窗口，在其消息被传送到该类的窗口过程之前，这些消息是由另一个窗口过程进行解释和处理的。

SetWindowLoog 函数通过改变与特定窗口相关的窗日过程，使系统调用新的窗口过程来创建子类，新的窗口过程替换了以前的窗口过程。应用程序必须通过调用 CallWindowsProc 来将新窗日过程没有处理的任何消息传送到以前的窗口过程中，这样就允许应用程序创建一系列窗口过程。

如果定义了 STRICT，那么 lpPrevWndFunc 参数具有 WNDPROC 数据类型。WNDPROC 类型说明如下：

LRESULT（CALLBACK• WNDPROC）（HWND， UINT WPARAM，LPARAM）

如果没有定义 STRICT，那么 lpPrevWndFunc 参数具有 FARPROC 数据类型。FARPROC 类型说明如下：int（FAR WINAPL•FARPROC）（）

在 C 语言中，FARPROC 申明表示为一个没有指定参数表的回调函数。然而在 C＋＋中；申明中的空参数表示该函数没有参数。这种微妙的区别有可能引起代码出错。下面是一种解决办法：#ifdef STRICT；WNDPROC MyWindowProcedure；#else；FARPROC MyindowProcedure；#endif IResult=CallWindowProc（MyWindowProcedure，…）

有关该函数的空参数表方面进一步的信息，清参考 Bjarne Stroustrup 编写的 C＋＋编程语言第 2 版。

对于 Windows NT：函数 CallWindowsProc function 进行了 Unicode 至 ANSf 转换处理。如果你直接调用该窗口过程，那么无法利用该转换。

速查：Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h;库文件：user32.lib：Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 版本实现。

2.21.2 DefWindowProc

函数功能： 该函数调用缺省的窗口过程来为应用程序没有处理的任何窗口消息提供缺省的处理。该函数确保每一个消息得到处理。调用 DefWindowProc 函数时使用窗口过程接收的相同参数。

函数原型：LRESULT DefWindowProc（HWND hWnd，UINT Msg，WPARAM wParam，LPARAM IParam)；

参数：

hWnd：指向接收消息的窗口过程的句柄。

Msg：指定消息类型。

wParam：指定其余的、消息特定的信息。该参数的内容与 Msg 参数值有关。

IParam：指定其余的、消息特定的信息。该参数的内容与 Msg 参数值有关。

返回值： 返回值就是消息处理结果，它与发送的消息有关。

备注： 对于 Windows CE；如果 Msg 为 WM_SETTEXT 那么返回 0。

当 DefWindowProc 处理 WM_DESTROY 消息时，它不自动调用 PostQutMessage。

速查： Windows NT 3.1 以上版本；Windows：95 以上版本：Windows CE 以上版本；头文件；winuser.h；库文件：user32.lib；Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 版本实现。

2.21.3 WindowProc

函数功能： 该函数是一个应用程序定义的函数。它处理发送给窗口的消息。WNDPROC 类型定义了一个指向该回调函数的指针。WindowProc 是用于应用程序定义函数的占位符。

函数原型： LRESULT CALLBACK WindowProc（HWND hwhd，uMsg，WPARAM wParam，LPARAM IParam）；

参数：

hwnd：指向窗口的句柄。

uMsg：指定消息类型。

wParam：指定其余的、消息特定的信息。该参数的内容与 UMsg 参数值有关。

IParam：指定其余的、消息特定的信息。该参数的内容与 uMsg 参数值有关。

返回值： 返回值就是消息处理结果，它与发送的消息有关。

速查： Windows NT:3.1 以上版本;Windows:95 以上版本;Windows CE:1.0 以上版本;头文件:winuser.h；库文件：用户自定义。

2.22 窗口属性函数（Window Property）

2.22.1 EnumProps

函数功能： 该函数将窗口属性表中的所有项列举出来，一个一个地传送给指定的回调函数，直到列举到最后一项，或者回调函数返回 FALSE 为止。

函数原型： int EnumProps（HWND hWnd，PROPENUMPROC lpEnumFunc）；

参数：

hWnd：指向要列举属性表内容的窗口。

lpEnumFunc：指向回调函数的指针。有关回调函数方面更多的信息，可参考 PropEnumProc 函数。

返回值： 返回值指定了回调函数返回的最后一个值、如果函数没有发现要列举的属性，那么它返回-1。

备注： 应用程序只能删除它增加进去的那些属性。它无法删除其他应用程序加进去的或者系统本身的属性。

速查： Windows NT：3.1 以上版本：Windows：95 以上版本；Window CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 版本实现。

2.22.2 EnumPropsEx

函数功能：该函数将窗口属性表中的所有项列举出来，依次传送给指定的回调函数，直到列举到最后一项，或者回调函数返回 FALSE 为止。

函数原型：int EnumPropsEx（HWND hWnd，PROPENUMPROCEX lpEnumFunc，LPARAM lParam）；

参数：

hWnd：指向要列举属性表内容的窗口。

lpEnumFunc：指向回调函数的指针。有关目调函数方面更多的信息，可参考 PropEnumProcEx 函数。

lParam：包含应用程序定义的、要传送给回调函数的数据。

返回值：返回值指定了回调函数返回的最后一个值。如果函数没有发现要列举的属性，那么它返回-1。

备注：应用程序只能删除它增加进去的那些属性。它无法删除其他应用程序加进去的或者系统本有的属性。

速查： Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持：头文件：Winser.h；库文件：user32.lib：URICOd6：在 Windows NT 环境中以 Unicode 和 ANSI 版本实现。

2.22.3 GetProp

函数功能：该函数从给定窗口的属性列表中检索数据句柄。给定的字符串标识了要检索的句柄。该字符串和句柄必须在前一次调用 SetProp 函数时已经加到属性表中。

函数原型：HANDLE GetProp（HWND hWnd，LPCTSTR lpString）；

参数：

hWnd：指向要搜索属性表的窗口。

LpString:指向以 null 结尾的字符串指针，或者包含一个标识字符串的原子。如果该参数是一个原子，那么它必须是使用 GlobalAddAtom 函数创建的。原子是 16 位的数据值，它必须是放置在 lpstring 参数的低位率中，而高位字必须为 0。

返回值：如果属性表中包含了给定的字符串，那么返回值为相关的数据句柄。否则，返回值为 NULL。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32。lib；Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 版本实现。

2.22.4 PropEnumProc

函数功能：该函数是一个应用程序定义的回调函数，它与 EnumProps 函数一同使用。该函数从窗口属性新检索属性项。PROPENUMPROC 类型定义了一个指向该回调函数的指针。PropEnumProc 是用于应用程序定义的函数名的占位符。

函数原型：BOOL CALLBACK PropEnumProc（HWND hwnd，LPCTSTR lpszString，HANDLE hData）；

参数：

hwnd：指向要列举属性表内容的窗口。

lpzString：指向以 null 结尾的字符串的指针。该字符串是属性表项的字符串部分。该字符串是在通过调用 SetProp 函数将属性增加到窗口属性表中时与数据句柄一起指定的。

hData：指向数据的句柄。该句柄是属性表项中的数据部分。

返回值：返回 TRUE 以继续列举属性表。返回 FALSE 停止列举属性表。

备注：下列限制适用该回调函数：该回调函数不应该产生对其他任务的控制或做一些有可能产生对其

他任务的控制的事情。

该回调函数可以调用 RemoveProp 函数。然而，RemoveProp 函数通过该回调函数的参数只能清除传给该回词函数的属性。该回调函数不应该试图增加属性。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；库文件；用户自定义；Unicode：定义为 Unicode 和 ANSI 原型。

2.22.5 PropEnumProcEx

函数功能：该函数是一个应用程序定义的回调函数，它与 EnumPropsEx 函数一同使用。该函数从窗口属性表中检索属性项。PROPENUMPROCEX 类型定义了一个指向该回调函数的指针。PropEnumProcEx 是用于应用程序定义的函数名的占位符。

函数原型 BOOL CALLBACK PropEnumProcEx(HWND hwnd,LPTSTR lpszStrng,HANDLE hData,DWORD dwData);

参数：

hwnd：指向要列举属性表内容的窗口。

lpszString：指向以 null 结尾的字符串的指针。该字符串是属性表项的字符串部分。该字符串是在通过调用 SetProp 函数将属性增加到窗口属性表中时与数据句柄一起指定的。

hData：指向数据的句柄。该句柄是属性表项中的数据部分。

dwData：应用程序定义的数据、该值被指定为在调用 EnumPropsEX 进行列举初始化时的 IParam 参数。

返回值：返回 TRUE 以继续列举属性表。返回 FALSE 停止列举属性表。

备注：下列限制适用该回调函数：

请回调函数不应该产生对其他任务的控制或做一些有可能产生对其他任务的控制的事情。该回调函数可以调用 RemoveProp 函数。然而，RemoveProp 函数通过该回调函数的参数只能清除传给该回调函数的属性。该回调函数不应该试图增加属性。

速查：Windows NT：3.1 以上版本；Windows：95 的以上版本；Windows CE：不支持；头文件：winuser.h；库文件：用户自定义；Unicode：定义为 Unicode 和 ANSI 原型。

2.22.6 RemoveProp

函数功能：该函数从指定的窗口的属性表中删除一项。指定的字符串标识了要删除的项。

函数原型：HANDLE RemoveProp (HWND hWnd, LPCTSTR lpString);

参数：

hWnd：指向要改变属性项的窗口的句柄。

lpString：指向以 null 结尾的字符串指针，或者包含一个标识字符串的原子。如果该参数是一个原子，那么它必须是使用 AddAtom 函数创建的。原子是 16 位的数据值，它必须是放置在 lpString 参数的低位字中，

而高位率必须为 0。

返回值：返回值标识了指定的字符串。如果该串无法在指定的属性表中发现，那么返回值为 NULL。

备注：应用程序必须释放与从属性表中清除的项相关的数据句柄。应用程序只能清除它加入的那些属性它不能清除其他应用程序或系统本身加入的属性。

RemoveProp 函数返回与该字符串相关的数据句柄，这样应用程序就可以释放与该句柄相关的数据。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；文件：user32.lib；Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 版本实现。

2.22.7 SetProp

函数功能：该函数在指定窗口的属性表中增加一个新项，或者修改一个现有项。如果指定的字符串不在属性表中，那么就增加该新的项，新项中包含该字符串和句柄，否则就用指定的句柄替换该字符串的全前句柄。

函数原型：BOOL SetProp（HWND hWnd，LPCTSTR lpString，HANDLE hData）；

参数：

hWnd：指向窗口的句柄，该窗口的属性表要接收一个新项。

lpString：指向以 null 结尾的字符串指针，或者包含一个标识字符串的原子。如果该参数是一个原子，么它必须是以前使用 GlobalAddAtom 函数创建的。原子是 16 位的数据值，它必须是放置在 lpstring 参数低位字中，而高位字必须为 0。

hData：指向要拷贝到属性表中的数据的句柄。该数据句柄可以标识任何对应用程序有用的值。

返回值：如果该数据句柄和字符串以加到属性表中，那么返回值为非零。如果该函数失败，那么返回为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：在清除窗口之前（也就足在处理 WM_DESTROY 消息之前），应用程序必须把它加到属性表的所项清除。应用程序必须使用 RemoveProp 函数来清除这些项。

速查：Windows NT：3.1 以上版本；Windows：95 以上版本；Windows CE：不支持；头文件：winuser.h；文件：user32.lib；Unicode：在 Windows NT 环境中以 Unicode 和 ANSI 版本实现。

第三章　图形设备接口函数

图形设备接口（GDI）提供了一系列的函数和相关的结构，应用程序可以使用它们在显示器、打印机或其他设备上生成图形化的输出结果。使用 GDI 函数可以绘制直线、曲线、闭合图形、路径、文本以及位图图像。所绘制的图形的颜色和风格依赖于所创建的绘图对象，即画笔、笔刷和字体。你可以使用画笔来绘制直线和曲线，使用笔刷来填充闭合图形的内部，使用字体来书写文本。

应用程序通过创建设备环境(DC),可以直接向指定的设备进行输出。设备环境是一个 GDI 管理的结构。其中包含一些有关设备的信息,比如它的操作方式及当前的选择。应用程序可使用设备环境函数来创建 DC。GDI 将返回一个设备环境句柄，在随后的调用中，该句柄用于表示该设备。例如，应用程序可以使用该句柄来获取有关该设备性能的一些信息，诸如它的类型（显示器、打印机或其他设备），它的显示界面的尺寸和分辨率等。

应用程序可以直接向一个物理设备进行输出，比如显示器或打印机；也可以向一个"逻辑"设备进行输出，比如内存设备或元文件。逻辑设备向应用程序所提供的保存输出结果的格式，可以很容易地将其发送到物理设备上。一旦应用程序将输出结果记录到了一个元文件中，那么该元文件就可以被使用任意多次，并且该输出结果可以被发送到任意多个物理设备上。

应用程序可以使用属性函数来设置设备的操作方式和当前的选择。操作方式包括文本和背景颜色，混色方式（也称为二元光栅操作，用于确定画笔或笔刷的颜色与绘图区域现有的颜色如何进行混色），映射方式（用于指定 GDI 如何将应用程序所用的坐标映射到设备坐标系统上）。当前的选择是指绘图时使用哪个绘图对象。

图形设备接口函数包括以下几类：

3.1　位图函数（Bltmap）

位图是一个图形对象，可将图像作为文件进行创建、处理（比例缩放、滚动、旋转和绘制）和存储。

位图函数提供了一系列处理位图的方法。

3.1.1AlphaBlend

函数功能：该函数用来显示透明或半透明像素的位图。

函数原型：AlphaBlend(HDC hdcDest,int nXOriginDest,int nYOriginDest,int nWidthDest,int hHeightDest,HDC hdcSrc,int nXOriginSrc,int nYOriginSrc,int nWidthSrc,int nHeightSrc,BLENDFUNCTION blendFunction)；

参数：

hdcDest：指向目标设备环境的句柄。

nXoriginDest：指定目标矩形区域左上角的 X 轴坐标，按逻辑单位。

nYOriginDest：指定目标矩形区域左上角的 Y 轴坐标，按逻辑单位。

nWidthDest：指定目标矩形区域的宽度，按逻辑单位。

hHeghtdest：指向目标矩形区域高度的句柄，按逻辑单位。

hdcSrc：指向源设备环境的句柄。

nXOriginSrc：指定源矩形区域左上角的 X 轴坐标，按逻辑单位。

nYOriginSrc：指定源矩形区域左上角的 Y 轴坐标，按逻辑单位。

nWidthSrc：指定源矩形区域的宽度，按逻辑单位。

nHeightSrc：指定源矩形区域的高度，按逻辑单位。

blendFunction：指定用于源位图和目标位图使用的 alpha 混合功能，用于整个源位图的全局 alpha 值和格式信息。源和目标混合功能当前只限为 AC_SRC_OVER。

返回值：如果函数执行成功，那么返回值为 TRUE；如果函数执行失败，那么返回值为 FALSE。

Windows NT：若想获取更多错误信息，请调用 GetLastError 函数。

备注：如果源矩形区域与目标矩形区域大小不一样，那么将缩放源位图与目标矩形区域匹配。如果使用 SetStretchBltMode 函数，那么 iStretchMode 的值是 BLACKONWHITE 和 WHITEONBLACK，在本函数中，iStretchMode 的值自动转换成 COLORONCOLOR。目标坐标使用为目标设备环境当前指定的转换方式进行转换。源坐标则使用为源设备环境指定的当前转换方式进行转换。如果源设备环境标识为增强型图元文件设备环境，那么会出错（并且该函数返回 FALSE）。如果目标和源位图的色彩格式不同，那么 AlphaBlend 将源位图转换以匹配目标位图。

AlphaBlend 不支持镜像。如果源或目标区域的宽度或高度为负数，那么调用将失败。

速查：Windows NT：5.0 及以上版本；Windows：98 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：作为一个资源包含在 msimg32.dll 中。

BitBlt

函数功能：该函数对指定的源设备环境区域中的像素进行位块（bit_block）转换，以传送到目标设备环境。

函数原型：BOOL BitBlt(HDC hdcDest,int nXDest,int nYDest,int nWidth,int nHeight,HDC hdcSrc,int nXSrc,int nYSrc,DWORD dwRop)；

参数：

hdcDest：指向目标设备环境的句柄。

nXDest：指定目标矩形区域左上角的 X 轴逻辑坐标。

nYDest：指定目标矩形区域左上角的 Y 轴逻辑坐标。

nWidth：指定源和目标矩形区域的逻辑宽度。

nHeight：指定源和目标矩形区域的逻辑高度。

hdcSrc：指向源设备环境的句柄。

nXSrc：指定源矩形区域左上角的 X 轴逻辑坐标。

nYSrc：指定源矩形区域左上角的 Y 轴逻辑坐标。

dwRop：指定光栅操作代码。这些代码将定义源矩形区域的颜色数据，如何与目标矩形区域的颜色数据组合以完成最后的颜色。

下面列出了一些常见的光栅操作代码：

BLACKNESS：表示使用与物理调色板的索引 0 相关的色彩来填充目标矩形区域，（对缺省的物理调色板而言，该颜色为黑色）。

DSTINVERT：表示使目标矩形区域颜色取反。

MERGECOPY：表示使用布尔型的 AND（与）操作符将源矩形区域的颜色与特定模式组合一起。

MERGEPAINT：通过使用布尔型的 OR（或）操作符将反向的源矩形区域的颜色与目标矩形区域的颜色合并。

NOTSRCCOPY：将源矩形区域颜色取反，于拷贝到目标矩形区域。

NOTSRCERASE：使用布尔类型的 OR（或）操作符组合源和目标矩形区域的颜色值，然后将合成的颜色取反。

PATCOPY：将特定的模式拷贝到目标位图上。

PATPAINT：通过使用布尔 OR（或）操作符将源矩形区域取反后的颜色值与特定模式的颜色合并。然后使用 OR（或）操作符将该操作的结果与目标矩形区域内的颜色合并。

PATINVERT：通过使用 XOR（异或）操作符将源和目标矩形区域内的颜色合并。

SRCAND：通过使用 AND（与）操作符来将源和目标矩形区域内的颜色合并。

SRCCOPY：将源矩形区域直接拷贝到目标矩形区域。

SRCERASE：通过使用 AND（与）操作符将目标矩形区域颜色取反后与源矩形区域的颜色值合并。

SRCINVERT：通过使用布尔型的 XOR（异或）操作符将源和目标矩形区域的颜色合并。

SRCPAINT：通过使用布尔型的 OR（或）操作符将源和目标矩形区域的颜色合并。

WHITENESS：使用与物理调色板中索引 1 有关的颜色填充目标矩形区域。（对于缺省物理调色板来说，这个颜色就是白色）。

返回值：如果函数成功，那么返回值非零；如果函数失败，则返回值为零。

Windows NT：若想获取更多错误信息，请调用 GetLastError 函数。

备注：如果在源设备环境中可以实行旋转或剪切变换，那么函数 BitBlt 返回一个错误。如果存在其他变换（并且目标设备环境中匹配变换无效），那么目标设备环境中的矩形区域将在需要时进行拉伸、压缩或旋转。

如果源和目标设备环境的颜色格式不匹配，那么 BitBlt 函数将源场景的颜色格式转换成能与目标格式匹配的格式。当正在记录一个增强型图元文件时，如果源设备环境标识为一个增强型图元文件设备环境，那么会出现错误。如果源和目标设备环境代表不同的设备，那么 BitBlt 函数返回错误。

Windows CE：在 Windows CE 1.0 版中，参数 dwRop 只可以指定为下列值：SRCCOPY、SRCAND、SRCPAINT、SRCINVERT。在 Windows CE 2.0 版中，参数 dwRop 可以是任何光栅操作代码值。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.1.2CreateBitmap

函数功能：该函数创建一个带有特定宽度、高度和颜色格式的位图。

函数原型：HBITMAP CreateBitmap(int nWidth,int nHeight, UINT cPlanes, UINT cBitsPeral,CONST VOID *lpvBits)；

参数：

nWiath：指定位图宽度、单位为像素。

NHeight：指定位图高度、单位为像素。

CPlanes：指定该设备使用的颜色位面数目。

CBitsPerPel：指定用来区分单个像素点颜色的位数（比特数目）。

LpvBits：指向颜色数据数组指针。这些颜色数据用来设置矩形区域内像素的颜色。矩形区域中的每一扫描线必须是双字节的整数倍（不足部分以 0 填充）。如果该参数为 NULL，那么就表示没有定义新位图。

返回值：如果函数成功，那么返回值是位图的句柄；如果失败，那么返回值为 NULL。若想获取更多错误信息，请调用 GetLastError 函数。

备注：在创建完位图之后，可以通过使用 SeletObject 函数把它选入到设备环境中。尽管函数 CreateBitmap 可以用来创建彩色位图，但由于性能方面的原因，应用程序使用 CreateBitmap 函数来创建单位色位图，创建彩色位图应该使用函数 CreateCompatibleBitmap。当由 CreateBitmap 创建而返回的彩色位图被选入到设备环境时，系统必须确保选入进去的设备环境格式与位图匹配。由于函数 CreateComapatbleBitmap 获取设备环境，所以它返回的位图与指定的设备环境有相同的格式。由于这个原因，对 SelectObject 的后续调用都要比从 CreateBitmap 函数创建返回的彩色位图调用快。

如果位图是单色的，那么对于目标设备环境而言，0 表示前景颜色，而 1 表示背景颜色。

如果应用程序将 nWidth 或 nHeight 参数设为 0，那么函数 CreateBitmap 返回的是只有一个含像素的单色位图的句柄。当不再需要位图时，可调用 DeleteObject 函数删除它。

Windows CE：参数 cPlanes 必须是 1。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.1.3CreateBitmaplndirect

函数功能： 该函数可以创建一个具有特定宽度、高度和颜色格式的位图。

函数原型： HBITMAP CreateBitmaplndirect(CONST BITMAP *lpbm)；

参数：

ipbm：指向 BITMAP 结构的指针。该结构包含有关位图的信息。如果应用程序将其成员 bmWidth 或 bmHeight 设为 0，那么 CreateBitmaplndirect 将返加一个只有 1 个像素点的单色位图的句柄。

返回值： 如果函数执行成功，那么返回值是指向位图的句柄；如果函数执行失败，则返回值为 NULL。若想获取更多错误信息，请调用 GetLastError 函数。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.4CreateCompatibleBitmap

函数功能： 该函数创建与指定的设备环境相关的设备兼容的位图。

函数原型： HBITMAP CreateCompatibleBitmap(HDC hdc,int nWidth,int nHeight)；

参数：

hdc： 设备环境句柄。

nWidth：指定位图的宽度，单位为像素。

nHeight：指定位图的高度，单位为像素。

返回值： 如果函数执行成功，那么返回值是位图的句柄；如果函数执行失败，那么返回值为 NULL。若想获取更多错误信息，请调用 GetLastError。

备注： 由 CreateCompatibleBitmap 函数创建的位图的颜色格式与由参数 hdc 标识的设备的颜色格式匹配。该位图可以选入任意一个与原设备兼容的内存设备环境中。由于内存设备环境允许彩色和单色两种位图。因此当指定的设备环境是内存设备环境时，由 CreateCompatibleBitmap 函数返回的位图格式不同。然而为非内存设备环境创建的兼容位图通常拥有相同的颜色格式，并且使用与指定的设备环境一样的色彩调色板。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.1.5CreateDIBBitmap

函数功能： 该函数由与设备无关的位图（DIB）创建与设备有关的位图（DDB），并且有选择地为位图置位。

函数原型： HBITMAP CreateDIBitmap(HDC hdc,CONST BITMAPINFONEADER *lpbmih,DWORD fdwlnit,CONST VOID *lpblnit,CONST BITMAPINFO *lpbmi,UINT fuUsage)；

参数：

hdc：设备环境句柄。

lpbmih：指向位图信息头结构的指针，它可以是下列操作系统位图信息头之一：

Windows NT 3.51 和早期：BITMAPINFOHEADER；Windows NT 4.0 和 Windows 95：BITMAPV4HEADER；Windows NT 5.0 和 Windows 98：BITMAPV5HEADER。

如果 fdwlnit 是 CBM_INIT,那么函数使用位图信息头结构来获取位图所需的宽度、高度以及其他信息。

注意高度若是正数,那么表示是自底向上 DIB,而负数表示为自顶向下 DIB,这种情况与 CreateDIBitmap 函数兼容。

Fdwlnit：位标识集。它指定系统如何对位图的位进行初始化。

下面是定义的位标志常量：

CBM_INIT：如果设置了该标志，那么系统将使用 lpblnit 和 lpbmi 两个参数指向的数据来对位图中的位进行初始化。如果没有该标志，那么表示上述两个参数指向的数据无效。如果 fdwlnit 为 0，那么系统不会对位图的位进行初始化。

lpblnit：该指针指向包含初始的位图数据的字节类型数组。数据格式与参数 lpbmi 指向的 BITMAPINFO 结构中的成员 biBitCount 有关。

lpbmi：指向 BITMAPINFO 结构的旨针。该结构描述了参数 lpbmi 指向的数组的维数和颜色格式。

fuUsage：表示 BITMAPINFO 结构的成员 bmiColors 是否初始化过，并且如果是，那么 bmiColors 是否包含明确的红、绿、蓝（RGB）值或调色板索引。参数 fuUsage 必须取下列值中的一个，这些值的含义为：

DIB_PAL_COLORS：表示提供一个颜色表，并且该表由该位图要选入的设备环境的逻辑调色板的 16 位索引值数组组成。

DIB_RGB_COLORS：表示提供一个颜色表，并且表中包含了原义的 RGB 值。

返回值：如果函数执行成功，返回值则是创建的位图的句柄；如果函数执行失败，那么返回值为 NULL，若想获取更多错误信息，请调用 GetLastError 函数。

备注：用于 fdwlnit 参数的 CBM__CREATDIB 标志不再被支持。当不再需要该位图时,可调用 DeleteObject 函数删除它。

ICM：参数 fuUsage 用来指定参数 lpbmi 指向的 BITMAPINFO 结构中的成员 bmiColors 是否包含颜色信息。如果 bmiColors 不包含颜色信息，那么不能进行位图的颜色管理。BITMAPINFO 中的 bmiColors 成员必须包含 BITMAPV4HEADER 或 BITMAPV5HEADER，以便能够进行颜色管理。在创建完位图之后，产生的位图的内容颜色不匹配。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.6CreateDIBSection

函数功能：该函数创建应用程序可以直接写入的、与设备无关的位图（DIB）。该函数提供一个指针，该指针指向位图位数据值的地方。可以给文件映射对象提供句柄，函数使用文件映射对象来创建位图，或者让系统为位图分配内存。

函 数 原 型 ： HBITMAP CreateDIBSection(HDC hdc,CONST BITMAPINFO *pbmi,UINT iUsage,VOID *ppvBits,HANDLE hSection,DWORD dwOffset)；

参数：

hdc：设备环境句柄。如果 iUsage 的值是 DIB__PAL__COLORS，那么函数使用该设备环境的逻辑调色板对与设备无关位图的颜色进行初始化。

pbmi：指向 BITMAPINFO 结构的指针，该结构指定了与设备无关位图的各种属性，其中包括位图的维数和颜色。

iUsage：指定由 pbmi 参数指定的 BITMAPINFO 结构中的成员 bmiColors 数组包含的数据类型（要么是逻辑调色板索引值，要么是原文的 RGB 值）。下列值是系统定义的，其含义为：

DIB__PAL__COLORS：表示成员 bmiColors 是 hdc 指定的设备环境的逻辑调色板，使用的是 16 位索引值

数组。

DIB__RGB__COLORS：表示结构 BITMAPINFO 中包含了 RGB 值的数组。

ppvBits：指向一个变量的指针，该变量接收一个指向 DIB 位数据值的指针。

hSection:文件映射对象的句柄。函数将使用该对象来创建 DIB(与设备无关位图)。该参数可以是 NULL。

如果 hSection 不是 NULL,那么它一定是文件映射对象的句柄。该对象是通过调用带有 PAGE__READWRITE 或 PAGE__WRITECOPY 标志的 CreateFileMapping 函数创建的。不支持只读的 DIB 类型。通过其他方法创建的句柄将会引起 CreateDIBSection 函数执行失败。

如果 hSection 不是 NULL，那么函数 CreateDIBSection 将在 hSection 引用的文件映射对象中偏移量为 dwOffset 处获取位图的位数据值。应用程序可以在以后通过调用 GetObject 函数来检索 hSection 句柄，其中 GetObject 函数使用了 GreateDIBSection 函数返回的 GBITMAP 类型的对象。

如果 hSection 为 NULL，那么系统将为与设备无关位图（DIB）分配内存。在这种情况下，函数 CreateDIBSection 将忽略参数 dwOffset，应用程序无法在以后获取指向内存的句柄。通过调用 GetObject 函数来填充的 DIBSECTION 结构成员 dshSection 也将成为 NULL。

DwOffset：指定从 hSection 引用的文件映射对象开始处算起的偏移量，超过这个偏移量的地方就是位图的位数据值开始存放的地方。在 hSection 为 NULL 时忽略该值。位图的位数据值是以 DWORD 为单位计算的。

返回值：如果函数执行成功，那么返回值是一个指向刚刚创建的与设备无关位图的句柄，并且*ppvBits 指向该位图的位数据值；如果函数执行失败，那么返回值为 NULL，并且*ppvBit 也为 NULL，若想获得更多错误信息，请调用 GetLastError 函数。

备注：正如上面提到过，如果 hSection 为 NULL，那么系统为 DIB 分配内存。当以后通过调用 DeleteObject 函数删除该 DIB 时，系统将关闭指向相应内存的句柄。如果 Hsection 不为 NULL，那么在调用 DeleteObject 删除该位图之后，必须自己关闭 hSection 内存句柄。

Windows NT：需要保证 GDI 子系统在自己绘制位图之前已经完成所有由创建的位图。访问位图应该是同时的。这可以调用函数来进行。其适用于任何指向位图的位数据值指针的情况，也包括在调用象这样的函数时传送指针的情形。

：如果（由参数指向）结构成员不包含或，那么不进行颜色管理。否则，允许有颜色管理，并且有与该位图有关的特定的颜色空间。

：参数要设为。除非使用的是位图。在这种情况下，应设为。然而，如果使用了，操作系统将忽略结构成员数组中的值。另外，应将参数设为，忽略参数，可将它设为 0。

在版中，由参数指向的结构必定规定每像素点为 1 或 2 位。

在版中，如果图像是调色板模式（通常是 1，2，4 和 8 格式）的，那么结构中必须包括颜色表。对于或非调色板式的图像，颜色表必须是项长度，这项必须指定红、绿和蓝 3 色的位掩码值。对于图像，将忽略颜色表，其像素必须按（BGR）格式存储。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.1.7ExtFloodFill

函数功能：该函数将使用当前刷子填充显示表面区域。

函 数 原 型： BOOL ExtFloodFill(HDC hdc,int nXStart,int nYStart,COLORREF crColor,UINT fuFillType)；

参数：

hdc：设备环境句柄。

nXSTart：指定要开始填充处的逻辑 X 轴坐标。

nYStart：指定要开始填充处的逻辑 Y 轴坐标。

crColor：指定要填充的边界或区域的颜色。crColor 的具体解释要根据参数 fuFillType 的值而定。

fuFillType：指定要进行的填充操作类型。该参数必须是下列值之一，这些值的含义如下：

FLOODFILLBORDER:表示填充区域是由 crColor 参数指定的颜色包围起来的部分。这种形式与 FloodFill 函数执行的填充类型一样。

FLOODFILLSURFACE：表示填充区域是由 crColor 指定的颜色来定义。填充操作向四周伸展，直到遇到这种颜色为止。这种操作式样对于带有多种颜色边界的填充区域有用。

返回值：如果函数执行成功，那么返回值为非零；如果函数执行失败，那么返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：下列原因可能引起函数执行失败：

填充无法完成。

指定的像素点有着参数 crColor（如果要求 FLOODFILLBORDER 操作样式）指定的边界颜色（即颜色相同）。

指定的像素点没有参数 crColor（如果要求 FLOODFILLSURFACE 操作样式）指定的颜色。

该点在剪辑区之外——也就是说在设备中不可见。

如果 fuFillType 参数为 FLOODFILLBORDER，那么系统认为要填充的区域是完全被参数 crColor 指定的颜色包围起来的。该函数从参数 nXStart 和 nYStart 指定的点开始填充，向四周继续，直到遇到边界为止。

如果 fuFillType 是 FLOODRILLSURFACE，那么系统就认为要填充的区域是单颜色的，函数从 nXStart 和 nYStart 两个参数指定的点开始填充区域，并向四周延伸，对包含参数 crColor 指定颜色的所有相邻区域进行填充。

只有支持光栅显示操作的设备和内存设备环境才支持 ExtFloodFill 函数。为了确定设备是否支持该技术，可使用函数 GetDeviceCaps。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.8GetBitmapDimensionEx

函数功能：该函数用来检索位图的大小（维数）。检索到的大小必须已经由 SetBitmapDimensionEX 函数设置完毕。

函数原型：BOOL GetBitmapDimensionEx(HBITMAP hBitmap, LPSIZE lpDimension)；

参数：

hBitmap：指向位图的句柄。

lpDimension：指向接收位图大小的 SIZE 结构的指针。

返回值：如果函数执行成功，那么返回值非零；如果函数执行失败，那么返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：该函数返回的数据结构中包含用于位图高度和宽度的域，如果其大小仍未设置，那么返回的结构将其设置为 0。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.9GetDIBColorTable

函数功能：该函数从 DIB 位图的颜色表中检索 RGB（红、绿、蓝）颜色值，此 DIB 位图是当前选入指

定设备环境中的位图。

函数原型：UINT GetDIBColorTable(HDC hdc,UINt uStartindex, UINT cEntries, RGBQUAD *pColors)

参数：

hdc：指定设备环境，DIB 类位图必须选入到该设备环境中。

uStartindex：一个从零开始的颜色表索引值，该索引值指定了要检索的第 1 个颜色色表项。

cEntries：指定要检索的颜色表项的数目。

pColors：指向一个缓冲区的指针，该缓冲区接收一个 RGBQUAD 结构数值，该结构包含 DIB 颜色表中的颜色信息。该缓冲区必须足够大，以包含象 cEntries 参数值一样多的 RGBQUAD 数据结构。

返回值：如果函数执行成功，那么返回值就是函数检索到的颜色表项的数目；如果函数执行失败，那么返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：GetDIBColorTable 函数应该是用来检索使用 1、4 或 8 位像素点格式的 DIB 位图的颜色表的，与位图相关的 BITMAPINFOHEADER 结构中的成员 biBitCount 规定了每个像素点的位数。biBitCount 值大于 8 的 DIB 类位图没有颜色表，但它们有相关的颜色掩码。可以调用 GetObject 函数来检索那些颜色掩码。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.10 GetDIBits

函数功能：GetDIBits 函数检取指定位图的信息，并将其以指定格式复制到一个缓冲区中。

函数原型：int GetDIBits(HDC hdc, HBITMAP hbmp, UINT uStartScan, UINT cScanLines, LPVOID lpvBits, LPBITMAPINFO lpbi, UINT uUsage)；

参数：

hdc：设备环境句柄。

hbmp：位图句柄。

uStartScan：指定检索的第一个扫描线。

cScanLines：指定检索的扫描线数。

lpvBits：指向用来检索位图数据的缓冲区的指针。如果此参数为 NULL，那么函数将把位图的维数与格式传递给 lpbi 参数指向的 BITMAPINFO 结构。

lpbi：指向一个 BITMAPINFO 结构的指外，此结构确定了设备无在位图的数据格式。

uUsage：指定 BITMAPINFO 结构的 bmiColors 成员的格式。它必须为下列取值：

DIB_PAL_COLORS：颜色表由指向当前逻辑调色板的 16 位索引值数组构成。

DIB_RGB_COLORS：颜色表由红、绿、蓝（RGB）三个直接值构成。

返回值：如果 lpvBits 参数非空，并且函数调用成功，那么返回值为从位图复制的扫描线数。

Windows 95 和 Windows 98:如果 lpvBits 参数为 NULL 并且 GetDIBits 成功地填充了 BITMAPINFO 结构，那么返回值为位图中总共的扫描线数。

Windows NT：如果 lpvBits 参数为 NULL 并且 GetDIBits 成功地填充了 BITMAPINFO 结构，那么返回值为非 0。如果函数执行失败，那么将返回 0 值。Windows NT:若想获得更多错误信息，请调用 callGetLastError 函数。

注释：如果所需要的 DIB 格式与其内部格式相匹配，那么位图的 RGB 值将被复制。如果不匹配，那么将合成一个颜色表。下表描述了针对每一种颜色格式所合成的颜色表。

1_BPP：颜色表中仅包含黑白表项。

4_BPP：颜色表由标准 VGA 定义的颜色组合而成。

8_BPP：颜色表由 GDI 定义的 256 色组合而成。

24_BPP：不返回颜色表。

如果 lpvBits 参数为一个有效指针，那么位图信息头结构的前 6 个成员必须初始化为 DIB 的大小和格式。

注意：位图信息头结构可为以下几种格式：

操作系统位图信息头结构(Operating System Bitmap Information Header)

Windows NT 3.51 及早期的 BITMAPINFOHEADER

Windows NT 4.0 及 Windows 95 中的 BITMAPV4HEADER

Windows NT 5.0 及 Windows 98 中的 BITMAPV5HEADER

通过将高度设为正数来指定一个自下而上的 DIB，而自上而下的 DIB 则通过设置一个负的高度值来指定。位图的颜色表将附加在 BITMAPINFO 结构的后面。

如果 lpvBits 为 NULL，那么 GetDIBits 将检查 lpbi 所指向的第一个结构的第一个成员。这一成员必须指定 BITMAPCOREHEADER 结构或位图信息头结构的字节数。函数将通过指定的大小来确定剩余成员如何被初始化。

如果 lpvBits 为 NULL，并且 BITMAPINFO 结构的 bit count 成员初始化为 0，那么 GetDIBits 将不填充 BITMAPCOREHEADER 结构或位图信息头结构的颜色表部分。这一技术可用来查询位图属性。

应用程序调用这个函数时必须将 hbmp 参数所标识的位图选择到一个设备环境中。

自下而上 DIB 的原点为位图的左下角，自上而下 DIB 的原点为其左上角。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.11 GetPixel

函数功能：该函数检索指定坐标点的像素的 RGB 颜色值。

函数原型：；COLORREF GetPixel(HDC hdc, int nXPos, int nYPos)

参数：

hdc：设备环境句柄。

nXPos：指定要检查的像素点的逻辑 X 轴坐标。

nYPos：指定要检查的像素点的逻辑 Y 轴坐标。

返回值：返回值是该象像点的 RGB 值。如果指定的像素点在当前剪辑区之外；那么返回值是 CLR_INVALID。

备注：该像素点必须在当前剪辑区的边界之内。并不是所有设备都支持 GetPixel 函数。应用程序应调用 GetDeviceCaps 函数来确定指定的设备是否支持该函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.1.12 GetStretchBltMode

函数功能：该函数获取当前伸展（也称展宽）模式。伸展模式定义了如何将颜色数据增加到位图中，或如何从位图中移走。当调用 StretchBlt 函数时，位图可能进行伸展或压缩处理。

函数原型：int GetStretchBltMode(HDC hdc)；

参数：

hdc：设备环境句柄，

返回值：如果函数执行成功，那么返回值是当前伸展模式；如果函数执行失败，那么返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastErrr 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.13 GradientFill

函数功能：该函数填充矩形和三角形结构。

函数原型：BOOL GradientFill(HDC hdc, CONST PTRIVERTEX pVertex, DWORD dwNumVertex, CONST PVOID pMesh, DWORD dwNumMesh, DWORD dwMode)；

