
学校的理想装备
电子图书·学校专集
校园网上的最佳资源
深入编程内幕
——Visual C++
BOOK
内部资料 非卖品

一 走进 Visual C++

Visual C++作为一个功能非常强大的可视化应用程序开发工具，是计算机界公认的最优秀的应用开发工具之一。Microsoft 的基本类库 MFC 使得开发 Windows 应用程序比以往任何时候都要容易。本光盘教学软件的目的就是为了让你学会在 Visual C++环境下，利用微软的基本类库 MFC 开发出功能强大的 Windows 应用程序。在本章节的内容当中，我们将向您介绍使用 VC 开发软件需要用到的一些基本概念，使用 MFC 进行编程的基本特点，以及 VISUAL C++集成开发环境提供的一系列编程辅助工具——WIZARD 的使用方法。

1 理解 VC 工程

Visual C++作为一种程序设计语言，它同时也是一个集成开发工具，提供了软件代码自动生成和可视化的资源编辑功能。在使用 Visual C++开发应用程序的过程中，系统为我们生成了大量的各种类型的文件，在本节中将要详细介绍 Visual C++中这些不同类型的文件分别起到什么样的作用，在此基础上对 Visual C++如何管理应用程序所用到的各种文件有一个全面的认识。

首先要介绍的是扩展名为 dsw 的文件类型，这种类型的文件在 VC 中是级别最高的，称为 Workspace 文件。在 VC 中，应用程序是以 Project 的形式存在的，Project 文件以.dsp 扩展名，在 Workspace 文件中可以包含多个 Project，由 Workspace 文件对它们进行统一的协调和管理。

与 dsw 类型的 Workspace 文件相配合的一个重要的文件类型是以 opt 为扩展名的文件，这个文件中包含的是在 Workspace 文件中要用到的本地计算机的有关配置信息，所以这个文件不能在不同的计算机上共享，当我们打开一个 Workspace 文件时，如果系统找不到需要的 opt 类型文件，就会自动地创建一个与之配合的包含本地计算机信息的 opt 文件。

上面提到 Project 文件的扩展名是 dsp，这个文件中存放的是一个特定的工程，也就是特定的应用程序的有关信息，每个工程都对应有一个 dsp 类型的文件。

以 clw 为扩展名的文件是用来存放应用程序中用到的类和资源的信息的，这些信息是 VC 中的 ClassWizard 工具管理和使用类的信息来源。

对应每个应用程序有一个 readme.txt 文件，这个文件中列出了应用程序中用到的所有的文件的信息，打开并查看其中的内容就可以对应用程序的文件结构有一个基本的认识。

在应用程序中大量应用的是以 h 和 cpp 为扩展名的文件，以 h 为扩展名的文件称为头文件。以 cpp 为扩展名的文件称为实现文件，一般说来 h 为扩展名的文件与 cpp 为扩展名的文件是一一对应配合使用的，在 h 为扩展名的文件中包含的主要是类的定义，而在 cpp 为扩展名的文件中包含的主要是类成员函数的实现代码。

在应用程序中经常要使用一些位图、菜单之类的资源，VC 中以 rc 为扩展名的文件称为资源文件，其中包含了应用程序中用到的所有的 windows 资源，要指出的一点是 rc 文件可以直接在 VC 集成环境中以可视化的方法进行编辑和修改。

最后要介绍的是以 rc2 为扩展名的文件，它也是资源文件，但这个文件中的资源不能在 VC 的集成环境下直接进行编辑和修改，而是由我们自己根据需要手工地编辑这个文件。

对于以 ico,bmp 等为扩展名的文件是具体的资源，产生这种资源的途径很多。使用 rc 资源文件的目的就是为了对程序中用到的大量的资源进行统一的管理。

2 MFC 编程特点

如果你曾经使用过传统的 windows 编程方法开发应用程序，你会深刻地体会到，即使是开发一个简单的 windows 应用程序也需要对 windows 的编程原理有很深刻的认识，同时也要手工编写很多的代码。因为程序的出错率几乎是随着代码长度的增加呈几何级数增长的，这就使得调试程序变得非常困难。所以传统的 windows 编程是需要极大的耐心和丰富的编程经验的。

近几年来，面向对象技术无论是在理论还是实践上都在飞速地发展。面向对象技术中最重要的就是“对象”的概念，它把现实世界中的气球、自行车等客观实体抽象成程序中的“对象”。这种“对象”具有一定的属性和方法，这里的属性指对象本身的各种特性参数。如气球的体积，自行车的长度等，而方法是指对象本身所能执行的功能，如气球能飞，自行车能滚动等。一个具体的对象可以有许多的属性和方法，面向对象技术的重要特点就是对象的封装性，对于外界而言，并不需要知道对象有哪些属性，也不需要知道对象本身的方法是如何实现的，而只需要调用对象所提供的方法来完成特定的功能。从这里我们可以看出，当把面向对象技术应用到程序设计中时，程序员只是在编写对象方法时才需要关心对象本身的细节问题，大部分的时间是放在对对象的方法的调用上，组织这些对象进行协同工作。

MFC 的英文全称是 Microsoft Fundation Classes，即微软的基本类库，MFC 的本质就是一个包含了许多微软公司已经定义好的对象的类库，我们知道，虽然我们要编写的程序在功能上是千差万别的，但从本质上来讲，都可以化归为用户界面的设计，对文件的操作，多媒体的使用，数据库的访问等等一些最主要的方面。这一点正是微软提供 MFC 类库最重要的原因，在这个类库中包含了一百多个程序开发过程中最常用到的对象。在进行程序设计的时候，如果类库中的某个对象能完成所需要的功能，这时我们只要简单地调用已有对象的方法就可以了。我们还可以利用面向对象技术中很重要的“继承”方法从类库中的已有对象派生出我们自己的对象，这时派生出来的对象除了具有类库中的对象的特性和功能之外，还可以由我们自己根据需要加上所需的特性和方法，产生一个更专门的，功能更为强大的对象。当然，你也可以在程序中创建全新的对象，并根据需要不断完善对象的功能。

正是由于 MFC 编程方法充分利用了面向对象技术的优点，它使得我们编程时极少需要关心对象方法的实现细节，同时类库中的各种对象的强大功能足以完成我们程序中的绝大部分所需功能，这使得应用程序中程序员所需要编写的代码大为减少，有力地保证了程序的良好的可调试性。

最后要指出的是 MFC 类库在提供的对象的各种属性和方法都是经过谨慎的编写和严格的测试，可靠性很高，这就保证了使用 MFC 类库不会影响程序的可靠性和正确性。

3 使用 Wizard

Visual C++是一种功能强大的通用程序设计语言，它提供了各种向导和工具帮助我们来实现所需的功能，在一定程度上实现了软件的自动生成和可视化编程。下面就为你介绍 VC 集成环境中几个最主要的开发工具的使用方法。

首先要介绍的是 Appwizard 工具，这个工具的作用是帮助我们一步步地生成一个新的应用程序，并且自动生成应用程序所需的基本代码。下面我们就介绍使用 Appwizard 生成一个应用程序的具体步骤。

- 单击 File 菜单 New 菜单项，系统弹出的对话让我们选择所要创建的文件类型，这里的文件分成了 Files，Project，Workspaces，Other documents 四种大类型，每一个类型下面又包含许多具体的文件类型，选中 Projects 标签，标签下的工作区中列出的是各种不同的应用程序类型，比如 dll 类型的动态链接库，exe 类型的可执行程序等，这里选中 MFC Appwizard(exe)选项，表示要创建的是一个使用 MFC 基本类
  库进行编程的可执行程序。如下图 1.1 所示：

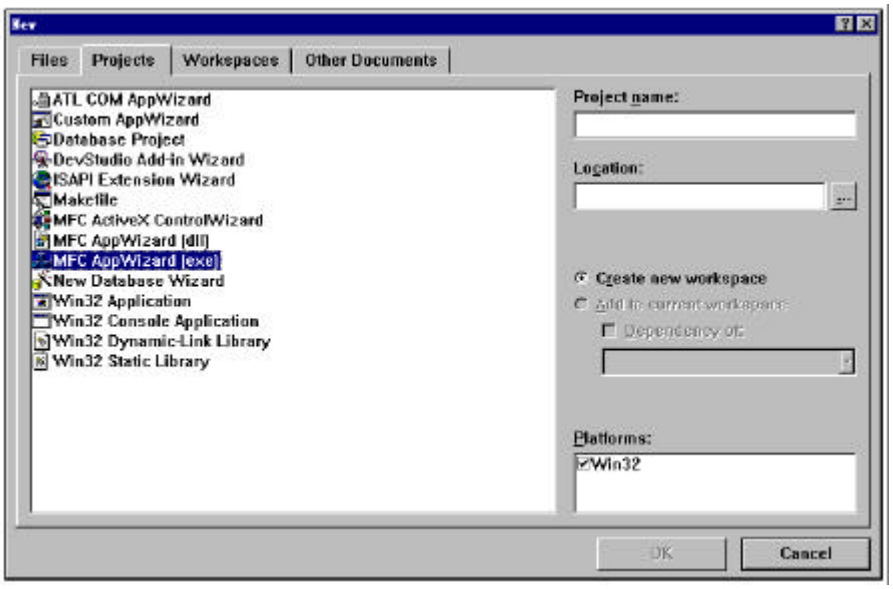


图 1.1

- 选好后在 project name 一栏中为程序起一个名字为 test，在 location 一栏中为程序定义文件存放的目录，对话框右下角的 platforms 一栏中的 Win32 项表示要创建的程序是建立在 32 位的 windows 平台基础上。单击ＯＫ按钮，就启动了使用ＭＦＣ方式开发应用程序的 Appwizard 功能。