参数：

hdc：指向目标设备环境的句柄。

pVertex：指向 TRIVERTEX 结构数组的指针，该数组中的每项定义了三角形顶点。

dwNumVertex：顶点数目。

pMesh：三角形模式下的 GRADIENT_TRIANGLE 结构数组，或矩形模式下的 GRADIENT_RECT 结构数组。

dwNumMesh：参数 pMesh 中的成员数目（这些成员是三角形或矩形）。

dwMode：指定倾斜填充模式。该参数可以包含下列值，这些值的含义为：

GRADIENT_FILL_RECT_H：在该模式下，两个端点表示一个矩形。该矩形被定义成左右边界具有固定颜色（由 TRIVERTEX 结构指定）。GDI 从上至下插入颜色，并填充内部区域。

GRADIENT_FILL_RECT_V：在该模式下，两个端点表示一个矩形。该矩形定义其顶部和底部边界的颜色为固定值（通过 TRIVERTEX 结构指定），GDI 从顶至底部边界插入颜色，并填充内部区域。

GRADIENT__FILL__TRIANGLE：在该模式下，TRIVERTEX 结构数组以及描述单个三角形的数组索引序列被传给 GDI。GDI 在三角形顶点之间进行线性插值，并填充内部区域。在 24 和 32 位 / 像素模式下，绘图是直接进行。在 16、8、4 和 1 位 / 像素模式中进行抖动处理。

返回值：如果函数执行成功，那么返回值为 TRUE；如果函数执行失败，则返回值为 FALSE。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：若想在矩形区域中加入一些平滑的阴影（底纹），请用三角形的三个顶点调用 GradientFill 函数。CDI 将进行线性插值，并填充矩形区域。在绘制矩形时可能使用两种阴影模式在水平模式中，矩形从左至右开始变暗，在垂直模式中则是从上至下进行。

GradientFill 函数使用网眼法（mesh method）来表示要绘制对象的端点。所有传给 GradientFill 的顶点都存储在 pVertex 数组中。参数 pMesh 指定了这些顶点如何连接形成一个对象。当填充矩形时，pMesh 指向一个 GRADIENT__RECT 结构数组。每一个 GRADIENT__RECT 结构指定了 pVertex 数组中两个顶点的索引值，这两个顶点形成一个矩形的左上角和右下角坐标。

在填充三角形情况下，pMesh 指向的是 GRADIENT__TRIANGLE 结构数组。每一个 GRADIENT__TRIANGLE 结构指定了 pVertex 数组中的 3 个顶点，这三个顶点形成一个三角形。

为了简化硬件加速，无需要求该例程在三角形内部具有像素完善特性。

若想了解更多信息，参考平滑成影、绘制带阴影的三解和矩形等方面的内容。

速查：Windows NT：5.0 及以上版本；Windows：98 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：作为一个资源包含在 msimg32.dll 中。

3.1.14 PlgBlt

函数功能：该函数对源设备环境中指定的矩形区域中的颜色数据位进行位块转换，并转换到目标设备环境中指定的平行四边形里。如果给定的位掩码句柄表示为有效的单色位图，那么函数使用该位图来对源矩形中的颜色数据进行掩码（屏蔽）操作。

函数原型： BOOL PlgBlt(HDC hdcDest, CONST POINT *lpPoint, HDC hdcSrc,int nXSrc, int nYSrc, int nWidth, int nHeight, HBITMAP hbmMask, int xMask, int yMask)；

参数：

hdcDest：指向目标设备环境的句柄。

lpPoint：指向代表目标平行四边形 3 个角的 3 个顶点的数组指针。源矩形的左上角映射到该数组的第 1 个顶点，右上角映射为数组中的第 2 个顶点，左下角映射成第 3 个顶点。而右下角则映射成平行四边形中隐含的第 4 个点。

hdcSrc：指向源设备环境的句柄。

nXSrc：指定源矩形左上角的 X 轴坐标，按逻辑单位。

nYSrc：指定源矩形左上角的 Y 轴坐标，按逻辑单位。

nWidth：指字源矩形的宽度，按逻辑单位。

nheight：指定源矩形的高度，按逻辑单位。

hbMask：指向可选的单色位图的句柄。该位图是用来对源矩形的颜色进行屏蔽用的。

xMask：指定单色位图左上角的 X 轴坐标。

yMask：指定单色位图左上角的 Y 轴坐标。

返回值： 如果函数成功，那么返回值非零；如果函数失败，则返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 平行四边形的第 4 个顶点（D）是通过把头 3 个顶点（A、B、C）当作矢量，并按 D＝B＋C＋A 计算来定义的。如果存在位掩码，那么掩码中的数值 1 表示应该将源像素的颜色拷贝到目标像素点上。掩码中的数值 0 表示不改变目标像素颜色。如果掩码矩形比源和目标矩形要小，那么该函数重复掩码模式。

在源设备环境中，允许有伸缩、平移和反射变换，然而禁止有旋转和剪切变换。如果掩码位图不是单色位图，那将会出现错误。在需要时，目标设备环境用的伸展模式是用来定义如何拉伸或压缩像素的。

当正在对一个增强型图元文件进行记录时，如果源设备环境标识为增强型图元文件设备环境，那么将出现错误。目标坐标根据目标设备环境进行变换，源坐标是根据源设备环境进行变换。如果源变换有旋转或剪切，那么会返回一个错误。如果目标和源矩形颜色格式不相同，那么 Plgblt 对源矩形进行转换，以与目标矩形匹配。不是所有设备都支持 PlgBlt 函数，若想了解更多信息，那么 GetDeviceCaps 函数中有关 RC_BITBLT 光栅特性的描述。如果源和目标设备环境代表不兼容设备，那么 PlgBlt 返回一个错误。

速查： Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.15 SetDIBits

函数功能： 该函数使用指定的 DIB 位图中发现的颜色数据来设置位图中的像素。

函数原型： int SetDIBits(HDC hdc, HBITMAP hbmp, UINT uStartScan, UINT cScanLines, CONST VOID *lpvBits,CONST BITMAPINFO *lpbmi, UINT fuColorUse)；

参数：

hdc：指向设备环境中的句柄。

hbmp：指向位图的句柄。函数要使用指定 DIB 中的颜色数据对该位图进行更改。

uStartScan：为参数 lpvBits 指向的数组中的、与设备无关的颜色数据指定起始扫描线。

cScanLines：为包含与设备无关的颜色数据的数组指定扫描线数目。

lpvBits：指向 DIB 颜色数据的指针，这些数据存储在字节类型的数组中，位图值的格式取决于参数 lpbmi 指向的 BITMAPINFO 结构中的成员 biBitCount。

lpbmi：指向 BITMAPINFO 数据结构的指针，该结构包含有关 DIB 的信息。

fuColorUse：指定是否提供了 BITMAPINFO 结构中的 bmiColors 成员，如果提供了，那么 bmiColors 是

否包含了明确的 RGB 值或调色板索引。参数 fuColorUse 必须取下列值，各值的含义为：

DIB_PAL_COLORS：颜色表由 16bit 的索引值数组组成。这些值可以对由 hdc 参数标识的设备环境中的逻辑调色板进行索引。

DIB_RGB_COLORS：提供了颜色表，并且表中包含了原义的 RGB 值。

返回值： 如果函数成功，那么返回值就是复制的扫描线数；如果函数失败，那么返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 当位图的位要索引到系统调色板时，可获取最佳的位图绘制速度。

应用程序可能通过调用 GetSystemPaletteEntries 函数来检索系统调色板颜色和索引。在检索到颜色和索引值之后，应用程序可以创建 DIB，有关更多的信息，请参考系统调色板(System Paletle)。

只有在参数 fuColorUse 设置为 DIB_PAL_COLORS 常量时才使用参数 hdc 标识的设备环境，否则会忽略 hdc 参数中的值。

在应用程序调用该函数时，必须把由参数 hbmp 标识的位图选入到设备环境中。

自底向上的 DIB 位图的起始点是该位图的左下角处，自顶向下的 DIB 位图的起源点在该位图的左上角处。

ICM：颜色管理照样进行。如果指定的 BITMAPINFO 结构不是 BITMAPV4HEADER 或 BITMAPV5HEADER，那么当前设备环境的颜色配置（profile）就用作源颜色配置。如果 BITMAPINFO 结构不是 BITMAPV4HEADER 或 BITMAPV5HEADER,那么使用 RGB 颜色。如果指定的 BITMAPINFO 结构是 BITMAPV4HEADER 或 BITMAPV5HEADER，那么与该位图有关的颜色配置(profile)被用作源颜色。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h：库文件：gdi32.lib。

3.1.16 SetBitmapDimensionEx

函数功能： 该函数为位图指定首选的大小。这些大小可以为应用程序所用，然而不能被系统所用。

函数原型： BOOL SetBitmapDimensionEx(HBITMAP bBitmap, int nWidth, int nHeight, LPSIZE lpSize)；

参数：

hBitmap：指向位图的句柄。该位图不可能是 DIB 类型的位图。

nWidth：指定位图的宽度，以 0.1 毫米为单位。

nHeight：指定位图的高度，以 0.1 毫米为单位。

lpSize：指向 SIZE 结构的指针。该结构用来接收该位图以前的大小。该指针可以为 NULL。

返回值： 如果该函数成功，那么返回值非零；如果该函数失败，那么返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 应用程序可以通过调用 GetBitmapDimensionEX 函数来检索位图的大小，该位图的大小是用函数 SetBitmapDimensionEX 设置的。

由 hBitmap 参数标识的位图不能是 DIB 类型，DIB 类型位图是由函数 GreateDIBSection 来创建的。如果该位图为 DIB 类型，那么函数 SetBitmapDimensionEX 会失败。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h：库文件：gdi32.lib。

3.1.17 MaskBlt

函数功能： 该函数使用特定的掩码和光栅操作来对源和目标位图的颜色数据进行组合。

函数原型： BOOL MaskBlt(HDC hdcDest, int nXDest, int nYDest, int nWidth, int nHeight, HDC hdcSrc,

int nXSrc, int nYSrc, HBITMAP hbmMask, int xMask, int yMask, DWORD dwRop)；

参数：

hdcDest：指向目标设备环境的句柄。

nXDest：指定目标矩形左上角的逻辑 X 轴坐标。

nYDest：指定目标矩形左上角的逻辑 Y 轴坐标。

nWidth：指定目标矩形和源位图的宽度，按逻辑单位。

nHeight：指定目标矩形和源位图的高度，按逻辑单位。

hdcSrc：指向源位图所在的设备环境，如果 dwRop 参数规定的光栅操作不包括源位图，那么该参数必须为 0。

nXSrc：指定源位图左上角的逻辑 X 轴坐标。

nYSrc：指定源位图左上角的 Y 轴逻辑坐标。

nbmMask：指向单色掩码位图的句柄，该位图与源设备环境中的彩色位图进行组合。

xMask：指定由参数 hbmMask 指向的掩码位图的水平像素偏移量。

yMask：指定由参数 hbmMask 指向的掩码位图的垂直像素偏移量。

dwRop：指定前景和背景光栅操作码，函数使用这些操作码为控制源和目标数据的组合。背景光栅操作码存储在该参数值的高位字中的高位字节中，而前景光栅操作代码雄在该参数值的高位字中的低位字节中，而低位字则忽略，并且应该为 0。宏 MAKEROP4 创建前景和背景光栅操作码这样的组合。

有关该函数场景中的前景和背景方面的讨论，可参考下面的备注一节。

至于有关公用的我栅操作码清单，可参考 BitBlt 函数。

返回值：如果函数执行成功，那么返回值为非零，如果函数失败，那么返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：在由 hbmMask 指定的掩码中，数值 1 表示在那个位置应使用 dwRop 指定的前景光栅操作码。数值 0 表示应使用 dwRop 指定的背景光栅操作码。如果光栅操作需要源对象，那么掩码矩形必须覆盖源矩形。如果没有覆盖，那么函数会执行失败。如果光栅操作没有要求源对象，那么掩码矩形必须覆盖目标矩形，如果没有覆盖，那么函数会失败。如果在调用函数时，源设备环境中实行旋转或剪切变换，那么会出现错误。然而，允许有其他类型的变换。如果源位图的颜色格式、模式和目标位图不一样，那么该函数对模式或源位图格式，或者两者进行转换，以与目标格式匹配。如果掩码位图不是单色位图，则会出现错误。当正在记录的是增强型图元文件时，如果源设备环境标识为增强型图元文件设备环境，那么会出现错误（该函数返回 FALSE）。不是所有设备都支持 MaskBlt 函数，应用程序应调用 GETDeviceCaps 函数来确定设备是否支持该函数。如果没有提供掩码位图，那么该函数非常类似 BitBlt，它使用前景光栅操作码。ICM：当出现尖头消息时，不进行颜色管理。

Windows CE：Windows CE 1.0 版只支持 SRCCOPY 和 SRCINVERT 光栅操作。该函数在 Windows CE 2.0 版与其他 Windows 桌面平台一样。

速查：Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.1.18 SetDIBColorTable

函数功能：该函数用来对目前进入指定设备环境的设备无关位图（DIB）颜色表中的项设置 RGB（红、绿、蓝）颜色值。

函数原型：UINT SetDIBColorTable(HDC hdc, UINT uStartindex, UINT cEntries, CONST RGBQUAD *pColors)；

参数：

hdc：指定设备环境。一个与设备无关位图必须被选入到该设备环境中。

uStartindex：一个从零开始的颜色表索引，该索引指定了要设置的第 1 个颜色表项。

cEntries：指定要设置的颜色表项的数目。

pColors：指向 RGBQUAD 结构数组的指针。该结构包含了用于 DIB 颜色表的新颜色信息。

返回值：如果函数执行成功，那么返回值就是该函数设置的颜色表项的数目；如果函数执行失败，那么返回值为 0。Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：使用该函数是用来对那些使用 1、4、或 8 位 / 像素格式的 DIB 位图的颜色表进行设置的。位图信息头结构中的成员 BitCount 表示了位图中每像素占多少位这种信息。

注意：位图信息头结构可以为：

Windows NT 3.51 和以前版；BITMAPINFOHEADER，BITMAPINFOHEADER 结构指定了每像素占多少位。BiBitCount 值大于 8 的设备无关位图（DIB）没有颜色表。

Windows NT 4.0 和 Windows 95：BITMAPV4HEADER，对于 Windows NT 4.0 和 Windows 95，与位图相关的 BITMAPV4HEADER 结构成员 bV 4BitCount 指定了每像素点占的位数。bV 4BitCount 值大于 8 的 DIB 没有颜色表。

Windows NT 5.0 和 Windows 98：BITMAPV5HEADER，对于 Windows NT 5.0 和 Windows 98，与位图相关的 BITMAPV4HEADER 结构中的成员 bV 5BitCount 指定了位图中每像素点占的位数。bV 5BitCount 值大于 8 的 DIB 位图没有颜色表。

ICM：不进行颜色管理。

速查：Windows NT：3.5 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.19 SetDIBitsToDevice

函数功能：该函数使用 DIB 位图和颜色数据对与目标设备环境相关的设备上的指定矩形中的像素进行设置。对于 Windows 98 和 Windows NT 5.0，函数 SetDIBitsToDevice 已经得到扩展，它允许 JPEG 图像作为源图像。

函数原型：int SetDIBitsToDevice(HDC hdc, int xDest, int Ydest, DWORD dwWidth, DWORD dwHeight, intXSrc, int Ysrc, UINT uStartScan, UINT cScanLines, CONST VOID *lpvBits, CONST BITMAPINFO *lpbmi, UINT fuColorUse)；

参数：

hdc：设备环境句柄。

XDest：指定目标矩形左上角的 X 轴坐标，按逻辑单位表示坐标。

YDest：指字目标矩形左上角的 Y 轴坐标，按逻辑单位表示坐标。

dwWidth：指定 DIB 的宽度，按逻辑单位表示宽度。

dwHeight：指定 DIB 的高度，按逻辑单位表示高度。

XSrc：指定 DIB 位图左下角的 X 轴坐标，按逻辑单位表示坐标。

YSrc：指定 DIB 位图左下角的 Y 轴坐标，按逻辑单位表示坐标。

uScanLines：指定 DIB 中的起始扫描线。

cScanLInes：指定参数 lpvBits 指向的数组中包含的 DIB 扫描线数目。

lpvBits：指向存储 DIB 颜色数据的字节类型数组的指针。关于更多的信息，请参考下面的备注一节。

lpbmi：指向 BITMAPINFO 结构的指针，该结构包含有关 DIB 的信息。

fuColorUse：指向 BITMAPINFO 结构中的成员 bmiColors 是否包含明确的 RGB 值或对调色板进行索引的值。有关更多的信息，请参考下面的备注部分。

参数 fuColorUse 必须是下列值之一，这些值的含义如下：

DIB__PAL__COLORS：表示颜色表由 16 位的索引值数组组成，利用这些值可对当前选中的逻辑调色板进

行索引。

DIB__RGB__COLORS：表示颜色表包含原义的 RGB 值。

返回值：如果函数执行成功，那么返回值是设置的扫描线数目；如果函数失败，那么返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

Windows 98、NT 5.0 及以后版本：如果驱动程序不支持传给 SetDIBitsToDevice 函数的 JPEG 文件图像，那么函数将失败，并返回 GDI_ERROR。

备注：当位图的位是对系统调色板进行索引时，可获得最佳的位图绘制速度。应用程序可以通过调用 GetSystemPaletteEntries 函数来检索系统调色板的颜色和索引值。在检索到颜色和索引值之后，应用程序可以创建 DIB。有关系统调色板方面更多的信息，请参考颜色方面的内容。

自底向上的 DIB 位图的起始点是在该位图的左下角，而自顶向下的 DIB 的起始点是在左上角。

为了减少对大型 DIB 位图的位进行设置所需的内存量,应用程序可以通过重复调用 SetDIBitsToBevice。每次将位图的不同部分放入到 lpvBits 数组来将输出捆绑在一起。参数 uStartScan 和 cScanLines 的值标明了 lpvBits 数组中包含的位图部分。在有一个全屏幕 MS DOS 会话在前台运行时，如果正在后台运行的一个进程调用了 SetDIBitsToDevice 函数，那么该函数会返回一个错误。

对于 Windows 98、Windows NT 5.0 及以后版本；如果 BITMAPINFOHEADER 中的成员 biCompression 为 BI_JPEG，那么 lpvBits 指向一个包含 JPEG 图像的缓冲区。BITMAPINFOHEADER 结构中的成员 biSizeimage 指定了该缓冲区的大小。参数 fuColorUse 必须设置为 DIB__RGB__COLORS。如果 BITMAPV4HEADER 中的成员 bV4SizeImage 指定了该缓冲区的大小。参数 fuColorUse 必须设为 DIB__RGB－COLORS。如果 BITMAPV5HEADER 结构中的成员 bV5Compression 为 BI__JPEG，那么参数 lpbBits 指向一个包含 JPEG 图像的缓冲区。DITMAPV5HEADER 结构中的成员 bV5SizeImage 指定了该缓冲区的大小，参数 fuColorUse 必须设为 DIB__RGB__COLORS。

ICM：进行颜色管理操作。如果指定的 BITMAPINFO 结构不是 BITMAPV4HEADER 或 BITMAPV5HEADER，那么当前设备环境的颜色配置(profile)就当作源颜色配置使用。如果 BITMAPINFO 结构不是 BITMAPV4HEADER 或 BITMAPV5HEADER,那么使用 RGB 颜色。如果指定的 BITMAPINFO 结构为 BITMAPV4HEADER 或 BITMAPV5HEADER，那么与该位图有关的颜色配置（profile）就用作源颜色。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.20 SetPixel

函数功能：该函数将指定坐标处的像素设为指定的颜色。

函数原型：COLORREF SetPixel(HDC hdc, int X, int Y, COLORREF crColor)；

参数：

hdc：设备环境句柄。

X：指定要设置的点的 X 轴坐标，按逻辑单位表示坐标。

Y：指定要设置的点的 Y 轴坐标，按逻辑单位表示坐标。

crColor：指定要用来绘制该点的颜色。

返回值：如果函数执行成功，那么返回值就是函数设置像素的 RGB 颜色值。这个值可能与 crColor 指定的颜我色有不同，之所以有时发生这种情况是因为没有找到对指定颜色进行真正匹配造成的；如果函数失败，那么返回值是 C1。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：如果像素点坐标位于当前剪辑区之外，那么该函数执行失败。

不是所有设备都支持 SetPixel 函数。有关详情，请参考 GetDeviceCaps。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文

件：wingdi.h；库文件：gdi32.lib。

3.1.21 SetPixeIV

函数功能： 该函数将指定坐标处的像素设置为与指定颜色最接近的颜色，该像素点必须在剪辑区和设备表面的可视部分内。

函数原型： BOOL SetPixeIV(HDC hdc, int X, int Y, COLORREF crColor)；

参数：

hdc：设备环境句柄。

X：指定要设置点的 X 轴坐标，按逻辑单位表示坐标。

Y：指定要设置点的 Y 轴坐标，按逻辑单位表示坐标。

crColor：指定要用来绘制像素点的颜色。

返回值： 如果函数成功，那么返回值非零；如果函数失败，那么返回值为零。

Windows NT：若想获得更多的错误信息，请调用 GetLastError。

备注： 不是所有设备都支持 SetPixeIV 函数 0。有关这方面的更多信息，请参考 GetDeviceCaps 函数中有关 RC_BITBLT 特性的描述。SetPixeIV 比 SetPixel 快，因为前者不需要返回实际绘制点的颜色值。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h：库文件：gdi32.lib。

3.1.22 SetSTretchBltMode

函数功能： 该函数可以设置指定设备环境中的位图拉伸模式。

函数原型： int SetSTretchBltMode(HDC hdc, int iStretchMode)；

参数：

hdc：设备环境句柄。

LStretchMode：指定拉伸模式。它可以取下列值，这些值的含义如下：

BLACKONWHITE：使用消除和现在的像素颜色值进行逻辑 AND（与）操作运算。如果该位图是单色位图，那么该模式以牺牲白色像素为代价，保留黑色像素点。

COLORONCOLOR：删除像素。该模式删除所有消除的像素行，不保留其信息。

HALFTONE：将源矩形区中的像素映射到目标矩形区的像素块中，覆盖目标像素块的一般颜色与源像素的颜色接近。在设置完 HALFTONE 拉伸模之后，应用程序必须调用 SetBrushOrgEx 函数来设置刷子的起始点。如果没有成功，那么会出现刷子没对准的情况。

STRETCH_ANDSCANS：与 BLACKONWHITE 一样。

STRETCH_DELETESCANS：与 COLORONCOLOR 一样。

STRECH_HALFTONE：与 HALFTONE 相同。

STRETCH_ORSCANS：与 WHITEONBLACK 相同。

WHITEONBLACK：使用颜色值进行逻辑 OR（或）操作，如果该位图为单色位图，那么该模式以牺牲黑色像素为代价，保留白色像素点。

返回值： 如果函数执行成功，那么返回值就是先前的拉伸模式，如果函数执行失败，那么返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 拉伸模式在应用程序调用 StretchBit 函数时定义系统如何将位图的行或列与显示设备上的现有像素点进行组合。

BLACKONWHITE(STRETCH_ANDSCANS)和 WHITEONBLACK(STRETCH_ORSCANS)模式典型地用来保留单色位图

中的前景像素。COLORONCOLOR(STRETCH_DELETESCANS)模式则典型地用于保留彩色位图中的颜色。

HALFTONE 模式比其他三种模式需要对源图像进行更多的处理，也比其他模式慢，但它能产生高质量图像，也应注意在设置 HALFTONE 模式之后，应调用 SetBrushOrgEx 函数以避免出现刷子没对准现象。

根据设备驱动程序的功能不同，其他一些拉伸模式也可能有效。

3.1.23 StretchBlt

函数功能： 函数从源矩形中复制一个位图到目标矩形，必要时按目前目标设备设置的模式进行图像的拉伸或压缩。

函数原型： BOOL StretchBlt(HDC hdcDest, int nXOriginDest, int nYOriginDest, int nWidthDest, int nHeighDest, HDC hdcSrc, int nXOriginSrc, int nYOriginSrc, int nWidthSrc, int nHeightSrc, DWORD dwRop)；

参数：

hdcDest：指向目标设备环境的句柄。

nXOriginDest：指定目标矩形左上角的 X 轴坐标，按逻辑单位表示坐标。

nYOriginDest：指定目标矩形左上角的 X 轴坐标，按逻辑单位表示坐标。

nWidthDest：指定目标矩形的宽度，按逻辑单位表示宽度。

nHeightDest：指定目标矩形的高度，按逻辑单位表示高茫。

hdcSrc：指向源设备环境的句柄。

nXOriginSrc：指向源矩形区域左上角的 X 轴坐标，按逻辑单位表示坐标。

nYOriginSrc：指向源矩形区域左上角的 Y 轴坐标，按逻辑单位表示坐标。

nWidthSrc：指定源矩形的宽度，按逻辑单位表示宽度。

nHeightSrc：指定源矩形的高度，按逻辑单位表示高度。

dwRop：指定要进行的光栅操作。光栅操作码定义了系统如何在输出操作中组合颜色，这些操作包括刷子、源位图和目标位图等对象。参考 BitBlt 可了解常用的光栅操作码列表。

返回值：如果函数执行成功，那么返回值为非零，如果函数执行失败，那么返回值为零。Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.1.24 StretchDIBits

函数功能： 该函数将 DIB 中矩形区域内像素使用的颜色数据拷贝到指定的目标矩形中。如果目标矩形比源矩形大小要大，那么函数对颜色数据的行和列进行拉伸，以与目标矩形匹配。如果目标矩形大小要比源矩形小，那么该函数通过使用指定的光栅操作对行列进行压缩。

函数原型： int StretchDIBits(HDC hdc, int XDest , int YDest , int nDestWidth, int nDestHeight, int XSrc, int Ysrc, int nSrcWidth, int nSrcHeight, CONST VOID *lpBits, CONST BITMAPINFO *lpBitsInfo, UINT iUsage, DWORD dwRop)；

参数：

hdc：指向目标设备环境的句柄。

XDest：指定目标矩形左上角位置的 X 轴坐标，按逻辑单位来表示坐标。

YDest：指定目标矩形左上角的 Y 轴坐标，按逻辑单位表示坐标。

nDestWidth：指定目标矩形的宽度。

nDestHeight：指定目标矩形的高度。

XSrc：指定 DIB 中源矩形（左上角）的 X 轴坐标，坐标以像素点表示。

YSrc：指定 DIB 中源矩形（左上角）的 Y 轴坐标，坐标以像素点表示。

nSrcWidth：按像素点指定 DIB 中源矩形的宽度。

nSrcHeight：按像素点指定 DIB 中源矩形的高度。

lpBits：指向 DIB 位的指针，这些位的值按字节类型数组存储，有关更多的信息，参考下面的备注一节。

lpBitsInfo：指向 BITMAPINFO 结构的指针，该结构包含有关 DIB 方面的信息。

iUsage：表示是否提供了 BITMAPINFO 结构中的成员 bmiColors，如果提供了，那么该 bmiColors 是否包含了明确的 RGB 值或索引。参数 iUsage 必须取下列值，这些值的含义如下：

DIB__PAL__COLOR：表示该数组包含对源设备环境的逻辑调色板进行索引的 16 位索引值。

DIB__RGB__COLORS：表示该颜色表包含原义的 KGB 值，若想了解更多的信息，请参考下面备注一节。

dwRop：指定源像素点、目标设备环境的当前刷子和目标像素点是如何组合形成新的图像。若想了解更多信息，请参考下面的备注一节。

返回值：如果函数执行成功，那么返回值是拷贝的扫描线数目，如果函数执行失败，那么返回值是 GDI__CRROR。

Windows NT：若想获取更多错误信息，请调用 GetLastError 函数。

Windows 98、Windows NT 5.0 及以后版本：如果设备驱动程序不支持传送给 StretchDIBits 的 JPEG 文件格式的图像，则该函数将失败，并返回 GEI_ERROR。

备注：自底向上的 DIB 的起始点为左下角，自顶向下 DIB 的起始点为左上角。

如果 nSrcWidth 和 nDestWidth 参数的符号不同。或是 nSrcHeight 和 nDesHeight 参数的符号不同。那么函数 StretchDIBits 将创建位图的镜像。如果 NsrcWidth 和 NdestWidth 符号不同，那么函数将沿着 X 轴创建位图镜像。如果 NsrcHeight 和 NdestHeight 符号不同，那么函数将沿着 Y 轴创建位图镜像。

对于 Windows 98、Windows NT 5.0 及以后版本：该函数允许将 JPEG 图像用作源图像，每个参数如何使用其实仍是一样的。

如果 BITMAPINFOHEADER 结构中的成员 biCompression 为 BI_JPEG，那么参数 lpBits 指向的是一个包含 JPEG 图像的缓冲区。BITMAPINFOHEADER 结构中的 biSizeImage 成员指定了该缓冲区的大小。参数 iUsage 必须设为 DIB_RGB_COLORS。dwRop 必须设为 SRCCOPY。

如果 BITMAPV4HEADER 结构中的成员 bV 4Compression 为 BI_JPEG，那么参数 lpBits 指向的是一个包含 JPEG 图像的缓冲区。BITMPV4HEADER 结构中的 bV4SizeImage 成员指定了该缓冲区的大小。参数 iUsage 必须设为 DIB_RGB_COLORS。参数 dwRop 必须设为 SRCCOPY。

如果 BITMAPV5HEADER 结构中的成员 bV 5Compression 为 BI_JPEG，那么参数 lpBits 指向的是一个包含 JPEG 图像的缓冲区。BITMPV5HEADER 结构中的 bV5SizeImage 成员指定了该缓冲区的大小。参数 iUsage 必须设为 DIB_RGB_COLORS。dwRop 必须设为 SRCCOPY。

为确保打印时可以有正确的图元文件假脱机操作。应用程序应在调用 StretchDIBits 函数之前调用 CHECKJPEGFORMAT 转义符，以确认打印机识别 JPEG 图像。

ICM：执行颜色管理。如果指定的 BITMAPINFO 结构中的 bmiHeader 不包含 BITMAPV4HEADER 或 BITMAPV5HEADER，那么当前设备环境的颜色配置（profile）被用作源颜色配置（profile）。如果没有颜色档案，那么就使用 RGB。如果指定的 BITMAPINFO 结构中的成员 bmiHeader 包含了 BITMAPV4HEADER 或 BITMAPV5HEADER，那么将把位图标头中指定的颜色配置（profile）用作源颜色配置。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.1.25 TransparentBltm

函数功能：该函数对指定的源设备环境中的矩形区域像素的颜色数据进行位块（bit_block）转换，并将结果置于目标设备环境。

函数原型：BOOL TransparentBltm(HDC hdcDest, int nXOriginDest, int nYOriginDest, int nWidthDest, int hHeightDest, HDC hdcSrc, int nXOriginSrc, int nYOriginSrc, int nWidthSrc, int nHeightSrc, UINT crTransparent)；

参数：

hdcDest：指向目标设备环境的句柄。

nXOriginDest：指定目标矩形左上角的 X 轴坐标，坐标以逻辑单位表示。

nYOriginDest：指定目标矩形左上角的 Y 轴坐标，坐标以逻辑单位表示。

nWidthDest：指定目标矩形的宽度。

nHeightDest：指定目标矩形高度的句柄。

hdcsrc：指向源设备环境的句柄。

nXOriginSrc：指定源矩形（左上角）的 X 轴坐标，坐标以逻辑单位表示。

nYOriginsrc：指定源矩形（左上角）的 Y 轴坐标，坐标以逻辑单位表示。

nWidthSrc：指定源矩形的宽度。

nHeightSrc：指定源矩形的高度。

crTransparent：源位图中的 RGB 值当作透明颜色。

返回值：如果函数执行成功，那么返回值为 TRUE；如果函数执行失败，那么返回值为 FALSE。

Windows NT：若想获取更多错误信息，请调用 GetLastError 函数。

备注：函数 TransparentBlt 支持 4 位 / 像素和 8 位 / 像素格式的源位图，使用 AlphaBlend 可以指定带有透明度的 32 位 / 像素格式的位图。如果源和目标矩形的大小不一致，那么将对源位图进行拉伸以与目标矩形匹配，当使用 SetStretchBltMode 函数时，BLACKONWHITE 和 WHITEONBLACK 两种 iStretchMode 模式将被转换成 TransparentBlt 函数的 COLORONCOLOR 模式。目标设备环境指定了用于目标坐标的变换类型，而源设备环境指定了源坐标使用的变换类型。如果源位图或目标位图的宽度或高度是负数，那么 TransparentBlt 函数也不对位图进行镜像。

速查：Windows NT：5.0 及以上版本；Windows：98 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：作为一个资源包含在 msimg32.dll 中。

3.1.26 CreateDiscardableBitmap

函数功能：该函数创建与指定设备兼容的位图，这种位图是已淘汰的，它具有与设备一样的位 / 像素格式和颜色调色板。应用程序可以选择这种位图作为与指定设备兼容的内存设备的当前位图。

函数原型：HBITMAP CreateDiscardableBitmap(HDC hdc, int nWidth, int nHeight)；

参数：

hdc：设备环境句柄。

nWidth：按位来指定位图的宽度。

nHeight：按位来指定位图的高度。

返回值：如果函数执行成功，返回值是指向位图的句柄；如果函数执行失败，那么返回值为 NULL。Windows NT：若想获取更多错误信息，请调用 GetLastError 函数。

3.1.27 FloodFill

函数功能： 该函数用当前刷子填充显示区域。该区域是由参数 crFill 指定的值包围起来的区域。

函数原型： BOOL FloodFill(HDC hdc, int nXStart, int nYStart, COLORREF crFill);

参数：

hdc：设备环境句柄。

nXStart：指定开始填充处的逻辑 X 坐标。

nYstart：指定开始填充位置的逻辑 Y 坐标。

crFill：指定要填充的区域边界的颜色。

返回值： 如果函数执行成功，则返回值为非零；如果函数执行失败，那么返回值为零。

Windows NT：若想获取更多的错误信息，请调用 GetLastError 函数。

备注： 下列原因可能导致该函数失败：填充无法完成。给定点的颜色与 crFill 参数指定的边界颜色相同。给定点在当前剪辑区之外——也就是在当前设备上不可见。

3.1.28 GetBitmapBits

函数功能： 该函数将指定位图的位拷贝到缓冲区里。

函数原型： LONG GetBitmapBits(HBITMAP hbmp, LONG cbBuffer, LPVOID lpvBits);

参数：

hbmp：指向感兴趣的位图的句柄。

cbBuffer：指定要从位图拷贝到缓冲区的字节数。

lpvBits：指向接收位图位数据的缓冲区指针。这些位是按字节类型存储在数组中的。

返回值： 如果该函数执行成功，那么返回值就是拷贝到缓冲区的字节数；如果该函数执行失败，那么返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

3.1.29 LoadBitmap

函数功能： 该函数从模块的可执行文件中加载指定的位图资源。该函数已经被函数 LoadImage 替代。

函数原型： HBITMAP LoadBitmap(HINSTANCE hInstance, LPCTSTR lpBitmapName);

参数：

hlnstance：指向模块实例的句柄。该模块的可执行文件包含了要加载的位图。

lpBitmapName：指向字符串（以 NULL 结束）批指针。该字符串包含了要加载的位图资源名称。另外一种方式就是该参数可以由低位字是资源标识符、高位字为 0 位形式组成。可以使用宏 MAKEINTRESOURCE 来创建这个参数值。

返回值：如果函数执行成功,则返回值是指向指定位图的句柄。如果函数执行失败,那么返回值是 NULL。

Windows NT：若想获取更多错误信息，请调用 GetLastError。

备注：如果由参数 lpBitmapName 指向的位图不存在,或者没有足够的内存来加载该位图,则函数失败。应用程序可以使用函数 LoadBitmap 来访问 Win32 API 使用的，预定义的位图。若要这么做，应用程序必须将 hlnstance 参数设为 NULL，并且 lpBitmapName 参数应取下列值：

OBM_BTNCORNERS OBM_OLD_RESTORE; OBM_BTSIZE OBM_OLD_RGARROW;

OBM_CHECK OBM_OLD_UPARROW; OBM_OLD_RESTORE; OBM_OLD_ZOOM;

OBM_CLOSE OBM_REDUCE; OBM_COMBO OBM_REDUCED; OBM_DNARROW

OBM_RESTORE; OBM_LFARROWD OBM_RGARROW1; OBM_LFARROW1

OBM_SIZE; OBM_MNARROW OBM_UPARROW; OBM_OLD_CLOSE OBM_UPARROWD;

OBM_OLD_DNARROW OBM_PARROW1; OBM_OLD_LFARROW OBM_ZOOM;

OBM_OLD_REDUCE OBM_ZOOMD

以 OBM_OLD 开头的位图名表示是比 3.0 更虫和 16 位版 Windows 系统使用的位图。

对于使用任何 OBM__常量的应用程序而言，在加入 WINDOWS.H 头文件之前必须定义常量 OEMRESOURCE。

应用程序必须调用 DeleteObject 函数来删除 LoadBitmap 函数返回的每一个位图句柄。

对于 Windows CE：当使用 LoadBitmap 函数时位图进行初始化时，该位图是只读的。当把位图选入到设备环境中时，无法更改设备环境（例如，加入文字），因为这样需要往位图写入的权利。

Windows CE 不支持参数 lpBitmapName 中的 OBM_*（以 OBM_开始的所有值）。

Windows CE 1.0 只支持 2 位灰阶的调色板，所以只可以使用 1 位来表示每个像素（单色。BMP）或 2 位来表示每个像素（2bp）的位图。

3.1.30 SetBitmapBits

函数功能： 该函数将位图的颜色数据位设置成指定值。

函数原型： LONG SetBitmapBits(HBITMAP hmbp, DWORD cBytes, CONST VOID (lpBits);

参数：

hbmp：指向要设置的位图的句柄。

cBytes：指定参数 lpBits 指向的数组的字节数。

lpBits：指向字节类型数组的指针。该数组中包含了指定位图的颜色数据。

返回值： 如果该函数执行成功，则返回值就是在设置位图位时使用的字节数；如果失败，则返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： lpBits 标识的数组必须是与 WORD 类型一致。

3.2 笔刷函数（Brush）

笔刷是一种绘图工具，Win32 应用程序可使用它绘制多边形、椭圆形和路径的内部。绘图应用程序使用笔刷绘制图形；字处理应用程序使用笔刷绘制水线；计算机辅助设计（CAD）应用程序使用笔刷绘制截面视图的内部；电子表格应用程序使用笔刷绘制饼图的扇形和直方图的方条。笔刷函数提供了一系列创建和使用笔刷的方法。

3.2.14.CreateBrushlndirect

函数功能： 该函数可以创建具有指定风格、颜色和模式的逻辑刷子。

函数原型： HBRUSH CreateBrushlndirect(CONST LOGBRUSH *lplb);

参数：

lplb：指向 LOGBRUSH 结构的指针，该结构包含与刷子有关的信息。

返回值： 如果函数执行成功，那么返回值标识一个逻辑刷子；如果函数执行失败，则返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：刷子就是系统用来对要填充图形的内部进行绘制的位图。

速查：Windows NT 3.1、Windows 95 以上。

3.2.24.CreateDIBPattemBrushPt

函数功能：该函数可以创建一个具有 DIB（与设备无在的位图）指定模式的逻辑刷子。

函数原型：HBRUSH CreateDIBPattemBrushPt(CONST VOID *lpPackedDIB, UINT iUsage)；

参数：

lpPackedDIB：指向包装的 DIB 的指针。这种 DIB 由 BITMAPINFO 结构后紧跟用于定义位图像素的字节型数组组成。对于 Windows 95 和 Windows 98，不支持从大于 8*8 像素的位图或 DIB 中创建刷子。如果指定了一个较大的位图，那么只使用该位图的一部分。

iUsage：指定 BITMAPINFO 结构中的成员 bmiColors 是否包含了一个有效的颜色表，如果是这样，那么该颜色表中的项是否包含明确的 RGB 或调色板索引，并且该颜色表由 16 位索引值数组组成，利用这些索引值可对刷子要选入的设备环境的逻辑调色板进行索引。