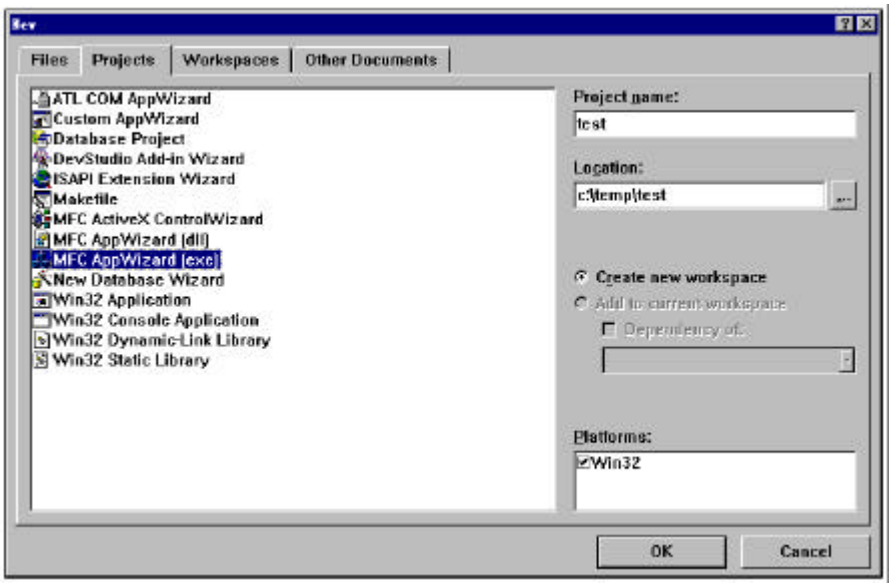


图 1.2

- Wizard 让我们选择程序的类型和程序中的资源所用的语种，这里不妨选择程序类型为单文档界面，语种为英语，然后单击ＮＥＸＴ按钮。

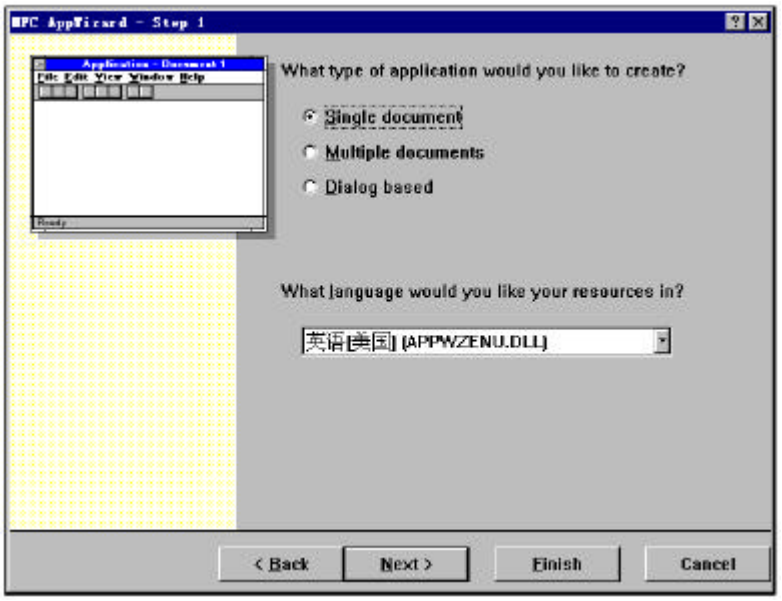


图 1.3

- Wizard 让我们选择是否需要提供数据库方面的支持，这里选择 NONE，然后单击 NEXT 按钮。

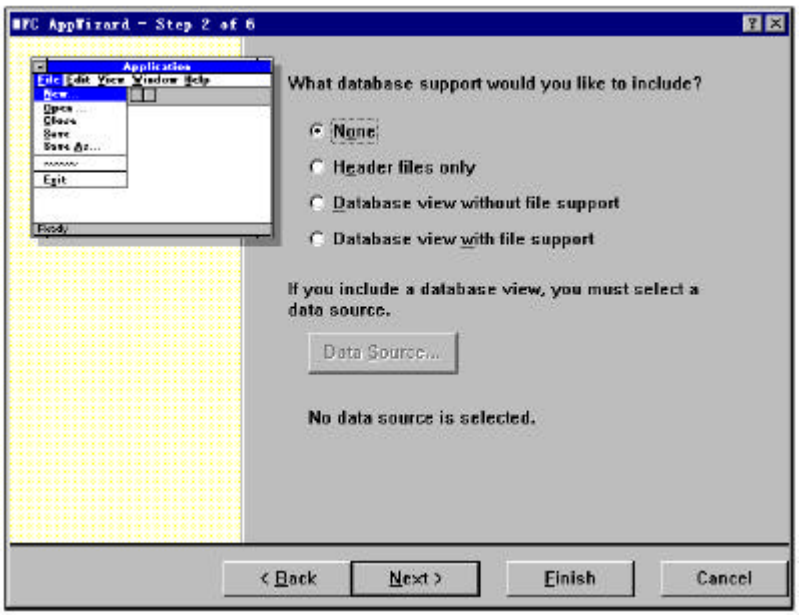


图 1.4

- 下面选择程序中对复合文档的支持，这里选择 NONE。

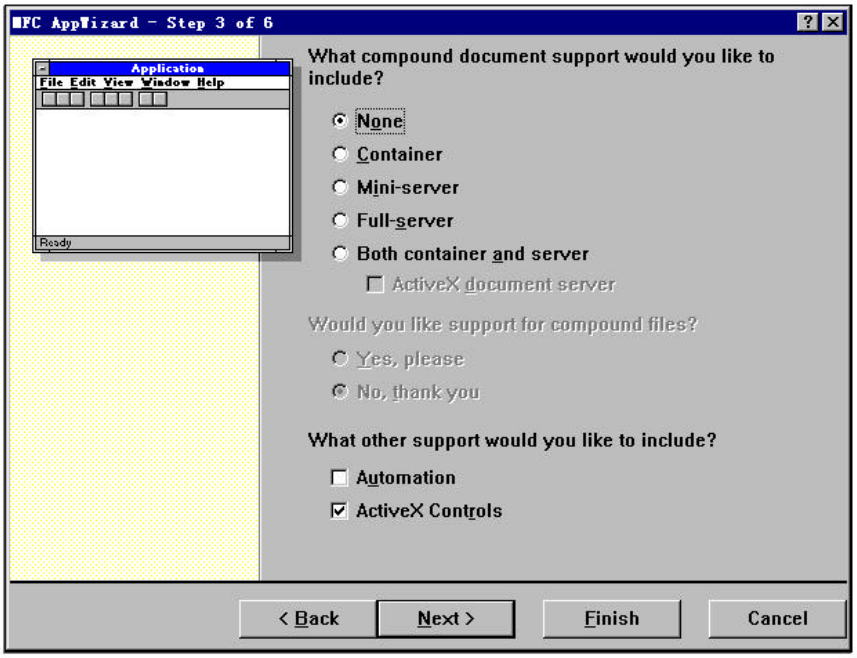
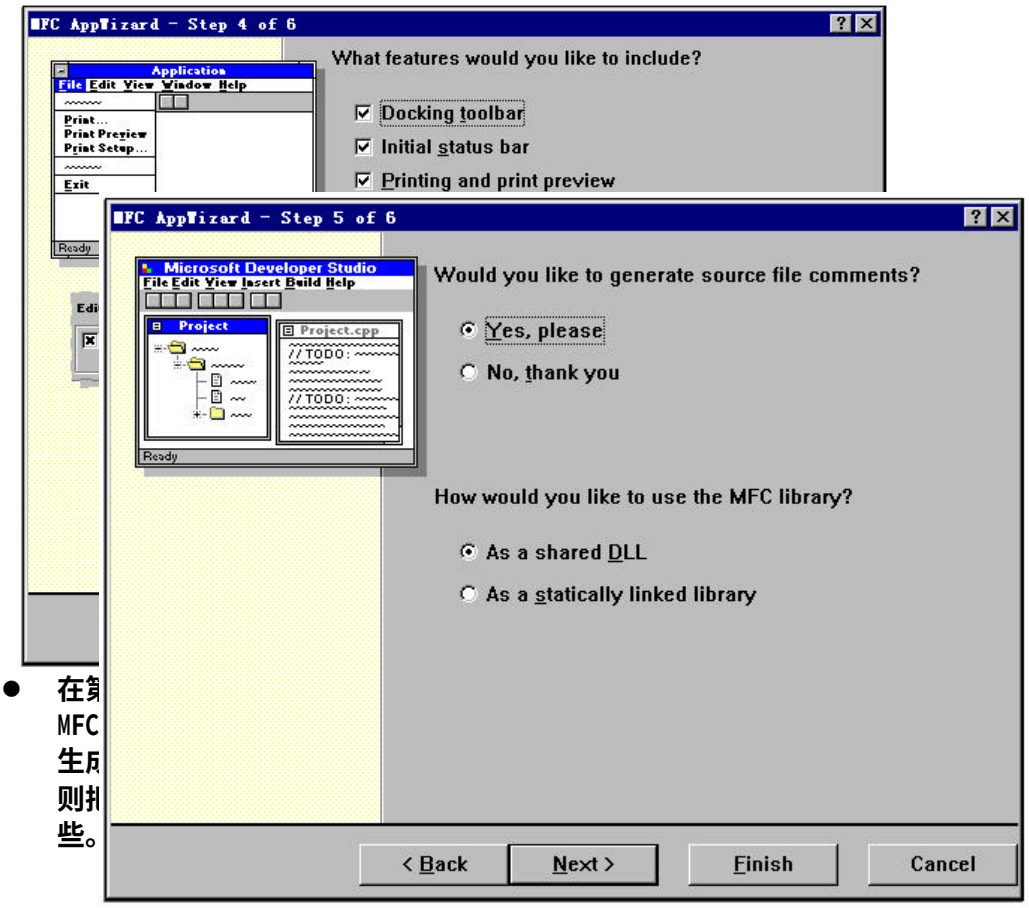


图 1.5

- 接着选择程序的其它一些特性，如提供对 WINSOCK 的支持等。这里对系统的缺省值不作改变,如下图 1.6 所示。单击 NEXT 按钮。

图 1.6



- 在第 使用 MFC 以后 生成 时， 则把 一 些。

图 1.7

- 进入 APPWIZARD 的最后一个步骤，对话框中的提示信息指明了系统将要自动创建的对象和相关文件，以及派生出这些对象的 MFC 的基类等内容。在这一步当中，我们还可以对视图类的基类进行选择，单击 FINISH 按钮。

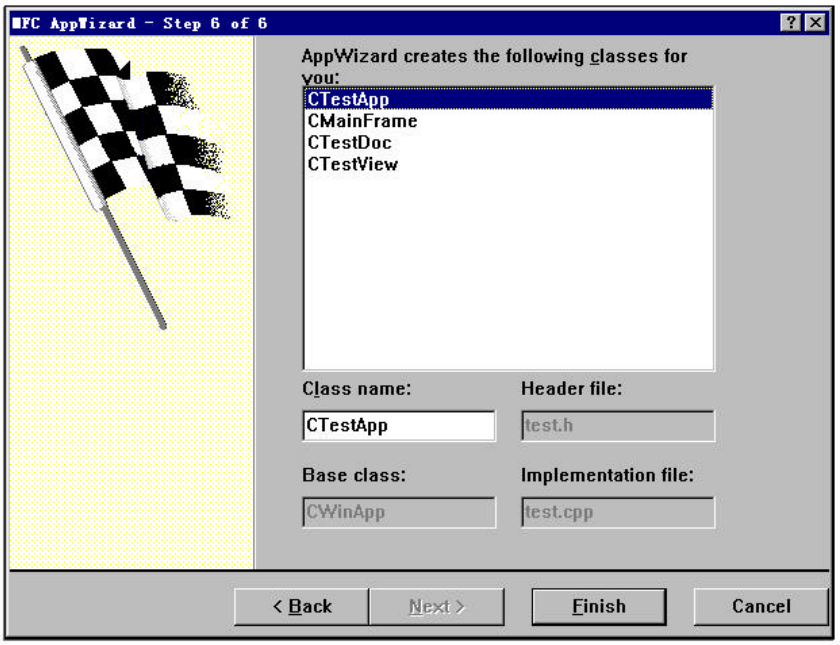


图 1.8

- 游览一下对话框中对将要生成的程序的有关信息的描述后单击 OK 按钮。系统就自动为我们生成一个使用 MFC 基本类库的应用程序的基本框架，在以后将会对这个框架的内容作详细的介绍。

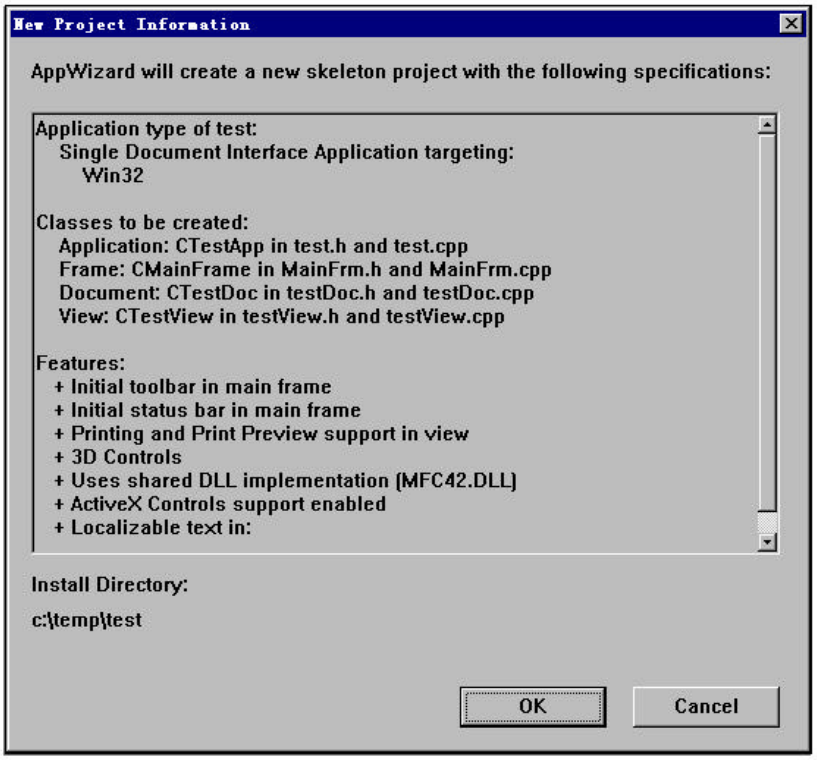


图 1.9

接下来介绍 VC 集成环境中提供的一个很重要的工具 CLASSWIZARD，它主要是用来管理程序中的对象和消息的，这个工具对于 MFC 编程显得尤为重要。 单击 VIEW 菜单的 CLASSWIZARD 项，就可以运行 MFC CLASSWIZARD，在这个对话框中就可以对程序中的对象和消息进行管理了。

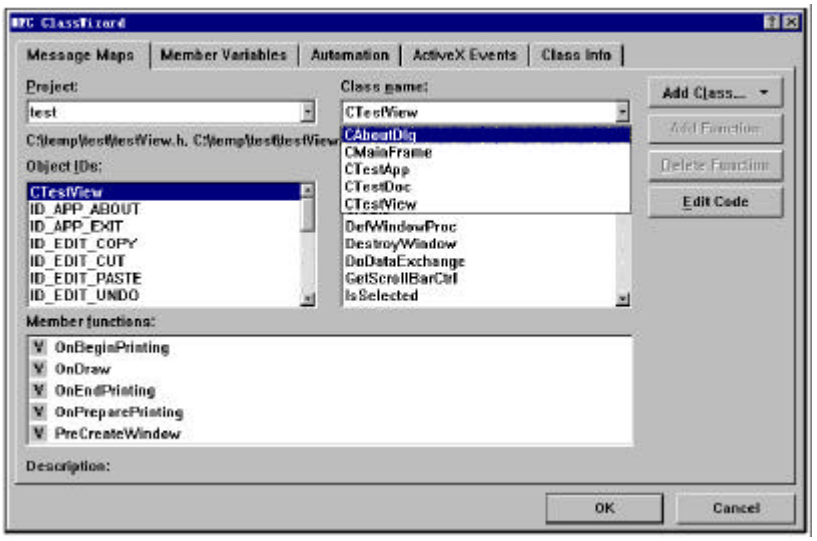


图 1.10

在对话框中的 MESSAGE MAPS 标签下，PROJECT 栏中的内容代表当前程序的名字。CLASSWNAME 下拉列表框列出的就是程序当前用到的所有类的名字，在 MESSAGE 一栏中列出的就是一个选中的类所能接收到的所有的消息，在 WINDOWS 程序设计中，消息是个极为重要的概念，用户通过窗口界面的各种操作最后都转化为发送到程序中的对象的各种消息，下面就向您介绍在 WINDOWS 程序设计中最常用的一些消息：

1 窗口消息：WM_CREATE，WM_DESTROY，WM_CLOSE

我们创建一个窗口对象的时候，这个窗口对象在创建过程中收到的就是 WM_CREATE 消息，对这个消息的处理过程一般用来设置一些显示窗口前的初始化工作，如设置窗口的大