DIB_PAL_COLORS：表示提供颜色表，该颜色表包含原义的 RGB 值。

返回值：如果该函数成功,那么返回值标识了一个逻辑刷子;如果该函数执行失败,那么返回值为 NULL。

Windows NT：若想获得更多错误信息，可调用 GetLastError 函数。

备注：刷子是指系统用来绘制要填充图形的内部区域的位图。

在应用程序调用 CreateDIBPatternBrushPt 创建完刷子之后，可以通过调用 SelectObject 函数来将该刷子选入任何设备环境中，当不需要该刷子时，可调用 DeleteObject 函数删除它。

ICM：在创建刷子时没有颜色。然而，当把刷子选入到一个 ICM 允许的设备环境中时将完成颜色管理。

Windows CE：Windows CE NO 版不支持 iUsage 参数的 DIB_PAL_COLORS 标志。

在 Windows CE 2.0 版中，参数 iUsage 设为 DIB_RGB_COLORS。当使用的是 8bpp 位图时，可将 iUsage 设为 DIB_PAL_COLORS，然而，在这种情况下，Windows CE 将忽略 BITMAPINFO 结构中成员 bmiColors 数组中的值。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.2.34.CreateHatchBrush

函数功能：该函数可以创建一个具有指定阴影模式和颜色的逻辑刷子。

函数原型：HBRUSH CreateHatchBrush(int fnStyle, COLORREF clrref)；

参数：

fnStyle：指定刷子的阴影样式。该参数可以取下列值，这些值的含义为：

HS_BDIAGONAL：表示 45 度向下，从左至右的阴影；

HS_CROSS：水平和垂直交叉险影；

HS_DIAGCROSS：45 度交叉阴影；

HS_FDIAGONAL：45 度向上，自左至右阴影；

HS_HORIZONTAL：水平阴影；

HS_VERTICAL：垂直阴影。

cirref：指定用于阴影的刷子的前景色。

返回值：如果函数执行成功，那么返回值标识为逻辑刷子；如果函数执行失败，那么返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：位图是指系统用来绘制要填充图形的内部区域的位图。

在应用程序调用 CreateHatchBrush 函数创建刷子之后，它可以通过调用 SelectObject 函数把该刷子选入到任何设备环境中。

如果应用程序使用带阴影的刷子，用合适的颜色填充父窗口和子窗口的背景，那么在绘刷子窗口的背景之前，有可能需要设置刷子的起始点。要做到这一点，可以在应用程序中调用 SetBrushOrgEx 函数。

当不再需要该刷子时，可调用 DeleteObject 函数删除它。

ICM：在创建刷子时没有颜色，然而在把该刷子选入到 ICM 许可的设备环境中时，将会完成颜色管理。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.2.44.CreatePatternBrush

函数功能：该函数可以创建具有指定位图模式的逻辑刷子，该位图不能是 DIB 类型的位图，DIB 位图是由 CreateDIBSection 函数创建的。

函数原型：HBRUSH CreatePatternBrush(HBITMAP hbmp)；

参数：

hbmp：指向用于创建逻辑刷子的位图。

对于 Windows 95 和 Windows 98，不支持从大于 8＊8 像素规模的位图或 DIB 中创建刷子。如果指定的位图比较大，那么只使用该位图中的一部分。

返回值：如果该函数执行成功，那么返回值标识为一个逻辑刷子，如果该函数执行失败，那么返回值为 NULL。对 Windows NT，若想获得更多错误信息，可调用 GetLastError 函数。

备注：具有某种模式的刷子实际上就是指系统用来绘制要填充图形的内部区域的位图。

在应用程序调用 Create Patterm Brush 创建刷子之后，可以通过调用 SelectObject 函数把该刷子选入到任何设备环境中。也可以使用 DeleteObject 函数删除该刷子，这并不影响有关的位图。因此，可以使用该位图来创建任意数目的模式刷子。

使用单位图（每像素占 1 位）创建的刷子具有它绘制的设备环境中的文本和背景颜色。像素位为 0 表示使用当前文本颜色绘制像素，为 1 则表示使用当前背景颜色绘制像素点。

hbmp 参数标识的位图不能是 DIB 类型的位图。DIB 位图是由函数 CreateDIBSection 创建的位图。如果该位图是 DIB 类型，那么函数 CreatePattemBrush 将失败。

ICM：在创建刷子时没有颜色。然而，当该刷子被选入到一个 ICM 许可的设备环境中时，将进行颜色管理。

Windows CE：Windows CE 象 NT 一样，支持任意的刷子大小。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.2.54.CreateSolidBrush

函数功能：该函数创建一个具有指定颜色的逻辑刷子。

函数原理：HBRUSH CreateSolidBrush(COLORREF crColor)；

参数：

crColor：指定刷子的颜色。

返回值：如果该函数执行成功，那么返回值标识一个逻辑实心刷子；如果函数失败，那么返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：实心刷子实际上就是指系统用来绘制要填充图形的内部区域的位图。

在应用程序调用 CreateSolidBrush 创建刷子以后，可以通过调用 SelectObject 函数把该刷子选入设备环境。

ICM：在创建刷子时没有颜色操作。 然而，当把该刷子选入 ICM 许可的设备环境中时，将进行颜色管理。

Windows CE：Windows CE 不支持抖动刷子。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.2.64.GetBrushOrgEx

函数功能：该函数可检索用于指定设备环境的当前刷子的起始点，该函数替代了函数 GetBrush Org。

函数原型：BOOL GetBrushOrgEx(HDC hdc, LPPOINT lppt)；

参数：

hdc：设备环境句柄。

lppt：指向 POINT 结构的指针。该结构按设备坐标来接收刷子的起始点。

返回值：如果该函数执行成功，则返回值为非零，如果该函数执行失败，那么返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：刷子是指系统用来对要填充的图形的内部区域进行绘制的位图。

刷子起始点是一套坐标，其值在 0－7 之间。该起始点指定了位图中某个像素的位置。缺省的刷子起始点坐标是（0，0）。对于水平坐标而言，值 0 相当于最左边的像素列。7 则对应于最右面像素列。对于垂直坐标而言，0 对应于最上面的像素行，值 7 对应于最下面的行。当系统在任何绘制操作开始时对刷子定位，它将刷子的起始点映射到窗口客户区中刷子起始点指定的位置上。例如，如果起始点设为（2，3），那么系统将刷子的起始点（0，0）映射到窗口客户区的（2，3）位置处。

如果应用程序使用刷子和合适的颜色来填充父窗口和子窗口两个窗口的背景，那么在绘制完父窗口之后但尚未绘制子窗口之前，有可能需要设置该刷子的起始点。

对于 Windows NT：系统自动跟踪所有窗口管理的设备环境的起始点，并在需要时调整刷子，以保持在表面上有一致的模式。

对于 Windows 98 和 95：不支持对刷子起始点的自动跟踪，应用程序在使用它之前，必须使用 UnrealizeObject，SetBrushOrgEx 和 SelectObject 函数来调整刷子。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.2.74.GetSysColorBrush

函数功能：该函数可以检索标识逻辑刷子的句柄，该刷子对应指定的颜色索引值。

函数原型：HBRUSH GetSysColorBrush(int nIndex)；

参数：

nIndex：指定颜色索引值，该值与用来绘制 21 个窗口元素之一的颜色对应。

返回值：返回值标识了逻辑刷子。

备注：刷子是指系统对要填充图形的内部区域进行绘制而使用的位图。应用程序可以通过调用 GetSysColor 函数来检索当前系统颜色。应用程序可以调用函数 SetSysColors 来设置当前系统颜色。

应用程序必须为使用系统刷子的窗口注册一个窗口类。

对于 Windows CE：在 Windows CE 中，赋给 COLOR_X 标志的值不同于 Windows 桌面平台上的赋值。因此，应该为参数 nIndex 指定 COLOR_X 标志的值，而不是相应的整数值。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

3.2.84.PatBlt

函数功能： 该函数使用当前选入指定设备环境中的刷子绘制给定的矩形区域。通过使用给出的光栅操作来对该刷子的颜色和表面颜色进行组合。

函数原型： BOOL PatBlt(HDC hdc, int nXLeft, int nYLeft, int nWidth, int nHeight, DWORD dwRop)；

参数：

hdc：设备环境句柄。

nXLeft：指定要填充的矩形左上角的 X 轴坐标，坐标按逻辑单位表示。

nYLeft：指定要填充的矩形左上角的 Y 轴坐标，坐标按逻辑单位表示。

nWidth：指定矩形的宽度，按逻辑单位表示宽度。

nHeight：指定矩形的高度，按逻辑单位表示高度。

dwRop：指定光栅操作码。该操作码可以取下列值，这些值的含义如下：

PATCOPY：将指定的模式拷贝到目标位图中。

PATINVERT：使用布尔 OR（或）操作符将指定模式的颜色与目标矩形的颜色进行组合。

DSTINVERT：将目标矩形反向。

BLACKNESS：使用物理调色板中与索引 0 相关的颜色填充目标矩形。（对于缺省的物理调色板而言，该颜色为黑色）。

WHITENESS：使用物理调色板中与索引 1 有关的颜色来填充目标矩形。（对于缺省的物理调色板而言，该颜色为白色）。

返回值： 如果函数执行成功，则返回值为非零；如果函数执行失败，则返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 该函数的参数 dwRop 取值限定为全部 256 个三元光栅操作有限子集。特别地，涉及源矩形的操作码不能使用。

并非所有设备都支持 PalBlt 函数。有关更多的信息，请参考函数 GetDeviceCaps 中有关 RC_BITBLT 特性的描述。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.2.94.SetBrushOrgEx

函数功能： 该函数设置刷子起始点，GDI 将该起始点赋给应用程序选入指定设备环境的下一个刷子。

函数原型： BOOL SetBrushOrgEx(HDC hdc, int nXOrg, int nYOrg, LPPOINT lppt)；

参数：

hdc：设备环境句柄。

nXOrg：按设备单位来指定新刷子起始点的 X 轴坐标，如果该值比刷子宽度还要大，那使将使用取模操作符来减少该值。（nXorg 对刷子宽度取模）。

nYOrg：按设备单位指定新刷子起始点的 Y 轴坐标，如果该值比刷子高度还要大，那么将使用取模运算符来减小该值。（nYorg 对刷子高度取模）。

lppt：指向 POINT 结构的指针。该结构接收前一个刷子的起始点信息。

如果不需要前一个刷子的起始点信息，那么该参数可以为 NULL。

返回值：如果函数执行成功，则返回值为非零，如果该函数失败，那么返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.2.10 CreateDIBPattermBrush

函数功能：该函数创建一个逻辑刷子，该刷子的模式由指定的 DIB（与设备无关位图）位图来指定。该刷子可以连续地选入到任何与支持光栅操作的设备相关的设备环境中。

函数原型：HBRUSH CreateDIBPattermBrush(HGLOBALhglbDIBPacked, UINTfuColorSpec)；

参数：

hglbDIBPacked:指向一个全局内存对象的句柄。该对象包含了一个包装的 DIB。这种 DIB 由 BITMAPINFO 结构后紧跟用于定义位图像素的字节型数组组成。

对于 Windows95 和 Windows98，不支持从大于 8*8 像素的位图或 DIB 中创建刷子。如果指定的位图较大，那么只使用该位图的一部分。

fuColorSpec：指出 BITMAPINFO 结构中的成员 bmiColors 是否已被初始化，并且如果是初始化了，那么该成员是否包含了明确的 RGB 值或逻辑调色板索引。参数 fuColorSpec 必须取下列值之一，这些值的含义如下：

DIB_PAL_COLORS：表示提供了颜色表，该颜色表由 16 位索引值数组组成，利用这些索引值可以对刷子要选入的设备环境的逻辑调色板进行索引。

DIB_RGB_COLORS：表示提供了颜色表，并且表中包含了原文的 RGB 值。

返回值：如果函数执行成功，那么返回值标识为逻辑刷子；如果函数失败，则返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：当应用程序把一个颜色的 DIB 模式刷子选入到单色设备环境中时，系统并不知道 DIB 中规定的颜色，相反地，它使用设备环境的当前背景色和前景色来显示模式刷子。映射成 DIB 中第 1 个颜色（在 DIB 颜色表中偏移量为 0）的像素是使用前景色显示的，映射成第 2 个颜色（在 DIB 颜色表中偏移量为 1）和像素则使用背景色来显示。当不再需要该刷子时，可调用函数 DeleteObject 删除它。

ICM：在创建刷子时无颜色操作，然而当把该刷子选入到一个 ICM 允许的设备环境中时，会进行颜色管理。

3.2.11 FixBrushOrgEx

函数功能：在 Win32API 中没有实现 FixBrushOrgEx 函数,在这里提供出来是考虑与 Win32 的兼容性，如果调用它，那么该函数什么也不做，并且返回 FALSE。

3.3 剪裁函数（Clipping）

剪裁是一种处理过程，它将输出到某个区域或路径中的内容限制在应用程序窗口的显示区内。剪裁函数提供了一系列处理剪裁区域的方法。

3.3.1ExcludeClipRect

函数功能：该函数的功能是创建一个新的剪切区域，该区域由一个现存的剪切区域减去一个特定的矩形区域而构成。

函数原型：int ExcludeClipRect(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect);

参数：

：设备环境句柄。

：定义了矩形区域左上角的逻辑 X 坐标。

：定义了矩形区域左上角的逻辑 Y 坐标。

：定义了矩形区域右上角的逻辑 X 坐标。

：定义了矩形区域右上角的逻辑 Y 坐标。

返回值：该返回值表明了新的剪切区域的复杂度，它可是如下几种形式：

NULLREGION：剪切区域为；

SIMPLEREGION：剪切区域是单个矩形；

COMPIEXREGION：剪切区域有多个矩形；

ERROR：剪切区域创建失败

注释：矩形的底边和右边并不排除在剪切区域之外。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.3.2ExtSelectClipRgn

函数功能：访函数通过特定的方式把一个特定的区域与当前的剪切区域合并在一起。

函数原型：int ExtSelectClipRgn (HDc hdc, HRGN hrgn, int fnMode);

参数：

hdc：设备环境句柄。

hrgn：标识被选择的区域。当指定 RGN_COPY 模式时，该句柄只能为 NULL。

fnMode：定义执行的运算模式。它必须是下列值之一：

RGN_AND：新的剪切区域与当前剪切区域的重迭部分合并，该区域由 hrgn 标识。

RGN_COPY：新的剪切区域是由 hrgn 标识区域的一个拷贝，这和 select clipRgn 是统一的。如果由 hrgn 所定义的区域为空。那么新的剪切区域就是缺省的剪切区域（缺省的剪切区域是个空区域）。

RGN_DIFF：新的剪切区域与由 hrgn 定义的区域之外的区域合并。

RGN_OR：新的剪切区域与当前的剪切区域合并，并且该区域由 hrgn 标识。

RGN_XOR：新的剪切区域与当前的剪切区域合并，并且该区域由 hrgn 指定，但是不包括任何重迭区域。

返回值：返回值表明了新的剪切区域的复杂度，它的值是如下几种：

NULLREGION：区域为空；

SIMPLEREGION：区域为单个矩形；

COMPLEXREGION：区域为多个矩形；

ERROR：发生了错误。

注释：当该函数调用发生错误时，由设备环境定义的以前的前切区域不受影响。ExtselectclipRgn 函数假设在设备单元中定义了该特定区域的坐标。使用的仅仅是由 hrgn 参数定义区域的一个拷贝，而该剪切区域自身还可被再使用，并且它还可被删除。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.3.3GetClipBox

函数功能：该函数得到一个能够完包含当前可见区域的最小矩形的大小。该可见区域由当前的剪切区域定义或由剪切路径所定义或者由任何重迭的窗口所定义。

函数原型：int GetClipBox(HDC hdc, LPRECT lprc)；

参数：

hdc：设备环境句柄。

lprc：RECT 结构的一个指针，用来接收矩形的大小。

返回值：如果该函数执行成功，那么它的返回值定义了剪切区域的复杂，返回值的意义为：

NULLREGIOW：区域为空；

SimpIEREGZO：区域为一单个矩形；

CompLEXREGIOW：区域为多个矩形；

ERROR：发生错误；

GetclipBox：返回基于给定设备一下文环境的逻辑坐标。

Windows 若想获得更多的错误信息，调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.3.4GetClipRgn

函数功能：该函数得到一个句柄，该句柄标识了由当前应用定义的剪切区域。

函数原型：int GetClipRgn(HDC hdc, HRGN hrgn)；

参数：

hdc：设备环境句柄。

hrgn：在该函数调用前的一个现存区域的句柄，当该函数调用完毕后。Hrgn 变为当前区域的拷贝的句柄。

返回值：如果该函数执行成功，并且给定设备环境没有剪切区域，那么返回值是零；如果该函数执行成功，并且给定设备环境有一个剪切区域，那么返回值是 1；如果发生了错误，那么返回值是 C1。

Windows NT：若想获得更多的错误信息，调用 GetLastError 函数。

注释：由应用定义的剪切区域是由 selectclipRgn 函数定义的区域，并不是当应用调用 BeginPaint 函数时创建的剪切区域。

如果该函数执行成功，hrgn 参数就是当前剪切区域的拷贝的句柄，该拷贝以后的改变并不影响当前的剪切区域。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.3.5GetMetaRgn

函数功能：该函数得到特定设备环境的当前元区域。

函数原型： int GetMetaRgn(HDC hdc, HRGN hrgn)；

参数：

hdc：设备环境句柄。

hrgn：在该函数调用之前的一个现存区域的句柄，当该函数调用完成之后，hrgn 变为当前区域的拷贝的句柄。

返回值： 如果成功，返回一个非零值；如果失败，返回零。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

注释：如果该函数调用成功，hrgn 是当前元区域的拷贝的句柄，该拷贝的变化并不影响当前的元区域。一个设备上下文环境的当前剪切区域是它的剪切区域和元区域的一个交集。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.3.6IntersectClipRect

函数功能： 该函数创建了一个新的剪切区域，该区域是当前剪切区域和一个特定矩形的交集。

函数原型： int IntersectClipRect(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect)；

参数：

hdc：设备环境句柄。

nLeftRect：定义矩形左上角的逻辑 X 坐标。

nTopRect：定义矩形左上角的逻辑 Y 坐档。

nRightRect：定义矩形右下角的 X 坐标。

nBottomRect：定义矩形右下角的 Y 坐档。

返回值： 返回值表明了新剪切区域的类型。

NULL REGION：区域为空；

SIMPLEREGION：区域为单个矩形。

CoupIExREGION：区域为多个矩形；

ERROR：发生错误（当前的剪切区域不变影响）。

注释：给定矩形的最下边和最右边不包括在剪切区域。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.3.7OffsetClipRgn

函数功能： 该函数按指定的位移，移动设备上下文环境的剪切区域。

函数原型： int OffsetClipRgn(HDC hdc, int nXOffset, int nYOffset)；

参数：

hdc：设备环境句柄。

nXOffset：定义左移或右移的逻辑单位数。

nYOffseet：定义上移或下移的逻辑单位数。

返回值： 返回值表明了该新区域的复杂度。

NULLREGION：区域为空；

SIMPIEREGION：区域是单个矩形。

COMPLEXREGION：区域为多个矩形；

ERROR：发生错误（当前的剪切区域不受影响）。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.3.8PtVisible

函数功能：该函数确定指定的点是否在设备环境的剪切区域之内。

函数原型：BOOL PtVisible(HDC hdc, int X, int Y)；

参数：

hdc：设备环境句柄。

X：定义点的逻辑 X 坐标。

Y：定义点的逻辑 Y 坐档。

返回值：如果该点在设备环境的剪切区域之内，则返回值非零；如果该点不在设备上描述表的剪切区域之内，则返回值为零。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.3.9RectVisible

函数功能：该函数确定指定矩形的任何部分是否在设备上下文环境的剪切区域之内。

函数原型：BOOL RectVisible(HDC hdc, CONSTRECT *lprc)；

参数：

hdc：设备环境句柄。

lprc：RECT 结构的指针，包括特定矩形的逻辑坐标。

返回值：如果特定矩形的一部分在剪切区域之内，则返回非零值；如果特定矩形的任何部分都不在剪切区域之内，则返回值为零。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.3.10 SelectClipRgn

函数功能：该函数选择一个区域作为指定设备环境的当前剪切区域。

函数原型：int SelectClipRgn(HDc hdc, HRGN hrgn)；

参数：

hdc：设备环境句柄。

hrgn：标识被选择的区域。

返回值：返回值表明了区域的复杂度，可以是下列值之一。

NULLREGION：区域为空；

SIMPLEREGION：区域为单个矩形；

COMPLEXREGION：区域为多个矩形；

ERROR：发生错误（以前的剪切区域不受影响）。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：只有选择区域的一个拷贝被使用，区域本身可被任何其他的设备环境所选择，并且它还被删除。

SelectclipRgn 函数假设指定区域的坐标在设备联合中已经定义。

要想删除一个设备环境的剪切区域，指定一个空（NULL）区域句柄。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.3.11 SetMetaRgn

函数功能： 该函数使指定设备环境的当前剪切区域与当前的元区域产生一个交集，并把该交集区域保存为指定设备环境的新的元区域。剪切区域被重置为空区域。

函数原型：

参数：

hdc：设备环境句柄。

返回值： 返回值表明了新剪切区域的复杂度，取如下值之一。

NULLREGION：空区域；

SIMPLEREGION：区域为单个矩形；

COMPLEXREGION：区域为多个矩形；

ERROR：发生错误（以前的剪切区域不受影响）。

注释：设备环境的当前剪切区域由它的剪切区域与元区域的交集来确定。只有在当通过调用 SaveDC 函数把一个应用的初始设备环境保存了之后 SetMetaRgn 函数才可被调用。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.3.12 SelectClipPath

函数功能： 该函数选择当前的路径作为设备环境的一个剪切区域。通过使用特定的模式把新的区域与任何存在的剪切区域合并。

函数原型： BOOL SelectClipPath(HDC hdc, int iMode)；

参数：

hdc：设备环境句柄。

iMode：定义使用路径的方法，具取值如下：

RGN_AND：新的剪切区包括当前剪切区域与当前路径的一个交集（重迭区域）。

RGN_COPY：新的剪切区域就是当前的路径。

RGN_DIFF：新的剪切区域包含除了当前路径外的当前剪切区域。

RGN_OR：新的剪切区域包含当前剪切区域与当前路径的并集。

RGN_XOR：新的剪切区域包含当前剪切区域与当前路径的并集但不包含重迭的区域。

返回值： 如果函数执行成功，返回非零值；如果函数执行失败，返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。GetLastError 函数可能会返回如下的代码：

ERROR_CAN_NOT_COMPLETE;ERROR_INYALID_PARAMETER;ERROR_NOT_ENOUGH_MEMORY。

注释：由 hdc 参数标识的设备环境必须包含一个闭合路径。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；

库文件：gdi32.lib。

3.4 颜色函数（Color）

颜色是组成 Win32 应用程序所生成的图片和图像的一个重要元素。 Win32 API 提供了一系列管理和使用画笔、笔刷、文本和位图的颜色的函数。

3.4.1AnimatePalette

函数功能： 该函数替换指定逻辑调色板上的入口点。

函数原型： BOOL AnimatePalette(HPALETTE hpal, UINT iStartindex, UINT cEntries, CONST PALETTEENTRY *ppe)；

参数：

hpal：逻辑调色板的句柄。

iStartIndex：指定要被替换的第一个逻辑调色板入口点。

cEntries：指定要被替换的入口点数目。

ppe：指向 PALETTEENTRY 数组结构第一个元素的指针，用来替换当前的入口点。

返回值： 如果执行成功，返回非零值；如果失败，返回值是零，若想获得更多错误信息，请调用 GetLastError 函数。

注释：通过调用 GetDeviceCaps 函数和定义 RASTERCAPS 常量，一个应用可以确定一个设备是否支持调色板操作。仅当 LOGPALETTE 结构的 palPalEntry 成员置有 PC_RESERVED 标志时，AnimatePalette 函数才能改变入口点。如果给定的调色板与一个活动窗口相联系，那么调色板中的色彩立即被替换。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.2CreateHalftonePalette

函数功能： 该函数创建一个指定设备环境的半色调调色板。

函数原型： HPALETTE CreateHalftonePalette(HDC hdc)；

参数：

hdc：设备环境句柄。

返回值：如果函数执行成功，返回值定义了一个逻辑半色调调色板；如果函数执行失败，返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：当一个设备环境的展开模式设为 HALFTONE 时应用就会创建一个半色调调色板，在 stretchBit 或 strechDIBits 函数调用之前由 createHalftonePalette 函数返回的逻辑半色调调色板会映射在设备环境中。当不再需要该调色板时，可以调用 DeleteObject 函数删除它。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.3CreatePalette

函数功能： 该函数创建一个逻辑彩色调色板。

函数原型： HPALETTE CreatePalette(CONST LOGPALETTE *lplgpl)；

参数：

lplgpl：指向 LOGPALETTE 结构的指针，该结构包含了逻辑调色板中的色彩信息。

返回值： 如果函数成功，则返回值是一个逻辑调色板的句柄；如果失败，返回值为 NULL（空）。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：通过调用 GetDevice Caps 函数和定义 RASTERCAPS 常量一个应用可以确定一个设备是否支持调色板操作。一旦应用创建了一个逻辑调色板，它可以通过调用 Select Palette 函数把该调色板选入设备环境中。通过调用 RealizePalette 函数选入设备环境中的调色板可被映射。

当不再需要该调色板时，可调用 DeleteObject 函数删除它，由于 Windows CE 并不仲裁前景和背景应用的调色板，调色板并不随着 Windows 系统颜色而自动退色，因此由该函数创建的调色板的颜色入口点数目和 LOGPALETTE 结构中的 PalNumEntries 相同。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.4.4GetColorAdjustment

函数功能： 该函数检取指定设备环境的颜色调整值。

函数原型： BOOL GetColorAdjustment(HDC hdc, LPCOLORADJUSTMENT lpca)；

参数：

hdc：设备环境句柄。

ipca：指向 ColorADJustMENT 结构的指针，它接受颜色调整值。

返回值： 函数成功，返回非零值，失败返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查： Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.5GetNearColor

函数功能： 该函数返回一个颜色值，该值和系统调色板的一个颜色相一致，当使用一个特定的颜色值时，该颜色被显示。

函数原型： COLORREF GetNearColor (HDC hdc, COLORREF crColor)；

参数：

hdc：设备环境句柄。

crColor：定义一个与请求颜色相一致的颜色值。

返回值： 如果函数执行成功，返回值与系统调色板中的一种颜色相一致；如果失败，返回值是 CLR_INVALID。

Windows NT：若想获得更多错误信息，请调用 GelLastError 函数。

Windows CE：GetNearestcolor 函数从选定的设备环境中返回的一种颜色和一个指定的颜色非常匹配。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文

件：wingdi.h；库文件：gdi32.lib。

3.4.6GetNearestPaletteIndex

函数功能：该函数检取指定逻辑调色板入口的索引，以与一个特定的颜色值相匹配。

函数原型：UINT GetNearestPaletteIndex(HPALETTE hpal, COLORREF crColor)；

参数：

hpal：标识一个逻辑颜色调色板。

crColor：指定一个匹配的颜色。

返回值：如果函数执行成功，返回值是一个逻辑调色板入口的索引；如果函数执行失败，返回值是CLR_INVALID。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：通过调用 GetDevicecaps 函数和指定 PASTERCAPS 常量，一个应用可以确定一个设备是否支持调色板操作。如果给定的逻辑调色板包含设置 PC_EXPLICT 标志的入口点，那么返回值是不确定的。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.4.7GetPaletteEntries

函数功能：该函数从给定的逻辑调色板中提出指定范围的调色板项目。

函 数 原 型 ： UINT GetPaletteEntries(HPALETTE hpal, UINT iStartIndex, UINT nEntries, LPPALETTEENTRY lppe)；

参数：

hpal：逻辑调色板句柄。

iStartIndex：指定要提取的逻辑调色板中的第一项。

nEntries：指定要提取的逻辑调色板中的项数。

lppe：指向接受调色项目的 PALETTEENTRY 结构数组的指针，该数组所含结构的数目至少为 nEntries 参数指定的数目。

返回值：如果函数成功，且逻辑调色板的句柄是一个可靠的指针（非零），返回值是从逻辑调色板提取的项目数。如函数成功，且逻辑调色板的指针为零，返回值是给定调色板的项数。如函数失败，返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.4.8GetSystemPaletteEntries

函数功能：该函数从与指定设备环境有关系的系统调色板中检取调色板入口点的范围。

函 数 原 型 ： UINT GetSystemPaletteEntries(HDC hdc, UINT iStartIndex, UINT nEntries, LPPALETTEENTRY lppe)；

参数：

hdc：设备环境句柄。

iStartIndex：指定从系统调色板中检取的第一个入口点。

nEntries：指定从系统调色板中检取入口点的数目。

lppe：指向 PALETTEENTRY 结构数组的指针，它检取调色板入口点，该数组至少要包含由 nEntries 参数所指定的结构数，如果该参数为 NULL（空），则该函数返回调色板中入口点的总数。

返回值：如果函数执行成功，返回值是从调色板中检取的入口点数；如果失败，返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：通过调用 GetDeviceCaps 函数和定义 RASTERCAPS 常用量，一个应用可以确定一个设备是否支持调色板操作。

Windows CE：如果和 hdc 相关的设备没有一个可设置的调色板，那么 GetSystemPaletteEntries 函数就会失败，在使用 GetsystempaletteEntries 之前，可使用 GetDeviceCaps 函数来确定考查该设备是否有一个可设置的调色板。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.4.9GetSystemPaletteUse

函数功能：该函数检取特定设备环境的系统调色板的当前状态。

函数原型：UINT GetSystemPaletteUse（HDc hdc）；

参数：

hdc：设备环境句柄。

返回值：如果成功，返回值是系统调色板的当前状态，它可是以下几种：

SYSPAL_NOSTATIC：系统调色板除了黑色和白色之外不包含其他静态颜色。

SYSPAL_STATIC：当一个应用映射它的逻辑调色板时，系统调色板包含的静态颜色不会变化。

SYSPAL-ERROR：给定的设备环境无效或者不支持彩色调色板。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：在缺省模式下当一个应用映射它的逻辑调色板时，系统调色板有 20 个静态色彩，不发生变化。通过调用 SetSystemPalette 函数一个应用可以访问这些色彩中的大部分。由 hdc 设备环境句柄必须支持彩色调色板。通过调用 GetDeviceCaps 函数和定义 RASTERCAPS 常量，一个应用就能确定一个设备是否支持彩色调色板。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.10 HTUI_ColorAdjustment

函数功能：该函数显示中间色调色彩调整的缺省用户接口。

函数原型：LOGN APIENTRY HTUI_ColorAdjustment(LPWSTR pCallertitle, HANDLE hDefDIB, LPWSTR pDefDIBTitle, PCOLORADJUSTMENT pColorAdjustment, BOOL ShowMonochromeOnly, BOOL UpdatePermission)；

参数：

pCallerTitle：指向调用的应用或设备标识的指针。PcallerTitle 的值将在对话框的调整中显示。如果该参数的值为空（NULL），则没有标认显示。

hDefDIB：设备独立位图的句柄，如果 hDefDIB 为空（NULL），则该函数使用 DIB 作为它的缺省图片进行色彩调整测试。如果 hDefDIB 为空，则在用户调整性能时，会显示三个标准图片之一：

RGB 色图表：基准色图表；NTSC 色图表。

PDefDIBTitle：一个字符串指针，它指向 DIB 图片的名字或已传递的 hDefDIB 的描述。

pColorAdjustment：指向 ColoRADJUDSTMENT 数据结构的指针。

ShowMonochromeOnly：限制位图的显示为单色，当输出设备是单色时可设置该参数。

UpdatePemission：允许 ColoRADJUSTMENT 结构更新，其值如下：

TRUE：颜色调整不仅限于当前用户界面，当退出半色调颜色调整用户界面时，ColoRADJustMENT 结构的改变将被保存。

FALSE：颜色调仅限于用户当前的会话。CoLORADJusTMENT 结构不变。

返回值：HTUL_ColorAdjustment 函数返回如下类型为 LONG 型的数值。

>0 用户选择更新 ColoRADJUSTMENT 结构；=0 用户选择取消更新 ColoRADJUSTMENT 结构；<0 发生了错误。

注释：一个应用可以通过关联 htui.dll 或使用 loadlibrary 和 GetProcAddres 函数来得到 htui.dll 的位置。

速查：Windows NT：5.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h。

3.4.11 HTUI_DeviceColorAdjustment

函数功能：该函数显示一个指定设备的调整颜色的用户界面，例如打印机。

函 数 原 型 ： LONG APIENTRY HTUI_DeviceColorAdjustment(LPSTR pDeviceName, PDEVHTADJDATA pDevHTAdjData);

参数：

pDeviceName：指向设备名的指针，在用户颜色调整界面的修改框中显示。

pDevHTAdjDate：指向一 DEVHTADJDATA 结构的指针。

返回值：HTUI_DeviceColorAdjustment 函数返回一个 LONG 型值。

>0 用户更新 DEVHTADJDATA 结构；=0 用户取消颜色调整；<0 发生错误。

速查：Windows NT：5.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h。

3.4.12 RealizePalette

函数功能：该函数从当前逻辑调色板中映射调色板入口点到系统调色板中。

函数原型：UINT RealizePalette(HDC hdc);

参数：

hdc：设备环境句柄。一个逻辑调色板被选择在该设备环境中。

返回值：如果函数成功，函数返回值定义口点的数目，这些逻辑调色板中的入口点映射到系统调色板中；如果失败，返回值是 GDI__ERROR。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：通过调用 GetDeviceCaps 函数和定义 RASTERCAPS 常量。一个应用可以确定一个设备是否支持调色板操作。

RealizePalette 函数修改与指定设备环境有关的设备的调色板，如果该设备环境是一个内存设备环境，那么选进该设备环境的位图颜色表将被修改，如果该设备环境是珍上显示设备环境，那么该设备的物理调色板将被修改。

逻辑颜色调色板是颜色密集型应用和系统之间的一个缓存，允许这些应用在不干扰其他窗口颜色的情况下使用任意多的颜色。

当焦点在一个应用窗口，并且它调用 RealizePalette 函数时，则系统试图映射尽可能多的颜色，这对于活动窗口的应用也同样正确。

Windows CE：Windows CE 在前景和背景调色板应用之间并不做出仲裁，前景应用完全控制了系统调色板，因此，对于前景应用 Windows CE 并不执行任何颜色匹配操作。它仅仅用 hdc 参数的调色板入口点来覆盖系统调色板的入口点。

Windows CE 背景应用中不支持 RealizePalette 函数。

如果与 hdc 相关的设备设置一个可设置的调色板，那么 RealizePalette 函数将执行失败，在使用 RealizePaletle 函数之前，先调用 GetDevicecaps 函数来断定一个设备是否有一个可设置的调色板。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.4.13 ResizePalette

函数功能：该函数增大或减小基于指定值的逻辑调色板的大小。

函数原型：BOOL ResizePalette(HPALETTE hpal, UINT nEntries)；

参数：

hpal：标识一个将发生变化的调色板。

nEntries：指定一个调整过大小的调色板的入口点数。

返回值：如果成功返回非零值；如果失败返回值是零。

Windows NT：若想获得更多错误信息，请调用函数 GetLastError。

注释：通过调用 GetDeviceCaps 和指定 RASTERCAPS 常量，一个应用可以确定一个设备是否指调色板操作。如果一个应用调用 ReSizeCaps 来减小一个调色板的大小。那么该调色板中剩余的入口点不发生变化，如果一个应用调用 ReSizePalette 函数来增大一个调色板的大小，那么增加的调色板入口点置为黑色（红色、绿色和蓝色值为 0）并且它们的标志置为 0。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.14 SelectPalette

函数功能：该函数选择指定的逻辑调色板到一个设备环境中。

函数原型：HPALETTE SelectPalette(HDC hdc, HPALETTE hpal, BOOL bForceBackground)；

参数：

hdc：设备环境句柄。

hpal：标识被选择的逻辑调色板。

bForceBackground：确定逻辑调色板是否被强行作为一个背景调色板，如果该值为 TRUE RealizePalette 函数就使逻辑调色板以最好的可能方式映射成物理调色板中已有的颜色，这种情况常发生；如果该值为 FALSE，RealizePalette 使逻辑调色板拷贝到设备调色板中，这时该应用在前景（如果 hdc 参数是一个内存设备环境，该参数被忽略）。

返回值：如果成功，返回值和设备环境以前的逻辑调色板相一致；如果失败返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：通过调用 GetDevicecaps 函数和定义 RASTERCaps 常数，一个应用可以确定一个设备是否支持调色板操作。

一个应用可以把一个逻辑调色板选入多个设备环境中，逻辑调色板的变化会影响所有的设备环境。

如果一个顶层窗口的每一个子窗口都映射自己的调色板，那么一个应用就可以在 bForceBackground 参数设为 TRUE 的情况下调用 SelectPalette。但是只有需要映射它的调色板的子窗口必须把 bForceBackground 置为 TRUE，其他的子窗口必须把该值设为 FALSE。

Windows CE：Windows CE 在前景应用和背景应用的调色板之间不做仲裁，因此 WindowsCE 忽略 bForceBackground 参数，把它当作始终是 FALSE。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.4.15 SetColorAdjustment

函数功能：该函数为一个使用指定值的设备环境设置颜色调整值。

函数原型：BOOL SetColorAdjustment(HDC hdc, CONST COLORADJUSTMENT *lpca)；

参数：

hdc：设备环境句柄。

lpca：指向 COLORADJUSTMENT 结构的指针，该结构包含颜色调整值。

返回值：如果成功，返回值非零；如果失败，返回值是零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：当调用 StretchBit 和函数 StretchDIBits，设为 HALFTONE 模式时颜色调整值用来调整源位图的输入颜色。

速查：Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.16 SetPaletteEntries

函数功能：该函数为一个逻辑调色板的入口点数组设置 RGB 颜色值和标志。

函数原型：UINT SetPaletteEntries(HPALETTE hpal, UINT iStart, UINT cEntries, CONST PALETTEENTRY *lppe)；

参数：

hpal：标识逻辑调色板。

iStart：指定第一个被设置的逻辑调色板入口点。

cEntries：指定被设置的逻辑调色板的入口点数目。

lppe：指向包含 RGB 值和标志的 PALETTEENTRY 结构数组的第一个元素。

返回值：如果成功，返回值是逻辑调色板中设置的入口点数目；如果失败，返回值为零。

Windows NT：若想获得更多错误的信息，请调用 GetLastError 函数。

注释：通过调用 GetDeviceCaps 函数和定义 RASTERCAPS 常数，一个应用可以确定一个设备是否支持调色板操作。

如果一个逻辑调色板已被选择并且被映射，那么该调色板的变化并不影响外部的物理调色板，RealizePalette 必须重新被调用以设置一个新的逻辑调色板到外部。

Windows CE：如果 startindex 调色板入口点数目大。那么 setpalettEntries 函数将执行失败。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.4.17 SetSystemPaletteUse

函数功能：该函数允许一个应用指定系统调色板包含 2 个或 20 个静态颜色。缺省的系统调色板包含20个静态颜色（为一个应用映射一个逻辑调色板时，静态颜色不发生变化）。

函数原型：UINT SetSystemPaletteUse(HDC hdc, UINT uUsage)；

参数：

hdc：设备环境句柄，该设备环境必须引用一个支持彩色调色板的设备。

uUsage：指定系统调色板的一个新的使用，该值可以是如下几种：

SYSPAL_NOSTATIC：系统调色板包含个静态颜色（黑色和白色）。

SYSPAL_NOSTATIC 256：Windows NT 5.0或以后版本，系统调色板不包含静态颜色。

SYSPAL_STATIC：系统调色板包含静态颜色，当一个应用映射它的逻辑调色板时这些静态颜色不发生变化。

返回值：如果成功，返回值是前一个系统调色板，它可以是SYSPAL_NOSTATIC。SYSPAL_NOSTATIC 256或SYSPAL_STATIC；如果失败，返回值是SYSPAL_ERROR。

Windows NT：若想获得更多错误信息，请调用GetLastError函数。

注释：通过调用GetDeviceCaps函数和指定RASTERCAPS常量，一个应用可以确定一个设备是否支持调色板操作。

当一个应用程序窗口移到前台，并且SYSPAL_NOSTATIC值已设置时，该应用必须调用GetSysColor函数来储存当前的系统颜色设值，它必须同时调用 setsysColor 函数仅使用黑色和白色来设置合理的值。当该应用回到后台或被中断，则以前的系统颜色必须被恢复。

如果该函数返回SYSPAL_ERROR，则指定的设备环境无效，或者不支持彩色调色板。当一个应用的窗口最大化并且有输入焦点时，必须调用该函数。当一个应用调用该函数并且uUsage设置为SYSPAL_NOSTATIC之后，它必须遵循如下的步骤：映射逻辑调色板。调用 GetSysColor 函数保存当前系统颜色设置。调用SetsysColor 函数使用白色和黑色设置系统颜色为合理值。例如，相邻或重迭的项目（如窗口框和边界）应分别设置为黑色和白色。

发送WM_SYSCOLORCHANGE消息到其他的顶层窗口，让它们使用新的系统颜色刷新屏幕。

当该应用窗口关闭或失去焦点时，它必须做如下事情：

调用SetsystemrPalette函数并把uUsage参数设置为SYSPAL_STATIC，映射逻辑调色板。

把系统颜色恢复为以前的值，发送WM_SYSColoRCHANGE消息。

速查：Windows NT：3.1及以上版本；Windows：95及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.18 UnrealizeObject

函数功能：该函数重置一个逻辑调色板。它指导系统去映射调色板，虽然它以前并没有被映射过，下一次该应用为一个指定的调色板调用函数时，该系统把该逻辑调色板完全重新映射到系统调色板中。

函数原型：BOOL UnrealizeObject(HGDIOBJ hgdiobj)；

参数：

hgdiobj：标识要被重置的逻辑调色板。

返回值：如果成功，返回值非零；如果失败，返回零。

Windows NT：若想获得更多错误信息，请调用GetLastError函数。

注释：UnrealizeObject不能与存储对象一起使用，通过调用Getstockobject(DEFAULT-PALETTE)得到的缺省调色板是一个存储对象。由hgdiobj所定义的调色板可以是一个设备环境的当前选择的调色板。

Windows 95 和 Windows 98 不支持自动跟踪画笔源点，在使用画笔之前，应用必须调用 UnrealizeObject SetBrushorgEX 和 SelectzObject 函数以定位该画笔。

Windows NT 5.0 或以后版本：如果 hgdiobj 是个画笔，Unrealizeobject 不做任何事情，并且该函数返回 TRUE。可以使用 SetBrusHOrgEX 来设置画笔的源点。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.4.19 UpdateColors

函数功能： 该函数通过把客户区域的当前颜色重新映射到发前被映射过的逻辑调色板，来更新指定设备环境的客户区域。

函数原型： BOOL UpdateColors(HDC hdc)；

参数：

hdc：设备环境句柄。

返回值： 如果成功返回值非零，如果失败，返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：通过调用 GetDeviceCaps 函数和设置 RASTERCAPS 常量，一个应用可以确定一个设备是否支持调色板操作，当系统调色板发生变化时，一个映射逻辑调色板非活动窗口可以通过调用 Updatecolors，作为一种选择方法刷新它的客户区域。

UpdateColors 函数更新一个客户区域比刷新该区域要快，但是，由于在系统调色板改变之前 UpdateColors 函数要进行基于每个像素颜色的色彩转换，因此，每一次调用该函数会导致一些颜色失真。

一旦收到 WM_PALETTECHANGED 消息，该函数必须马上被调用。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5 坐标空间及映射函数（Coordinate Space and Transformation）

Win32 应用程序使用坐标空间和映射函数对输出的图形进行比例缩放、旋转、转换、剪裁和反射。

坐标空间是基于笛卡尔坐标系统的一个平面空间。该坐标系统要求有两个垂直相交的、长度相等的坐标轴。共有 4 种坐标空间：现实坐标、页面坐标、设备坐标、物理设备坐标（显示区，或桌面，或打印纸的页面）。映射方式就是改变（"映射"）对象的大小、方向和形状的一种算法。

3.5.1ClientToScreen

函数功能： 该函数将指定点的用户坐标转换成屏幕坐标。

函数原型： BOOL ClientToScreen(HWND hWnd,LPPOINT lpPoint)；

参数：

：用户区域用于转换的窗口句柄。

：指向一个含有要转换的用户坐标的结构的指针，如果函数调用成功，新屏幕坐标复制到此结构。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

注释：函数用屏幕坐标取代 POINT 结构中的用户坐标，屏幕坐标与屏幕左上角相关联。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

3.5.2DptoLP

函数功能： 该函数将设备坐标转变为逻辑坐标，转变依赖于设备的图形模式，窗口和坐标的起点及范围的设置，和转换的内容。

函数原型： BOOL DptoLP(HDC hdc, LPPOINT lpPoints, int nCount)；

参数：

hdc：指向设备环境的句柄。

lpPoints：指向 POINT 结构数组的指针，每个 POINT 结构中的 X 和 Y 坐标将被转换。

NCount：规定数组中点的数目。

返回值： 如果函数调用成功，返回值为非零值。否则为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：如果设备坐标超过 27 位或如果转换的逻辑坐标超过 32 位，DptoLP 函数调用失败，在溢出的情况下，所有点的结果不能定义。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.3CombineTransform

函数功能： 该函数连接两个全局空间到页空间的转换。

函数原型： BOOL CombineTransform(LPXFORM lpxformResult, CONST XFORM *lpxform1, CONST XFORM *lpxform2)；

参数：

lpxformResult：指向接收结合转换的 XFORM 结构的指针。

lpxformal：指向标识第一次转换的 XFORM 结构的指针。

Lpxform2：指向标识第二次转换的 XFORM 结构的指针。

返回值： 如果调用函数成功，返回值为非零值，否则为零。

注释：联合转换的作用与先用第一个转换，然后再用第二个相同转换。三个转换不需要分明的，例如：Lpxforml 可象 lpxformResult 一样指向同样的 XFORM 结构。

3.5.4GetCurrentPositionEx

函数功能： 该函数获取逻辑坐标中的当前位置。

函数原型： BOOL GetCurrentPositionEx(HDC hdc, LPPOINT lpPoint)；

参数：

hdc：指向设备环境的句柄。

lpPoint：指向接收当前位置坐标的 POINT 结构的指针。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.5GetMapMode

函数功能：该函数获取当前映射方式。

函数原理：int GetMapMode(HDC hdc)；

参数：

hdc：指向设备环境的句柄。

返回值：如果函数调用成功，返回值规定映射方式；否则，返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：下面的列表描述了不同的映射方式。

MM_ANISOTROPIC：逻辑单位转换成具有任意比例轴的任意单位，用 SetWindowsEx 和 SetViewportExtEx 函数指定单位、方向和需要的比例。

MM_HIENGLISH：每一个逻辑单位被映射为 0.001 英寸、X 正方向向左，Y 正方向向上。

MM_ISOTROPC:逻辑单位映射为具有均等刻度轴的任意单位,也就是一个 X 轴的单位等于沿 Y 轴的单位，用 SetWindowsEx 和 SetViewportExtEx 函数指定单位和轴的方向，绘图设备界面根据需要进行调整以保证 X 轴和 Y 轴单位保持一样大小的尺寸。当设置窗口范围时视察口将被调整以达到保持单位统一。

MM_LOENGLISH：每个逻辑单位被映射为 0.01 英寸，X 正方向向右，Y 正方向向上。

MM_TEXT：每个逻辑单位被映射为一个设备像素，X 正方向向右，Y 正方向向下。

MM_TWIPS：逻辑单位映射为具有均等刻度轴的任意单位，也就是一个 X 轴的单位等于沿 Y 轴的单位，用 SetWindowsEx 和 SetViewportExtEx 函数指定单位和轴的方向，绘图设备界面根据需要进行调整以保证 X 轴和 Y 轴单位保持一样大小的尺寸。当设置窗口范围时视察口将被调整以达到保持单位统一。

MM_LOENGLISH：每个逻辑单位被映射为 0.01 英寸，X 方向向向右，Y 正方向向上。

MM_TEXT：每个逻辑单位被映射为一个设备像素，X 正方向向右，Y 正方向向下。

MM_TWIPS：每个逻辑单位被映射为一个打印机点的二十分之一（1 / 1440 英寸），X 正方向向右，Y 正方向向上。

速查：Windows NT：3.1 及以上版本；Windows：98 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.6GetViewportExtEx

函数功能：该函数获取指定的设置环境的当前坐标的 X 和 Y 的范围。

函数原型：BOOL GetViewportExtEx(HDC hdc, LPSIZE lpSize)；

参数：

hdc：指向设备环境的句柄。

lpSize：指向结构的指针，X 和 Y 的范围以设备单位放在此结构中。

返回值：如果函数调用成功，返回值为非零值，否则为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.7GetGraphicsMode

函数功能： 该函数为规定的设备环境获取当前的绘图模式。

函数原型： int GetGraphicsMode(HDC hdc)；

参数：

hdc：指向设备环境的句柄。

返回值： 如果函数调用成功，返回值为当前的绘图模式，其值为以下值中之一：

GM_COMPATBLE：当前绘图模式为兼容性的绘图响应模式，一个与 16 位 windows 兼容的绘图模式，在这种绘图模式中，应用程序不能为指定设备环境设置或修改全局转换，该兼容性的绘图模式是缺省的绘图模式。

GM_ADVANCED：在 Windows NT 和 Windows 98 中，当前绘图模式是高级的绘图模式上，一个允许全局转换的模式，在这种模式中，应用程序可以为指定设置环境设置或改变全局转换。Windows 95：GM_ADVANCED 值不被支持，否则返回值是零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：应用程序可以通过调用 SetGraphicsMode 函数为设备描述设置表格模式。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：Use gdi32.lib。

3.5.8GetViewportOrgEx

函数功能： 该函数为指定设备环境获取坐标原点的 X 坐标和 Y 坐标。

函数原型： BOOL GetViewportOrgEx(HDC hdc, LPPOINT lpPoint)；

参数：

hdc：指向设备环境的句柄。

lpPoint：指向 POINT 结构的指针，该结构存放原点坐标。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.9GetWindowsExtEx

函数功能： 该函数为指定的设备环境获取窗口的 X 和 Y 的范围。

函数原型： BOOL GetWindowsExtEx(HDC hdc, LPSIZE lpSize)；

参数：

hdc：指向设备环境的句柄。

lpSize：指向结构的指针，X、Y 的范围以页面空间单位存放在此结构中。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.10 GetWindowOrgEx

函数功能： 该函数为指定的设备环境获取窗口原点的 X 坐标和 Y 坐标。

函数原型： BOOL GetWindowOrgEx(HDC hdc, LPPOINT lpPoint)；

参数：

hdc：指向设备环境的句柄。

lpPoint：指向 POINT 结构的指针，该结构以页面为单位存放窗口原点的坐标。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.11 GetWorldTransform

函数功能： 该函数获取当前全局空间到页面空间的转换。

函数原型： BOOL GetWorldTransform(HDC hdc, LPXFORM lpXform)；

参数：

hdc：指向设备环境的句柄。

lpXform：指向 XFORM 结构的指针，该结构存放当前全局空间到页面空间的转换值。

返回值： 如果函数调用成功，返回值为非零值。否则为零。

速查： Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.12 LptoDP

函数功能： 该函数把逻辑坐标转换为设备坐标，此转换依赖于设备环境的映射方式，原点的设置、窗口和观察口的范围及全局转换。

函数原型： BOOL LptoDP(LPPOINT lpPoints, int nCount, HDC hdc)；

参数：

hdc：指向设备环境的句柄。

lpPoints：指向 POINT 结构数组的指针，每一个 POINT 结构中的 X 坐标和 Y 坐标将被转换。

nCount：指定数组中点的数目。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

注释：如果逻辑坐标超过 32 位，或如果转换的设备坐标超过 27 位，那么该函数调用失败，在此溢出的情况下，所有点的结果是未定义的。

3.5.13 MapwindowPoints

函数功能： 该函数把相对于一个窗口的坐标空间的一组点映射成相对于另一窗口的坐标空 的一组点。

函数原型： int MapwindowPoints(HWND hWndFrom, HWND hWndTo,LPPOINT lpPoints, UINT cPoints)；

参数：

hWndfrom:转换点所在窗口的句柄,如果此参数为 NULL 或 HWND_DESETOP 则假定这些点在屏幕坐标上。

hWndTo：转换到的窗口的句柄，如果此参数为 NULL 或 HWND_DESKTOP，这些点被转换为屏幕坐标。

lpPoints：指向 POINT 结构数组的指针，此结构数组包含要转换的点，此参数也可指向 RECT 结构，在此情况下，Cpoints 参数应设置为 2。

cPoints：指定 LpPoints 参数指向的数组中 POINT 结构的数目。

返回值：如果函数调用成功，返回值的低位字是每一个源点的水平坐标的像素数目，以便计算每个目标点的水平坐标；高位字是每一个源点的垂直坐标的像素的数目，以便计算每个目标点的垂直坐标，如果函数调用失败，返回值为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

3.5.14 ModifyWorldTransform

函数功能：该函数用指定的方式修改与设备环境有关的全局转换。

函数原型：BOOL ModifyWorldTransform(HDc hdc, CONST XFORM *lpXform, DWORD iMode)；

参数：

hdc：指定环境设备。

lpXform：指向 XFORM 结构的指针，该结构用于修改给定 0 设备环境的全局转换。

iMode：指定转换数据如何改变当前域转换，此参数为下列值之一：

MWT_IDENTTTY：用等同矩阵重新设置当前全局转换，如规定此方式，由 lpXform 指向的 XFORM 结构被忽略。

MWT_LEFTMULTIPLY：用 XFORM 结构中的数据乘以当前转换，XFORM 结构中的数据为左乘矩阵，且与当前转换数据为右乘矩阵。

返回值：如果函数调用成功，返回值为非零值，否则为零。

速查：Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.15 OffsetViewportOrgEx

函数功能：该函数用指定的水平位移和垂直位移改变设备环境的观察窗原点。

函数原型：BOOL OffsetViewportOrgEx(HDC hdc, int nXOffset, int nYOffset, LPPOINT lpPoint)；

参数：

hdc：指向设备环境的句柄。

nXOffset：指定以设备单位为单位的水平位移。

nYOffset：指定以设备单位为单位的垂直位移。

lpPoint:指向 POINT 结构的指针,先前的观察窗原点以设备单位放在该结构中,如果 lpPoint 为 NULL,则先前的观察窗原点没有被返回。

返回值：如果函数调用成功，返回值为非零值，否则为零。

注释：新原点为当前原点与水平位移和垂直位移的总和。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.16 OffsetWindowOrgEx

函数功能： 该函数用指定的水平位移和垂直位移改变设备环境的窗口原点。

函数原型： BOOL OffsetWindowOrgEx(HDC hdc, int nXOffset, int nYOffset, LPPOINT lpPoint)；

参数：

hdc：指向设备环境的句柄。

nXOffset：指定以逻辑单位为单位的水平位移。

nYOffset：指定以逻辑单位为单位的垂直位移。

lpPoint：指向 POINT 结构的指针，先前原点的逻辑坐标存放在此结构中，如果 lpPoint 是 NULL，那么先前原点没有被返回。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.17 ScaleViewportExtEx

函数功能： 该函数用指定的乘数与除数按比例修改设备环境的视窗。

函数原型： BOOL ScaleViewportExtEx(HDC hdc, int Xnum, int Xdenom, int Ynum, int Ydenom, LPSIZE lpSize)；

参数：

hdc：指向设备环境的句柄。

Xunm：指定乘以当前水平范围的数值。

Xdenom：指定除以当前水平范围的数值。

Ynum：指定乘以当前垂直范围的数值。

Ydenom：指定除以当前垂直范围的数值。

lpSize：指向 Size 结构的指针，先前视窗范围（以设备单位）存放在此结构中，如 lpSize 参数的值为 NULL，则什么也没返回。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

注释：观察口范围修改如下：

xNewve=(xOldVE*Xnum)/Xdenom; yNewve=(yOldVE *Ynum)/Ydenom

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.18 ScaleWindowExtEx

函数功能： 该函数用指定的乘数和除数按比例来改变设备环境窗口。

函数原型： BOOL ScaleWindowExtEx(HDC hdc, int Xnum, int Xdenom, int Ynum, int Ydenom, LPSIZE lpSize)；

参数：

hdc：指向设备环境的句柄。

Xnum：指定乘以当前水平范围的数值。

Xdenom：指定除以当前水平范围的数值。

Ynum：指定乘以当前垂直范围的数值。

Ydenom：指定除以当前垂直范围的数值。

lpSize：指向 Size 结构的指针，先前的窗口范围（以设备单位）存放在此结构中，如 lpSize 参数的值为 NULL，则什么也没返回。

返回值：如果函数调用成功，返回值为非零值，否则为零。

注释：窗口范围修改如下：

xNewwe=(xOldWE*Xnum)/Xdenom; yNewwe=(yOldWE*Ynum)/Ydenom

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.19 ScreenToClient

函数功能：该函数把屏幕上指定点的屏幕坐标转换成用户坐标。

函数原型：BOOL ScreenToClient(HWND hWnd, LPPOINT lpPoint)；

参数：

hWnd：指向窗口的句柄，此窗口的用户空间将被用来转换。

lpPoint：指向 POINT 结构指针，该结构含有要转换的屏幕坐标。

返回值：如果函数调用成功，返回值为非零值，否则为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：该函数应用 hWnd 参数标识的窗口和 POINT 结构给定的屏幕坐标来计算用户坐标，然后以用户坐标来替代屏幕坐，新坐标是相对于指定窗口的领域的左上角。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

3.5.20 SetGraphicsMode

函数功能：该函数为指定的设备环境设置图形模式。

函数原型：int SetGraphicsMode(HDC hdc, int iMode)；

参数：

hdc：指向设备环境的句柄。

iMode：指定图形模式，该参数可为下列值之一：

GM_COMPAIBLE：设置与 16 位 Windows 相兼容的图形模式，这是缺省的模式。如果指定此值，应用程序只能通过调用设置窗口视窗范围和原点的函数来改变全局到设备的转换，但不是通过利用 SetWorldTransform 或 ModifyWorldTransform，调用这些函数将失败，设置窗口和视窗范围和原点的函数的例子为 SetViewportExtEx 和 SetwindowExtEx 函数。

GM_ADVANCED：Windows NT 和 Windows 98：设置高级图形模式，允许全局转换。如果应用程序设置或改变指定设备环境的全局转换，必须规定该值在这种模式中，所有的图形，包括文本输出，全部转换为设备环境规定的全局到设备的转换。

Windows 95：不支持 GM_ADVANCED，当操作增强的元文件时，Windows 95 试图使 Windows95 中的增强的元文件看起来与在 Windows NT 上操作一样。为完成此功能，Windows 95 当操作指定的增强元文件记录时，可以模仿 GM_ADVANCED 模式。

返回值：如果调用成功，返回值为老图形模式，调用失败，返回值为零。若想获得更多错误信息，请调用 GetLastError 函数。

备注：根据图形模式，有 3 种不同的表格输出：

TextOutput：在 GM_COMPATIBLE 模式中，TrueType（或矢量字体）文本输出时的操作方式就该 DC 中全局到设备转换而言很象光栅字体文本输出。TrueType 文本总是按从左到右和从上到下的顺序写，即使图形的剩余部分在 X 或 Y 轴被翻动，只有 TrueType(或矢量字体)文本的高被定为合适的高度，在 GM_COMPATIBLE 模式中写非水平文本的唯一办法，是为该设备环境中选举的字体指定非零表格的大小和方向。

在 GM_ADVANCED 模式中 TrueType（或矢量字体）文本输出完全转换为设备环境中域到设备的转换，光栅字体只有很受局限的转换能力（通过某些整型系数来伸展）。图形设备界面（GDI）试图创造出最好的输出。

Rectangle Exclusion：如果设置缺省的 GM_COMPATIBLE 图形模式，当画长方形时，系统不包括底部和最右的边。GM_ADVANCED 图形模式应用于画底边和右边包括在内的长方形。

Arc Drawing：如果设置缺省的 GM_COMPATIBLE 图形为模式，GDI 用设备空间中当前弧的方向来画弧，根据此协议，弧不管页面到设备的转换，此转换要求沿着 X 或 Y 轴的翻动。如果设置了 GM_ADVANCED 图形模式，GDI 总是在逻辑空间逆时针方向画弧。这就是说在 GM_COMPATIBLE 图形模式中，弧控制点和弧本身，都全遵从设备环境的全局到设备的转换。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.21 SetMapMode

函数功能：该函数设置指定设备环境的映射方式，映射方式定义了将逻辑单位转换为设备单位的度量单位，并定义了设备的 X、Y 轴的方向。

函数原型：int SetMapMode(HDC hdc, int fnMapMode)；

参数：

hdc：指向设备环境的句柄。

fnMapMode：指定新的映射方式，此参数可以是下面列出的任何一个值。

MM_ANISOTROPIC：逻辑单位转换成具有任意比例轴的任意单位，用 SetWindowExtEx 和 SetViewportExtEx 函数可指定单位、方向和比例。

MM_HIENGIISH：每个逻辑单位转换为 0.001 英寸，X 的正方面向右，Y 的正方向向上。

MM_HIMETRIC：每个逻辑单位转换为 0.01 毫米，X 正方向向右，Y 的正方向向上。

MM_ISOTROPIC：逻辑单位转换成具有均等比例轴的任意单位，即沿 X 轴的一个单位等于沿 Y 轴的一个单位，用和函数可以指定该轴的单位和方向。图形设备界面（GDI）需要进行调整，以保证 X 和 Y 的单位保持相同大小（当设置窗口范围时，视口将被调整以达到单位大小相同）。

MM_LOENGIISH：每个逻辑单位转换为英寸，X 正方向向右，Y 正方向向上。

MM_LOMETRIC：每个逻辑单位转换为毫米，X 正方向向右，Y 正方向向上。

MM_TEXT：每个逻辑单位转换为一个设置备素，X 正方向向右，Y 正方向向下。

MM_TWIPS；每个逻辑单位转换为打印点的 1 / 20（即 1 / 1400 英寸），X 正方向向右，Y 方向向上。

返回值：如果函数调用成功，返回值指定先前的映射方式，否则，返回值为零，若想获得更多错误信息，请调用 GetLastError 函数。

备注：MM_TEXT 方式允许应用程序以设备像素为单位来工作，像素的大小根据设备不同而不同。MM_HIENLISH, MM_HIMETRIC, MM_LOENGLISH, MM_LOMETRIC 和 MM_TWIPS 方式对必须用物理意义单位（如英寸或毫米）制图的应用程序是非常有用的。MM_ISOTROPIC 方式保证了 1：1 的纵横比。MM_HIENLISH 方式允许对 X 和 Y 坐标分别进行调整。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.22 SetWindowOrgEx

函数功能： 该函数用指定的坐标设置设备环境的窗口原点。

函数原型： BOOL SetWindowOrgEx(HDC hdc, int X, int Y, LPPOINT lpPoint)；

参数：

hdc：指向设备环境的句柄。

X：指定新窗口原点的逻辑 X 坐标。

Y：指定新窗口原点的逻辑 Y 坐标。

lpPoint：指向 POINT 结构的指针，先前的窗口原点存放在此结构中，如果 lpPoint 的值为 NULL，则什么也没返回。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.23 SetWorldTransform

函数功能： 该函数为指定的设备环境设置全局空间和页面空间之间的二维的线性转变，此转换可用于比例缩放、旋转、剪切或翻译图形的输出。

函数原型： BOOL SetWorldTransform(HDC hdc, CONST XFORM *lpXform)；

hdc：指向设备环境的句柄。

lpxform：指向 XFORM 结构的指针，此结构含有转换数据。

返回值： 如果函数调用成功，返回值为非零值，否则为零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 对任何全局空间中的坐标点（X、Y），页面空间中的转换坐标点（X、Y）以下式来决定。

X'=X*eM11+y*eM21+eDx, Y'+X*eM12+y*eM22+eDy

转换矩阵为下列矩阵：

|eM11 eM120|

|eM21 eM220|

|eDx eDy |映射方式（由当前窗口和视口原点定义的）用于定义单位和比例。全局转换常用于以不依靠设备的方式来缩放或旋转逻辑图像。缺省全局转换是偏移为零的等同矩阵。除非先调用 SetGraphicsMode 函数将给定设备环境的图形模式设置为 GM_ADVANCED，否则 SetworldTranform 函数调用将失败，同样，也不可能重新把设备环境的图形方式设置成缺省的 GM_COMPATIBLE 方式，除非全局转已经通过调用 SetWorldtransform 或 ModifyworldTransform 函数首先重新设置缺省的等同转换。 **速查：** Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.5.24 SetViewportExtEx

函数功能： 该函数用指定的值来设置指定设备环境坐标的 X 轴、Y 轴范围。

函数原型： BOOL SetViewportExtEx(HDC hdc, int nXExtent, int nYExtent, LPSIZE lpSize)；

参数：

hdc：指向设备描述表的句柄。

nXExtent：指定观察口以设备单位为单位的水平轴的范围。

nYExent：指定观察口以设备单位为单位的垂直轴的范围。

lpSize：指向 Size 结构的指针，先前的设备单位为单位的视口范围存放在此结构中，如 lpSize 值为 NULL，则什么也没返回。

返回值：如果函数调用成功，返回值为非零值，否则为零。

备注：当设置下面的映射方式时，对函数 SetWindowExtEx 和 SetViewportExtEx 的调用被忽略：

MM_HIENGLISH; MM_HIMETRIC; MM_LOENGLISH; MM_LOMETRIC; MM_TEXT; MM_TWIPS

当设置 MM_ISOTROPIC 方式时，应用程序在调用 SetViewportExtEx 之前必须调用 SetWindowExtEx 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.6 设备环境函数（Device Context）

设备环境是一个结构，它定义了一系列图形对象及其相关的属性，以及会影响输出结果的绘图方式。这些图形对象包括：画笔（用于画直线），笔刷（用于绘图和填充），位图（用于屏幕的拷贝或滚动），调色板（用于定义可用的颜色集），剪裁区（用于剪裁和其他操作），路径（用于绘图和画图操作）。设备环境函数用于对设备环境进行创建、删除或获取信息。

3.6.1CancelDC

函数功能：该函数的功能是把设备上下文环境中悬而未决的操作取消。

函数原型：BOOL CancelDC(HDc hdc)；

参数：

hdc：标识设备上下文环境。

返回值：如果函数执行成功，返回值为非零，如果执行错误，返回值为零。

Windows NT：若想获得错误信息，请调用 GetLastError 函数。

注释：CancelDc 函数一般由多线程应用程序使用，用来取消一个冗长的绘画操作，如果线程 A 激活了一个冗长的绘画操作，线程 B 可以通过调用此函数来取消它。

如果一个操作被取消，那么受影响的线程将返回一个错误，并且该绘画操作的结果是不确定的，如果一个程序中并没有绘画操作，但调用了该函数，其结果也是不确定的。

速查：Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.6.2ChangeDisplaySettings

函数功能：该函数把缺省显示设备的设置改变为由 lpDevMode 设定的图形模式，要改变一个特定显示设备的设置，请使用 ChangeDisplaySettingEx 函数。

函数原型：LONG ChangeDisplaySettings (LPDEVMODE lpDevMode, DWORD dwflags)；

参数：

lpDevMode：指向一个描述转变图表的 DEVMODE 的指针。DEVMODE 的 dmSize 参数必须依 DEVMODE 结构的大小、字节进行初始化，dmDriveExtra 参数必须初始化来显示 DEVMODE 随后的驱动数据的字节数，另外还可以选用以下**参数：**

dmBitsPerPel 每个像素的位数，dmPelsWidth 像素宽度，dmPelsHeight 像度高度，dmDisplayFlage 模式标志。

dmDisplayFrequency 模式频率。

dmposition 在多显示配置中设备的位置（适用于 Windows 98、Windows NT 5.0 及以后版本）。

除了设置好 DEVMOD 结构中诸多元素的值之外，还必须要正确地设置 dmFields 元素中的标志。这些标志表明了 DEVMODE 结构中哪个元素在改变显示设置时使用了。如果在 dmFields 中没有设置正确的位，那么显示设置将不会发生变化。请设置好以下的标志：

DM_BITSPERPEL 使用 dmBitsPerPel 的值，DM_PELSWIDTH 使用 dmPelsWidth 的值，

DM_PELSHEIGHT 使用 dmPelsHeight 的值，DM_DISPLAYFLAGS 使用 dmDisplayFlags 的值，

DM_DISPLAYFREQUENCY 使用 dmDisplayFrequency 的值。

DM_POSITION 使用 fdmPosition 的值（适用于 Windows98、WindowsNT5.0）。

如果 lpDevMode 为空。那么显示设置就使用注册表中的所有当前值。在显示模式动态地调整之后要想再回到缺省的模式，最简单的办法就是把 lpDevMode 参数置为空，使 dwFlags 参数置为 0。

dwflags：表明了图形模式如何改变，它可能是如下的几种形式中的一种：

0：表明当前屏幕的图形模式要动态地改变。

CDS_UPDATEREGISTRY：表明当前屏幕的图形模式会动态地变化，并且该图形模式会更新注册表。该模式信息存贮在用户档案中。

CDS_TEST：系统检测是否要设置被请求的图形模式。

CDS_FULLSCREEN：从本质上讲该模式是暂时的。

CDS_GLOBAL：该设置保存在全局设置区域内， 因此它们会影响所有的用户。该标志仅在与标志一起使用时才有效。CDS_SET_PRIMARY：该设备成为首要设备。

CDS_RESET：设置要改变，即使请求的设置与当前设置一样。CDS_NORESET：设置保存在注册表中，但是它不起作用，该标志只有与 CDS_UPDATEREGISTRY 标志一起使用时才有效。

指定 CDS_TEST 允许一个应用确定哪个图形模式真正的有效。但并不会使系统变为那个有效的图形模式。

如果 CDS_UPDATEREGISTRY 被指定并且它可能会动态地改变图形模式。则注册表中保存该信息并且返回 DISP_CHANGE_SUCCESSFUL 如果不可能使用图形模式动态地改变，则注册表中保存该信息并且返回 DISP_CHANGE_RESTART。

Windows NT:如果指定了 CDS_UPDATEREGISTRY 并且在注册表中不能保存该信息,则图形模式不会改变，并且返回 DISP_CHANGE_NOTUPDATERD。

返回值：ChangeDisplaySettings 函数的返回值如下：

DISP_CHANGE_SUCCESSFUL：设备改变成功。

DISP_CHANGE_RESTART：为使图形模式生效计算机必须重新启动。

DISP_CHANGE_BADFLAGS：标志的无效设置被传送。

DISP_CHANGE_NOTUPDATED：在 WindowsNT 中不能把设置写入注册表。

DISP_CHANGE_BADPARA：一个无效的参数被传递。它可以包括一个无效的标志或标志的组合。

DISP_CHANGE_FAILED：指定图形模式的显示驱动失效。

DISP_CHANGE_ADMODE：不支持图形模式。

注释：为了保证传递给 ChangeDisplaySetting 的 DEVMODE 结构是有效的，并且仅包含显示驱动支持的值，可以使用由 EnumDisplaySettings 函数返回的 DEVMODE。

当显示模式被动态地改变时,WM_DISPLAYCHANGE 消息带着如下的消息参数传递给所有正在运行的应用：

wParam 每像素点的新位数，LOWORD（lParam）新像素宽度，HIWORD（lParam）新像素高度。

速查：Windows NT:3.51 及以上版本;Windows:95 及以上版本;Windows CE:不支持;头文件:winuser.h;库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.3ChangeDisplaySettingsEx

函数功能：该函数把显示设备在 lpszDeviceName 参数中定义的设置，改变为在 lpDevMode 参数中定义的图形模式。

函数原型：LONG ChangeDisplaySettingsEx(LPCSTR lpszDeviceName, LPDEVMODE lpDevMode, HWND hwnd, DWORD dwflags LPVOID lParam)；

参数：

lpszDeviceName：指向一个以 null 结尾的字符串的指针，该字符串指定了一个显示设备，该函数从该显示设备中得到它的图形模式的信息。

查看 EnumDisplayDevice 函数，可得与这些显示设备有关的名字的更详细的信息，lpszDeviceName 参数可以为 NULL（空），NULL 值定义了缺省的显示设备。

lpDevMode：指向 DEVMOD 结构的指针，该结构描述了要经向的图形模式 dmSize 元素，必须以字节为单位，初始化为 DEVMODE 结构的尺寸，可以使用 DEVMODE 结构的如下元素：

dmBitsPerpel 每像素字节数，dmPelsWidth 像素宽度，dmPelsHeight 像素高度，

dmDisplayFlags 模式标志，dmDisplayFrequency 模式频率，

dmPosition：Windows98 和 WindowsNT5.0 以后版本，在多显示配置下的设备位置。

为了设置前面的 DEVMODE 元素，必须在 dmFields 元素中设置适当的标志，这些标志表明 DEVMODE 结构的哪些元素是用来改变显示设置，如果在 dmFields 中没有设置适当的位，则显示设备不会被改变。

设备如下的一个或多个标志：

DM_BITSPERPEL：使用 dmBitsPerPel 值，DM_PELSWIDTH：使用 dmPelsWidth 值。

DM_PELSHEIGHT：使用 dmPelsHeight 值，DM_DISPLAYFLAGS：使用 dmDisplayFlags 值。

DM_DISPLAYFREQUENCY：使用 dmDisplayFrequency 值。

DM_POSITION：Windows98, Windows NT 5.0 和以后版本，使用 dmPosition 值。

如果 lpDevMode 为 NULL，使用注册表中的所有当前值来进行显示设置，在一个动态模式改变之后恢复缺省模式的最简单的办法是把 lpDevMode 设为 NULL 和把 dwFlags 参数置为 0。

hwnd：必须为 NULL。

dwflags：表明图形模式如何改变，dwflags 可取如下值之一：

0：当前屏幕的图形模式将动态地改变。

CDS_UPDATEREGISTRY：当前屏幕的图形模式将动态地改变并且在注册表中图形模式将会更新，模式信息存储在 USER 描述文件中。

CDS_TEST：如果请求的图形模要被设置，则系统进行测试，CDS_FULLSCREEN 从本质上讲这种模式是暂时的。（对于 Windows NT：如果改变到另一个桌面，或从另一个桌面改变，那么该模式将不被重置）。

CDS_GLOBAL：这些设置将保存在全局设置区内，因此它们对所有用户都有作用，该标志只有与 CDS_UPDATEREGISTRY 标志一起设置时才有效。

CDS_SET_PRIMARY：该设备将成为原始设备。

CDS_RESET：即使请求设置与当前设置相同，也会改变设置。

CDS_SETRECT：在 WM_DISPLAYCHANGE 消息中指定的发送到所有应用程序的 lParam 消息参数，被当作 RECT 结构的指针，该结构在一个多屏幕环境中定义了显示设备的位置。如果 iParam 参数为 NULL，意味着显示设备将从多屏幕环境中分离。

CDS_NOREST：设置保存在注册表中，但不起作用，该标志只有和 CDS_UPDATEREGISTRY 标志一起指定时才有效。

指定 CDS_TEST 能让一个应用程序确定哪一种图形模式真正有效而不导致系统改变到那种图形模式如果指定了 CDS_UPDATEREGISTRY 并且可能动态改变图形模式，则这些信息存储在注册表中并返回 DISP_CHANGE_SUCCESSFUL，如果不能动态改变图形模式，则信息存储在注册表中并返回

DISP_CHANGE_RESTART。

Windows：如果指定 CDS_UPDATEREGISTRY，并且信息不能保存在注册表中，那么图形模式不会改变，并返回 DISP_CHANGE_NOTUPDATED。

lParam：必须为空。

返回值：ChabngeDisplaySettingEx 函数返回如下一值：

DISP_CHANGE_SUCCESSRJL：设置改变成功。

DISP_CHANGE_RESTART：为了图形模式正确地运行，计算机必须重新启动。

DISP_CHANGE_BADFLAGS：传送了一个无效的标志集。

DISP_CHANGE_NOTUPDATED：Windows NT 不能把设置写入注册表。

DISP_CHANGE_BADPARAM：传送了一个无效参数，包括一个无效标志或几个无效标志的组合。

DISP_CHANGE_FAILED：指定图形模式显示驱动失效。

DISP_CHANGE_BADMODE：不支持该图形模式。

注释：为了保证传送给 ChangeDisplaySettingEx 函数的 DEVMODE 结构是有效的，并且仅包含显示驱动支持的值，请使用 EnumDisplaySetting 函数返回的 DEVMODE 结构。

当显示模式动态改变时,WM_DISPLAYCHAGE 消息带着如下的消息参数传送给所有正在运行的应用程序：

wParam 像素的新位数，LOWORD(lParam)新像素宽度；HIWORD(iParam)新像素高度。

速查：Windows NT：5.0 及以上版本；Windows：98 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.4CreateCompatibleDC

函数功能：该函数创建一个与指定设备兼容的内存设备上下文环境（DC）。

函数原型：HDC CreateCompatibleDC(HDC hdc)；

参数：

hdc：现有设备上下文环境的句柄，如果该句柄为 NULL，该函数创建一个与应用程序的当前显示器兼容的内存设备上下文环境。

返回值：如果成功，则返回内存设备上下文环境的句柄；如果失败，则返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：内存设备上下文环境是仅在内存中存在的设备上下文环境，当内存设备上下文环境被创建时，它的显示界面是标准的一个单色像素宽和一个单色像素高，在一个应用程序可以使用内存设备上下文环境进行绘图操作之前，它必须选择一个高和宽都正确的位图到设备上下文环境中，这可以通过使用 CreateCompatibleBitmap 函数指定高、宽和色彩组合以满足函数调用的需要。

当一个内存设备上下文环境创建时，所有的特性都设为缺省值，内存设备上下文环境作为一个普通的设备上下文环境使用，当然也可以设置这些特性为非缺省值，得到它的特性的当前设置，为它选择画笔，刷子和区域。

CreateCompatibleDc 函数只适用于支持光栅操作的设备，应用程序可以通过调用 GetDeviceCaps 函数来确定一个设备是否支持这些操作。

当不再需要内存设备上下文环境时，可调用 DeleteDc 函数删除它。

ICM：如果通过该函数的 hdc 参数传送给该函数设备上下文环境(Dc)对于独立颜色管理（ICM）是能用的，则该函数创建的设备上下文环境(Dc)是 ICM 能用的，资源和目标颜色间隔是在 Dc 中定义。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.5CreateDC

函数功能：该函数通过使用指定的名字为一个设备创建设备上下文环境。

函数原型：HDC CreateDC(LPCTSTR lpszDrive, LPCTSTR lpszDevice, LPCTSTR lpszOutput, CONST DEVMODE *lplnitData);

参数：

lpszDriver：

Windows NT：指向一个以 Null 结尾的字符串的指针，该字符串为显示驱动指定 DISPLAY 或者指定一个打印驱动程序名，通常为 WINSPOOL。

Windows 95 和 Windows 98：在基于 32 位的应用程序中，该参数被忽略或者为 Null。但有一个例外，可以通过指定以 null 结尾的 DISPLAY 来得到一个显示设备上下文环境，如果该参数为 DISPLAY。其他所有的参数必须为 Null。

lpszDevice:指向一个以 null 结尾的字符串的指针,该字符串指定了正在使用的特定输出设备的名字，它不是打印机模式名。LpszDevice 参数必须被使用。

lpszOutput：该参数在 32 位应用中被忽略；并置为 Null，它主要是为了提供与 16 位应用程序兼容，更多的信息参见下面的注释部分。

LplnitData：指向包含设备驱动程序的设备指定初始化数据的 DEVMODE 结构的指针，DocumentProperties 函数检索指定设备获取已填充的结构，如果设备驱动程序使用用户指定的缺省初始化值。则 lplnitData 参数必须为 Null。