小，背景颜色等，WM_DESTROY 消息指示窗口即将要被撤消，在这个消息处理过程中，我们就可以做窗口撤消前的一些工作。WM_CLOSE wm_close 消息发生在窗口将要被关闭之前，在收到这个消息后，一般性的操作是回收所有分配给这个窗口的各种资源。在 windows 系统中资源是很有限的，所以回收资源的工作还是非常重要的。

2 键盘消息：WM_CHAR，WM_KEYDOWN，WM_KEYUP

这三个消息用来处理用户的键盘数据，当用户在键盘上按下某个键的时候，会产生 WM_KEYDOWN 消息，释放按键的时候又回产生 WM_KEYUP 消息，所以 WM_KEYDOWN 与 WM_KEYUP 消息一般总是成对出现的，至于 WM_CHAR 消息是在用户的键盘输入能产生有效的 ASCII 码时才会发生。这里特别提醒要注意前两个消息与 WM_CHAR 消息在使用上是有区别的。在前两个消息中，伴随消息传递的是按键的虚拟键码，所以这两个消息可以处理非打印字符，如方向键，功能键等。而伴随 WM_CHAR 消息的参数是所按的键的 ASCII 码，ASCII 码是可以区分字母的大小写的。而虚拟键码是不能区分大小写的。

3 鼠标消息：WM_MOUSEMOVE，WM_LBUTTONDOWN， WM_LBUTTONUP， WM_LBUTTONDBCLICK，WM_RBUTTONDOWN， WM_RBUTTONUP，WM_RBUTTONDBCLICK

这组消息是与鼠标输入相关的，WM_MOUSEMOVE 消息发生在鼠标移动的时候，剩余的六个消息则分别对应于鼠标左右键的按下、释放、双击事件，要指出的是 WINDOWS 系统并不是在鼠标每移动一个像素时都产生 MOUSEMOVE 消息，这一点要特别注意。

4 另一组窗口消息：WM_MOVE ，WM_SIZE ，WM_PAINT

当窗口移动的时候产生 WM_MOVE 消息，窗口的大小改变的时候产生 WM_SIZE 消息，而当窗口工作区中的内容需要重画的时候就会产生 WM_PAINT 消息。

5 焦点消息 WM_SETFOCUS，WM_KILLFOCUS

当一个窗口从非活动状态变为具有输入焦点的活动状态的时候，它就会收到 WM_SETFOCUS 消息，而当窗口失去输入焦点的时候它就会收到 WM_KILLFOCUS 消息。

6 定时器消息：WM_TIMER

当我们为一个窗口设置了定时器资源之后，系统就会按规定的时间间隔向窗口发送 WM_TIMER 消息，在这个消息中就可以处理一些需要定期处理的事情。

最后要指出的一点是，在 WINDOWS 环境下，消息的来源是多方面的，最常见的是用户的操作产生消息，系统在必要的时候也会向程序发送系统消息，其他在运行中的程序也可以向程序发送消息。此外，在程序的内部，也可以根据需要在适当的时候主动产生消息，比如主动产生 WM_PAINT 消息以实现需要的重画功能。

上面介绍了 MESSAGE 栏中主要的消息，在 MEMBER FUNCTION 一栏中列出的是目前被选中的类已经有的成员函数。这些成员函数一般说来是与这个类可以接收的消息一一对应的。也就是说，一个成员函数一般总是用来处理某个特定的消息。如果在 MESSAGE 栏中的某个消息在程序中需要处理，但目前还没有相应的类成员函数，比如这里选中 WM_TIMER 这个消息，它目前还没有相应的对应的类的成员函数，

单击 ADD FUNCTION 按钮，

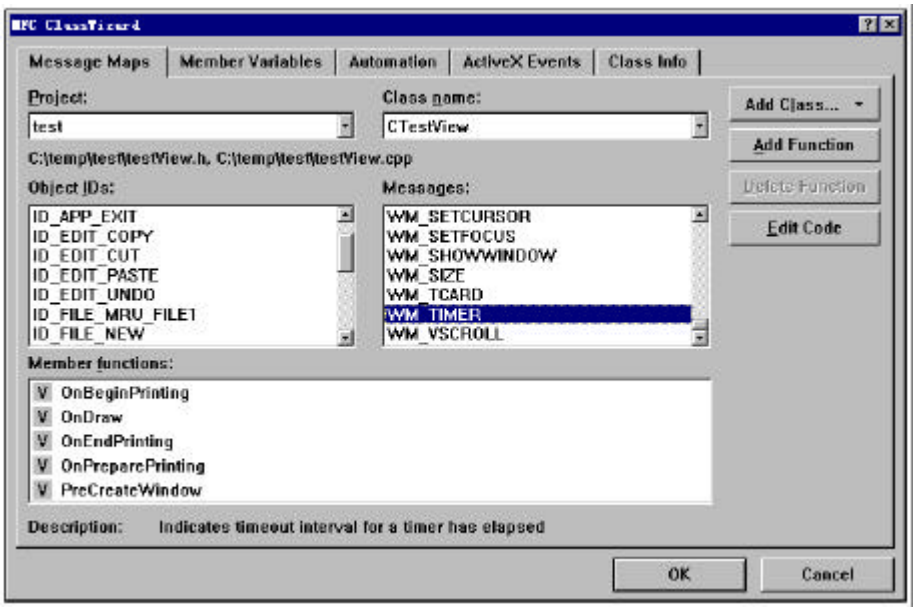


图 1.11

系统就自动为 WM_TIMER 消息在类中添加了对应的成员函数 ONTIMER，单击 EDITCODE 按钮，可以发现系统已经自动生成了完成 ONTIMER 函数所需的基本代码，我们只要在这些基本代码的基础上再添加所需要的代码就可以了。

注意对话框中的 ADD CLASS 按钮，它用来往当前应用程序中添加一个新的类 。

单击后选中 NEW 菜单，

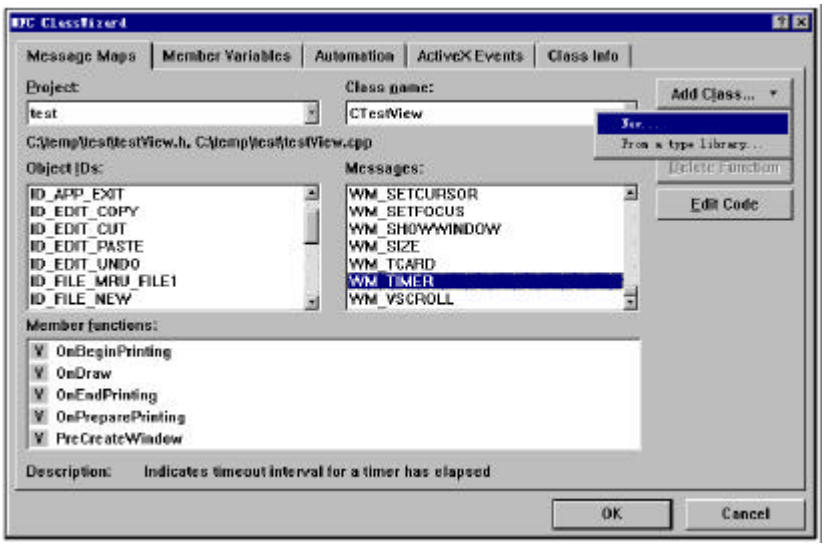


图 1.12

系统弹出了 NEW CLASS 对话框用于生成一个新的类。在这个对话框中需要为类起个名字，设置类文件的名字，另外还要在 BASE CLASS 一栏 的下拉列表框中选择某个已有的类作为基类，设好需要的信息后单击 OK 就生成了一个新的类。

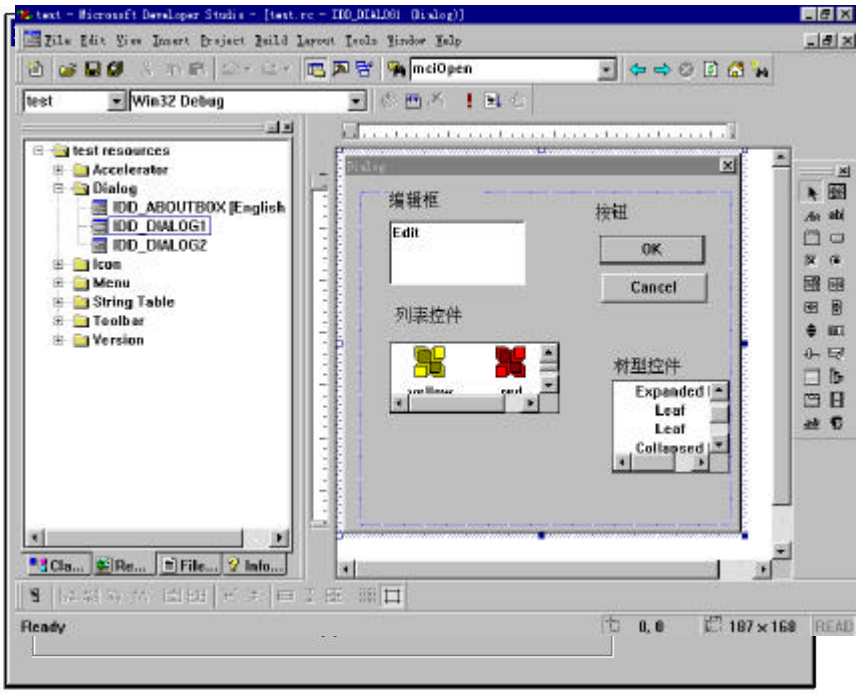


图 1.13

CLASS WIZARD 还有一些很强大的功能，这里就不再详细介绍，你会在不断的学习中慢慢地了解和掌握。

最后介绍一下集成环境提供的一个重要工具 RESOUCR EDITOR，也就是资源编辑器。在 VC 开发的应用程序中要用到大量的位图，菜单，工具条，对话框等各种资源。这些资源对于程序而言是相对独立的，所以可以对它们进行单独的编辑，然后使用在程序中。而 RESOUCE EDITOR 正是为编辑资源提供了一种可视化的开发方法。极大地减轻了程序员的负担。