返回值：成功，返回值是特定设备的设备上下文环境的句柄；失败，返回值为 Null。

Windows NT：若想获得更多错误信息，可调用 GetLastError 函数。

注释：16 位的 Windows 应用程序使用 lpszOutput 参数指定一通讯口的名字或者指定打印到文件，基于 32 位的 Windows 应用程序不需要指定通讯口的名字，32 位应用程序使可以通过调用 StartDoc 函数来完成打印到文件的功能，在调用 StartDoc 函数时，使用 DOCINFO 结构。该结构的 lpszOutput 成员指定了输出文件名的路径。当不再需要该设备上下文环境时可调用 DeleteDc 函数删除它。

ICM:通过设置 DEVMODE 结构(由 plnitDufa 参数指定)的 dmlCMMethod 元素为合适值，可使 ICM 激活。

Windows CE：如果没有给 lpszDriver 参数驱动程序名，则 Windows 返回一个显示设备上下文环境，Windows CE 忽略 lpszDevice 参数，Windows CE 把 lplnitData 和 lpszOutput 参数传送给驱动程序而不做任何修改。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib，Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.6CreatelC

函数功能：该函数为指定设备创建一个信息描述表，该信息描述表能在不创建设备上下文环境的情况下提供一种快速得到设备信息的方法。

函数原型：HDC CreatelC(LPCTSTR lpszDriver, LPCTSTR lpszDevice, LPCTSTR lpszOutput, CONST DEVMODE *lpdvmlnit);

参数：

lpszDriver：指向一个以 null 结尾的字符串，该字符串指定了设备驱动名（如：Epson）。

lpszDevice：指向一个以 null 结尾的字符串，该字符串指定了一个正在使用的输出设备的名字，就如打印管理器显示的那样（如 Epson Fx_80），它不是打印机模式名，必须使用 lpszDevice 参数。

LpszOutput：指向一个以 null 结尾的字符串，该字符串指定物理输出介质（文件或输出端口）的文件

名或设备名。该参数被忽略，它的存在仅仅是提供函数原型以保持与 16 位 Window API 中使用的原型一致。

lpdvmlnit：指向 DEVMODE 结构的指针。该结构包含设备驱动器的指定初始化数据。DocumentProperties 函数为可检索指定设备获取已填充的结构，如果设备驱动器使用由用户指定的缺省初始化数据时，lpdvmlnit 参数必须为 Null。

返回值： 如果成功，返回一个信息描述表的句柄；失败，则返回 Null。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

注释：如果一个应用程序调用一个 GDI 绘画函数并且提供一个句柄来确认一个信息描述表的话，将会发生错误，当不再需要该信息描述表时可调用 DeleteDc 函数删除它。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib，Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.7DeleteDC

函数功能： 该函数删除指定的设备上下文环境（Dc）。

函数原型： BOOL DeleteDC(HDC hdc)；

参数：

hdc：设备上下文环境的句柄。

返回值： 成功，返回非零值；失败，返回零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：如果一个设备上下文环境的句柄是通过调用 GetDC 函数得到的，那么应用程序不能删除该设备上下文环境，它应该调用 ReleaseDC 函数来释放该设备上下文环境。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.8DeleteObject

函数功能： 该函数删除一个逻辑笔、画笔、字体、位图、区域或者调色板，释放所有与该对象有关的系统资源，在对象被删除之后，指定的句柄也就失效了。

函数原型： BOOL DeleteObject(HGDIOBJ hObject)；

参数：

hObject：逻辑笔、画笔、字体、位图、区域或者调色板的句柄。

返回值： 成功，返回非零值；如果指定的句柄无效或者它已被选入设备上下文环境，则返回值为零。

注释：当一个绘画对象（如笔或画笔）当前被选入一个设备上下文环境时不要删除该对象。当一个调色板画笔被删除时，与该画笔相关的位图并不被删除，该图必须单独地删除。

Windows CE：当对象在当前被选入一个设备上下文环境时，DeleteObject 函数返回错误。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.9DeviceCapabilities

函数功能： 该函数检索打印机设备驱动器的性能。

函数原型： DWORD DeviceCapabilities(LPCTSTR pDevice, LPCTSTR pPort, WORD fwCapability, LPTSTR

pOutput, CONST DEVMODE *pDevMode);

参数：

pDevice：指向以 null 结尾的字符串的指针，该字符串包含打印机名，注意是打印机名，而不是打印驱动器名。

nPort：指向以 null 结尾的字符串的指针，该字符串包含了设备连接的通讯口的名字，如 LPT1。

fwCapablity：指定要查询的性能，该参数可取下列值之一：

DC_BINADJUST：Windows 95 检索在 DEVMODE 结构中指定的纸张资源的页面位置，DEVMODE 结构由指定，返回值可能如下：

DCBA_FACEDOWNCENTER, DCBA_FACEUPCENTER, DCBA_FACEUPLEFT, DCBA_FACEUPRIGHT, DCBA_FACEDOWNNONE, DCBA_FACEDOWNCENTER, DCBA_FACEDOWNLEFT, DCBA_FACEDOWNRIGHT, windows 98 不支持。

DC_BINNAMES：拷贝一个包含一系列纸张接收器名的数组。这个数组的格式为 char PaperName [cBinMax] [cchBinName]，其中 cchBinName 为 24。如果 pOutput 参数为 NULL，返回值是所需的接收器项数，否则返回值是接收器的拷贝数。

DC_BINS：检索一系列可用的纸张接收器，该函数把列表作为一个 WORD 数组拷贝到 pOutput 参数中，如果 pOutput 为 NULL，则函数返回所支持的接收器的数目，以使应用程序有机会分配准确尺寸的缓冲区，至于有关接收器的更多信息，请参见结构的元素的描述。

DC_COPIES：返回设备可以打印的拷贝数。DC_DRIVER 返回打印驱动程序的版本数。

DC_DATATYPE_PRODUCED：在 Windows 95 中返回值是打印驱动程序支持的数据类型数，如果函数返回-1，驱动程序只能理解 RAW 数据类型，被支持的数据类型名拷贝到一个数组中，在调用 StartDoc 函数指定该数据类型时，使用 DOCINFO 结构中的名字。Windows 98 不支持。

DC_DUPLEX：返回双向支持的级别，如果打印机有双向打印的能力，函数返回 1，否则值为零。

DC_EMF_COMPLIANT：在 Windows 95 中确定打印机驱动程序是否支持增强图元文件（EMF），返回值为 1 表明打印机驱动支持 EMF，-1 表示不支持 EMF，Windows 98 不支持。

DC_ENUMRESOLUTIONS：返回一系列可用的分辨率，如果 pOutput 为 Null，则函数返回可用的分辨率配置，分辨率由一对 LONG 类型整数来分别表示水平和垂直分辨率（每英寸点数）。

DC_EXTRA：返回打印驱动 DEVMODE 结构的设备专用部分所需的字节。

DC_FIELDS：返回打印驱动 DEVMODE 结构的 dmFields 元素。DmFilelds 元素指定结构的设备无关部分中哪些成员是打印驱动支持的。

DC_FILEFEPENDENCIES：返回安装驱动程序时必须装载的系列文件。如果 pOutput 参数为 Null 函数返回文件个数，否则 pOutput 用 char [chFileName, 64]形式指向一个文件名数组，每个文件名都是以 Null 结束的字符串。

DC_MAXEXTENT：返回 POINTS 结构，该结构包含打印机驱动程序的 DEVMODE 结构的 dmPaperLength 和 dmPaperWidth 元素可指定的最大纸张规格，POINTS 结构的 x 元素包含最大 dmPaperWidth 值。Y 元素包含最大 dmPaperLength 值。

DC_MINEXTENT：返回 POINTS 结构，该结构包含打印机驱动程序的 DEVMODE 结构的 dmPaperLengh 和 dmPaperWidth 元素可指定的最小纸张规格，POINTS 结构的 x 元素包含最大 dmPaperWidth 值，y 元素包含最大 dmPaperLength 值。

DC_ORIETATION：返回设备纵向和横向打印之间的关系，以纵向打印逆时针旋转产生横向打印时旋转的度数来表示，返回值如下：

0：没有横向，90：纵向旋转 90 度产生横向，270：纵向旋转 270 度产生横向。

DC_PAPERNAMES：检索一系列支持的纸名（如 Letter 或 Lega）。如果 pOutput 参数为 Null，则函数返回可用的打印机规格数。否则 pOutput 以 char [cpaperNames, 64]的形式指向一个打印纸名数组，每一个纸张名都是以 null 结束的字符串。

DC_PAPERS：检索一系列支持的纸张规格，函数把列表作为一个 WORD 数组拷贝到 pOutput 中，并且返回数组中的项数。如果 pOutput 为 Null，则函数返回所支持的纸张规格数以使应用程序有机会分配准确尺寸的缓冲区，有关打印纸规格的更多信息，请参见 DEVMOVE 结构中关于 dmPagerSize 成员的描述。

DC_PAPERSIZE:以十分之一毫米为单位,把所有支持的打印机尺寸,拷贝到由 pOutput 参数指定的 POINT 结构数组中去。如果打印机在 DMORIENT_PORTRAIT 方向，则返回纸张尺雨的宽（x 维）和长（y 维）。

DC_SIZE：返回打印机驱动程序 DEVMODE 结构的 dmSize 元素值。

DC_TRUETYPE：检索驱动程序使用 TrueType 字体的能力，对于 DC_TRUETYPE，pOutput 参数应为 NULL，返回如下一值或多值，其值的含义为：

DCTT_BITMAP：设备可以按图形方式打印 TrueType 字体。DCTT_DOWNLOAD：设备能够卸载 TrueType 字体。

DCTT_DOWNLOAD_OUTLINE：在 Windows 95 和 Windows 98 中设备可以卸载 TrueType 字体。

DCTT_SUBDEV：设备可以用 TrueType 字体替代设备字体。

DC_VERSION：返回打印机驱动程序遵循的指定版本。

Poutput：指向一个字节数组，数组的格式依赖于 fwCapabilities 参数的设置。如果 pOutput 为零，则 DeiceCapabilities 返回输出数据所需的字节数。

PdevMode：指向一个 DEVMODE 结构，如果该参数为 NULL，DeviceCapabilities 则检索指定打印机驱动程序的当前缺省的初始化值，否则该函数检索包含在由 pDevMode 指向的结构中的值。

返回值：如果成功，返回值依赖于 fwCapabilities 参数的设置，如果失败，则返回 C1。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：在 16 位 Windows 版本中，DeviceCapabilities 函数在打印机驱动程序中执行，并且需要调用 LoadLibrary 和 GetProcAddress 函数得到该函数的指针，但这些都不再需要了。因为 DeviceCapabilities 是 WinAPI 的一部分。

在打印机驱动程序中不需要调用 LoadLibraryp 函数，由 DevMode 参数指向的 DEVMODE 结构，可以通过调用 DocumentProperties 函数而得到。

速查：Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：winspool.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.10 DrawEscape

函数功能：该函数存取视频显示的画图能力，该视频显示不能直接通过图形设备接口（GDI）使用。

函数原型：int DrawEscape(HDC hdc, int nEscape, int cblnput, LPCSTR lpszlnData)；

参数：

hdc：指定视频显示器的设备上下文环境的句柄。

nEscape：指定要执行的转义函数。

cblnput：指定由 lpszlnData 参数指向的数据字节数。

lpszlnData：指定的转义所需输入结构的指针。

返回值：返回值表示函数的结果。如果成功，返回值大于零，但检测执行的 QUERYESCSUPPORT 绘图转义除外。如果转义没有执行，则返回值为零，如果发生错误，则返回值小于零。

Winodws NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：当一个应用程序调用 DrawEscape 函数时，由 Input 和 lpszlnData 指定的数据直接传送给指定的显示驱动程序。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.6.11 Enumdisplaydevices

函数功能：该函数可得到系统中显示设备的信息。

函数原型：BOOL Enumdisplaydevices(PVOID Unused, DWORD iDevNum, PDISPLAY_DEVICE lpDisplayDevice, DWORD dwFlags);

参数：

Unused：该参数当前不用，应设为 Null。

iDevNum：指定感兴趣的显示设备的索引值，操作系统通过索引值确定每一个显示设备。索引值是连续的整数。从 0 开始，例如：如果一个系统有三个显示设备，那么它们的索引值为 0、1、2。

lpdisplayDevice：DISPLAY_DEVICE 结构的指针，该结构检索由 iDevNum 指定的显示设备的信息，在调用 EnumDisplayDevices 之前，必须以字节为单位把 DISPLAY_DEVICE 结构中 cb 元素初始化为 DISPLAY_DEVICE 结构的大小。

dwFlags：约束函数行为的一组标志，当前没有定义标志。

返回值：如果成功，则返回值非零；如果失败，则返回值为零；如果 iDevNum 大于最大的设备索引，则函数失败。

注释:为了查询系统的所有显示设备,在一个循环中调用该函数开始时 iDevNum 设为 0,并增加 iDevNum,直到函数失败。

速查：Windows NT：5.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.12 EnumDisplaySettings

函数功能：该函数得到显示设备的一个图形模式设备，通过对该函数一系列的调用可以得到显示设备所有的图形模式信息。

函数原型：BOOL EnumDisplaySettings(LPCTSTR lpszDeviceName, DWORD iModeNum, LPDEVMODE lpDevMode);

参数：

lpszDeviceName：指向一个以 null 的结尾的字符串，该字符串指定了显示设备。此函数将获得该显示设备的图形模式信息。该参数可以为 NULL。NULL 值表明调用线程正运行在计算机的当前显示设备上。如果 lpszDeviceName 为 NULL，该字符串的形式为\\.\displayx，其中 x 的值可以为 1、2 或 3。对于 Windows 95 和 Windows 98，lpszDeviceName 必须为 NULL。

iModeNum：表明要检索的信息类型，该值可以是一个图形模式索引，也可以是下列一值：

ENUM_CURRENT_SETTINGS：检索显示设备的当前设置。

ENUM_REGISTRY_SETTINGS：检索当前存储在注册表中的显示设备的设置。

图形模式索引值从零开始，要得到一个显示设备的所有图形模式信息，可以一系列地调用 EnumDisplaySettings 函数,并且 iModeNum 显为一个非零值时,则函数返回的信息是最近一次使用 iModeNum 置为零调用该函数时存储的信息。

lpDevMode：DEVMODE 结构的指针，该结构存储指定图形模式的信息，在调用 EnumDisplaySettings 之前，设置 dmSize 为 sizeof(DEVMODE)，并且以字节为单位，设置 dmDriveExtra 元素为接收专用驱动数据可用的附加空间。

EnumDisplaySettings 函数设置如下五个 DEVMODE 元素的值：dmBitsPerpel、dmPelsWidth、dmPelsHeight、dmDisplayFlags、dmDisplayFrequency。

返回值：如果成功，返回非零值；如果失败，返回零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：如果 iModeNum 大于显示设备最后的图形模式索引，那么函数就会失败，如同在 iModeNum 参数中描述的那样，使用这种方法可以枚举显示设备所有的图形模式。

速查：Windows NT：3.51 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.13 EnumObjectsProc

函数功能：该函数是应用程序定义的回调函数，为 EnumObjects 函数一起使用。此函数用来处理对象数据，GOBJECTENUMPROC 类型定义了指向该回调函数的指针，EnumObjectsProc 是应用程序定义函数名字的一个占位符。

函数原型：VOID CALLBACK EnumObjectsProc(PVOID lpLogObject, LPARAM lpData)；

参数：

lpLogObject：指向描述对象特性的 LOGPEN 或 LOGBRUSH 结构的指针。

lpData：指向应用程序定义的数据的指针，该数据由 EnumObjects 函数传送。

返回值：无。

注释：应用程序必须通过传送函数的地址给 EnumObjects 函数来注册该函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：用户自定义。

3.6.14 EnumObjects

函数功能：该函数为指定的设备上下文环境枚举可用的笔或画笔，对于每个可用对象，该函数调用应用程序定义的回调函数，提供数据描述该对象。Enumobjects 继续调用回调函数直到回调函数返回零或所有的对象都已枚举为止。

函数原型：int EnumObjects(HDC hdc, int nObjectType, GOBJENUMPROC lpObjectFunc, LPARAM lParam)；

参数：

hdc：设备上下文环境句柄。

nObjectType：指定对象类型，该参数可为 OBJ_BRUSH 或 OBJ_PEN。

lpObjectFunc：指向应用程序定义的回调函数的指针，关于回调函数的更多信息，参见 EnumObjectsProc 函数。

LParam：指向应用程序定义的数据的指针，该数据与对象信息一起传送给回调函数。

返回值：返回值表明回调函数最后的一次返回值并且由用户定义；如果有太多的对象而无法枚举，则返回值为－1，在这种情况下回调函数不被调用。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.6.15 GetCurrentObject

函数功能：该函数得到一个从特定类型中选定目标的设备对象句柄。

函数原型：HGDIOBJ GetCurrentObject(HDC hdc, UINT uObjectType)；

参数：

hdc：设备上下文环境句柄。

uObjectType：指定要查询的对象类型，该参数可取如下一值：

OBJ_PEN：笔；OBJ_BRUSH：返回当前选择的画笔；OBJ_PAL：返回当前选择的调色板；

OBJ_FONT：返回当前选择的字体；OBJ_BMAP：返回当前选择的位图。

返回值：如果成功，返回指定对象的句柄；如果失败，则返回 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：应用程序可以使用 GetCurrentObject 和 GetObject 函数来检索对当前选入给定设备上下文环境的图形对象的描述。

Windows CE：Windows CE1.0 版不支持 uObjectType 参数的 OBJ_PAL 标志值，在 WindowsCE2.0 版中，该函数与在 Windows 桌面平台完全相同。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.16 GetDC

函数功能：该函数检索一指定窗口的客户区域或整个屏幕的显示设备上下文环境的句柄，以后可以在 GDI 函数中使用该句柄来在设备上下文环境中绘图。

GetDCEx 函数是 GetDC 的一个扩展，它能使应用程序更多地控制在客户区域内如何或是否发生剪切。

函数原型：HDC GetDC(HWND hWnd)；

参数：

hWnd：设备上下文环境被检索的窗口的句柄，如果该值为 NULL，GetDC 则检索整个屏幕的设备上下文环境。

Windows 98，WindowsNT 5.0 或以后版本：如果该参数为 Null，GetDC 检索首要显示器的设备上下文环境，要得到其他显示器的设备上下文环境，可使用 EnumDisplayMonitors 和 CreateDc 函数。

返回值：如果成功，返回指定窗口客户区的设备上下文环境；如果失败，返回值为 Null。

Windows NT：若想获得更多错误信息，可调用 GetLastError 函数。

注释：GetDC 函数根据指定的等级类型检索指定窗口普通的、典型的或特有的设备上下文环境。

对于普通设备上下文环境，GetDC 在每次检索的时候部分分配给它缺省特性，对于典型和特有的设备上下文环境，GetDC 不改变先前设置的特性。

在使用普通设备上下文环境绘图之后，必须调用 ReleaseDc 函数释放该设备上下文环境，典型和特有设备上下文环境不需要释放，设备上下文环境的个数仅受有效内存的限制。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

3.6.17 GetDCBrushColor

函数功能：该函数确认将要返回画笔颜色的设备上下文环境（DC）。

函数原型：GetDCBrushColor(HDC hdc)；

参数：

hdc：设备上下文环境的句柄，该设备上下文环境的画笔颜色将被返回。

返回值：如果成功，返回值是当前 Dc 的画笔颜色的颜色引用；如果失败，返回值为 CLR_INVALID。

注释：GetDCBrushColor 函数返回先前的 DC_BRUSH 颜色，即使系统备用对象 DC_BRUSH 没有选入设备上下文环境。有关设置设备上下文环境画笔颜色的更多信息，参见 SetDCBrushColor。

ICM：如果 ICM 可用就执行颜色管理。

速查：Windows NT：5.0 及以上版本；Windows：98 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：msimg32.dll。

3.6.18 GetDCEx

函数功能：该函数检索指定窗口客户区域或整个屏幕的显示设备上下文环境的句柄，在随后的 GDI 函数中可以使用该句柄在设备上下文环境中绘图。

函数原型：HDC GetDCEx(HWND hWnd, HRGN hrgnClip, DWORD flags)；

参数：

hWnd：窗口的句柄，该窗口的设备上下文环境将要被检索，如果该值为 NULL，则 GetDCEx 将检索整个屏幕的设备上下文环境。

Windows98、Windows NT 5.0 和以后版本：如果该参数为 NULL，GetDCEx 函数检索首要显示器的设备上下文环境，要得到其他显示器的设备上下文环境使用 EnumDisplayMonitors 和 CreateDC 函数。

hrgnCLip：指定一剪切区域，它可以与设备上下文环境的可见区域相结合。

flags：指定如何创建设备上下文环境，可取下列值的组合：

DCX_WINDOW：返回与窗口矩形而不是与客户矩形相对应的设备上下文环境。

DXC_CACHE:从高速缓存而不是从 OWNDC 或 CLASSDC 窗口中返回设备上下文环境。从本质上覆盖 CS_OWNDC 和 CS_CLASSDC。

DCX_PARENTCLIP：使用父窗口的可见区域，父窗口的 WS_CIPCHILDREN 和 CS_PARENTDC 风格被忽略，并把设备上下文环境的原点，设在由 hWnd 所标识的窗口的左上角。

DCX_CLIPSIBLINGS：排除 hWnd 参数所标识窗口上的所有兄弟窗口的可见区域。

DCX_CLIPCHILDREN：排除 hWnd 参数所标识窗口上的所有子窗口的可见区域。

DCX_NORESETATTRS：当设备上下文环境被释放时，并不重置该设备上下文环境的特性为缺省特性。

DCX_LOCKWINDOWUPDATE:即使在排除指定窗口的 LockWindowUpdate 函数调用有效的情况下也许会绘制，该参数用于在跟踪期间进行绘制。

DCX_EXCLUDERGN：从返回设备上下文环境的可见区域中排除由 hrgnClip 指定的剪切区域。

DCX_INTERSECTRGN：对 hrgnClip 指定的剪切区域与返回设备描述的可见区域作交运算。

DCX_VALIDATE：当与 DCX_INTERSECTUPDATE 一起指定时，致使设备上下文环境完全有效，该函数与 DCX_INTERSECTUPDATE 和 DCX_VALIDATE 一起使用时与使用 BeginPaint 函数相同。

返回值：如果成功，返回值是指定窗口设备上下文环境的句柄，如果失败，返回值为 Null。HWnd 参数的一个无效值会使函数失败。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：除非显示设备上下文环境属于一个窗口类，在画图操作之后一定要调用 ReleaseDC 函数释放设备上下文环境。因为只有 5 个公用设备上下文环境在任何给定的时间都有效。释放设备上下文环境失败导致其他应用程序不能访问该设备上下文环境。

如果当窗口类注册时，CS_CLASSDC、CS_OWNDC 或 CS_PARENTDC 被指定为 WNDCLASS 结构的风格，那么该函数返回一个属于该窗口类的设备上下文环境。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

3.6.19 GetDCOrgEx

函数功能：该函数得到设备上下文环境的最终转换原点，最终转换原点指定了一个位移。系统使用该位移把设备转换成客户坐标。

函数原型：BOOL GetDCOrgEx(HDC hdc, LPPOINT lpPoint)；

参数：

hdc：指定设备上下文环境，该设备上下文环境的最终转换原点将要被检索。

lpPoint：指向 POINT 结构的指针，该函数将把它置为最终转换原点，以设备坐标的形式。

返回值：如果成功，返回非零值，如果失败，返回零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：最终转换原点与屏幕的物理原点有关。

速查：Windows NT：3.5 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.6.20 GetDCPenColor

函数功能：该函数设置当前 DC 笔颜色为指定颜色，如果设备不能提供指定颜色值，将返回最近的物理颜色。

函数原型：GetDCPenColor(HDC hdc)；

参数：

hdc：设备上下文环境的句柄，该设备上下文环境的笔颜色将被返回。

返回值：如果成功，则返回先前 DC 笔颜色的颜色引用；如果失败，则返回 CLR_INVALID。

注释：即使存储对象 DC_PEN 设备被选入 DC，GetDCPenColor 函数也将返回先前 DC_PEN 颜色，更多信息参见 Setting the pen 或 Brush Color 和 SetDCPenColor。

ICM：如果 ICM 可用则执行颜色管理。

速查：Windows NT：5.0 及以上版本；Windows：98 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：msimg32.lib。

3.6.21 GetDeviceCaps

函数功能：该函数检索指定设备的设备指定信息。

函数原型：int GetDeviceCaps(HDC hdc, int nlndex)；

参数：

hdc：设备上下文环境的句柄。

nIndex：指定返回项，该参数取下列一值。

DRIVERVERSION：设备驱动程序版本。

TECHNOLOGY：设备技术，它可以是下列一值：

DT_PLOTTER：矢量绘图仪；DT_RASDISPLAY：光栅显示器；

DT_RASPRINTER：光栅打印机；DT_RASCAMERA：光栅照相机；

DT_CHARSTREAM：字符流；DT_METAFILE：图元文件；

DT_DISPFILE：显示器文件。

如果 hdc 参数指定图元文件的设备上下文环境，则设备技术就是给 CreateEnhMetaFile 函数的引用设

备，使用 GetObjectType 函数可以确定它是否是一增强元文件设备上下文环境。

HORZSIZE：物理屏幕的宽度（毫米）；VERTSIZE：物理屏幕的高度（毫米）；HORZRES：屏幕的宽度（像素）；

VERTRES：屏幕的高度（光栅线）；

LOGPIXELSX：沿屏幕宽度每逻辑英寸的像素数，在多显示器系统中，该值对所显示器相同；

LOGPIXELSY：沿屏幕高度每逻辑英寸的像素数，在多显示器系统中，该值对所显示器相同；

BITSPIXEL：像素相连颜色位数；PLANES：颜色位面数；NUMBRUSHES：设备指定同画笔数；

NUMPENS：设备指定笔数；NUMFONTS：设备指定字体数；

NUMCOLORS：设备颜色表的入口数，如果设备的色深不超过 8 位像素。对于超过色深的设备返回－1；

ASPECTX：用于画线的设备像素的相对宽度；ASPECTY：用于画线的设备像素的相对高度；

ASPECTXY：用于画线的设备像素的对角线宽度；PDEVLCESIZE：保留；

CLIDCAPS：显示设备支持剪切性能的标志。如果设备可剪切为一个长方形，则为 1，否则为 0；

SIZEPALETTE：系统调色板中的入口数目，只有在设备驱动器在 RASTERCAPS 索引中设置 RC PALETFE 位时该索引值才是有效的。且该索引值只能用于 16 位 Windows 的驱动器；

NUMRESERVED：系统调色板中保留的入口数目，只有在设备驱动器在 RASTERLAP 索引中设置 RC PALETFE 位时，该索引值才是有效的且该索引值只有能于 16 位的 Windows 驱动器；

COLORRES：实际位像的实际设备颜色，只有设备驱动器在 RASTERLAP 索引中设置 RCPALETFE 位时，该索引值才是有效的且该索引值只能用于 16 位的 Windows 驱动器；

PHYSICALWIDTH：对于打印设备，为以设备单位的物理页面宽度。例如一个在 8.5*11 纸上设置为 600dpi 的打印机的设备物理单位宽度值为 5100，注意物理页面总是大于可打印的页面面积，且从不小于；

PHPSICALHEIGHT：为以设备单位的物理页面宽度。例如一个在 8.5*11 纸上设置为 600dpi 的打印机的设备物理单位高度值为 6600；

PHYSICALOFFSETX：对于打印设备，从物理页面左边到打印页面右边的距离，例如，一个在 8.5*11 纸上设置为 600dpi 的打印机，不能在超出左边 0.25 处打印，且有一个 150 备单位的水平物理位移；

PHYSICALOFFSETY；对于打印设备。从物理页面上面到打印页面上边的距离。例如一个在 8.5*11 打印纸上设置为的打印机，不能在超出上边的地方打处，且有一个设备单位的物理位移；

VREFRESH：Windows NT：对于显示设备。设备的当前垂直刷新率以每秒中的循环次数为单位 0 或 1 刷新率代表显示硬件的缺省刷新率，此缺省刷新率通常通过设置显卡或主板的跳线来改变，或通过一个不使用 Win32 显示函数比如 ChangeDisplay Setting 的一个配置程序来设置；

DESKTOPHORZRES：Windows NT：可视桌面的以像素为单位的宽度。如果设备支持一个可视桌面或双重显示则此值可能大于 VERTRES；

SCALINGFACTORX：打印机 x 轴的比例系数；SCALINGFACTORY：打印机 y 轴的比例系数。

BLTALIGNMENT：在 Windows NT 中作为像素倍数的水平绘图调整，对于最好的绘图操作，窗口绘图应该是水平调整到此值的倍数。0 显示设备为加速的，且可用任何调整。

SHADEBLENDCAPS：在 Windows 98、Windows NT 5.0 和以后版本中此值显示设备的阴影和混合特性。

SB_CONST_ALPHA：处理 BLENDFUNCTION 结构中的 Source constantAlpha 元素，并通过 AlphaBlend 数中的 blendFunction 参数来指定；

SB_GRAD_RECT：进行 Gradientfill 矩形填充的能力。SB_GRAD_TRI；进行 Gradientfill 三角形填充的能力；

SB_NONE:设备不支持这些特性中的任何一个。SB_PIXEL_ALPHA:处理 AlphaBlond 中每一个像素 Alphx；

SB_PREMULT_ALPHA：在 Alphablend 中对 alpha 进行预乘；

RASTERCAPS：设备所支持的光栅性能，可以是下列值的某种组合；

RC_BANDING：需要联合支持。RC_BITBLT：支持传送位图。

RC_BITMAP64：支持大于 64K 的位图。RC_DI_BITMAP：支持 SetDIBits 和 GetDIBits 函数。

RC_DIBTODEV：支持 SetDIBits To Device 函数；RC_FLOODFILL：支持连续填充

RC_GDI20_OUTPUT：支持 16 位 Windows 2.0 特征；RC_PALETTE：指定一个基于调色板的设备。

RC_SCALING：支持缩放；RC_STRETCHBLT：支持 StretchBlt 函数。

RC_STRETCHDIB：stretchDIBits 函数。

CURVECAPS：显示设备所支持的曲线性能，可以是下列值的某种组合。

CC_NONE：不支持绘制曲线；CC_CHORD：支持绘制弦；CC_CIRCLES：支持绘制圆。

CC_ELLIPSES：支持绘制椭圆；CC_INTERIORS：支持内部填充；CC_PIE：支持绘制扇形图。

CC_ROUNDRECT：支持绘制圆角矩形；CC_STYLED：支持绘制带风格的边界。

CC_WIDE：支持绘制宽的边界；CC_WIDESTYLED：支持绘制宽的、带风格的边界。

LINECAPS 设备所支持的画线性能，可以是下列值的某种组合：

LC_NONE：不支持绘制线段；LC_INTERIORS：支持内部填充；LC_MARKER：支持绘制标记符。

LC_POLYLINE：支持折线；LC_POLYMARKER：支持多种标记符；LC_STYLED：带风格的线段。

LC_WIDE：支持画宽线；LC_WIDESTYLED：支持宽的带风格的线段。

POLYGONALCAPS 设有所支持的多边形性能。可以是下列值的某种组合。

PC_NONE：不支持绘制多边形；PC_INTERIORS：支持内部填充；PC_POLYGON：支持绘制间隔式填充多边形。

PC_RECTANGLE：支持绘制矩形；PC_SCANLINE：支持绘制扫描线；PC_STYLED：支持绘制带风格的边界。

PC_WIDE：支持绘制宽边界；PC_WIDESTYLED：支持绘制宽的带风格的边界。

PC_WINDPOLYGON：支持绘制折线式填充多边形。

TEXTCAPS 设备所支持的文字性能，可以是下列值的某种组合：

TC_OP_CHARACTER：支持字符输出精度；TC_OP_STROKE：支持笔画输出精度。

TC_CP_STROKE：支持笔画剪切精度；TC_CR_90：支持字符作 90 度旋转；

TC_CR_ANY：支持字符作任意角度旋转；TC_SF_X_YINDEP：支持 x 和 y 方向的独立缩放。

TC_SA_DOUBLE：支持把字符放大一倍；TC_SA_INTEGER：支持整数倍缩放。

TC_SA_CONTIN：支持以任何倍数的严格缩放；TC_EA_DOUBLE：支持字符加重。

TC_IA_ABLE：支持斜字体；TC_UA_ABLE：支持下划线；TC_SO_ABLE：支持删除线。

TC_RA_ABLE：支持光栅字体；TC_VA_ABLE：支持矢量字体；TC_RESERVED：保留、必须为零。

TC_SCROLLBLT：不支持用位快传递来滚动，注意这可能事与愿违。

返回值：返回值指定所需项目的值。

注释：GetDeviceCaps 提供下列六个索引以代替打印机消失。

PHYSICALWIDTH GETPHYSPAGESIZE; PHYSICALHEIGHT GETPHYSPAGESIZE

PHYSICALOFFSETX GETPRINTINGOFFSET;

PHYSICALOFFSETY GETPHYSICALOFFSET;

SCALINGFACTORX GETSCALINGFACTOR;

SCALINGFACTORY GETSCALINGFACTOR;

Windows CE：Windows CE 不支持 nIndex 参数取下列值：

VREFRESH; DESKTOPHORZRES; DESKTOPVERTRES;BLTALIGNMENT

Windows CE 1.0 不支持 nIndex 参数取下列值：

PHYSICALWIDTH; PHYSICALHEIGHT; PHYSICALOFFSETX; PHYSICALOFFSETY

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.22 GetObject

函数功能：该函数得到指定图形对象的信息，根据图形对象，函数把填满的或结构，或表项（用于逻辑调色板）数目放入一个指定的缓冲区。

函数原型：int GetObject(HGDIOBJ hgdiobj, int cbBuffer, LPVOID lpvObject)；

参数：

hgdiobj：指向感兴趣的图形对象的句柄，它可以是这样的一个句柄：一个逻辑位图、一个刷子、一种字体、一个调色板、笔或通过调用 CreateDIBsection 函数创建的与设备无关位图。

cbBuffer：指定将要写到缓冲区的信息的字节数目。

lpvObject：指向一个缓冲区的指针，该缓冲区将要检索指定图形对象的信息。

下面列出的是缓冲区为每种图形对象类型可接收的信息和类型，可用 hgdiobj 来指定，写入*lpvObject:HBITMAP BITMAP。

HBITMAP：如果 cbBjffer 被设置为 sizeof(DIBSECTION)或 sizeof(BITMAP)，则从对 GreatDIBSection 函数的 DIBSECTION 调用中返回。

HPALETTE：逻辑调色板入口数的 WORD 数目。

HPEN：从对 ExtCreatePen 函数的 LXTLOGPEN 调用中返回。

HPENLOGPEN; HBRUSH LOGBRUSH; HFONT LOGFONT

如果 lpvObject 参数为 Null，则函数返回值为指定图形对象需要把信息贮存到缓冲区的字节数目。

返回值：如果函数调用成功，且 lpvObject 为一个有效指针，则返回值为贮存到缓冲区的字节数目；如果函数调用成功，且 lprObject 为 Null，则返回值为需要容纳的贮存到缓冲区的信息字节数目；如果函数调用失败，则返回值为 0。

Windows NT：若想获得更多错误信息，可调用 GetLastError 函数。

注释：lpvObject 参数指向的缓冲区一定要足够大以接收图形对象的信息。

如果 hgdiobj 标识一个由调用 GreateDIBSection 创建的位图，且指定的缓冲区足够大，则 GetObject 函数返回一个 DIBSECTION 结构。另外，DIBSECTION 中的 BITMAP 结构中的 bmBits 元素含有一个指向位图位值的指针。

如果 hgdiobj 标识了一个通过其他途径创建的位图，则 GetObject 只返回位图的宽、高和颜色格式信息，通过调用 GetDIBits 或 GetBitmapBits 函数可以得到位置的位值。

如果 hgdiobj 标识了一个逻辑调色板，则 GetObject 检索一个 2 字节的整数，该整数指定调色板中的项数，函数不检索定义调色板的 LOGPALETTE 结构，为检索有关调色板项的信息，应用程序可以调用 GetPaletteEntries 函数。

Windows CE：在 Windows CE 1.0 中，当用在 DIB 上中，GetObject 总返回一个 BITMAP。Windows CE 1.0 不支持 lpvObject 参数的 HPALETTE 值。此函数在 Windows CE 2.0 与在 Windows 桌面上一样。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.23 GetObjectType

函数功能：该函数确定指定对象的类型。

函数原型：DWORD GetObjectType(HGDIOBJ h)；

参数：

h：图形对象的句柄。

返回值：如果成功，返回值标识该对象，它可取如下值：

OBJ_BITMAP：位图（Bitmap）；OBJ_BRUSH：画笔（Brush）；OBJ_FONT：字体（Font）；OBJ_PAL：调色板（palette）；

OBJ_EXTPEN：扩展笔（Extendedpen）；OBJ_REGION：区域（Region）；OBJ_DC：设备上下文环境（Devicecontext）；

OBJ_MEMDC：存设备上下文环境；OBJ_METAFILE：图元文件；OBJ_ENHMETAFILE：增强图元文件；

OBJ_ENHMETADC：增强图元文件设备上下文环境；

如果失败，返回值为零，若想获得更多错误信息，请调用 GetLastError 函数。

Windows CE：Windows CE 不支持下列**返回值：**

OBJ_EXTPEN；OBJ_METADC；OBJ_METAFILE；OBJ_ENHMETAFILE；OBJ-ENHMETADC

Windows CE 1.0 版不支持 OBJ_PAL 返回值。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.24 GetStockObject

函数功能：该函数检索预定义的备用笔、刷子、字体或者调色板的句柄。

函数原型：HGDIOBJ GetStockObject(int fnObject)；

参数：

fnObject：指定对象的类型，该参数可取如下值之一；

BLACK_BRUSH：黑色画笔；DKGRAY_BRUSH：暗灰色画笔；DC_BRUSH：在 Windows98,Windows NT 5.0 和以后版本中为纯颜色画笔，缺省色为白色，可以用 SetDCBrushColor 函数改变颜色，更多的信息参见以下的注释部分。GRAY_BRUSH：灰色画笔；HOLLOW_BRUSH：空画笔（相当于 HOLLOW_BRUSH）；

LTGRAY_BRUSH：亮灰色画笔；NULL_BRUSH：空画笔（相当于 HOLLOW_BRUSH）；

WHITE_BRUSH：白色画笔；BLACK_PEN：黑色钢笔；

DC_PEN：在 Windows98、Windows NT 5.0 和以后版本中为纯色钢笔，缺省色为白色，使用 SetDCPenColor 函数可以改变色彩，更多的信息，参见下面的注释部分。

WHITE_PEN：白色钢笔；ANSI_FIXED_FONT：在 Windows 中为固定间距（等宽）系统字体；

ANSI_VAR_FONT：在 Windows 中为变间距（比例间距）系统字体；

DEVICE_DEFAUCT_FONT：在 WindowsNT 中为设备相关字体；

DEFAULT_GUI_FONT：用户界面对象缺省字体，如菜单和对话框；

OEM_FIXED_FONT：原始设备制造商（OEM）相关固定间距（等宽）字体；

SYSTEM_FONT：系统字体，在缺省情况下，系统使用系统字体绘制菜单，对话框控制和文本；

SYSTEM_FIXED_FONT：固定间距（等宽）系统字体，该对象仅提供给兼容 16 位 Windows 版本；

DEFAULT_PALETTE：缺省调色板，该调色板由系统调色板中的静态色彩组成。

返回值：如果成功，返回值标识声请的逻辑对象，如果失败，返回值为 NULL。

WindowsNT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：仅在 CS_HREDRAW 和 CS_UREDRAW 风格的窗口中使用 DKGRAY_BRUSH、GRAY_BRUSH 和 LTGRAY_BRUSH 对象。

如果在其他风格的窗口中使灰色画笔，可能导致在窗口移动或改变大小之后出现画笔模式错位现象，原始储存画笔不能被调整。

HOLLOW_BRUSH 和 NULL_BRUSH 储存对象相等。

由 DEFAULT_GUI_FONT 储存对象使用的字体将改变。当想使用菜单、对话框和其他用户界面对象使用的字体时请使用此储存对象。

不必要通过调用 DeleteObject 函数来删除储存对象。

Windows 98、Windows NT 5.0 和以后版本：DC_BRUSH 和 DC_PEN 都能与其他储存对象如 BLACK_BRUSH 和 BLACK_PEN 相互交换关于检索当前钢笔和画笔颜色的信息,请参见 GetDCBrushColor 和 GetDCPencolor，带 DC BRUSH 或 DC PEN 参数的 Getstockobject 函数可以与 SetDCPenColor 和 SetDCBrushColor 函数相互交换使用。

Windows CE：Windows CE 不支持 fnObject 参数的如下值：

ANSI_FIXED_FONT、ANSI_VAR_FONT、OEM_FIXED_FONT、SYSTEM_FIXED_FONT

Windows CE1.0 版本不支持 fnObject 的 DEFAULT_PALETTE 值。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；头文件：gdi32.lib。

3.6.25 ReleaseDC

函数功能：函数释放设备上下文环境（DC）供其他应用程序使用。函数的效果与设备上下文环境类型有关。它只释放公用的和设备上下文环境，对于类或私有的则无数。

函数原型：int ReleaseDC(HWND hWnd, HDC hdc)；

参数：

hWnd：指向要释放的设备上下文环境所在的窗口的句柄。

hDC：指向要释放的设备上下文环境的句柄。

返回值：返回值说明了设备上下文环境是否释放；如果释放成功，则返回值为 1；如果没有释放成功，则返回值为 0。

注释:每次调用 GetWindowDC 和 GetDC 函数检索公用设备上下文环境之后,应用程序必须调用 ReleaseDC 函数来释放设备上下文环境。

应用程序不能调用 ReleaseDC 函数来释放由 CreateDC 函数创建的设备上下文环境,只能使用 DeleteDC 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h；库文件：user32.lib。

3.6.26 ResetDC

函数功能：该函数根据指定结构中的信息更新给定打印机或绘图仪的设备上下文环境。

函数原型：HDC ResetDC(HDC hdc, CONST DEVMODE *lpInitData)；

参数：

hdc：将要更新的设备上下文环境的句柄。

lpInitData：指向包含新设备上下文环境信息的 DEVMODE 结构的指针。

返回值：如果成功，返回值为原始设备上下文环境的句柄；如果失败，返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

注释：当一个窗口按收一个 WM_DEVMODECHANGE 消息时，应用程序一般将使用函数 ResetDC，ResetDC 函数也可以在打印文档的时候改变纸张定位和纸张接收器。ResetDC 函数不能用来改变驱动程序名、设备名或者输出端口。当用户改变通讯口连接或者改变设备名时，应用程序必须删除原始设备上下文环境，并根据新的信息创建一个新的设备上下文环境。应用程序可以把一信息设备上下文环境传递给 ResetDC 函数。这种情况下，ResetDC 通常会返回一个打印机设备上下文环境。

ICM：由 hdc 指定的设备上下文环境的颜色文件将根据 DEVMODE 结构 lpInitData 元素中的信息进行重新设置。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

3.6.27 RestoreDC

函数功能：该函数恢复设备上下文环境（DC）到指定状态，该设备上下文环境的恢复是通过使状态信息出栈而进行的。该堆栈由先前调用 SaveDC 函数时创建的。

函数原型：BOOL RestoreDC(HDC hdc, int nSavedDC)；

参数：

hdc：设备上下文环境句柄。

nSaveDC：指定将要被恢复的设备上下文环境的实例，如果该参数为正，则 nSaveDC 代表要恢复的设备上下文环境的一个指定实例。如果该参数为负，则 nSaveDC 代表与当前设备上下文环境有关的一个实例。

返回值：如果成功，返回非零；如果失败，返回零。

注释：堆栈可以包含设备上下文环境多个实例的状态信息，如果给定参数指定的状态不在堆栈的顶部，那么 RestoreDC 就删除栈顶和指定实例之间的所有状态信息。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.28 SaveDC

函数功能：该函数通过把数据描述选定对象和图形模式（如位图、画笔、调色板、字体、笔、区域、绘图模式、映射模式）拷贝到描述表推栈中为保存，指定设备上下文环境的当前状态。

函数原型：int SaveDC(HDC hdc)；

参数：

hdc：要保存的设备上下文环境的句柄。

返回值：如果成功，返回值标识保存的设备上下文环境，如果失败，返回零。

注释：SaveDc 函数可以用来保存设备上下文环境状态的任何数目的实例和保存任何次数。

一个被保存的状态以后可以用 RestoreDc 函数进行恢复。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.29 SelectObject

函数功能：该函数选择一对象到指定的设备上下文环境中，该新对象替换先前的相同类型的对象。

函数原型：HGDIOBJ SelectObject(HDC hdc, HGDIOBJ hgdiobj)；

参数：

hdc：设备上下文环境的句柄。

hgdiobj：被选择的对象的句型，该指定对象必须由如下的函数创建。

位图：CreateBitmap, CreateBitmapIndirect, CreateCompatible Bitmap, CreateDIBitmap, CreateDIBsection（只有内存设备上下文环境可选择位图，并且在同一时刻只能一个设备上下文环境选择位图）。

画笔：CreateBrushIndirect, CreateDIBPatternBrush, CreateDIBPatternBrushPt,

CreateHatchBrush, CreatePatternBrush, CreateSolidBrush。

字体：CreateFont, CreateFontIndirect。

笔：CreatePen, CreatePenIndirect。

区域：CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreateRectRgn, CreateRectRgnIndirect。

返回值：如果选择对象不是区域并且函数执行成功，那么返回值是被取代的对象的句柄；如果选择对象是区域并且函数执行成功，返回如下一值；

SIMPLEREGION：区域由单个矩形组成；COMPLEXREGION：区域由多个矩形组成。NULLREGION：区域为空。

如果发生错误并且选择对象不是一个区域，那么返回值为 NULL，否则返回 GDI_ERROR。

注释：该函数返回先前指定类型的选择对象，一个应用程序在它使用新对象进行绘制完成之后，应该用新对象替换原始的缺省的对象。

应用程序不能同时选择一个位图到多个设备上下文环境中。

ICM：如果被选择的对象是画笔或笔，那么就执行颜色管理。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.6.30 SetDCBrushColor

函数功能：该函数把当前设备上下文环境（DC）画笔颜色设置为指定颜色值。如果设备不能提供指定的颜色值，那么就把颜色设为最近的物理颜色。

函数原型：COLORREF SetDCBrushColor(HDC hdc, COLORREF crColor);

参数：

hdc：设备上下文环境句柄。

crColor：指定新的画笔颜色。

返回值：如果成功，返回值指定先前设备上下文环境画笔颜色为 COLORREF 值；如果失败，返回 CLR_INVALID。

注释：当在 一个设备上下文环境中选择系统备用 DC_BRUSH 时，所有的以后绘制操作将使用设备上下文环境画笔颜色，直到系统备用画笔被取消选择为止，缺省 DC_BRUSH 颜色是 WHITE。

该函数将返回先前的 DC_BRUSH 颜色，即使在 DC 中没有选择系统备用画笔 DC_BRUSH，但是，除非系统备用 DC-BRUSH 被选择，否则绘制操作将不使用它，设置颜色可参照设置笔和画笔颜色的例子。

使用 DC_BRUSH 或 DC_PEN 参数的 GetStockObject 函数可以与 SetDCPenColor 和 SetDCBrushColor 函数互换。

ICM：如果 ICM 可用就执行颜色管理。

速查：Windows NT：5.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.6.31 SetDCPenColor

函数功能：该函数把当前设备上下文环境（DC）的笔颜色设置为指定颜色值。如果设备不能提供指定的颜色值，颜色就设为最近的物理颜色。

函数原型：COLORREF SetDCPenColor(HDC hdc, COLORREF crColor);

参数：

hdc：设备上下文环境的句柄。

crColor：指定新笔颜色。

返回值：如果成功，则返回值指定先前 DC 笔颜色为某个 COLORREF 值；如果失败，返回 CLR_INVALID。

注释：即使系统备用笔 DC_PEN 没有被选入 DC，该函数仍将返回先前 DC_PEN 的颜色，但是在绘制操作中这将不被使用，直到系统备用 DC_PEN 被选入 DC 为止。

设置颜色可以参见设置笔或画笔颜色的例子，使用 DC_BRUSH 或 DC_PEN 参数的 GetStockObject 函数可与 SetDCPenColor 和 SetDCBrushColor 函数交换使用。

ICM：如果 ICM 可用就执行颜色管理。详情参见 ForegroundldleProc 钩子处理过程。

速查：Windows NT：5.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.7 填充图形函数（Filled Shape）

填充图形是一些几何图形，其轮廓由当前的画笔绘制，内部由当前的笔刷填充。共有 5 种填充图形：椭圆，弦图，饼图，多边形，矩形。填充图形函数用于对填充图形进行操作。

3.7.1Chord

函数功能：该函数画一段圆弧，圆弧是由一个椭圆和一条线段（称之为割线）相交限定的闭合区域。此弧由当前的画笔画轮廓，由当前的画刷填充。

函数原型：BOOL Chord(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect, int nXRadial1, int nYRadial1, int nXRadial2, int nYRadial2)；

参数：

hdc：设备环境的句柄，圆弧出现在其中。

nLeftRect：指定边界矩形左上角的 X 坐标。

nTopRect：指定边界矩形左上角的 Y 坐标。

nRightRect：指定边界矩形右上角的 X 坐标。

nBottomRect：指定边界矩形右上角的 Y 坐标。

nXRadial1：指定定义圆弧的起始 R 的射线端点的 X 坐标。

nYRadial1：指定定义圆弧的起始半径端点的 Y 坐标。

nXRadial2：指定定义圆弧的终止处的半径端点的 X 坐标。

nYRadial2：指定定义圆弧的终止处的半径端点的 Y 坐标。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值为 0。

Windows NT：若想获得更多更多的错误信息，请调用 GetLastError 函数。

备注：圆弧的曲线是符合边界矩形的椭圆定义的。曲线开始于椭圆与第一条半径的交点，以逆时针方向延伸到椭圆与第二条半径的交点处。（一条半径是指从椭圆的中心到椭圆上指定端点之间的线段），圆弧被从第一条半径的端点到第二条半径的端点的一条线段和曲线封闭起来。若起始点和终止点生命，则画整个椭圆。

当前位置既不会被圆弧使用也不会被圆弧修改。

Windows 95 和 Windows 98：边界矩形的坐标值之和不能超过 32767。nLeftRect 和 nRightRect 或 nTopRect 和 nBottomRect 之和不能不能超过 32767。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.7.2Ellipse

函数功能：该函数用于画一个椭圆，椭圆的中心是限定矩形的中心，使用当前画笔画椭圆，用当前的画刷填充椭圆。

函数原型：BOOL Ellipse(HDC hdc, int nLeftRect, int nTopRect, nRightRect, int nBottomRect)；

参数：

hdc：设备环境句柄。

nLeftRect：指定限定矩形左上角的 X 坐标。

nTopRect：指定限定矩形左上角的 Y 坐标。

nRightRect：指定限定矩形右上角的 X 坐标。

nBottomRect：指定限定矩形右上角的 Y 坐标。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：椭圆既不使用也不改变当前位置。

Windows 95 和 Windows 98：限定矩形的坐标值之和不能超过 32767。nLeftRect 与 nRigthRect 或 nTopRectn 与 BottomRect 之和不能超过 32767。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.7.3FillRect

函数功能：该函数用指定的画刷填充矩形，此函数包括矩形的左上边界，但不包括矩形的右下边界。

函数原型：int FillRect(HDC hdc, CONST RECT *lprc, HBRUSH hbr)；

参数：

hdc：设备环境句柄。

lprc：指向含有将填充矩形的逻辑坐标的 RECT 结构的指针。

hbr：用来填充矩形的画刷的句柄。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：由参数 hbr 定义的画刷可以是一个逻辑现刷句柄也可以是一个颜色值，如果指定一个逻辑画刷的句柄，调用下列函数之一来获得句柄；CreateHatchBrush、CreatePatternBrush 或 CreateSolidBrush。此外，你可以用 GetStockObject 来获得一个库存画刷句柄。如果指定一个颜色值，必须是标准系统颜色（所选择的颜色必须加 1）如 FillRect(hdc, &rect, (HBRUSH)(COLOR_ENDCOLORS+1))，参见 GetSysColor 可得到所有标准系统颜色列表。

当填充一个指定矩形时，FillRect 不包括矩形的右、下边界。无论当前映射模式如何，GDI 填充一个矩形都不包括右边的列和下面的行。

Windows CE：在 Windows CE1.0 版中，参数 hbr 不能是一个彩色画刷。在 Windows CE2.0 版中，此函数和 Windows 桌面平台上的相同。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.7.4frameRect

函数功能： 该函数用指定的画刷为指定的矩形画边框。边框的宽和高总是一个逻辑单元。

函数原型： int frameRect(HDC hdc, CONST RECT *lprc, HBRUSH hbr)；

参数：

hdc：将要画边框的设备环境句柄。

lprc：指向包含矩形左上角和右上角逻辑坐标的结构 RECT 的指针。

hbr：用于画边框的画刷句柄。

返回值： 如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：由参数 hbr 定义的画刷必须是由 CreateHatchBrush、CreatePatternBrush 或 CreateSolidBrush 创建的，或者是由使用 GetStockObject 获得的。

如果 RECT 结构中的底部成员的值少于或等于顶部成员，或右部成员少于或等于左部成员，此函数画不了矩形。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.7.5Invertrect

函数功能： 该函数通过对矩形内部的像素点进行逻辑 NOT 操作而将窗口中的矩形反转。

函数原型： BOOL Invertrect(HDC hdc, CONST RECT *lprc)；

参数：

hdc：设备环境句柄。

lprc：指向包含将被反转的矩形的逻辑坐标的 RECT 结构的指针。

返回值： 如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 在单色屏幕上，InvertRect 将白色像素变为黑色，将黑色像素变为白色，在彩色屏幕，反转依赖于颜色是如何从屏幕上产生的。对同一个矩形调用两次 InvertRect 将使显示恢复为以前的色彩。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.7.6Pie

函数功能： 该函数画一个由椭圆和两条半径相交闭合而成的饼状楔形图，此饼图由当前画笔画轮廓，由当前画刷填充。

函数原型： BOOL Pie(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect, int nXRadial1, int nYRadial1, int nXRadial2, int nYRadial2)；

参数：

hdc：设备环境句柄。

nLeftRect：指定限定矩形左上角的 X 坐标。

nTopRect：指定限定矩形左上角的 Y 坐标。

nRigthRect：指定限定矩形右下角的 X 坐标。

nBottomRect：指定限定矩形右下角的 Y 坐标。

nXRadial1：指定第一条半径的端点的 X 坐标。

nYRadial1：指定第一条半径的端点的 Y 坐标。

nXRadial2：指定第二条半径的端点的 X 坐标。

nYRadial2：指定第二条半径的端点的 Y 坐标。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows：要得到更多的错误信息，调用 GetLastError。

备注：饼图的曲线是由符合限定矩形的椭圆定义的。曲线的起始点在椭圆与第一条半径的交点处，然后沿逆时针方向延伸，直到椭圆与第二条半径的交点处（一条半径是指从椭圆中心到椭圆上指定点之间的线段）。

函数 Pie 不使用和改变当前的位置。

Windows 95 和 Windows 98：限定矩形的坐标之和不能超过 32767。nLeftRect 和 nRightRect 之和或 nTopRect 和 nBottomRect 之和均不能超过 32767。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.7.7Polygon

函数功能：该函数画一个由直线相闻的两个以上顶点组成的多边形，用当前画笔画多边形轮廓，用当前画刷和多边形填充模式填充多边形。

函数原型：BOOL Polygon(HDC hdc, CONST POINT *lpPoints, int nCount)；

参数：

hdc：设备环境句柄。

lpPoints：指向用于指定多边形顶点的 POINT 结构数组的指针。

nCount：指定数组中顶点个数，此值必须大于等于 2。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：此多边形通过画一条从最后一个顶点到第一个顶点的线段而自动闭合起来。函数 Polygon 不使用和改变当前位置。

Windows CE：1.0 版本只支持凸多边形。在 Windows CE 2.0 版本中，此函数等同于在 Windows 桌面平台上的函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.7.8PolyPolygon

函数功能：该函数画一系列的多边形，每一个多边形都用当前的画笔画轮廓，用当前的画刷和多边形填充模式画填充。此函数画的多边形可以重叠。

函数原型：BOOL PolyPolygon(HDC hdc, CONST POINT *lpPoints, CONST INT *lpPolyCounts, int nCount)；

参数：

hdc：设备环境句柄。

lpPoints：指向定义多边形顶点的 POINT 结构数组的指针，各多边形是连续定义的，每个多边形通过

画一条从最后中一个顶点到第一个顶点的线段而自动闭合起来，每个顶点应被定义一次。

lpPolyCounts：指向整数数组的指针，每个整数指定相应多边表的点数，每个整数必须大于等于 2。

nCount：指定多边形的总个数。

返回值：如果函数调用成功，返回值非零，否则返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：此函数不使用和修改当前位置。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.7.9Rectangle

函数功能：该函数画一个矩形，用当前的画笔画矩形轮廓，用当前画刷进行填充。

函数原型：BOOL Rectangle(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect);

参数：

hdc：设备环境句柄。

nLeftRect：指定矩形左上角的逻辑 X 坐标。

nTopRect：指定矩形左上角的逻辑 Y 坐标。

nRightRect：指定矩形右上角的逻辑 X 坐标。

nBottomRect：指定矩形右上角的逻辑 Y 坐标。

返回值：如果函数调用成功，返回值非零，否则返回值为 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：此函数不使用和改变当前位置。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.7.10 RoundRect

函数功能：该函数画一个带圆角的矩形，此矩形由当前画笔画轮廓，由当前画刷填充。

函数原型：BOOL RoundRect(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect, int nWidth, int nHeight);

参数：

hdc：设备环境句柄。

nLeftRect：指定矩形左上角的 X 坐标。

nTopRect：指定矩形左上角的 Y 坐标。

nRightRect：指定矩形右上角的 X 坐标。

nbottomRect：指定矩形右上角的 Y 坐标。

nWidth：指定用来画圆角的椭圆的宽。

nHeight：指定用来画圆角的椭圆的高。

返回值：如果函数调用成功，则返回值非空，否则返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：此函数不使用和改变当前位置。

Windows 95 和 Windows 97：限定矩形的坐标之和不能超过 32767。nLeftRect 和 nRightRect 之和或

nTopRect 和 nBottomRect 之和均不能超过 32767。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.8 字体和文本函数（Font and Text）

字体用于在视频显示器或其他输出设备上绘制文本。Win32 API 提供了一系列用于安装、选择和查询各种字体的字体和文本函数。

3.8.1AddFontResource

函数功能： 该函数从指定的文件里增加字体资源到系统字体表，这些字体可随后被任何基于 Win32 的应用程序用来作正文输出。

函数原型： int AddFontResource(LPCTSTR lpszFilename)；

参数：

lpszFilename：指向含有一个有效的字体文件的文件名，它是以\0 结束的字符串的指针，此文件名可以指定一个.FON 字体资源文件、一个.FNT 未加工位图字体文件、一个.TTF 未加工 TrueType 文件或一个.FON TrueType 资源文件。

返回值： 如果函数调用成功，则返回值为增加的字体数；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 任何增加或删除系统字体表中字体的应用程序都必须以发 WM_FONTCHANGE 消息给操作系统中所有最顶层的窗口来通过其他窗口字体的改变，应用程序调用 SendMessage 和设置参数 hwnd 为 HWND_BROADCAST 来发送消息。

当一个应用程序不再需要一种由调用 AddFontResource 加载进来的字体资源，应该用 RemoveFontResource 来删除这种资源。

Windows CE：版本 1.0 仅支持光栅字体。Windows CE 2.0 版本支持使用 TrueType 字体或光栅字体其中之一的系统，字体类型（光栅或 TrueType）在系统设计时就已确定，不能被一个应用程序修改。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.2CreateFont

函数功能： 该函数创建一种有特殊性的逻辑字体，此逻辑字体可以在后面被任何设备选择。

函数原型： HFONT CreateFont(int nHeight, int nWidth, int nEscapement, int nOrientation, int fnWeight, DWORD fdwltalic, DWORD fdwUnderline, DWORD fdwStrikeOut, DWORD fdwCharSet, DWORD fdwOutputPrecision, DWORD fdwClipPrecision, DWORD fdwQuality, DWORD fdwPitchAndFamily, LPCTSTR lpszFace)；

参数：

nHeight：指定字体的字符单元或字符的逻辑单位高度，字符的高度值（也被称为 em 高度）是指字符单元高度值减去内部标头值。字体映射器以如下方式解释 nHeight 指定的值，各值含义为：

>0：字体映射器转换这个值以设备单位，并和已有字体的单元高度相匹配。

0：字体映射器转换在选择匹配时用一个缺省的高度值。

<0：字体映射器转换这个值到设备单位，并将它的绝对值和已有字体的字符高度相匹配。

比较所有的高度，字体映射器选择不超过要求大小的最大字体。

此映射当字体第一次被使用时发生。

对于 MM_TEXT 映射方式，可以用下面的公式为一种指定了点大小的字体确定高度：

nHeight=-MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY),72)

nWidth：指定所要求字体的字符的逻辑单位的平均宽度。如果此值为 0，字体映射器选择一个 closest match 值，closest match 值是由比较当前设备的特征系数与可使用字体的数字化特征系数之差的绝对值而确定的。

nEscapement：指定移位向量和设备 X 轴之间的一个角度，以十分之一度为单位。移位向量平行于正文行的基线。

Windows NT：当图形设备设置为 GM_ADVANCED 时，可以不依赖字符串的字符的定位角而指定字符串的移位角。

当图形模式被设置为 GM_COMPATIBLE 时，nEscapement 同时指定移位角和定位角，可以设置 nEscapement 和 nOrientation 为相同的值。

Windows 95：nEscapement 同时指定移位角和定位角，可设置 nEscapement 和 nOrientation 为相同的值。

nOrientation：指定每个字符的基线和设备 X 轴之间的角度。

FnWeight：在 0 到 1000 之间指定字体的权值，如 400 表示标准体，700 表示黑（粗）体，如果此值为 0，则使用缺省的权值。

为方便定义，可使用如下值：

FW_DONTCARE：0；FW_THIN；100；FW_EXTRALIGHT；200；FW_ULTRALIGHT；200；FW_LIGHT；300；

FW_NORMAL：400；FW_REGULAR；400；FW_MEDIUM；500；FW_SEMIBOLD；600；FW_DEMIBOLD；600；

FW_BOLD：700；FW_EXTRABOLD；800；FW_ULTRABOLD；800；FW_HEAVY；900；FW_BLACK；900。

fdwItalic：如果设置为 TRUE 则指定斜体。

fdwUnderline：如果设置为 TRUE，则指定加下划线的字全。

fdwStrikeOut：如果设置为 TRUE，则 strikeout 指定字体。

fdwCharSet：指定字符集，下列值是预定义的：

ANSI_CHARSET; BALTIC_CHARSET; CHINESEBIG5_CHARSET; DEFAULT_CHARSET;

EASTEUROPE_CHARSET; GB2312_CHARSET; GREEK_CHARSET; HANGUL_CHARSET; MAC_CHARSET; OEM_CHARSET; RUSSIAN_CHARSET; SHIFTJIS_CHARSET;

SYMBOL_CHARSET; TURKISH_CHARSET。

韩国 Windows：JOHAB_CHARSET;

中东地区 Windows：HEBREW_CHARSSET; ARABIC_CHARSET

泰国 Windows：THAI_CHARSET

OEM_CHARSET 指定的字符集与操作系统有关。

可以使用 DEFAULT_CHARSET 值来允许字体的名字和大小来充分描述逻辑字体。如果指定的字体名不存在，任何字符集的字体都可以替代指定的字体，所以应该小心地用 DEFAULT_CHARSET 来避免不期望的结果出现。

操作系统中存在其他字符集的字体。如果一个应用程序用一种未知字符集的字体，则应用程序不会试图去翻译或解释用那种字体写出来的字符串。

在字体映射过程中此参数很重要。为确保获得一致的结果，指定一个特殊的字符集。如果在 lpszFace 参数中指定了一个字体名，确定 fdwCharSet 值与由 lpszFace 指定的字体字符集是否匹配。

fdwOutputPrecision：指定输出精度，输出精度义输出与要求的字体高度、宽度、字符定位、移位、字符间距和字符类型的匹配程序，它可取下列值之一：

OUT_CHARACTER_PRECIS；未用。

OUT_DEFAULT_PRECIS：指定缺省的字体映射器状态。

OUT_DEVICE_PRECIS：指示字体映射器在当系统里有多种字体使用同一个字体使用同一个名字时选择一种设备字体。

OUT_OUTLINE_PRCIS：在 Windows NT 中此值指示字体映射器从 TrueType 和其他基于边框的字体中选择。

OUT_RASTER_PRECIS：指示字体映射器在当系统里有多种字体使用同一个名字时选择一种光栅字体。

OUT_STRING_PRECIS：此值没有被字全映射器使用，但是当扫描字体被列举时作为返回值。

OUT_STROKE_PRECIS：在 Windows NT 中此值没有被字体映射器使用，但是当 TrueType 字体、其他基于边框的字体和向量字体被列举时，作为返回值。

Windows 95：此值没有被字体映射器使用，但是当 TrueType 字体或向量字体被列举时，作为返回值。

OUT_TT_ONLY_PRECIS：指示字体映射器仅从 TrueType 字体中选择，如果系统中没有安装 TrueType 字体，则字体映射返回缺省状态。、

OUT_TT_PRECIS：指示字体映射器在当系统里有多种同名的字体时选择一种 TrueType 字体。

当操作系统含有多种与指定名字同名的字体时，应用程序可以使用 OUT_DEVICE_PRECIS,OUT_RASTER_PRECIS 和 OUT_TT_PRECIS 值来控制字体映射器如何选择一种字体，例如，如果操作系统含有名字 Symbol 的光栅和 TrueType 两种字体，指定 OUT_TT_PRECIS 使字体映射器选择 TrueType 方式。指定 OUT_TT_ONLY_PRECIS 使字体映射器选择一种 TrueType 字体，尽管这会给 TrueType 字体换一个名字。

fdwClipPrecision；指定裁剪精度，裁剪精度定义如何裁剪部分超出裁剪区的字符，它可取一个或多个下列值：

CLIP_DEFAULT_PRECIS：指定缺省裁剪状态。CLIP_CHARACTER_PRECIS：未用。

CLIP_STROKE_PRECIS：未被字体映射器使用，但是当光栅字体、向量字体或 TrueType 字体被列举时作为返回值。在 Windows 环境下，为保证兼容性，当列举字体时这个值总被返回。

CLIP_MASK：未用。CLIP_EMBEDDED：要使用嵌入式只读字体必须使用此标志。

CLIP_LH_ANGLES：当此值被使用时，所有字体的旋转依赖于坐标系统的定位是朝左的还是朝右的。

如果未使用此值，设备字体总是逆时针方向旋转，但其他字体的旋转依赖于坐标系统的定向。要得到更多关于坐标系统定向的信息，参见参数 orientation。

CLIP_TT_ALWAYS：未用。

fdwQuality：指向输出质量，输出质量定义 GDI 如何仔细地将逻辑字体属性与实际物理字体属性相匹配。它可取下列值之一：

DEFAULT_QUALITY：字体的外观不重要。

DRAFT_QUALITY：字体外观的重要性次于使用 PROOF_QUALITY 时，对 GDI 光栅字体，缩放比例是活动的，这意味着多种字体大小可供选择，但质量可能不高，如果有必要，粗体、斜体、下划线、strikeout 字体可被综合起来使用。

PROOF_QUALITY：字符质量比精确匹配逻辑字体字体属性更重要。对 GDI 扫描字体，缩放比例是活动的，并选择最接近的大小。尽管当使用 PROOF_QUALITY 时，选择字体大小并不完全匹配，但字体的质量很高，并没有外观上的变形。如果有必要，粗体、斜体、下划线、strikeout 字体可被综合起来使用。

fdwPitchAndFamily：指定字体间距和字体族，低端二位指定字体的字符间距，它可取下列值之一：

DEFAULT_PITCH；FIXED_PITCH；VARIABLE_PITCH

高端四位指定字体族，可取下列值之一：

FF_DECORATIVE：新奇的字体，如老式英语（Old English）。FF_DONTCARE：不关心或不知道。

FF_MDERN：笔划宽度固定的字体，有或者无衬线。如 Pica、Elite 和 Courier New。

FF_ROMAN：笔划宽度变动的字体，有衬线。如 MS Serif。

FF_SCRIPT：设计成看上去象手写体的字体。如 Script 和 Cursive。

FF_SWISS：笔划宽度变动的字体，无斜线。如 MS Sans Serif。

应用程序可以用运算符 OR 将字符间距和字体族组合起来给 fdwPitchAndFamily 赋值。

字体族描述一种字体的普通外观，当所有的精确字样都不能使用时，可用它们来指定字体。

lpszface：指向指定字体的字样名的、以\0 结束的字符串指针，字符串的长度不能超过 32 个字符（包括字符\0），函数 EnumFontFamilies 可用来列举所有当前可用字体的字样名。

如果 lpszFace 为 NULL 或指向一个空串，GDI 使用能匹配其他属性的第一种字体。

返回值： 如果函数调用成功，返回值是一种逻辑字体句柄；如果函数调用失败，返回值为 NULL。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 当一种字体不再使用时，可用 DeleteObject 来删除。

为保护那些提供字体给 Windows 和 Windows NT 的卖主的版权，基于 Win32 的应用程序总是列出所选择字体的准确名字。由于不同的系统会使用不同的字体，不要认为所选择字体就是要求的字体。例如，如果要求名叫 Palatino 的字体，但系统没提供那样一种字体，则字体映射器将会以一种不同名但有相似属性的字体取而代之。系统总是将用户选择的字体名报告出来。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.3CreateFontIndirect

函数功能： 该函数创建一种在指定结构定义其特性的逻辑字体。这种字体可在后面的应用中被任何设备环境选作字体。

函数原型： HFONT CreateFontIndirect(CONST LOGFONT *lplf)；

参数：

lplf：指向定义此逻辑字体特性的 LOGFONT 结构的指针。

返回值： 如果函数调用成功，返回值是逻辑字体的句柄；如果函数调用失败，返回值是 NULL。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 函数 CreateFontIndirect 创建一种在结构 LOGFONT 中定义特性的逻辑字体。当这种字体被函数选择时，GDI 的字体映射器会努力将此逻辑字体与现有物理字体相匹配，如果不能找到精确匹配，将会提供另一种选择，其特性与所要求的特性尽可能地匹配。

当一种字体不再需要进，可调用 DeleteObject 删除它。

Windows CE：1.0 版本只支持光栅字体。Windows CE 版本 2.0 支持使用 TrueType 字体和光栅字体其中之一的系统。字体类型（光栅或 TrueType）是在系统设计时就已选择，不能被应用程序改变。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.4CreateScalablefontResource

函数功能： 该函数为一种可升级的字体创建一个字体资源文件。

函数原型： BOOL CreateScalablefontResource(DWORD fdwHidden, LPCTSTR lpszFontRes, LPCTSTR lpszFontFile, LPCTSTR lpszCurrentPath)；

参数：

fdwHidden：指定此字体是否是一种嵌入式只读字体，此参数可取下列值之一：

0：此字体有读写权限。

1：此字体有只读权限，并且必须对系统的其他应用程序隐藏起来，当此值设置时，此字体不能被函数

EnumFonts 和 EnumFontFamilies 列举出来。

lpszFontRes：指向指定此函数创建的字体资源文件名字的，以\0 结束的字符串的指针。

lpszfontFile：指向以\0 结束的字符串的指针，该字符串指定用于创建字体资源文件的可升级字全文件的名字。

lpszCurrentPath：指向指定可升级字体文件路径的，以\0 结束的字符串的指针。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：函数 CreateScalableFontResource 被那些安装 TrueType 字体的应用程序使用。一个应用程序可用 CreateScalableFontResource 来创建一个字体资源文件（通常有.FOT 扩展名），然后用函数 AddFontResource 安装字体。TrueTpe 字体文件（通常有.TTF 扩展名）必须在 WINDOWS 目录下的 SYSTEM 子目录中，被函数 AddFontResource 使用。

CreateScalableFontResource 目前只支持 TrueType 技术升级字体。

当参数 lpszFontFile 只指定一个文件名和扩展名时，lpszCurrentPath 必须指定一个路径。

当参数 lpszFontFile 指定了一个完整的路径，则 lpszCurrentPath 必须为 NULL，或者为指向 NULL 的指针。

当参数 lpszFontFile 指定了一个文件名和扩展名，且 lpszCurrentPath 中指定路径时，在 lpszFontFile 中的字符串被拷贝到.FOT 文件，象属于此资源的.TTF 一样。当 AddFontResource 被调用时，操作系统假定.TTF 文件被拷贝到 SYSTEM 目录下（或者有网络安装的情况下到主 Windows 目录下）。当 CreateScalableFontResource 被调用时，.TTF 文件不必在这个目录下，因为 lpszCurrentPath 包含了目录信息，在这种情况下创建的资源不包含绝对路径，可以被任意安装使用。

当 lpszFontFile 指定了路径，且 lpszCurrentPath 被指定为 NULL，则在 lpszFontFile 中的字符串被拷贝到.FOT 文件，在这种情况下，当函数 AddFontResource 被调用时，.TTF 文件必须在参数 lpszFontFile 指定的位置处，当 CreateScalableFontResource 调用时，lpszCurrentPath 参数不需要，这种情况下创建的资源含有对路径和驱动器的绝对访问，当.TTF 被移到不同的位置时，就不能使用。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.5DrawText

函数功能：该函数在指定的矩形里写入格式化文本，根据指定的方法对文本格式化（扩展的制表符，字符对齐、折行等）。

函数原型：int DrawText(HDC, hdc, LPCTSTR lpString, int nCount, LPREST lpRect, UINT uFormat)；

参数：

hdc：设备环境句柄。

lpString：指向将被写入的字符串的指针，如果参数 nCount 是 C1，则字符串必须是以\0 结束的。

如果 uFormat 包含 DT_MODIFYSTRING，则函数可为此字符串增加 4 个字符，存放字符串的缓冲区必须足够大，能容纳附加的字符。

nCount：指向字符串中的字符数。如果 nCount 为 C1，则 lpString 指向的字符串被认为是以\0 结束的，DrawText 会自动计算字符数。

lpRect：指向结构 RECT 的指针，其中包含正文将被置于其中的矩形的信息（按逻辑坐标）。

uFormat：指定格式化正文的方法。它可以下列值的任意组合，各值描述如下：

DT_BOTTOM：将正文调整到矩形底部。此值必须和 DT_SINGLELINE 组合。

DT_CALCRECT：决定矩形的宽和高。如果正文有多行，DrawText 使用 lpRect 定义的矩形的宽度，并扩展矩形的底训以容纳正文的最后一行，如果正文只有一行，则 DrawText 改变矩形的右边界，以容纳下正文

行的最后一个字符，上述任何一种情况，DrawText 返回格式化正文的高度而不是写正文。

DT_CENTER：使正文在矩形中水平居中。

DT_EDITCONTROL：复制多行编辑控制的正文显示特性，特殊地，为编辑控制的平均字符宽度是以同样的方法计算的，此函数不显示只是部分可见的最后一行。

DT_END_ELLIPSIS 或 DT_PATH_ELLIPSIS：可以指定 DT_END_ELLIPSIS 来替换在字符串末尾的字符，或指定 DT_PATH_ELLIPSIS 来替换字符串中间的字符。如果字符串里含有反斜扛，DT_PATH_ELLIPSIS 尽可能地保留最后一个反斜杠之后的正文。

DT_EXPANDTABS：扩展制表符，每个制表符的缺省字符数是 8。

DT_EXTERNALLEADING：在行的高度里包含字体的外部标头，通常，外部标头不被包含在正文行的高度里。

DT_INTERNAL：用系统字体来计算正文度量。

DT_LEFT：正文左对齐。

DT_MODIFYSTRING：修改给定的字符串来匹配显示的正文，此标志必须和 DT_END_ELLIPSIS 或 DT_PATH_ELLIPSIS 同时使用。

DT_NOCLIP：无裁剪绘制当 DT_NOCLIP 使用时 DrawText 的使用会有所加快。

DT_NOPREFIX：关闭前缀字符的处理，通常 DrawText 解释助记前缀字符，&为给其后的字符加下划线，解释&&为显示单个&。指定 DT_NOPREFIX，这种处理被关闭。

DT_RIGHT：正文右对齐。

DT_RTLREADING：当选择进设备环境的字体是 Hebrew 或 Arabicf 时，为双向正文安排从右到左的阅读顺序都是从左到右的。

DT_SINGLELINE：显示正文的同一行，回车和换行符都不能折行。

DT_TABSTOP：设置制表，参数 uFormat 的 15″C8 位（低位字中的高位字节）指定每个制表符的字符数，每个制表符的缺省字符数是 8。

DT_TOP：正文顶端对齐（仅对单行）。DT_VCENTER：正文水平居中（仅对单行）。

DT_WORDBREAK：断开字。当一行中的字符将会延伸到由 lpRect 指定的矩形的边框时，此行自动地在字之间断开。一个回车一换行也能使行折断。

DT_WORD_ELLIPSIS：截短不符合矩形的正文，并增加椭圆。

注意：DT_CALCRECT, DT_EXTERNALLEADING, DT_INTERNAL, DT_NOCLIP, DT_NOPREFIX 值不能和 DT_TABSTOP 值一起使用。

返回值：如果函数调用成功，返回值是正文的高度；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：函数 DrawText 用设备环境中的字体选择、正文颜色和背景颜色来写正文，除非 DT_NOCLIP 被使用，DrawText 裁剪正文，所以它不会出现在指定矩形的外面，除 DT_SINGLELINE 格式化，其余的格式都认为正文有多行。

如果选择的字体对指定的矩形而言太大，DrawText 不会试图去换成一种小字体。

Windows CE：如果为参数 uFormat 指定 DT_CALCRECT 值，必须为 lpRect 指向的 RECT 结构设置 right 和 bottom 成员。Windows CE 不支持 uFormat 为 DT_EXTERNALLEADING。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.6DrawTextEx

函数功能：该函数在指定的矩形内绘制正文。

函数原型：int DrawTextEx(HDC, hdc, LPTSTR lpchText, int cchText, LPRECT lprc, UINT dwDTFormat,

LPDRAWTEXTPARAMS lpDTParams)；

参数：

hdc：设备环境句柄。

lpchText：指向将被绘制的字符串的指针，如果参数 cchText 设置为 C1，则此字符串必须以\0 结束。如果 dwDTFormat 包含 DT_MODIFYSTRING，此函数可以给字符串增加 4 个字符，存放字符串的缓冲区必须能足以容纳增加字符。

CchText：为 lpchText 指定的字符串指定字符数。如果字符串是以\0 结束的，则此参数可设为 C1。

lprc：指向结构 RECT 的指针，其中含有正文将格式化于其中的矩形的逻辑坐标信息。

DwDTFormat：指定格式选项，此参数取值参见 DrawText\uformat。

dwDTParams：指向 DRAWTEXTPARAMS 结构的指针，该结构指定附加的格式选项，此参数可以为 NULL。

返回值：如果函数调用成功，返回值是正文的高度；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.7EnumFontFamiliesEx