单击 FILE 菜单的 OPEN 菜单项，然后在对话框中选择打开 TEST.RC 文件，就可以开始使用资源编辑器了。在左边的工作区中按类型列出了程序中用到的所有的资源，双击其中的某个类型，比如双击 MENU 资源，MENU 目录的下面列出的就是系统已经有的 MENU 类型的资源，选中其中一个并双击，在右边的工作区中列出了这个资源当前的样子，我们就可以在工作区中对资源进行可视化的编辑和修改了。

图 1.14

如何添加一个资源呢？单击 INSERT 菜单，选中 RESOURCE 菜单项，系统弹出 INSERT RESOURCE 对话框。如图 1.15。

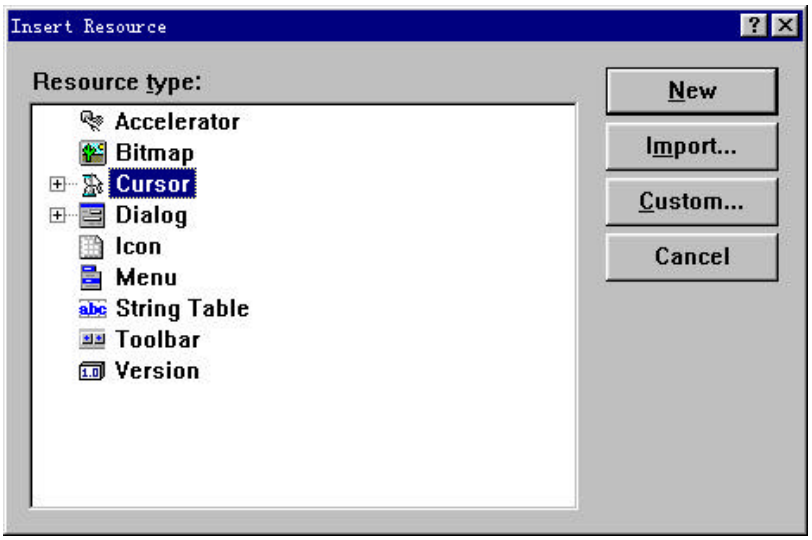


图 1.15

在图 1.15 这个对话框，在这个对话框中选中一种资源类型，比如选择 CURSOR 类型，然后单击 NEW 按钮。在左边的工作区中就出现了我们新生成的资源的标识符，双击这个标识符，在右边的工作区中就可以对这个新的指针形状资源进行可视化编辑了。如图 1.16。

通过这部分内容的介绍，相信您已经对使用 VISUAL C++开发 MFC 应用程序的

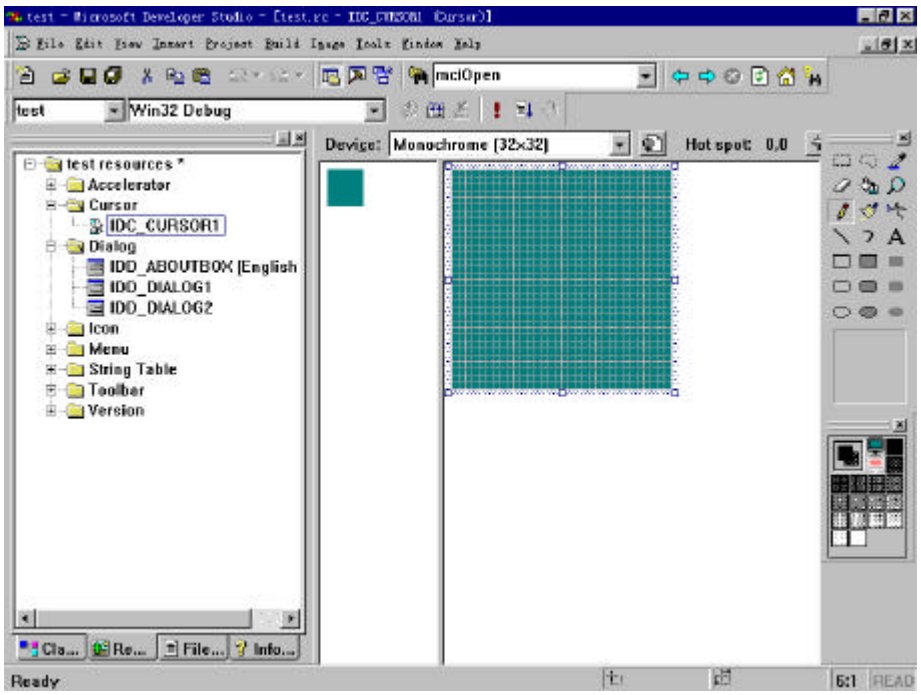


图 1.16

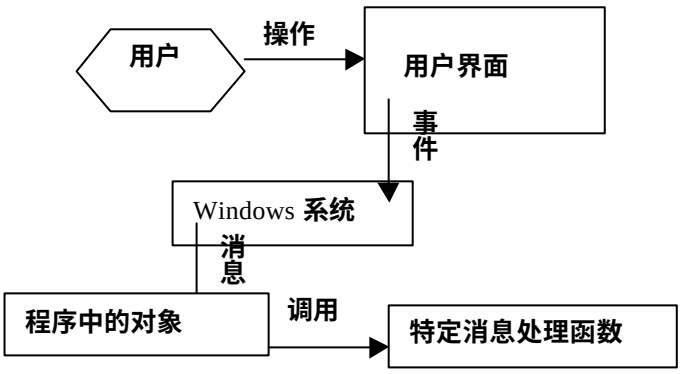
基本步骤有了认识。在下一章的内容当中，我们将结合 WINDOWS 的工作原理，详细地向您解释 MFC 类库的基本结构，以及 MFC 应用程序的基本框架——文档/视图结构。

二 MFC 程序结构分析

1 WINDOWS 程序工作原理

WINDOWS 程序设计是一种完全不同于传统的 DOS 方式的程序设计方法，它是一种事件驱动方式的程序设计模式。在程序提供给用户的界面中有许多可操作的可视对象。用户从所有可能的操作中任意选择，被选择的操作会产生某些特定的事件，这些事件发生后的结果是向程序中的某些对象发出消息，然后这些对象调用相应的消息处理函数来完成特定的操作。WINDOWS 应用程序最大的特点就是程序没有固定的流程，而只是针对某个事件的处理有特定的子流程，WINDOWS 应用程序就是由许多这样的子流程构成的。

从上面的讨论中可以看出，WINDOWS 应用程序在本质上是面向对象的。程序提供给用户界面的可视对象在程序的内部一般也是一个对象，用户对可视对象的操作通过事件驱动模式触发相应对象的可用方法。程序的运行过程就是用户的外部操作不断产生事件，这些事件又被相应的对象处理的过程。下面是 WINDOWS 程序工作原理的示意图。



2 建立应用程序

在介绍 AppWizard 的时候，我们已经建立了一个名字为 TEST 的工程，事实上这个框架程序已经可以编译运行了。在 BUILD 菜单中选择 REBUILD ALL 菜单项，系统开始编译由 APPWIZARD 自动生成的程序框架中所有文件中的源代码，并且链接生成可执行的应用程序。在 BUILD 菜单中选择 EXECUTE 菜单项，应用程序就开始开始运行了，虽然我们没有编写一行代码，但是可以看出由系统自动生成的应用程序的界面已经有了一个标准 WINDOWS 应用程序所需的几个组成部分，我们要做的事情是往这个应用程序添加必要的代码以完成我们所需要的功能。

接下来将要对 WINDOWS 自动生成的这个应用程序框架作详细的介绍，让你对 MFC 方式的 WINDOWS 应用程序的工作原理有全面的认识，只有这样你才会知道应该如何往程序框架当中添加需要的代码。