函数功能：该函数列举系统里所有符合由 LOGFONT 结构指定的字体特性的字体。此函数基于字样名或字符集或两者来枚举字体。

函数原型： int EnumFontFamiliesEx(HDC hdc, LPLOGFONT lpLogfont, FONTENUMPROC lpEnumFontFamExProc, LPARAM lParam, DWORD dwFlags)；

参数：

hdc：设备环境句柄。

lpLogfont：指向结构 LOGFONT 的指针，结构里含有要列举字体的信息。此函数检查下列成员：

lfCharset 如果设置为 DEFAULT_CHARSET，此函数列举所有字符集里的所有字体。如果设置了一个合法字符集值，此函数列举指定字符集里的字体。

lfFaceName 如果设置为空串，此函数列举每一个可用字样名的一种字体。如果设置了一个指定的字样名，此函数列举指定名字的所有字体。

LfPitchAndFamily 除 Hebrew 和 Arabic 以外，对操作系统的所有语言形式都设置为 0。对语言而言，给 lfPitchAndFamily 设置 MONO_FONT 值只列举那些能在该字体里提供所有代码页字符的字体。

lpEnumFontFamexProc：指向应用程序定义的回调函数的指针，要得到更多的善于回调函数的信息，参见函数 EnumFontFamExProc。

lParam：指定一个 32 位应用程序定义的值，此函数传递此值和字体信息给回调函数。

dwFlags：保留值，必为 0。

返回值：此返回值是由回调函数返回的最后一个值，此值依赖于提供给指定设备的字体族。

备注：EnumFontFamiliesEx 不使用字样名 tagged 来标识字符集，而改为传递正确的字样名和单独的字符集值给回调函数，此函数基于 LOGFONT 结构里的 lfCharset 和 lfFacename 值来列举字体。

如果 lfCharset 的值为 DEFAULT_CHARSET 且 lfFaceName 是一个空串，则此函数列举为每一个字符集的每个字列举一种字全，如果 lfFaceName 非空，则此函数无论字符集如何，都列举指定字样里的每一种字体。

如果 lfCharset 是一个有效的字符集值且 lfFaceName 是空串，则此函数列举指定字符集的每一种字体。如果 lfFaceName 非空，则此函数列举有着指定字样和字符集的每一种字体。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.8EnumFontFamExProc

函数功能：该函数是和函数 EnumFontFamiliesEx 一起使用的应用程序定义的回调函数，它用来处理字体。为每一种列举的字体调用一次，类型 FONTENUMPROC 定义了一个指向此回调函数的指针，EnumFontExProc 是这个应用程序定义的函数的名字的占位符。

函数原型：int CALLBACK EnumFontFamExProc(ENUMLOGFONTEX *lpelfe, NEWTEXTMETRICEX *lpntme, int FontType, LPARAM lParam)；

参数：

lpelfe：指向含有字体的逻辑属性的 ENUMLOGFONTEX 结构的指针。

lpntme：指向含有字体物理属性的结构的指针。此函数对 TrueType 字体使用结构，对其他字体使用结构 NEWTEXTMETRICEX。

FontType：指定字体类型，此参数可为下列值的组合。

DEVICE_FONTTYPE, RASTER_FONTTYPE, TRUETYPE_FONTTYPE

lParam：指定由函数 EnumFontFamiliesEx 传来的应用程序定义的数据。

返回值：要继续列举，返回值必须非零，要停止列举，返回值必须是为 0。

备注：一个应用程序必须通过将此回调函数的地址传给函数 EnumFontFamiliesEx 来注册回调函数。

与回调函数 EnumFontFamProc 不同的是，EnumFontFamProc 接收关于一种字体的扩充信息，结构 NEWTEXTMETRICEX 包括了文字（字符集）的定位名字，结构 NEWTEXTMETRICEX 包含了字体覆盖特性。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：用户自定义。

3.8.9GetAspectRatioFilterEx

函数功能：该函数获得当前纵横比过滤器的设置。

函数原型：BOOL GetAspectRatioFilterEx(HDC hdc, LPSIZE lpAspectRatio)；

参数：

hdc：设备环境句柄。

lpAspectRatio：指向结构 SIZE 的指针，用于接收当前纵横比过滤器。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：纵横比是指在给定设备上一个像素的宽与高之比。

系统提供了一个特殊的过滤器——纵横比过滤器用来选择为特殊设备设计的字体，一个应用程序可以通过调用函数 SetMapperFlags 来指示系统只接收那些符合指定纵横比的字体。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

函数功能：该函数从当前 TrueType 字体里取得给定范围的连续字符的逻辑单位宽度，此函数仅对 TrueType 字体适用。

函数原型：

参数：

：设备环境句柄。

：在当前字体里指定连续字符组的第一个字符。

：在当前字体里指定连续字符组的最后一个字符。

：指向结构 ABC 数组的指针，当函数返回时该结构用于接收字符宽度。此数组应该至少含有与和指定

的字符范围有相等个数的 ABC 结构。

返回值： 如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

：若想获得更多错误信息，请调用函数。

备注： 光栅器（）在一个大小已被选择的特定点之后提供了一个 ABC 字符间隔。A 间隔是指在放置字符之前加在当前位置的间隔。B 间隔是指字符黑色部分的宽度。C 间隔是指加到当前位置的间隔，用于给字符的右边提供空白。宽度的总和由 A＋B＋C 给出。

当得到一个字符的非 A 或 C 宽度，则此字符包括突出部分。

要转换 ABC 宽度为字体设计单位，应用程序应使用存贮于结构 OUTLINETEXTMETRIC 中成员里的值，此值可由调用函数获得。

要得到非字体里的字符宽度，应用程序应使用函数。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下以 Unicode 和 ANSI 两种方式实现。

3.8.10 GetCharABCWidthsFloat

函数功能： 该函数取得当前字体里在一个指定范围里的连续字符的逻辑单位宽度。

函数原型： BOOL GetCharABCWidthsFloat(HDC hdc, UINT iFirstChar, UINT iLastChar, LPABCFLOAT lpABCF)；

参数：

hdc：设备环境句柄。

iFirstChar：指定要寻求其 ABC 宽度的连续字符组的第一个字符的代码点。

iLastChar：指定要寻求其 ABC 宽度的连续字符组的最后一个字符的代码点，这个范围是包含在内的。如果指定的最后一个字符先于指定的第一个字符，则将返回错误信息。

lpABCF：指向含有 ABCFLOAT 结构数组的、应用程序定义的缓冲区的指针，该结构用于函数返回时接收字符宽度。

返回值： 如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 与函数 GetCharABCWidths 只返回 TrueType 字体的宽度不一样，GetCharABCWidthsFloat 可为任何字体取得宽度。此函数返回的宽度是按 IEEE 浮点格式。

如果没有标识当前的 world-to-device 转换，返回值可能是非零数值，即使相应的值在设备空间里是整数。

A 间隔是在放置字符之前加在当前位置的间隔。B 间隔是指字符黑色部分的宽度。C 间隔是指加到当前位置的间隔，用于给字符的右边提供空白。宽度总和由 A＋B＋C 给出。

ABC 间隔是沿着选字符体的字符基线来测量的，对那些超出当前选择字体范围的字符使用缺省字符的 ABC 宽度。

速查： Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.11 GetCharacterPlacement

函数功能： 该函数得到一个字符串的参数，如字符宽度、脱字符号定位、字符串的定序和字符翻译，返回信息的类型依赖于参数 dwFlags 和在给定显示环境里的当前选择字体，此函数将信息复制到指定的 GCP_RESULTS 结构里或一个或多个由此结构指定的数组里。

函数原型： DWORD GetCharacterPlacement(HDC hdc, LPCTSTR lpString, int nCount, int nMaxExtent, LPGCP_RESULTS lpResults, DWORD dwFlags)；

参数：

hdc：设备环境句柄。

lpString：指向将要处理的字符串的指针。

nCount：指定字符串里的字符个数。

nMaxExtent：指定字符串处理的最大宽度（逻辑单位）。如果要处理的字符超进此范围，则忽略不计。为任何要求的定序或字符数组进行的计算都只应用到包括进来的字符上。只有当参数 dwFlags 设置为 GCP_MAXEXTENT 时此参数才有用。当此参数处理输入的字符串时，每个字符及其宽度都被加入到输出宽度和其他数组里，只要总的宽度没有超过最大值。一旦达到最大值，处理就会停止。

lpResults：指向结构 GCP_RESULTS 的指针，该结构用于接收此函数的运行结果。

dwFlags：指定如何处理字符串并放入指定的数组里，此参数可取一个或多个下列值：

GCP_CLASSIN：指定数组 lpClass 里含有对字符的预置分类。此分类可以与输出时的相同，如果对一个字符的特殊分类是未知的，则数组里的相应位置必须设置为 0。要得到更多关于分类的信息，参见 GCP_RESULTS。只有当 GetFontLanguageInfo 返回 GCP_REORDER 标志值，此值才是有用的。

GCP_DIACRITIC：决定如何处理字符串里的变量符。如果来设置此值，变音符被当作零宽度的字符。例如，一个 Hebrew 字符中可以包含读音符，但可以不显示它们。

用 GetFontLanguageInfo 可以确定一种字体是否支持读音符。如果支持，可以根据应用程序的需要在 GetCharacterPlacement 调时或使用或不使用 GCP_DIACRITIC 标志。

GCP_DISPLAYZWG：用于那些需要重新排序，需要根据字符在一个单位中的不同位置而改变字符形状的语言，非显示字符经常出现在其代码页里。例如，在 Hebrew 代码页里，有 Left-To-Right 和 Right-To-Left 标记，用于决定输出字符串里字符的最终位置。通常这些标志不被显示并且会从 lpGlyphs 和 lpDx 数组里删除。可以用 GCP_DISPLAYZWG 标志来显示这些字符。

GCP_GLYPHSHAPE：指定将显示的字符串里的部分或所有字符使用与当前代码页里所选字体的标准形状不同的形状来显示，有一些语言，如阿拉伯语，若不设置此值将不支持字形创建。通常，如果 GetFontLanguageInfo 为一个字符串返回此值，则此值必须用于 GetCharacterPlacement。

GCP_JUSTIFY：调整数组 lpDx 里的宽度，使字符串长度和 nMaxextent 相同。GCP_JUSTIFY 只能和 GCP_MAXEXTENT 一起使用。

GCP_JUSTIFYIN：指定输入时数组 lpDx 包含的对齐权值。通常，一个对齐权值可取 0 或 1，当取 1 时，表示给字符的宽度可以为对齐而调整。对那些 GetFontLanguageInfo 函数将返回 GCP_KASHIDA 标志的语言，对齐权值可以是 GCP_ARAJUST_*值之一。

GCP_KASHIDA：使用 Kashidas 和（或）调整宽度来修改字符串的长度，使它能与 nMaxExtent 给定的值相等，在数组 lpDx 中，负对齐指数表示 Kashida。GCP_KASHIDA 只能与 GCP_JUSTIFY 组合使用，并且该字体（和语言）必须支持 Kashidas。GetFontLanguageInfo 用确定当前字体是否支持 Kashidas。

使用 Kashidas 来调整字符串会导致所需要符号数超过输入字符串的字符数。因此，当使用 Kashidas 时，应用系统不能认为设置与输入字符串同样大小的数值就足够了。（最大值可能大约为 dxPageWidth/dxAveCharWidth，其中 dxPageWidth 是文档的宽度，dxAveCharWidth 是指 GetTextMetrics 返回时的平均字符宽度）。

注意:仅仅因为 GetFontLanguageInfo 返回 GCP_KASHIDA 并不意味着它在调用 GetCharacterPlacement 中被使用，只表明它可供选择。

GCP_LIGATE：当需要捆绑时使用捆绑。当一个符号被两个或多个字符使用时，应用捆绑。例如：字母 a 和 e 可捆绑?，但这需要语言和字体都支持所需要的符号。（所给的例子在缺省的英语里就不能进行）。

用 GetFontLanguageInfo 来确定当前字体是否支持捆绑。如果支持，要有一个确定将捆绑字符数的特定最大值，将此值填放数组 lpGlyphs 中的第一个元素，如果要求通常的捆绑，设置该值为 0，如果未提供

GCP_LIGATE 值，不会发生捆绑。参见 GCP_RESULTS 可得到更多信息。

如果对字符集 GCP_REORDER 是必需的，但却没有被设置，则输出将是无意义的，除非进来的字符串已经是以可视的顺序排列的。(即第一次调用 GetCharacterPlacement 放入 lpGcpResults->lpOutString 后的结果是第二次调用的输入字符串)。

注意：仅因为 GetFontLanguageInfo 返回 GCP_LIGATE 值并不表示在调用 GetCharacterPlacement 时使用了该值，只表示该值可用而已。

GCP_MAXEXTENT：计算字符串的宽度只计算结果的逻辑单位宽度，不超过参数 nMaxExtent 指定的值。

GCP_NEUTRALOVERRIDE：只针对某些语言，忽略中性处理并将它们作为匹配字符串阅读顺序的强字符。只能和 GCP_REORDER 同时使用。

GCP_NUMERICOVERRIDE：只针对某些语言，忽略对数字的处理，把它们作为符合字符串阅读顺序的强字符，只能和 GCP_REORDER 同时使用。

GCP_NUMERICSLATIN：只对 Arabic/Thai。对数字使用标准 Latin 符号，忽略系统的缺省值。要确定一种字体的语言是否提供此选项，用 GetStringTypeEx 来查看该语言是否支持一种以上的数字格式。

GCP_NUMERICSLOCAL：只对 Arabic/Thai。对数字使用标准 Latin 符号，忽略系统的缺省值。要确定一种字体的语言是否提供此选项，用 GetStringTypeEx 来查看该语言是否支持一种以上数字格式。

GCP_REORDER：对字符串重排序，用于那些不是 SBCS 的且有从左到右阅读顺序的语言。如果没有给定此值，字符串被认为已经是按显示顺序排列的。

若此标志用于 Semitic 语言并且使用了 lpClass 数组，则数组的前两个元素是用来指定字符串范围之外部分的阅读顺序的。GCP_CLASS_PREBOUNDRTL 和 GCP_CLASS_PREBOUNDLTR 可用来设置该顺序。如果不要求预先设置的顺序，则赋该值为 0。如果设置了 GCPCLASSIN 标志，则这些值可以组合起来使用。

如果 GCP_REORDER 值没有给定,则 lpString 参数用于将语言进行可视排序,并且忽略字段 lpOutString 和 lpOrder。

用 GetFontLanguageInfo 来确定当前的字体是否支持重新排序。

GCP_SYMSWAPOFF 只对 Semitic 语言使用。指定可交换字符不重新设置。例如，在一个从右到左的字符串里’(’和’)’不被交换。

GCP_USEKERNING：当创建宽度数组时，在字体里（如果存在）使用字距调整对。用 GetFontLanguage 可确定当前字体是否支持字距调整对。

注意：不能因为 GetFontLanguage 返回了 GCP_USEKERNIN 值就认为该值被用于调用 GetCharacterPlacement，这只说明该值可以被选用。大多数 TrueType 字体都有一个字距调整表，但却不能使用它。

建议应用程序使用 GetFontLanguageInfo 来确定 GCP_DIACRITIC, GCP_DBCS, GCP_USEKERNING, GCP_LIGATE, GCP_REORDER, GCP_GLYPHSHAPE 和 GCP_KASHIDA 等值对当前所选字体是否有效。如果无效，GetCharacterPlacement 将不采用这些值。

GCP_NODIACRITICS：不再有定义，故不能使用。

返回值：如果函数调用成功,返回值和 GetTextextentPoint32 的返回值一样,是字符串的宽度和高度；如果函数调用失败，返回值是零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：GetCharacterPlacement 用来确保应用程序是否可以正确处理正文，不管国际通用设置和可用字体的类型如何。应用程序在调用函数 ExtTextOut 之前调用此函数，并用它来代替 GetTextExtentPoint32（偶尔也代替函数 GetCharWidth32 和 GetCharABCWidths)。

用 GetCharacterPlacement 来获得字符间隔和指数数组并不总是必需的，除非对齐和字距调整是必需的。对于非 Latin 字体，应用程序可以在 ExtTextOut 交付正文时提高速度，只要在调用 ExtTextOut 之前用 GetCharacterPlacement 来获得字符间的间隔和指数数组。这对于重复移交相同正文或利用字符间隔来给脱字符号定位特别有用。如果在调用 ExtTextOut 时要使用 lpGlyphs 输出数组，则 ETO_GLYPH_INDEX 标

志必须被设置。

GetCharacterPlacement 检查 GCP_RESULTS 结果里的 lpOrder, lpDx, lpCarePos, lpOutString,lpGlyphs 等成员，如果这些成员没有被设置为 NULL，为确保获得有效的信息，应用程序有责任在调用此函数前将成员放置到有效的地址，并且在调用以后要检查成员的值。如果给定了 GCP_JUSTIFY 或 GCP_USEKERNING 值，lpDx 各 / 或 lpCaretPos 成员必须有有效地址。如果 GCP_JUSTIFYIN 被设置，lpDx 成员也必须有合法地址。

在进行对齐计算时，如果字符串的结尾字符是空格，此函数将养活字符串的长度值并且在计算前删除这些空格。如果此数组全由空格组成，则函数返回出错信息。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.12 GetCharWidth32

函数功能：该函数用于取得当前字体在一个指定范围内的连续字符的逻辑坐标宽度。

函数原型：BOOL GetCharWidth32(HDC hdc, UINT iFirstChar, UINT iLastChar, LPINT lpBuffer)；

参数：

hdc：设备环境句柄。

iFirstChar：指定连续字符组里的第一个字符。

iLastChar：指定连续字符组里的最后一个字符，它不能在指定的第一个字符之前。

lpBuffer：指向用于接收宽度的缓冲区的指针。

返回值：如果函数调用成功，返回值非零，如果函数调用失败，返回值是零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：边界是包含在内的，即返回的宽度包括由参数 iFirstChar 和 iLastChar 指定的字符的宽度。如果一个字符在当前字体中不存在，则它被赋予缺省字体符的宽度。

速查：Windows NT：3.5 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.13 GetCharWidth32

函数功能：该函数用于得到当前字体在一个指定范围内的连续字符的逻辑坐标宽度。

函数原型：BOOL GetCharWidth32(HDC hdc, UINT iFirstChar, UINT iLastChar, LPINT lpBuffer)；

参数：

hdc：设备环境句柄。

iFirstChar：指定连续字符组里的第一个字符。

iLastChar：指定连续字符组里的最后一个字符，这只能在指定的第一个字符之前。

lpBuffer：指向接收宽度的缓冲区的指针。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回值是零。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：边界是包含在内的，即返回的宽度里包含了由参数 iFirsstChar 和 iLastChar 指定的字符的宽度，如果一个字符在当前字体里不存在，则给它赋予缺省字符的宽度值。

速查：Windows NT：3.5 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.14 GetFontData

函数功能： 该函数得到一种字体的度量数据。

函数原型： DWORD GetFontData(HDC hdc, DWORD dwTable, DWORD dwOffset, LPVOID lpvBuffer, DWORD cbData)；

参数：

hdc：设备环境句柄。

dwTable：指定字体度量表的名字，从度量表中可获得度量数据，此参数可确定归档于微软公司出版的 TrueType 字体文字规格说明中的度量表之一。如果此参数为零，得到的信息开始于字体文件的起始处。

dwOffset：指定从字体度量表的起点开始的偏移量，以确定此函数获取信息的开始位置。如果此参数为 0，则取回的信息开始于由 dwTable 指定的表的起点。如果此参数值大于或等于表的长度，将引发错误。

lpbBuffer：指向缓冲区的指针，该缓冲区用于接收字体信息，如果此参数 NULL，此函数返回能容纳字体信息的缓冲区的大小。

cbData：指定要取回的信息的字节数。如果此参数为 0，GetFontData 返回由参数 dwTable 指定的数据的大小。

返回值： 如果函数调用成功，返回值是返回的字节数；如果函数调用失败，返回值是 GDI_ERROR。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：应用程序可以不时地用函数 GetFontData 来将一种 TrueType 字体文件一起保存。要实现这一眯，应用程序检查 OUTLINETEXTMETRIC 结构里的 otmfsType 成员来确定字体是否嵌入。如果 otmfsType 的第一位被设置，则该字体不允许嵌入。如果第 1 位被清除，字体可被嵌入。如果第 2 位被设置，表示嵌入是只读的。如果允许嵌入，则应用程序可通过将 dwTable, dwOffset 和 cbData 置为 0 来取得整个字体文件。

如果应用程序试图用此函数来取得一种非 TrueType 字体的信息，将引发错误。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.15 GetFontLanguageInfo

函数功能： 该函数返回指定的设备环境里所选字体的信息。应用程序通常用这些信息和 GetCharacterPlacement 来准备用于显示的字符串。

函数原型： DWORD GetFontLanguageInfo(HDC hdc)；

参数：

hdc：设备环境句柄。

返回值： 返回值标识了当前选择字体的特性。当此字体是标准化的且可以被当作简单字体时，函数返回 0，如果返回 GCP_ERROR，表明发生错误，否则，函数返回下列值的组合：

GCP_DBCS：字符集是 DBCS。GCP_DIACRITIC：字体 / 语言包含读音符号。

FLI_GLYPHS：字体包含一些特殊符号，通常不能使用代码页。用 GetCharacterPlacement 来访问这些符号，此值仅作为信息，不是用来传给 GetCharacterPlacement。

GCP_GLYPHSHAPE：字体 / 语言在每个代码点或每个代码点的组合处（支持整形和 / 或捆绑）都有多个字形，并且字体含有高级的字形表，给特殊的形状提供特殊的字形。如果给定了此值，数组 lpGlyphs 将和 GetCharacterPlacement 一起使用，并且当字符串被绘制时，ETO_GLYPHINDEX 被传递给 ExtTextOut。

GCP_KASHIDA：字体 / 语言允许 Kashidas。

GCP_LIGATE：字体 / 语言包含可被特殊字符组合取代的捆绑字形。

GCP_USEKERNING：字体含有字距调整表，可用来在字符和字形之间提供更好的间隔。

GCP_REORDER：语言要求为显示重新排序。如 Hebrew 或 Arabic。

这些返回值，被 FLI_MASK 掩蔽后，可直接传给 GetCharacterPlacement。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：4.0 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.16 GetFontUnicodeRanges

函数功能：该函数返回关于一种字体支持哪些字符的信息，这些信息以 GLYPHSET 结构返回。

函数原型：WINGDIAPI DWORD WINAPI GetFontUnicodeRanges(HDC hdc, LPGLYPHSET lpgs)；

参数：

hdc：设备环境句柄。

lpgs：指向含有 GLYPHSET 结构的缓冲区的指针，如果此参数为 NULL，此函数返回指向当前字体的 GLYPHSET 结构的指针。

返回值：如果函数调用成功，返回一个指向当前设备环境中字体的结构的指针；如果函数调用失败，返回值是 0。

速查：Windows NT：5.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.17 GetGlyphIndices

函数功能：该函数将一个字符串转为字形下标的数组。此函数可用来确定一种字体里是否存在某个字形。

函数原型：WINGDIAPI DWORD WINAPI GetGlyphIndices(HDC hdc, LPCTSTR lpstr, int c, LPWORD pgi, DWORD fl)；

参数：

hdc：设备环境句柄。

lpstr：指向将被转换的字符串的指针。

c：在参数中的字符数。

pgi：与字符串的字符对应的字形下标数组。

fl：用来表示当一个字形不被支持时如何处理，此参数可取下列值：

GGI_MARK_NONEXISTING_GLYPHS：将不被支持的字形与十六进制的 0XFFFF 进行掩码操作。

返回值：如果函数调用成功，返回转换的字符数；如果函数调用失败，返回值是 GDI_ERROR。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：5.0 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.18 GetGlyphOutline

函数功能：该函数取得被选进指定设备环境的 TrueType 字体的字符轮廓或位图。

函数原型：DWORD GetGlyphOutline(HDC hdc, UINT uChar, UINT uFormat, LPGLYPHMETRICS lpgm, DWORD cbBuffer, LPVOID lpvBuffer, CONST MAT2 *lpmat2)；

参数：

hdc：设备环境句柄。

uChar：指定被返回其数据的字符。

uFormat：指定函数取得的数据的格式。可用下列值之一：各值含义为：

GGO_BITMAP：函数获得字形位图。要得到善于内存分配的信息，参见后面备注部分。

GGO_NATIVE：函数获得光栅器（rasterrizer）的本地格式的曲线数据点，并使用字体的设计单位，当指定了此值，由 lpMatrix 指定的任何变换都被忽略。

GGO_METRICS：函数只获得由 lpgm 指定的 GLYPHMETRICS 结构。其余缓冲区被忽徊，此值影响函数失败时返回值的含义，参见后面的返回值部分。

GGO_GRAY2_BITMAP：函数获得含 5 级灰色的字形位图。

GGO_GRAY4_BITMAP：函数获得含 17 级灰色的字形位图。

GGO_GRAY8_BITMAP：函数获得含 65 级灰色的字形位图。

注意：对 GGO_GRAYnBITMAP 值，函数获得 n*n+1 级灰色的字形位图。

lpgm：指向结构 GLYPHMETRICS 的指针，用于描述字表在字符单元的放置。

cbBuffer：指向定缓冲区的大小，该缓冲区用于复制轮廓字符的信息。如果此值为零，函数返回需要的缓冲区大小。

lpvBuffer：指向缓冲区的指针，该缓冲区用于复制轮廓字符的信息，如果此值为 NULL，函数返回需要的缓冲区大小。

lpmat2：指向 MAT2 结构的指针，该结构为字符信息转换矩阵。

返回值：如果指定了 GGO_BITMAP, GGO_GRAY2_BITMAP, GGO_GRAY4_BITMAP, GGO_GRAY8_BITMAP，或 GGO_NATIVE 值且函数调用成功，返回值将大于 0，否则，返回值是 GDI_ERROR。如果指定了上述之一值，但缓冲区或地址是 0，则返回需要的缓冲区的字节数。

如果 GGO_METRICS 被指定且函数调用失败，返回值是 GDI_ERROR。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：GetGlyphOutline 返回的字形轮廓是一种配有网络的字形（配有网络的字形是指一种被修改的字形，使其点阵图尽可能地与字形的原始设计一致）。如果一个应用程序需要一种无修改的字形轮廓，应该在那些大小等于字体的 em 单位的字体中要求一个字符的字形轮廓，字体的 em 单位值存在于结构 OUTLINETEXTMETRIC 的 otmEMSquare 成员中。

由 GetGlyphOutline 指定 GGO_BITMAP 返回的字形位图是一种双字对齐、面向行的单色位图，当指定 GGO_GRAY2_BITMAP，返回的位图是一种双字对齐、面向行的，其值在 0－4 之间的字节数组。当 GGO_GRAY8_BITMAP 指定时，返回的位图是一种双字对齐、面向行的，其值在 0－16 之间的字节数组。当指定 GGO_GRAY8_BITMAP 时，返回的位图是一种双字对齐，面向行的，其值在 0－255 之间的字节数组。应用程序可以指定 lpMatrix 参数里的 2－对－2 转换矩阵来将以位图格式获得的字符旋转。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.19 GetKemingPairs

函数功能：该函数取得指定设备环境里的当前选择字体的字符紧缩对。

函数原型：DWORD GetKemingPairs(HDC hdc, DWORD nNumPairs, LPKERNINGPAIR lpkmpair)；

参数：

hdc：设备环境句柄。

nNumPairs：指定填入 lpkmpair 数组中的紧缩对个数。如果当前字体有多于 nNumPairs 个字符紧缩对，将返回出错信息。

lpkmpair：指向 KERNINGPAIR 结构数组的指针，该数组用于接收字符紧缩缩对。此数组必须至少有 nNumPairs 个结构，如果此参数为 NULL，则此函数返回当前字体的所有字符紧缩对数目。

返回值：如果函数调用成功，返回字符紧缩的个数；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.20 GetOutlineTextMetrics

函数功能：该函数获得 TrueType 字体的正文度量。

函数原型：UINT GetOutlineTextMetrics(HDC hdc, UINT cbData, LPOUTLINETEXTMETRIC lpOTM)；

参数：

hdc：设备环境的句柄。

cbData：指定正文度量将返回于其中的数组的字节数。

lpOTM：指向结构 OUTLINETEXTMETRIC 的指针。如果此参数为 NULL，函数返回接收度量数据所需要的缓冲区大小。

返回值：如果函数调用成功，返回值是非零值或所需缓冲区大小；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：结构 OUTLINETEXTMETRIC 包含了大多数提供给 TrueType 字体的正文度量信息（包括一个 TEXTMETRIC 结构），返回在 OUTLINETEXTMETRIC 里的大小是以逻辑单位计算的，它们依赖于当前映射模式。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.21 GetRasterizerCaps

函数功能：该函数返回表示系统中是否安装有 TrueType 字体的标志。

函数原型：BOOL GetRasterizerCaps(LPRASTERIZER_STATUS lprs, UINT cb)；

参数：

lprs：指向 RASTERIZER_STATUS 结构的指针，该结构用于接收光栅器的信息。

cb：指定复制到参数指向的结构中的字节数。

返回值：如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：GetRasterizerCaps 使应用程序和打印驱动器能够确定 TrueType 字体是否被安装。

如果结构 RASTERIZER_STATUS 中的 wFlags 设置了 TT_AVAILABLE 标志，表示至少有一种 TrueType 字体被安装，如果 TT_ENABLED 标志被设置，表示系统能够使用 TrueType 字体。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.22 GetTabbedTextExtent

函数功能：该函数计算一个字符串的宽度和高度。如果字符串含有一个或多个制表符，则字符串的宽度基于指定的制表位。GetTabbedTextExtent 用当前所选的字体来计算字符串的尺寸。

函数原理： DWORD GetTabbedTextExtent(HDC hdc, LPCTSTR lpString, nCount, nTabPositions, LPINT lpnTabStopPositions)；

参数：

hDC：设备环境句柄。

lpString：指向字符串的指针。

nCount：指定正文字符串中的字符数。

nTabPositions：指定 lpnTabStopPositions 指向的数组中制表位位置的个数。

lpnTabStopPositions：指向含有制表位位置的数组指针，(按设备单位)。制表位必须按升序排列，最小的 X 值应是数组中的第一项。

返回值： 如果函数调用成功，返回值是字符串的尺寸。高度放于高位字，宽度位于低位字，如果函数调用失败，返回值是 0，如果 hDC 无效或 nTabPositionsn 小于 0 都将使 TabPositions 调用失败。

Windows NT： 若想获得更多错误信息，请调用 GetLastError 函数。

备注： 当前的裁剪不影响 GetTabbedTextExtent 返回的宽度和高度。

由于有的设备并不将字符放于规则的单元数组里(即紧缩字距)，故一个字符中字符的宽度和或许并不等于字符串的宽度。

如果参数 nTabPositions 的值为 0 且 lpntabStopPositions 为 NULL，则制表符被扩展为平均字符宽度的 8 倍。如果 nTabPositions 的值为 1，则制表位被 lpnTabStopPositions 指向的数组的第一个间隔值分开。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.23 GetTextAlign

函数功能： 该函数获得指定的设备环境下的文字对齐方式的设置。

函数原型： UINT GetTextAlign(HDC hdc)；

参数：

hdc：设备环境句柄。

返回值： 如果函数调用成功，返回值是文字对齐标志的状态。要了解返回值的更多信息，参见备注。返回值是下列值的组合：

TA_BASELINE：基准点在正文的基线上。TA_BOTTOM：基准点在限定矩形的下边界上。

TA_TOP：基准点在限定矩形的上边界上。TA_CENTER：基准点与限定矩形的中心水平对齐。

TA_LEFT：基准点在限定矩形的左边界上。TA_RIGHT：基准点在限定矩形的右边界上。

TA_RTLREADING：对于中东 Windows 版，正文从右到左的阅读顺序排列，与缺省的从左到右正好相反。只有当被选择的字体是 Hebrew 或 Arabic 时，此值才有用。

TA_NOUPDATECP：每次输出调用后当前状态不改变。TA_UPDATECP：每次输出调用后当前状态改变。

若当前字体有一条缺省的垂直基线（如 Kanji），下列值用于取代 TA_BASELINE 和 TA_CENTER，各值含义为：

VTA_BASELINE：基准点在正文的基线上。VTA_CENTER：基准点与限定矩形的中心垂直对齐。

如果函数调用失败，返回值是 GDI-ERROR。

Windows：若想获得更多的错误信息，请调用 GetLastError 函数。

备注： 限定矩形是指能将正文字符串的所有字符单元限定于其中的矩形。其尺寸可调用 GetTextextentPoint32 来获得。

文字对齐标志决定 TextOut 和 ExtTextOut 如何将正文字符串与基准点对齐。

文字对齐标志不必是单个的标志位，可以等于 0。标志必须按相关的组来检查，如下所示：

TA_LEFT, TA_RIGHT 和 TA_CENTER, TA_BOTTOM, TA_TOP 和 TA_BASELINE；

TA_NOUPDATECP 和 TA_UPDATECP。

如果当前字体有缺省的垂直基线，相关的标志如下所示：

TA_LEFT, TA_RIGHT 和 VTA_BASELINE, TA_BOTTOM, TA_TOP 和 VTA_CENTER；

TA_NOUPDATECP 和 TA_UPDATECP。

要验证一个特定的标志在返回值中被设置，应用程序必须执行以下步骤：

对该标志及其相关标志实施位 OR 操作；

对结果和返回值实施位 AND 操作；

检查结果值和标志是否相等。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.24 GetTextCharacterExtra

函数功能： 该函数取得指定设备环境中当前字符间隔。

函数原型： int GetTextCharacterExtra(HDC hdc)；

参数：

hdc：设备环境句柄。

返回值： 如果函数调用成功，返回值是当前字符间隔；如果函数调用失败，返回值是 08x8000000。

Windows NT： 若想获得更多错误信息，请调用 GetLastError 函数。

备注： 字符间隔定义沿基线的额外空间（按逻辑单位），用于调用 TextOut 或 ExtTextOut 时增加到每个字符，象画一条线一样。此间隔用于扩展正文行。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.25 GetTextColor

函数功能： 该函数取得指定设备环境的当前正文颜色。

函数原型： COLORREF GetTextColor(HDC hdc)；

参数：

hdc：设备环境句柄。

返回值： 如果函数调用成功，返回值是作为一个 COLORREF 的当前正文颜色；如果函数调用失败，返回值是 CLR_INVALID。

备注： 正文颜色即调用 TextOut 或 ExtTextOut 来绘制字符的前景颜色。

速查： Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib。

3.8.26 GetTextExtentPoint

函数功能： 该函数取得一个指定字符串里的字符数，该字符串将符合一个指定的空间，并且将其中每一个字符的范围放入一个数组。（一个正文的范围是指空间开始处到一个字符的间距）。此函数对自动换行的计算非常有用。

函数原型：BOOL GetTextExtentPoint(HDC hdc, LPCTSTR lpszStr, int cchString, int nMaxExten, LPINT lpnFit, LPINT alpDx, LPSIZE lpSize)；

参数：

hdc：设备环境句柄。

lpszStr：指向以\0（Null）结束的字符串的指针，函数取得该字符中的范围。

cchString：指定由 lpszStr 指向的字符串的字节数。

nMaxExtent：为格式经的字符串指定最大可允许的逻辑单位宽度。

lpnFit：指向整数的指针，该整数用于接收符合指定范围的字符的最大个数，该范围由 nMaxExtent 指定。如果 lpnFit 为 NULL，则 nMaxExtent 无效。

alpDx：指向整数数组的指针，该数组用于接收部分字符串范围。此数组的每一个元素给出一个间隔值，（按逻辑单位），表示字符串的开始处和在由 nMaxExtent 指定的空间里的一个字符之间的距离。尽管此数组应当至少有 cchString 指定那么多个元素，但函数向数组中填入其范围的字符数却由 lpnFit 确定。如果 alpDx 为 NULL，则函数不计算局部字符串宽度。

lpSize：指向结构 SIZE 的指针，该结构在函数返回时含有字符串的尺寸（按逻辑单位）。

返回值：如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备 注：如果 lpnFit 和 alpDx 的值均为 NULL，则调用 GetTextExtentPoint 等同于调用 GetTextExtentExPoint。

Windows CE：cchString 指定由 lpszStr 指向的字符串的字符个数而不是字节数。在 Windows 桌面平台下也是如此。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.27 GetTextExtentPoint32

函数功能：该函数计算指定的正文字符串的高度和宽度。

函数原型：BOOL GetTextExtentPoint32(HDC hdc, LPCTSTR lpString, int cbString, LPSIZE lpSize)；

参数：

hdc：设备环境句柄。

lpString：指向正文字符串的指针。此字符串不必以\0 结束，因为 cbString 指定了字符串的长度。

cbString：指向字符串中的字符数。

lpSize：指向 SIZE 结构的指针，该结构中字符串的尺寸将被返回。

返回值：如果函数调用成功，返回值是非零值，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：GetTextExtentPoint32 用当前所选字体来计算字符串尺寸，按逻辑单位计算的高和宽都没有考虑裁剪取的情况。

由于有的设备紧缩字符，因此一个字符串里字符的范围之和或许不等于字符串的范围。

计算的字符宽度考虑了由 SetTextCharacterextra 设备的字符间隔。

Windows CE 环境下 GetTextExtentPoint32 与 GetTextExtentPoint 相同。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：2.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.28 GetTextFace

函数功能： 该函数取得被选进指定设备环境的字体字样名。

函数原型： int GetTextFace(HDC hdc, int nCount, LPTSTR lpFaceName)；

参数：

hdc：设备环境句柄。

nCount：指定缓冲区能容纳的字符数。

lpFaceName：指向缓冲区的指针，该缓冲区用于接收字样名。如果此参数为 NULL，函数返回名字里的字符个数，包括结束的\0 字符。

返回值：如果函数调用成功，返回值为复制到缓冲区的字符的个数；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注： 字样名被复制为一个以\0 结束的字符串。如果字样名长于由 nCount 指定的字符数，则名字被截短。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.29 GetTextMetrics

函数功能： 该函数给指定的缓冲区里填入当前选择字体的度量值。

函数原型： BOOL GetTextMetrics(HDC hdc, LPTEXTMETRIC lptm)；

参数：

hdc：设备环境句柄。

lptm：指向结构 TEXTMETRIC 的指针，该结构用于接收度量值。

返回值： 如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：Windows CE 环境下，对任何给定的字体，其度量值并不必需与 Windows 桌面平台下的正文度量值相匹配。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.30 PolyTextOut

函数功能： 该函数在指定设备环境下以当前所选的字体和正文颜色绘制多个字符串。

函数原型： BOOL PolyTextOut(HDC hdc, CONST POLYTEXT *pptxt, int cStrings)；

参数：

hdc：设备环境句柄。

pptxt：指向 POLYTEXT 结构数组的指针，该数组描述将被绘制的字符串。数组中一每一个字符对应一个结构。

cString：指定 pptxt 数组中的 POLYTEXT 结构数。

返回值： 如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：每一个 POLYTEXT 结构都包含有基准点的坐标，以便在 Windows 下对齐相应的正文字符串。一个

应用程序可以调用 SetTextAlign 来指定如何使用基准点。可调用 GetTextAlign 来确定指定设备环境的当前对齐方式。要绘制单个正文字符串，应用程序应该调用 ExtTextOut。

速查：Windows NT：3.1 及以上版本；Windows：不支持；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.31 RemoveFontResource

函数功能：该函数从系统字体表中除去在指定文件里的字体。

函数原型：BOOL RemoveFontResource(LPCTSTR lpFileName)；

参数：

lpFileName：指向以\0 结束的字符串的指针，该字符串表示字体资源文件的名字。

返回值：如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0 。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：一个应用程序如果增加或除去系统字体表中的字体，应该发一个 WM_FONTCHANG 消息给系统中的所有顶层窗口，通知这个变化。应用程序通过设置参数 hwnd 为 HWND_BROADCAST 来调用 SendMessage 就可以发消息。