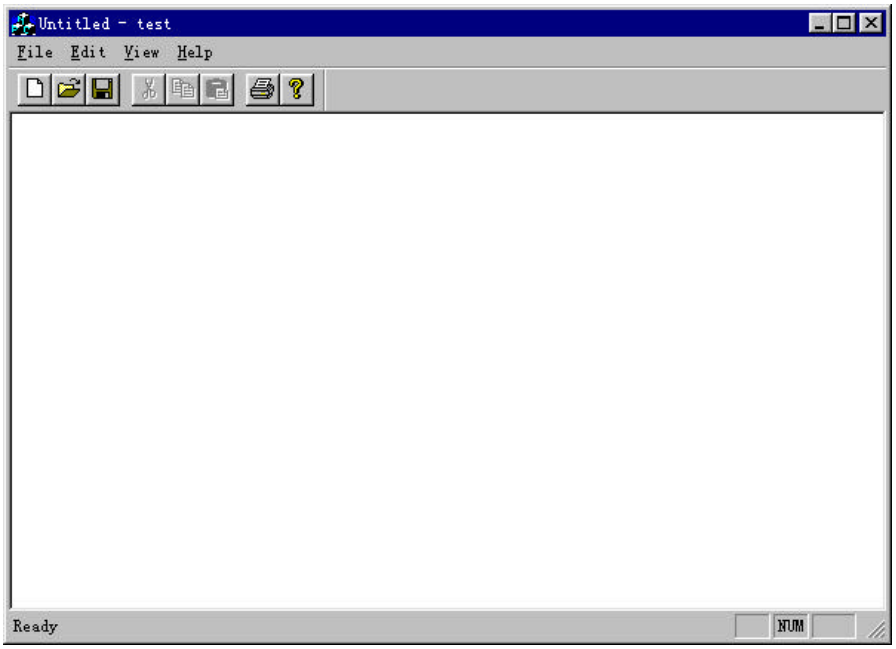
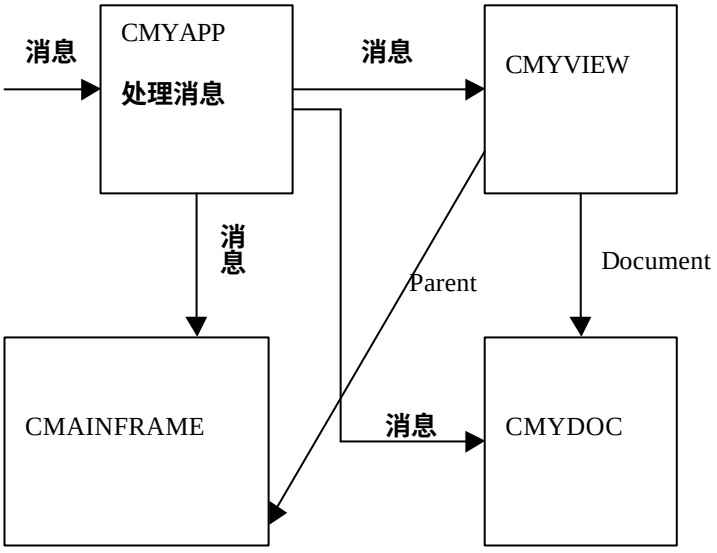


图 2.1

3 程序结构剖析

为了让您对 MFC 方式的程序的框架有一个总体的认识，这里设计了一个表示程序中的主要类之间的关系的图表：



这个图表表示了使用 MFC 方式的应用程序的四个主要类之间的关系，从中可以看出，CMYAPP 类主要的作用是用来处理消息的，它统一管理程序收到的所有的消息，然后把消息分配到相应的对象。CMAINFRAME 是 CMYVIEW 的父类，也就是说视窗 VIEW 显示在主框窗 MAINFRAME 的客户区中。类 CMYVIEW 的作用是显示数据，而数据的来源是类 CMYDOC，在 MFC 程序中，程序的数据是放在文档当中的，而显示数据则是利用视窗方式，文档与视窗分离带来的好处就是一个文档可以同时具有多个视窗，每个视窗只显示文档中的一部分数据，或者以特定的风格显示文档中的数据。文档与视窗分离的另一个好处就是在程序中可以处理多个文档，通过对不同的视窗的处理达到对不同的文档分别处理的目的。

使用过传统的 WINDOWS 编程方法的人都知道，在应用程序中有一个重要的函数 WINMAIN（），这个函数是应用程序的基础，用户的操作所产生的消息正是经过这个函数的处理派送到对应的对象中进行处理。在 MFC 方式的 WINDOWS 应用程序中，用来处理消息的是系统自动生成的 MFC 中的类 CWINAPP 的派生类 CMYAPP，下面就从这个类开始介绍应用程序的框架。

3．1 类 CMYAPP

类 CMYAPP 是应用程序运行的基础，注意这一行代码，可以看出这个类是由 MFC 中的类 CWINAPP 派生来的。在这个类中除了有一般类都有的构造函数，一个重要的成员函数就是 INITINSTANCE，我们知道，在 WINDOWS 环境下面可以运行同一程序的多个实例，函数 INITINSTANCE 的作用就是在生成的一个新的实例的时候，完成一些初始化的工作。注意这一行代码，它的作用就是生成一个 CMYAPP 类型的对象，生成的时候系统会主动调用 INITINSTANCE 函数完成一些必要的初始化工作。

下面研究 INITINSTANCE 函数所做的事情，注意这一行代码，它定义了一个文档模板对象指针 PDOCTEMPLATE，通过 NEW 操作符，系统动态生成了这个文档模板对象，然后使用 ADDDOCTEMPLATE 函数把这个文档模板对象加入到应用程序所维护的文档模板链表当中，这个文档模板 PDOCTEMPLATE 的作用就是把程序用到的框架窗口，CMAINFRAME，文档 CMYDOC，视窗 CMYVIEW 与应用对象 CMYAPP 联系起来。

CMYAPP 类提供了用户与 WINDOWS 应用程序之间进行交流的界面。在生成这个类的对象后，这个对象自动地把自身与 WINDOWS 系统建立联系，接收 WINDOWS 传送的消息，并交给程序中相应的对象去处理，这就免去了程序员许多的工作，使得开发 C++的 WINDOWS 程序变得简单方便。

3．2 类 CMAINFRAME

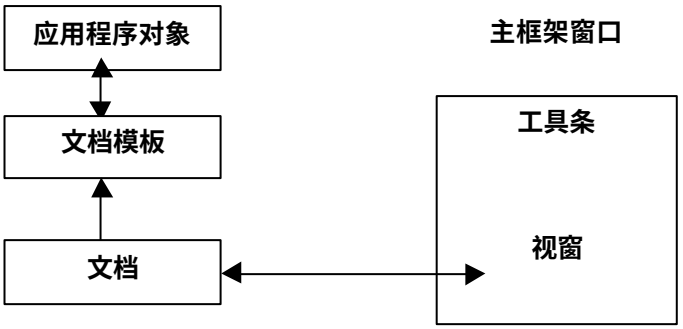
类 CMAINFRAME 是由 MFC 中的 CFRAMEWND 派生来的，所以它也是一个框架窗口。前面已经指出，CMAINFRAME 是类 CMYVIEW 的父类，也就是说 CMYVIEW 类的对象显示在主框架窗口的客户区中。在类 CMAINFRAME 中，系统已经从类 CFRAMEWND 那里继承了处理窗口的一般事件的 WINDOWS 消息，比如改变窗口的大小，窗口最小化等等的成员函数，因此编程的时候程序员不需要再关心此类消息的处理，从而减轻了程序员的负担。当然，如果确实需要重新编写处理此类消息的成员函数，则需要对原有的成员函数进行重载。

在 MFC 程序中，我们并不需要经常对 CMAINFRAME 类进行操作，更多的是对视窗类进行操作，达到对程序中的数据进行编辑和修改的目的。

最后要指出的是，在 MFC 方式的程序中，当程序的一个实例被运行的时候，系统根据前面在 CMYAPP 类中介绍的文档模板对象自动生成类 CMAINFRAME，CMYVIEW，CMYDOC 的对象，而不需要程序员主动地去创建这些类的对象。

3．3 类 CMyView 与 CMyDoc

之所以把 CMyView 类和 CMyDoc 类一起介绍是因为这两个类是密切相关的，
下面的框图可以说明文档与视窗的关系。



在这个框图当中，文档是由文档模板对象生成的，并由应用程序对象管理，而用户则是通过与文档相联系的视窗对象来存储、管理应用程序的数据，用户与文档之间的交互则是通过与文档相关联的视窗对象来进行的。

生成一个新的文档的时候，MFC 程序同时生成一个框架窗口，并且在框架窗口的客户区中生成一个视窗对象作为框架窗口的子窗口，这个子窗口以可视化的方式表现文档中的内容。视窗的重要功能就是负责处理用户的鼠标、键盘等操作，通过对视窗对象的处理达到处理文档对象的目的。

要指出的一点是，WINDOWS 应用程序分单文档界面 SDI 和多文档界面 MDI 两种，在单文档界面中，文档窗口与主框架窗口是同一概念。而这时的视窗对象则是显示在文档窗口的客户区当中。我们先前生成的 TEST 程序使用的就是单文档界面方式，此时文档窗口是主框架窗口，即类 CMAINFRAME 的对象。

下面将以一个例子来说明这两个类之间的关系。

前面已经提到，文档类是用来存放程序中的数据的，所以我们首先在文档类 CMyDoc 中加入一个成员变量用来存放数据。

在左边的工作区用右键单击 CMyDoc 选项，在弹出的菜单中选中 Add member variable 菜单项。

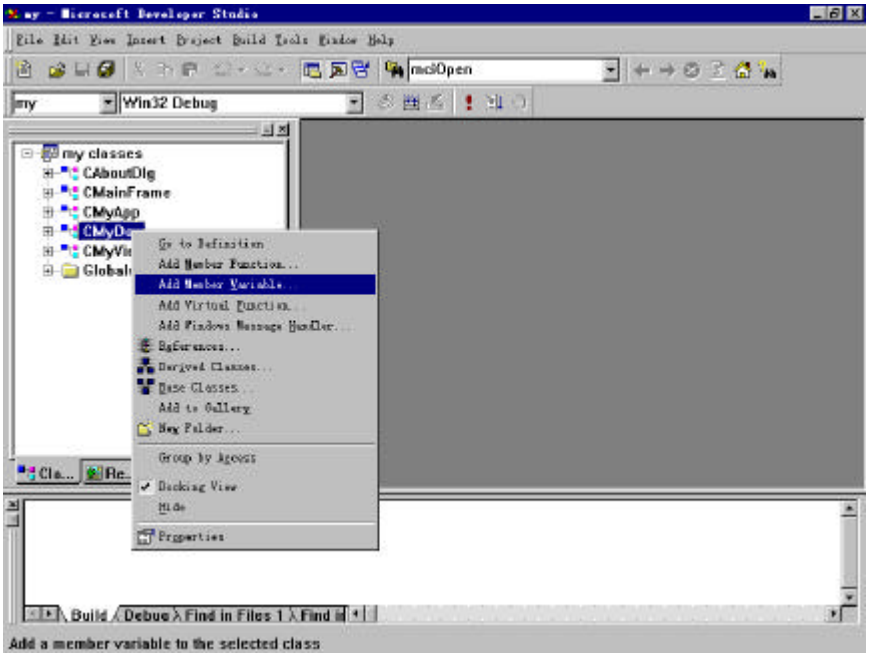


图 2.2

系统弹出 Add Member Variable 对话框。Variable Type 一栏用来输入成员变量的类型。这里设置为 CString，即字符串类型，Variable Declaration 一栏用来输入变量的名

字，这里不妨输入为 mystring，Access 组合框用来设置成员变量的访问权限，缺省为 Public，设好后单击 OK 按钮关闭对话框。如下图 2.3 所示：

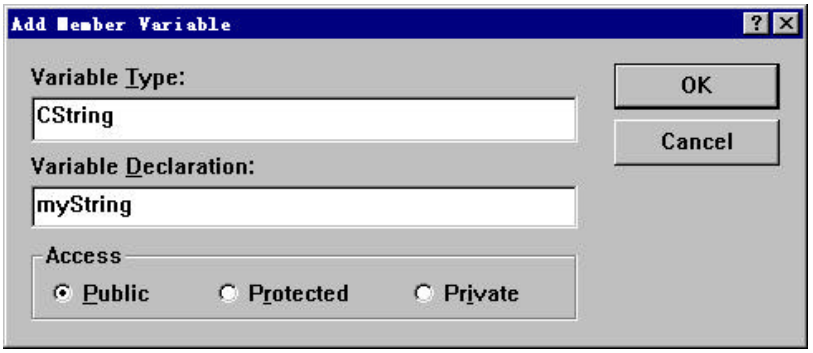


图 2.3

这时，如果打开类 CMyDoc 的头文件、可以发现其中已经自动加入了我们定义的公有变量 mystring。这个变量就可以作为我们的文档类的数据存储空间，因为 mystring 是公有成员，它就可以被文档对应的视窗所处理了。

在 VIEW 菜单中选择 ClassWizard 菜单项，系统打开 MFC ClassWizard 对话框，接下来我们要为视窗类添加处理键盘事件的成员函数。在 Classname 一栏中选中类 CMyView，然后在 messages 一栏中选中消息 wm_char，单击 add function 按钮，系统就自动往 CMyView 类中添加了处理 wm_char 消息的成员函数的框架。单击 edit code 按钮，接下来对 OnChar 这个成员函数进行编辑和修改。

可以看出系统已经自动在这个成员函数中添加了 CMyView 的基类 CView 的 WM_CHAR 消息的处理函数。注意这一行代码：

```
BEGIN_MESSAGE_MAP(CMyView, CView)
//{{AFX_MSG_MAP(CMyView)
ON_WM_CHAR()
ON_WM_LBUTTONDOWN()
ON_WM_CANCELMODE()
//}}AFX_MSG_MAP
// Standard printing commands
ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW,
  CView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

它被放在 mfc 的消息映射宏 BEGIN_MESSAGE_MAP 中，它的作用就是把 windows 系统发来的 WM_CHAR 消息连接到 CMyView 类的成员函数 OnChar 上，即把这个成员函数作为处理 WM_CHAR 消息的过程。接下来我们就往这个成员函数中添加处理 WM_CHAR 消息的具体代码。

首先在 OnChar 函数中添加如下的代码：

```
void CMyView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
// TODO: Add your message handler code here and/or call default
   CMyDoc * pdoc;
   pdoc=GetDocument();
}
```

这段代码的作用是首先定义一个指向文档类对象的指针 pdoc，然后利用 CMyView 类的成员函数 getdocument（）获取指向当前视窗类所对应的文档类对象的指针，并把这个指针赋给定义的文档类型指针 pdoc，这样我们在后面就可以用“pdoc_>mystring”的方式访问文档类中定义的公有数据成员 mystring 了。

接着往函数 OnChar 中添加如下的代码：

```
pdoc_>mystring=nChar;
CClientDC mydc(this);
mydc.TextOut(0,0,pdoc_>mystring,
pdoc_>mystring.GetLength());
```

这段代码中的第一行代码的作用是根据从消息 WM_CHAR 中传来的参数 nchar，也就是键盘中输入的字符的 ASCII 码，把输入的字符添加到文档中的字符串对象 mystring 中。

在介绍第二行代码前要先介绍设备描述表的概念。设备描述表也称为设备上下文，在 windows 环境中，当需要对一个对象，如打印机，屏幕，窗口等进行输出时，就必须先获取这个对象的设备描述表，然后通过这个设备描述表来进行输出。使用设备描述表带来的最大的好处就是输出格式的一致性，因为输出不再是直接针对具体的设备，而是通过统一格式的设备描述表间接地实现。第二行代码的作用就是定义并生成了一个当前视窗的客户区的设备描述表对象 MYDC，其中的参数 THIS 是面向对象语言中的一个重要的关键字。指代成员函数所在类的对象的指针。在生成了视窗的客户区的设备描述表 MYDC 之后，我们可以利用它在视窗的客户区中输出数据了。

这段代码的第三行就是调用设备描述表 MYDC 的方法 TEXTOUT，在视窗的客户区中输出文档中的字符串 MYSTRING 了。

我们在前面曾经指出，一个文档可以对应多个视窗。如果用户通过某个视窗更改了文档中的数据，就比如上面的代码当中，我们通过视窗 CMYWIVE 更改了文档中的字符串对象 MYSTRING，那么系统又如何维护同一文档的不同的视窗显示的数据的一致性呢？我们接着在 OnChar 函数中输入如下的代码：

```
pdoc_>UpdateAllViews(this,0L,0);
```

这行代码的作用就是通知本视窗所在的文档的所有其他的视窗，文档中的数据已经更新，这些视窗应该重新从文档中取回数据用来显示，这样就维持了同一文档的所有视窗的数据的一致性。这一行是视窗类中对文档的数据作了修改以后需要加的一条典型语句。

接下来运行这个程序，在 BUILD 菜单中选择 REBUILD ALL 菜单项来编译连接应用程序，然后单击 BUILD 菜单的 EXECUTE 菜单项运行程序，从键盘上输入一些字符，可以发现这些字符显示在视窗也就是主窗口的客户区当中。

而这些字符的实际位置是存放在文档对象的成员变量 MYSTRING 这个字符串中。改变一下窗口的大小，可以发现显示的数据都没有了，这是因为我们在窗口尺寸改变的时候没有把数据重新输出到窗口的客户区中。关闭应用程序，找到 CMyView 类的成员函数 ONDRAW，在其中添加如下的代码：