如果存在一种字体的重要引用，必须保证相关资源的存在，直到没有设备环境再使用它为止。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.32 SetMapperFlags

函数功能：该函数改变字体映射程序用于将逻辑字体映射为物理字体的算法。

函数原型：；DWORD SetMapperFlags(HDC hdc, DWORD dwFlag)；

参数：

hdc：设备环境句柄，设备环境中含有字体映射标志。

DwFlag：指定字体映射函数是否将字体的纵横比与当前设备的纵横比相匹配。如第零位被设置，映射函数只选择匹配的字体。

返回值：如果函数调用成功，返回值是字体映射标志的前一个值；如果函数调用失败，返回值是 GDI_ERROR。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：如果 dwFlag 被设置但没有匹配的字体，Windows 操作系统将选择一个新的纵横比并且取得一种与之相匹配的字体。DwFlag 的其余位必须是 0。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.33 SetTextAlign

函数功能：该函数为指定设备环境设置文字对齐标志。

函数原型：UINT SetTextAlign(HDC hdc, UINT fMode)；

参数：

hdc：设备环境句柄。

fMode：使用下列值的掩码指定文件对齐方式。从水平和垂直对齐标志里只能选择一个。另外，对改变前状态的两个标志只能选择一个。各值含义参见 GetTextAlign。缺省值是 TA_LEFT, TA_TOP 和 TA_NOUPDATECP。

返回值：如果函数调用成功，返回值是文字对齐方式的前一个设置；如果函数调用失败，返回值是 GDI_ERROR。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：TextOut 和 ExtTextOut 用文字对齐标志来将一个正文字符串在显示时定位或定位于其他设备。该示志指定基准点与限定正文的矩形的位置关系，基准点可以是当前位置，也可是传给正文输出函数的一个点。限定正文的矩形是正文字符串里的字符单元形成的。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.34 SetTextCharacterExtra

函数功能：该函数设置字符间隔，字符间隔加到每一个字符上，包括间隔字符，只要系统在写一个正文行。

函数原型：int SetTextCharacterExtra(HDC hdc, nCharExtra)；

参数：

hdc：设备环境句柄。

nCharExtra：指定被加到每个字符上的额外空间（按逻辑单位）。如果当前映射模式不是 MM_TEXT，则 nCharExtra 被转换，并且舍入到最接近的像素上。

返回值：如果函数调用成功，返回值是字符间隔的先前值；如果函数调用失败，返回值是 0x80000000。

Windows NT：要得到更多的错误信息，调用 GetLastError 函数。

速查：头文件：wingdi.h；库文件：gdi32.lib。

3.8.35 TabbedTextOut

函数功能：该函数将一个字符串写到指定的位置，并按制表位位置数组里的值展开制表符。正文以当前选择的字体、背景色和字体写入。

函数原型：；LONG TabbedTextOut(HDC hdc, int X, int Y, LPCTSTR lpString, int nCount, int nTabPositions, LPINT lpn TabStopPositions, int nTabOrigin)；

参数：

hDC：设备环境句柄。

X：字符串开始点的 X 坐标（按逻辑单位）。

Y：字符串开始点的 Y 坐标（按逻辑单位）。

lpString：指向将被绘制的字符串的指针，此字符串不必以\0 结束，因为 nCount 指定的字符串的长度。

nCount：指定字符串里的字符数。

nTabPosition：指定制表位位置数组里的值的个数。

lpnTabStopPositions：指向数组的指针，该数组含有制表位位置（按逻辑单位）。制表位必须按升序保存，最小的 X 值必须是数组的第一项。

Windows 95：一个制表位可被指定为负值，这会使在制表位上正文右对齐，而不是左对齐。

nTabOrigin：指定制表符展开的开始位置的 X 坐标（按逻辑单位）。

返回值：如果函数调用成功，返回值是字符串的尺寸（按逻辑单位）。高位字表示高度，低位字表示宽度；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：如果 nTabPositions 值为 0，且 lpnTabStopPositions 值为 NULL 则制表符将会按平均字符宽度的 8 位来扩展。

如果 nTabPositions 值为 1，则制表位按在 lpnTabStopPositions 中的第一个值指定的距离来分隔。

如果 lpnTabStopPositions 数组包含一个以上的话，则制表位被设为数组里的每一个值，共为 lpnTabStopPositions 个。

NtabOrigin 参数允许一个应用程序为一行多次调用 TabbedTextOut。如果应用程序多次调用 TabbedTextOut，nTabOrigin 每次都设置相同的值，则此函数在相对于 nTabOrigin 指定的位置处展开所有的制表符。

缺省地，TabbedTextOut 不会使用和改变当前位置，当一个应用程序需要在调用 TabbedTextOut 时改变当前位置，可以通过设置 wFlags 为 TA_UPDATECP，调用 SetTextAlign 来实现。当该标志被设置时，系统会在随后调用 TabbedTextOut 时忽略参数 X 和 Y 的值，而使用当前位置。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.36 SetTextJustification

函数功能：该函数指定一定数量的空间，让系统增加到一个正文字符串的间隔字符上，当应用程序调用 TextOut 或 ExtTextOut 时，空间就地加进去。

函数原型：BOOL SetTextJustification(HDC hdc, int nBreakExtra, int nBreakCount)；

参数：

hdc：设备环境句柄。

nBreakExtra：指定将被加到正文行的总的额外空间（按逻辑单位）。如果当前映射模式不是 MM_TEXT，则由 nBreakExtra 标识的值将被转换，并舍入到最近的像素上。

nBreakCount：指定一行中间隔字符的个数。

返回值：如果函数调用成功，返回值非零；如果函数调用失败，返回是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。 **备注：**间隔符通常空格符（ASCII32），但一种字体可以象定义其他字符一样来定义它。GetTextMetrics 可用来得到一种字体的间隔符。 TextOut 将指定的额外空间分散到一行中，甚至在间隔符中间。 GetTextExtenPoint32 总是和 SetTextJustification 一起使用。GetTextextentPoint32 在进行调整前计算给定行的宽度。必须先知道这个宽度值才能恰当计算出 nBreakExtra。 SetTextJustification 可用来调整含有使用不同字体的多个字符串的一行，在这种情况下，每一个字符串必须被单独调整。 由于在调整时会发生舍入错误，系统保留了一个运行错误项来定义当前错误值，当调整一行包含了多个操作时，GetTextExtentPoint 会在计算下一个操作范围时自动地使用这个错误项，并允许 TextOut 混合该错误到新的操作里。当一行被调整后，这个错误项必须被清除，以避免被组合到下一行。可将 nBreakExtra 设为 0，调用 SetTextJustification 来清除这个项。 **速查：**Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib。

3.8.37 TextOut

函数功能：该函数用当前选择字符、背景颜色和正文颜色将一个字符串写到指定位置。

函数原型： BOOL TextOut (HDC hdc, int nXStart, int nYStart, LPCTSTR lpString, int cbString)；

参数：

hdc：设备环境句柄。

nXStart：指定用于字符串对齐的基准点的逻辑 X 坐标。

nYStart：指定用于字符串对齐的基准点的逻辑 Y 坐标。

lpString：指向将被绘制字符串的指针。此字符串不必为以\0 结束的，因为 cbString 中指定了字符串的长度。

cbString：字符串的字符数。

返回值： 如果函数调用成功，返回值非零；如果函数调用失败，返回值是 0。

Windows NT：若想获得更多错误信息，请调用 GetLastError 函数。

备注：对基准点的解释依赖于当前字符对齐模式，一个应用程序可以调用 GetTextAlign 来获得该模式，调用 SetTextAlign 来更改该模式。

缺省地，此函数不能使用和修改当前位置，但一个应用程序可以在每次在指定设备环境中调用 TextOut 之前，通过将参数 fMode 设为 TA_UPDATECP 值来调用 SetTextAlign，以允许系统在调用 TextOut 时修改当前位置，当上述标志设置时，系统会忽略随后调用 TextOut 中的 nXStart 和 nYStart 值。

当把函数 TextOut 放入一个路径括号内时，系统为包含每一个字符并加上字符值的 TrueType 正文产生一个路径，产生的区域是字符框加上正文，而不是正文本身，可以在把 TextOut 放入路径括号之前设置背景模式为透明的，这样就能得到由 TrueType 正文的轮廓封闭而成的区域。下面是说明这个过程的例程代码：

GetClientRect(hwnd, &r); BeginPath(hdc); TextOut(hdc, r.left, r.top, "Defenestration can be hazardous", 4); EndPath(hdc); SelectClipPath(hdc, RGN_AND); FillRect(hdc, &r, GetStockObject(GRAY_BRUSH))。

速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：不支持；头文件：wingdi.h；库文件：gdi32.lib；Unicode：在 Windows NT 环境下实现为 Unicode 和 ANSI 两种版本。

3.8.38 EnumFontFamilies

函数功能： 该函数列举指定设备环境下的指定字体族里的字体。

函数原型：int EnumFontFamilies (HDC hdc, LPCTSTR lpszFamily, FONTENUMPROC lpEnumFontFamProc, LPARAM lParam)；

参数：

hdc：设备环境句柄。

lpszFamily：指向以\0 结束的字符串的指针，该字符串指定期望的字体族的名字。如果 lpszFamily 为 NULL，则 EnumFontFamilies 随机地选择并列举每一个有效字体族的一种字体。

lpEnumFontProc：指定应用程序定义的回调函数的过程实例地址，参见 EnumFontFamilies 可得到关于回调函数的更多信息。

lParam：指向应用程序供给的数据的指针，该数据和字体信息一起传给回调函数。

备注：对每一种由参数指定其字样名的字体，EnumFontFamilies 获得此字体的信息，并将其传给由参数 pEnumFontFamProc 指向的函数。应用程序定义的回调函数可按期望处理这些字体信息，当再没有有效字体或回调函数返回零时，列举才停止。

Windows CE 版本 1.0 只支持光栅字体。Windows CE 版本 2.0 支持那些使用早期 TrueType 或光栅字体两者之一的系统。字体类型（TrueType 或光栅）是在系统设计时选择的，不能被应用程序修改。

3.8.39 EnumFonts

函数功能： 该函数列举一个指定设备可用的字体，对那些有指定字样名的字体，EnumFonts 取得该字体的信息，并将信息传给应用程序定义的回调函数。回调函数可以按期望处理字体信息。当再没有字体可列举或回调函数返回零时，列举停止。

函数原型：int EnumFonts(HDC hdc, LPCTSTR lpFaceName, FONTNUMPROC lpFontfunc, LPARAM lParam)；

参数：

hdc：设备环境句柄。

LpFaceName：指向以\0 结束的字符串的指针，该字符串指定所期望字体的字样名。如果此值为 NULL，则 EnumFonts 随机选择并列举每个有效字样的一种字体。

LpFontFunc：指向应用程序定义的回调函数的指针。要得到更多的信息，参见 EnumFontsProc。

：指向任何应用程序定义的数据的指针，这些数据和字体信息一起传给回调函数。

返回值：返回值是由函数返回的一个值。其含义为应用程序定义。

备注：用 EnumFontFamiliesEx 来代替 EnumFonts。EnumFontFamiliesEx 与 EnumFonts 的不同之处在于前者取得了与一种 TrueType 字体相联系的风格名。用 EnumFontsFamiliesEx，可以取得关于字体风格的信息，这是用 EnumFonts 无法获得的。Windows CE：版本 1.0 只支持光栅字体。

Windows CE 版本 2.0 支持那些使用 TrueType 字体和光栅字体其中之一的系统。字体类型（光栅或 TrueType）是在系统设计时选择的，不能被应用程序修改。

3.8.40 GetCharWidth

函数功能： 该函数取得当前字体在一个指定范围内的连续字符的逻辑坐标宽度。

函数原型：BOOL GetCharWidth(HDC hdc, UINT iFirstChar, UINT iLastChar, LPINT lpBuffer)；

参数：

hdc：设备环境句柄。

iFirstChar：指定连续字符组的第一个字符。

iLastChar：指定连续字符组的最后一个字符，它不能在指定的第一个字符之前。

lpBuffer：指向接收宽度的缓冲区的指针。

返回值：如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：边界是包含在内的,即返回的宽度里包含了由 IfirstChar 和 iLastChar 指定的字符的宽度范围。如果一个字符在当前字体里不存在，则给字赋予缺省字符的宽度范围。

3.8.41 GetCharWidth

函数功能： 该函数取得当前字体在一个指定范围内的连续字符的逻辑坐标宽度。

函数原型：BOOL GetCharWidth(HDC hdc, UINT iFirstChar, UINT iLastChar, LPINT lpBuffer)；

参数：

hdc：设备环境句柄。

iFirstChar：指定连续字符组的第一个字符。

iLastChar：指定连续字符组的最后一个字符，它不能在指定的第一个字符之前。

lpBuffer：指向接收宽度的缓冲区的指针。

返回值：如果函数调用成功，返回值非零，如果函数调用失败，返回值是 0。

Windows NT：若想获得更多的错误信息，请调用 GetLastError 函数。

备注：边界是包含在内的,即返回的宽度里包含了由 IfirstChar 和 iLastChar 指定的字符的宽度范围。如果一个字符在当前字体里不存在，则给字赋予缺省字符的宽度范围。

3.8.42 EnumFontFamProc

函数功能：该函数是由应用程序定义的与函数 EnumFontFamilies 一起使用的回调函数，它接收用于描述与可用字体的数据。类型 FONTENUMPROC 定义了一个指向此回调函数的指针，EnumFontFamProc 是应用程序定义的函数的名字的占位符。

函数原型： int CALLBACK EnumFontFamProc(ENUMLOGFONT FAR *lpelf, NEWTEXTMETRIC FAR *lpntm, int FontType, LPARAM lParam)；

参数：

lpelf：指向结构 ENUMLOGFONT 的指针，该结构包含字体的逻辑属性。该结构是局部定义的。

lpntm：指向结构 NEWTEXTMETRIC 的指针，该结构包含 TrueType 字体的物理属性。如果该字体不是 TrueType 字体，此参数为指向结构 TEXTMETRIC 的指针。

FontType：指定字体的类型，此参数可以是下列值的组合：

DEVICE_FONTTYPE, RASTER_FONTTYPE, TRUETYPE_FONTTYPE。

lParam：指向由函数 EnumFontFamilies 传来的应用程序定义的数据的指针。

返回值：要继续列举，返回值必须是非零值；要停止列举，返回值必须为 0。

备注：应用程序必须将此回调函数的地址传给 EnumFontFamilies 以将函数注册。

RASTER_FONTTYPE, DEVICE_FONTTYPE 和 TRUETYPE_FONTTYPE 等常数可用 AND 操作符连接起来决定字体的类型。如果 RASTER_FONTTYPE 位被设置，则该字体是一种光栅字体。如果 TRUETYPE_FONTTYPE 位被设置，该字体是一种 TrueType 字体。如果上述两位都未被设置，则该字体是一种向量字体。当一种设备（如激光打印机）支持下载 TrueType 字体或该字体是一种驻留设备的字体，则 DEVICE_FONTTYPE 位被设置，当设备是一种显示设备、点阵打印机或其他光栅设备，则该位为 0。应用程序也可用 DEVICE_FONTTYPE 来区别图形设备接口（GDI）提供的光栅字体和设备提供的字体。GDI 可以为 GDI 提供的光栅字体模拟粗体、斜体、下划线和删除线（strikeout）属性，但却不支持设备提供的字体。

Windows CE：字体的物理属性总是保存在 TEXTMETRIC 结构中。

Windows CE 版本 1.0 不支持 TrueType 字体。在该版本中，参数 FontType 的值始终是 RASTER_FONTTYPE。

在 Windows CE 版本 2.0，FontType 可依赖于 Windows CE 平台取值为 RASTER_FONTTYPE 或 TRUETYPE_FONTTYPE，任何给定的 Windows CE 平台只支持光栅字体和 TrueType 字体其中之一，字体类型（光栅或 TrueType）是在系统设计时选择的，应用程序不能修改。

3.8.43 EnumFontsProc

函数功能：该函数是一个应用程序定义的回调函数，处理由 EnumFonts 获得的字体数据。EnumFontsProc 是应用程序定义的函数的名字的占位符。

函数原型： int CALLBACK EnumFontsProc(lplf lplf, lptm lptm, DWORD dwType, LPARAM lpData)；

参数：

lplf：指向结构 LOGFONT 的指针，该结构含有字体的逻辑属性。

lptm：指向结构 TEXTMETRIC 的指针，该结构含有字体的物理属性。

dwType：指定字体类型，此参数可取下列值的组合：

DEVICE_FONTTYPE, RASTER_FONTTYPE, TRUETYPE_FONTTYPE。

lpData：指向由 EnumFonts 传来的应用程序定义的数据的指针。

返回值：如果要继续列举，返回值必须是非零值；要停止列举，返回值必须是 0。

备注：RASTER_FONTTYPE 和 DEVICE_FONTTYPE 可由 AND 操作符组合起来决定字体类型。参数 FontType 的 RASTER_FONTTYPE 用来指定该字体是光栅字体还是向量字体。如果该位是 1，字体是光栅字体；该位是 0，则是向量字体。FontType 的 DEVICE_FONTTYPE 位用于确定字体是基于设备的还是基于图形设备接口(GDI)的。如果该位是 1，表明该字体是基于设备的，如果该位是 0，则是基于 GDI 的字体。

如果只要基字体被列举，设备就能进行正文转换（比例变换、倾斜等），则用户应该查询设备的正文转换能力，以确定设备还能直接提供哪些字体。

应用程序应该将 EnumFontsProc 的地址传给 EnumFonts，将 EnumFontsProc 注册。

Windows CE：版本 1.0 不支持 TrueType 字体。参数 TrueType 的值必须是 RASTER_FONTTYPE。

Windows CE 版本 2.0 支持 TrueType 和光栅两种字体。在版 2.0 里，FontType 依赖于 Windows CE 平台可取 RASTER_FONTTYPE 或 TRUETYPE_FONTTYPE。任何 Windows CE 平台只能支持光栅字体和 FontType 字体两者之一。

3.9 ICM 2.0 函数

Microsoft Windows 98 和 Windows NT 5.0 所使用的颜色管理方案称为 Image Color Management 版本 2.0，或 ICM2.0，由一系列函数组成。

3.10 直线和曲线函数（Line and Curve）

直线和曲线用于在光栅设备上绘制输出图形。一条直线是光栅显示器上的一系列高亮像素点（或打印纸上的一系列点），由两个点进行标识：起点和终点。一条规则曲线也是光栅显示器上的一系列高亮像素点（或打印纸上的一系列点），符合某个二次曲线段的周界（或一部分）。不规则曲线则是由不符合二次曲线段的一系列像素点组成。

3.11 元文件函数（Metafile）

元文件是一个结构的集合，这些结构是以与设备无关的格式存储图像。设备无关是元文件与位图的差异之一。与位图不同，元文件保证是与设备无关的。不过，元文件有一个缺点：它通常比位图的绘图速度慢。因此，如果一个应用程序要求有较快的绘图速度，而不需要具有设备无关性，则应该用位图代替元文件。

元文件函数提供了一些对元文件进行操作的方法。

3.12 多显示器支持函数（Multple Display Monltors）

每个 Windows 工作站所支持的显示器个数是不受限制的。可以用创建邻接区域的方式安排多个显示器。每个显示器的大小和颜色深浅都可以独立设置。

所有的显示器屏幕一起构成了一个虚拟屏幕。桌面窗口覆盖整个虚拟屏幕，而不仅仅是某个显示屏幕。由于现有的应用程序都要求显示器具有一个原点坐标（0，0），所以虚拟屏幕必须在某个显示器上包含原点坐标（0，0），这个显示器就被看作是主显示器。

每个物理显示设备都由一个 HMONITOR 类型的显示器句柄表示。一个显示器在它的整个生存期间具有相同的 HMONITOR 值。

任何显示设备环境（DC）的 Win32 函数所返回的值都是主显示器的 DC。要想获取其他显示器的 DC，可使用 EnumDisplayMonitors 函数。系统对每个显示器调用回调函数，为该显示器传入一个 DC 值。用户可以使用该 DC 在该显示器上绘图。

3.13 绘图和画图函数（Painting and Drawing）

绘图和画图函数为应用程序提供了一系列在窗口中绘图的方法，以及如何创建和使用显示设备环境（DC）的方法。

3.14 路径函数（Path）

一个路径是指一个或多个被填充、被绘制轮廓或既被填充又被绘制轮廓的图形（或形状）。Win32 应用程序将路径用作很多用途，在绘图和画图应用程序中使用路径。计算机辅助设计（CAD）应用程序用路径来创建唯一剪裁区，绘制不规则形状的轮廓，以及填充不规则形状的内部。路径函数用于创建、改变和绘制路径。

3.15 画笔函数（Pen）

画笔是 Win32 应用程序用于绘制直线和曲线的图形工具。画图应用程序使用画笔来画手画线、直线以及曲线。计算机辅助设计（CAD）应用程序用画笔来画可见线、隐藏线、截面线、中心线等等。字处理和桌面出版应用程序用画笔来画边界和水线。电子表格应用程序用画笔来指明图表的趋向，以及勾勒直方图和饼图的轮廓。画笔函数提供了一系列使用画笔的方法。

3.16 打印和打印假脱机函数（Printing and Print Spooler）

Microsoft Windows 和 Windows NT 提供了一套完整的函数，使应用程序可以在不同的设备上进行打印，如激光打印机，向量绘图仪，光栅打印机，以及传真机等。

3.17 矩形函数（Rectangle）

Win32 应用程序使用矩形来指定显示屏幕上或窗口中的一个矩形区域。矩形函数用于对矩形进行操作。

3.18 区域函数（Region）

区域是指一个可被填充、着色、转换和加外框的形状，包括矩形、多边形或椭圆（或这几种形状的组合），用于完成击键测试（测试光标位置）。

区域函数用于对区域进行操作。

第四章　网络服务函数

网络函数允许网络上的不同计算机的应用程序之间进行通讯。

网络函数用于在网络中的各计算机上创建和管理共享资源的连接，例如共享目录和网络打印机。

网络接口包括 Windows 网络函数、Windows 套接字（Socket）、NetBIOS、RAS、SNMP、Net 函数，以及网络 DDE。Windows 95 只支持这些函数中的一部分。

网络服务函数包括以下几类：

DLC 函数（DLC）

数据连接控制(DLC)接口是一个具有特殊目的的、不可路由的协议。它不是用于运行 Windows 和 Windows NT 的计算机之间的通讯，而是为运行 Windows 或 Windows NT 的计算机与 IBM 主机或直接连接到网络上的打印机之间提供了连通性。

网络函数（Net）

对于基于 OS/2 的服务器来说，Microsoft LAN Manager 所支持的 Net 函数提供了很多网络操作系统所需的功能，这些功能在本地操作系统中被忽略了。Windows NT、Windows 95 和 Windows 98 具有很多内置的网络功能，因此，有些原始的 Net 函数就不再支持了。

Windows NT、Windows 95 和 Windows 98 支持多种网络函数。Net 函数集提供了一些其他网络函数来覆盖的附加功能。另外，还可以使用这些函数来监测和管理基于 OS/2 的 LAN Mansger 服务器。

NetBIOS 函数

Win32 应用程序可以使用 Network Basic Input/Output System（NetBIOS）接口与网络中的其他计算机上的应用程序进行通讯。

NetBIOS 接口包括一系列显式命令，由一个被称为网络控制块（NCB）的结构提供。应用程序可以对任何支持 NetBIOS 接口的协议发出 NetBIOS 命令。

网络 DDE 函数（Networking DDE）

一个进程可以使用 Win32 API 提供的网络动态数据交换（DDE）函数与在网络中的不同计算机上运行的进程建立会话。

RAS 服务器管理函数（RAS Server Administration）

在 Windows NT 4.0 上，可使用 RAS 服务器管理函数来实现 RAS 服务器管理功能。Windows 95 不提供 RAS 服务器支持。

远程访问服务函数（Remote Access Service）

使用远程访问服务（RAS）可以使远程用户犹如直接连接到计算机网络上一样地访问一个或多个 RAS 服务器。

远程访问服务函数用于实现远程访问服务功能。

服务函数（Service）

Win32 API 提供了一套完整的服务函数。这些函数应该可以代替 NetService 函数，除非需要控制 LANManager2.x 服务器上的服务。

服务函数用于控制服务。一个服务就是一个应用程序，管理员可以使用服务控制程序接口来控制服务。

Windows 网络函数（Windows Networking）

Windows 提供的 Windows 网络（Wnet）函数使用户可以在应用程序中实现网络功能，而不需使用特殊的网络供应程序或物理的网络实现。原因是 Wnet 函数是网络无关的。

第五章　国际特性函数

这些特性有助于用户编写国际化的应用程序。Unicode 字符集使用 16 位的字符值来表示计算过程中所用的字符，比如各种符号，以及很多编程语言。国家语言支持（NLS）函数可帮助用户将应用程序本地化；输入方法编辑器（IME）函数（在 Windows 亚洲版中可用）用于帮助用户输入包含 Unicode 和 DCBS 字符的文本。

国际特性函数包括以下几类：

输入方法编辑器函数（Input Method Editor）

输入方法编辑器(IME)有助于简化用户的文本输入过程(文本中包含 Unicode 字符和双字节字符 DBCS)。

输入方法编辑器函数用于创建和管理 IME 窗口。

国家语言支持函数（National Language Support）

使用国家语言支持函数可以帮助 Win32 应用程序支持世界各地的不同语言，满足不同地区用户的特殊需要。

Unicode 和字符集函数（Unicode and Character Set）

Win32 API 通过 Unicode 和传统字符集可以支持国际上的很多不同的书写语言。Unicode 是一种世界通用的字符编码标准，它使用 16 位的字符值来表示各种字符，包括技术符号和出版所用的特殊字符。传统字符集是指以前所用的字符编码标准，比如 Windows ANSI 字符集，它是使用 8 位的字符值或 8 位值的组合来表示在指定的语言或地理区域中所用的字符。

Unicode 和字符集函数用于对字符集进行操作。

第六章　系统服务函数

系统服务函数为应用程序提供了访问计算机资源以及底层操作系统特性的手段，比如访问内存、文件系统、设备、进程和线程。应用程序使用系统服务函数来管理和监视它所需要的资源。例如，应用程序可使用内存管理函数来分配和释放内存，使用进程管理和同步函数来启动和调整多个应用程序或在一个应用程序中运行的多个线程的操作。

系统服务函数提供了访问文件、目录以及输入输出（I/O）设备的手段。应用程序使用文件 I/O 函数可以访问保存在指定计算机以及网络计算机上的磁盘和其他存储设备上的文件和目录。这些函数支持各种文件系统，从 FAT 文件系统，CD-ROM 文件系统（CDFS），到 NTFS。

系统访问函数为应用程序提供了一些可以与其他应用程序共享代码或信息的方法。例如，可以将一些有用的过程放到 DLL 中，使它们对所有的应用程序都可用。应用程序只需使用 DLL 函数将动态链接库加载进来并获取各过程的地址，就可以使用这些过程了。通讯函数用于向通讯端口写入数据及从通讯端口读出数据，并控制这些端口的操作方式。有几种内部通讯（IPC）的方法，比如 DDE、管道（Pipe）、邮槽（Mailslot）和文件映射。对于提供安全属性的操作系统来说，应用程序可使用安全函数来访问安全数据，并保护这些数据不会被有意或无意地访问或破坏。

使用系统服务函数可以访问有关系统和其他应用程序的信息。应用程序可用系统信息函数来确定计算机的特别属性，比如是否出现鼠标、显示屏幕上的元素具有多大尺寸。注册和初始化函数用于将应用程序的特殊信息保存到系统文件中，以便于该应用程序的新实例对象，甚至其他应用程序都可以获取和使用这些信息。

应用程序使用系统服务函数可以处理执行过程中的的一些特殊情况，比如错误处理、事件日志、异常处理。还有一些属性可用于调试和提高性能。例如，使用调试函数可对其他进程的执行过程进行单步控制，而性能监视函数则可对某个进程的执行路径进行跟踪。

系统服务函数还提供了一些特性，可用于创建其他类型的应用程序，比如控制台应用程序和服务。

系统服务函数包括以下几类：

访问控制函数（Access Control）

Microsoft Windows NT 所提供的安全功能对 Win32 应用程序是自动使用的。在系统中运行的每个应用程序都受由 Windows NT 的特殊配置所提供的安全功能所影响。Windows NT 是支持 Win32 安全功能的唯一平台。

Windows NT 的安全功能对大多数 Win32 函数的影响都是最小的，不需要安全功能的 Win32 应用程序不必合并任何特殊代码。不过，你可使用 Windows NT 的安全属性向 Win32 应用程序提供一些服务。

访问控制函数提供了一系列控制访问 Win32 对象（比如文件）、管理函数（比如设置系统时间或审核运行动作的函数）的 Windows NT 安全模型。

原子函数（Atom）

原子表格是一个系统定义的表格，用于保存字符串和相应的标识符。应用程序将一个字符串放到原子表格中，并接受一个 16 位的整数（称为一个原于），用于访问该字符串。放到原子表格中的字符串被称为原子名字。

原子函数提供了一系列对原子进行添加、删除、初始化等的操作。

客户服务器访问控制函数（Client/Server Access Control）

客户/服务器访问控制函数包括三类：

用于模拟客户机。

用于检查和设置私有对象上的安全描述符。

用于生成安全时间日志中的审核消息。

剪贴板函数（Clipboard）

剪贴板是由一系列函数和消息组成，Win32 应用程序可使用它来传输数据。由于所有的应用程序都可以访问剪贴板，所以数据可以很容易地在应用程序之间或一个应用程序内部进行传输。

通讯函数（Communication）

通讯资源是一个物理或逻辑设备，用于提供双向的异步数据流。例如，串行端口、并行端口、传真机以及调制解调器都是通讯资源。对于每个通讯资源都有一个服务供应程序（包含一个库或驱动程序），使应用程序可以访问该资源。通讯函数是通讯设备所使用的函数。

控制台函数（Console）

Microsoft Windows 和 Windows NT 提供了控制台函数，用于管理字符模式的应用程序（这种应用程序未提供自己的图形用户界面）的输入和输出（I/O）

数据解压库函数（Data Decompression Library）

数据解压库函数在 LZEXPAND.DLL 中声明，用于对压缩的文件进行解压。

调试函数（Debugging）

调试器是一个应用程序，开发人员可使用它来检查和改正编程错误。Win32 API 的调试函数为用户提供了一系列的调试手段。

设备输入和输出函数（Device Input and OutPut）

Win32 应用程序使用设备输入和输出控制与设备驱动程序进行通讯。被访问的设备由设备句柄标识；而设备驱动程序要完成的动作则由控制代码来指定。

动态数据交换函数（Dynamic Data Exchange）

Win32 API 为不能使用"动态数据交换管理库（DDEML）"的应用程序提供了一系列实现动态数据交换的函数。

动态数据交换管理函数（Dynamic Data Exchange Management）

动态数据交换（DDE）是一种内部通讯方式，即使用共享内存在应用程序之间交换数据。应用程序可以使用 DDE 进行一次性的数据传输，以及数据的即时交换和更新。

动态数据交换管理函数为用户提供了一系列管理动态数据交换的手段。

动态链接库函数（Dynamic-Link Library）

动态连接库（DLL）是由函数和数据组成的一些模块。一个 DLL 是由它的调用模块（.EXE 或.DLL）在运行时加载的。当一个 DLL 被加载后，它就被映射到其调用进程的地址空间中。

DLL 可以定义两种函数：外部的和内部的。外部函数可以被其他模块调用，内部函数只能在声明它的 OLL 内部被调用。尽管 DLL 可以输出数据，但它的数据通常只能由它的函数使用。

DLL 提供了一种使应用程序模块化的方法，这样就可以更容易地更新和重用程序的功能。DLL 也有助于在几个应用程序同时使用相同的功能时减少内存开销，因为虽然每个应用程序都拥有一份数据的备份，但它们可以共享代码。

错误函数（Error）

写得好的应用程序应包括一些能够处理意外错误并可从错误中顺利恢复的代码。当发生错误时，应用程序可能需要用户进行干预，或自己恢复。在一些极端情况下，应用程序对能会将用户从系统中退出或关机。错误函数为用户痰腥 LL 内部被调用。尽管 DLL 可以输出数据，但它的数据通常只能由它的函数使用。

DLL 提供了一种使应用程序模块化的方法，这样就可以更容易地更新和重用程序的功能。DLL 也有助于在几个应用程序同时使用相同的功能时减少内存开销，因为虽然每个应用程序都拥有一份数据的备份，但它们可以共享代码。

错误函数（Error）

写得好的应用程序应包括一些能够处理意外错误并可从错误中顺利恢复的代码。当发生错误时，应用程序可能需要用户进行干预，或自己恢复。在一些极端情况下，应用程序对能会将用户从系统中退出或关机。错误函数为用户提供了一些进行错误处理的方法。

事件日志函数（Event Logging）

很多应用程序都在不同的属性错误日志中记录错误和事件。这些属性错误日志具有不同的格式，并显示不同的用户界面，而且无法将数据合并起来得到一个完整的报告。因此，用户必须要检查各种数据来诊断问题。Windows NT 的事件日志为应用程序（和操作系统）提供了一种标准、集中的方法，来记录重要的软件和硬件事件。事件日志服务将事件从不同的地方保存到一个称为"事件日志"的集合中。Windows NT 还提供了一个事件浏览器和编程接口，用于查看日志和检查日志。事件日志函数提供了一系列编写和检查事件日志的方法。

文件函数（File）

文件是计算机存储信息的基本单位，不同的信息可分别存放在不同的文件中。应用程序可使用文件函数对文件进行输入和输出（I/O）操作。

文件安装库函数（File Installation Library）

Win32 API 包含一个文件安装库，应用程序使用它可以更容易地安装文件，使安装程序能分析当前已安装的文件。

文件映射函数（File Mapping）

文件映射函数用于对文件映射对象进行操作。

文件系统函数（File System）

Win32 应用程序依赖文件系统来保存和获取存储设备上的信息。文件系统提供了应用程序在与存储设备相关的个别卷上创建和访问文件及目录时所需的底层支持。

每个文件系统都由一个或多个驱动程序和所支持的动态链接库（定义文件系统的数据格式和特性）组成。它们确定了文件名的约定、安全性及可恢复性的级别，以及输入输出（I/O）操作的一般性能。文件系统函数用于对文件系统进行操作。

句柄和对象函数（Handle and Object）

对象是一个表示系统资源的数据结构，比如表示一个文件、线程或图像。应用程序不能直接访问对象所表示的对象数据或系统资源，而是必须使用对象句柄。对象句柄可用于检查和修改系统资源。每个句柄在一个内部维护的表中都有一项。在这些项中包含资源的地址以及标识资源类型的方法。句柄和对象函数用于对句柄和对象进行操作。

Hook 函数

Hook 是系统消息处理机制中的一部分。在系统消息处理机制中，应用程序可安装一个子程序来监视系统中的消息传送情况，并可处理某些类型的消息（在这些消息到达目的窗口过程之前）。Hook 函数用于对 Hook 进行操作。

lmageHlp 函数

lmageHlp 函数由 IMAGEHLP DLL 提供。lmageHlp 函数可用于 PE 格式的图像。PE 图像由一个兼容的 Win32 连接程序提供，比如由 Microsoft Developer Studio 提供。

超大整数操作函数（Large Integer Operations）

Win32 API 提供了一系列超大整 邮槽是一种单向的内部处理通讯（IPC）机制。Win32 应用程序可以在邮槽中保存消息，邮槽的所有者可以获取保存在其中的消息。这些消息通常是通过网络发送到一台指定的计算机上，或发送到某个指定域中的所有计算机上。域是一组工作站和服务器，共享一个组名。

可以选择使用命名管道来代替邮槽进行内部处理通讯。命名管道是两个进程交换消息的一种简单方法。而邮槽则是一个进程向多个进程广播消息的一种简单方法。需要考虑的重要一点是邮槽使用邮包，而命名管道则不用。邮槽函数可用于创建邮槽、设置或获取邮槽信息。

内存管理函数（Memory Management）

内存管理函数用于分配和使用内存。

管道函数（Pipe）

管道是一段共享内存，用于进程通讯。创建管道的进程称为管道服务程序。连接管道的进程称为管道客户程序。某个进程向管道中写入信息，然后其他进程从管道中读出信息。管道函数用于创建、管理和使

用管道。

电源管理函数（Power Management）

电源管理函数用于对计算机的电源进行管理。

进程和线程函数（Process and Thread）

一个 Win32 应用程序由一个或多个进程组成。在最简单的条件下，一个进程就是一个可执行程序，在该进程的环境中运行一个或多个线程。线程是操作系统分配处理器时间的基本单位。一个线程可以执行进程代码的任何部分，包括正被其他线程执行的部分。一个"纤度"（Fiber）是一个执行单位，必须由应用程序手工调度。"纤度"在调度它的线程环境中运行。

作业对象允许进程组被作为一个单位进行管理。作业对象是可命名、可得到及可共享的对象，用于控制与其相关的进程的属性。在作业对象上完成的操作会影响所有与该作业对象相关的进程。

进程和线程函数包括三类函数：进程和线程函数、作业对象函数、"纤度"函数。

注册函数（Registry）

注册表是一个系统定义的数据库，应用程序和系统构件可使用它来保存和获取配置数据。注册函数用于对注册表进行操作。

字符串处理函数（string Manipulation）

字符串处理函数用于对字符串进行处理。

结构化的异常处理函数（Structured Exception Handling）

异常是在程序执行过程中发生的一种事件，发生异常时需要执行正常的控制流程以外的代码。共有两种异常：硬件异常和软件异常。硬件异常是由 CPU 引发的，可能由于执行了某些指令序列而产生，比如除零操作，或访问一个无效的内存地址。软件异常是由应用程序或操作系统显式地引发。例如，当系统检测出一个无效的参数值时就会引发一个异常。

结构化的异常处理是一种同时处理软件异常和硬件异常的机制。因此，在程序中可用作对硬件和软件异常一起进行处理。使用结构化的异常处理使用户可以完全控制对异常的处理，为调试器提供支持，并且对所有编程语言和机器都是可用的。

同步函数（Synchronization）

Win32 API 提供了各种方法来调整执行过程中的多个进程。同步函数为线程提供了一系列对资源访问进行同步的机制。

系统信息函数（System Information）

系统信息函数用于修改系统的配置、设置和属性。

系统消息函数（System Message）

系统消息函数用于向一些系统部件发送系统消息，比如应用程序、网络驱动器、系统级设备驱动器等。

系统关机函数（System Shutdown）

应用程序可使用系统关机函数将当前的用户退出系统、关机，或锁定工作站。

磁带备份函数（Tape Backup）

备份应用程序可使用磁带备份函数从磁带中读取数据，向磁带中写入数据，初始化磁带，以及获取磁带或磁带驱动信息。

时间函数（Time）

Microsoft Windows 和 Windows NT 提供了各种日期和时间函数，用于获取和设置系统及个别文件的日期和时间。

使用时间函数可以检查和修改日期及时间。

计时器函数（Timer）

计时器是一个内部例程，它反复地测量一个指定的时间间隔（以毫秒为单位）。

计时器函数用于对计时器进行操作。

工具帮助函数（Tool Help）

由"工具帮助库"所提供的函数可使用户更容易地获取有关当前正在执行的 Win32 应用程序的信息，为用户提供工具帮助服务。

窗口站和桌面函数（Window Station and Desktop）

窗口工作站和桌面函数主要是为 Win32 服务的开发人员提供的，用于对新的窗口工作站和桌面功能进行操作。开发由登录用户使用的典型应用程序的开发人员不必考虑窗口工作站和桌面。

Windows NT 4.0 访问控制函数（Windows NT Access-Control）

Windows NT 4.0 访问控制函数用于对安全描述符和访问控制列表（ACL）进行操作。在更高版本的 Windows NT 中也支持这些函数。

Windows NT 4.0 访问控制函数是 Microsoft Win32 提供的三套访问控制函数之一。

WinTrust 函数

WinTrust 函数用于对指定的主题进行指定确认。