```
void CMyView::OnDraw(CDC* pDC)
{
     CMyDoc* pDoc = GetDocument();
     ASSERT_VALID(pDoc);
     // TODO: add draw code for native data here
     pDC_>TextOut(0,0,pDoc_>mystring);
}
```

当窗口的大小改变的时候，应用程序会调用 ONDRAW 这个函数，我们添加的这行代码的作用就是把字符串对象 MYSTRING 重新显示在窗口的客户区当中，这样在窗口大小改变的时候，数据依然显示在窗口客户区中。

再次编译运行这个程序，可以发现窗口大小的改变不再影响数据的显示了。

在前面的内容当中，我们已经介绍了使用 MFC 编制程序的基本结构。MFC 的内容非常丰富，下面我们将针对软件的基本任务：接受用户输入、处理程序输出、进行文件处理以及数据库访问技术，向您介绍如何使用 MFC 编写程序，实现这些基本的功能。

三 深入 MFC 类库

1 处理用户输入

程序从用户那里得到数据，经过自己的处理，再把计算的结果输出到屏幕、打印机或者其他的输出设备上，这是软件工作的基本模型。消息和键盘消息是最基本的输入信息，除此之外，MFC封装了一系列的使用户可以进行可视化的输入的界面对象，比方说对话框以及可以布置在上面的编辑框、按钮、列表控件、树形控件等等。使程序支持用户输入的手段更加丰富。

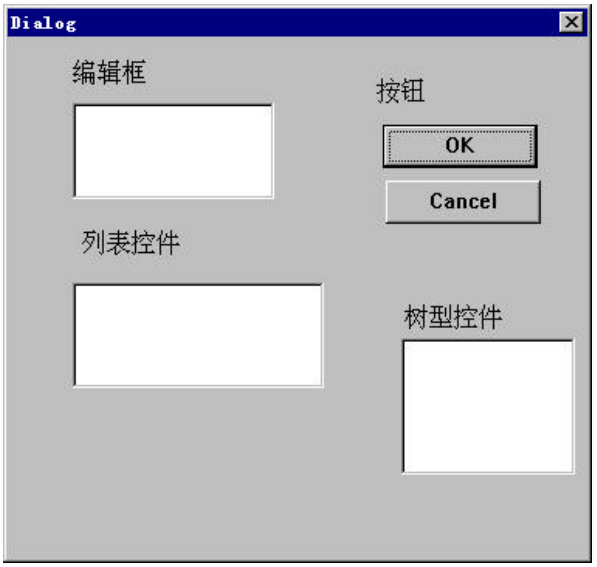


图3.1

1．1 定义对话框资源

下面我们通过一个例子来介绍如何设计一个基于对话框的界面，接受用户输入的数据，并且将它们以图形化的方式显示出来。我们将制作这样的一个有用的程序。它访问一个保存歌曲曲目的数据库，用户可以通过对话框让用户定义一个曲目表，并选定一张背景图，然后在一个没有系统菜单的窗口客户区上滚动显示曲目的名字。在许多的娱乐场所的大屏幕上我们都可以看到类似的东西。在这部分的内容当中，我们着重介绍如何获得和处理用户输入数据的工作，在后面的处理用户输出的内容当中，我们还将使用这个例子来说明如何在屏幕上进行输出工作。

下面我们来介绍如何定义一个对话框资源。在WORKSPACE窗口当中RESOURCE一页，在DIALOG小图标上面单击鼠标右键，弹出菜单，选择INSERT命令,如图3.2所示：

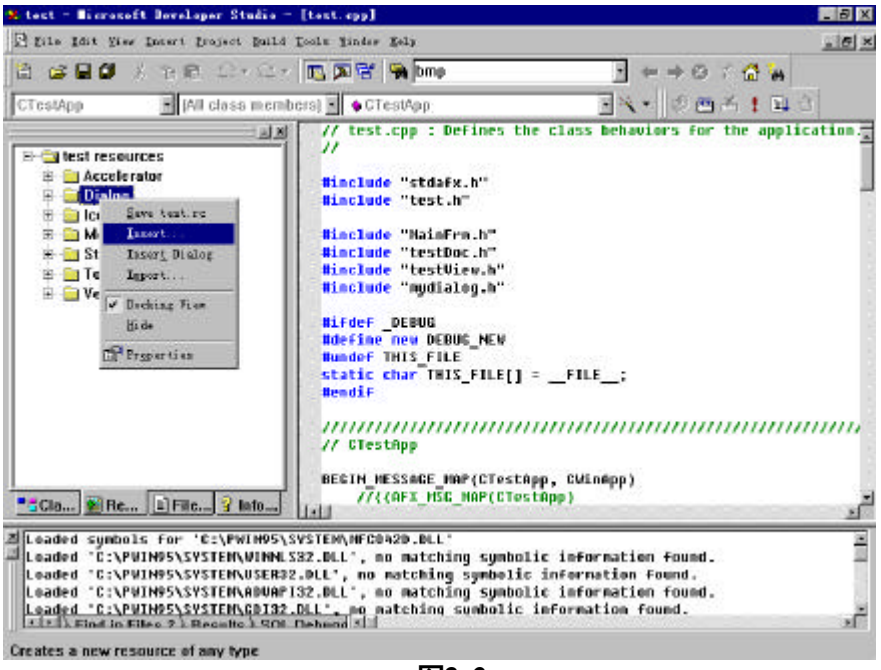


图3.2

接下来在弹出的资源类型对话框中选择DIALOG，表示添加一个对话框资源。单击NEW按钮。

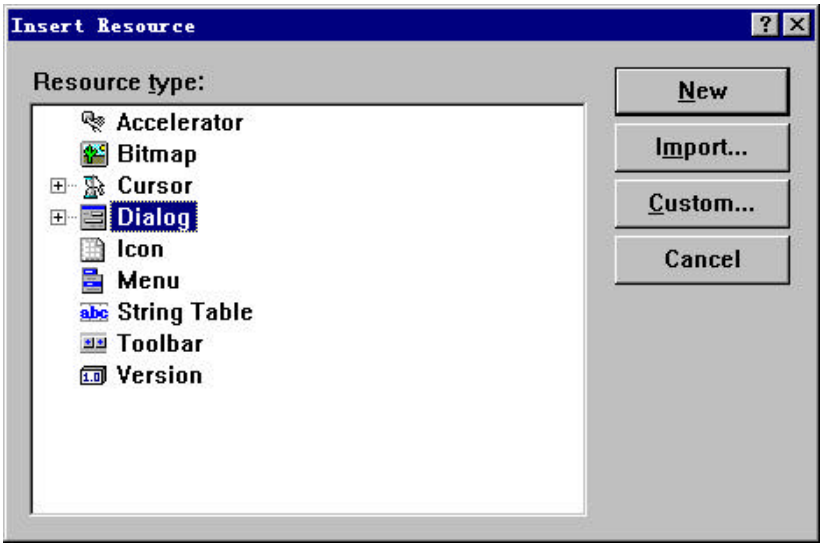


图3.3

接下来我们就开始布置这个对话框。对话框上面已经有了两个按钮：OK和CANCEL。如下图3.4所示。它们的作用分别是确认用户通过对话框进行的输入和取消前面的输入工作。把它们的标题分别改为确定和取消。布置其它的控件。每一个对话框以及对话框上面的每一个控件都有一个ID号码，它们的定义包括在RESOURCE.H 这个头文件当中。比方说，这个对话框的ID号是IDD_DIALOG2，确认按钮的ID号是IDOK，取消按钮的ID号是IDCANCEL，MFC将通过ID号来访问这些资源。单击DIALOG工具条上面的TEST按钮可以测试对话框运行的效果。需要注意的是我们这里我们定义的对话框只是一个资源，如果要使这个对话框真正实现它的功能，必须在程序当中定义一个使用这个资源的对话框类。

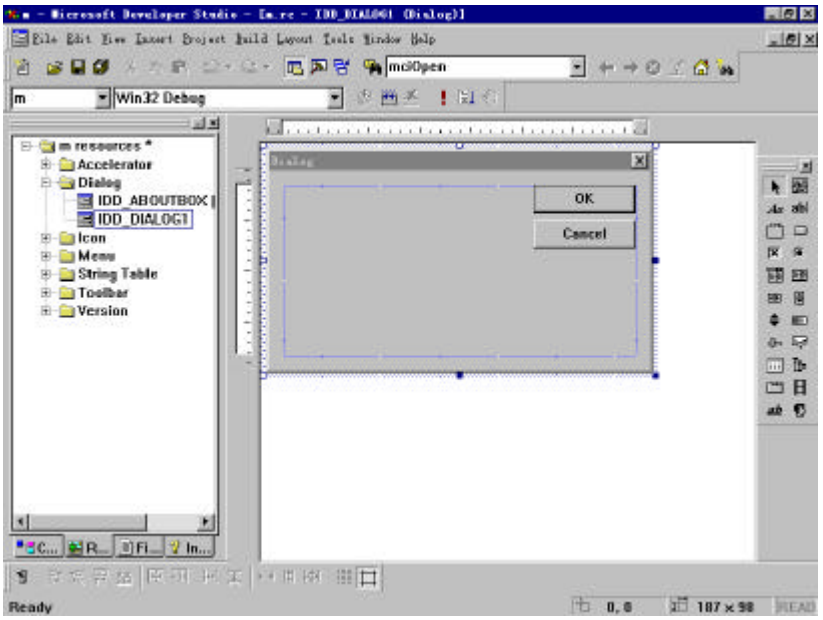


图3.4

1．2 定义对话框类

下面我们就定义一个对话框类。在VIEW菜单当中选择CLASS WIZARD命令，单击ADD CLASS按钮，在弹出的菜单当中选择NEW命令，在NAME一栏当中输入新类的名字，在BASE CLA

SS列表框当中选择需要继承MFC当中的哪一个类。在DIALOG ID列表框当中选择对话框资源的ID号码，在这个实例当中，我们不使用OLE AUTOMATION，所以在这个组框当中选择NONE。在FILE NAME一栏显示的是这个类的定义写在哪一个文件当中。

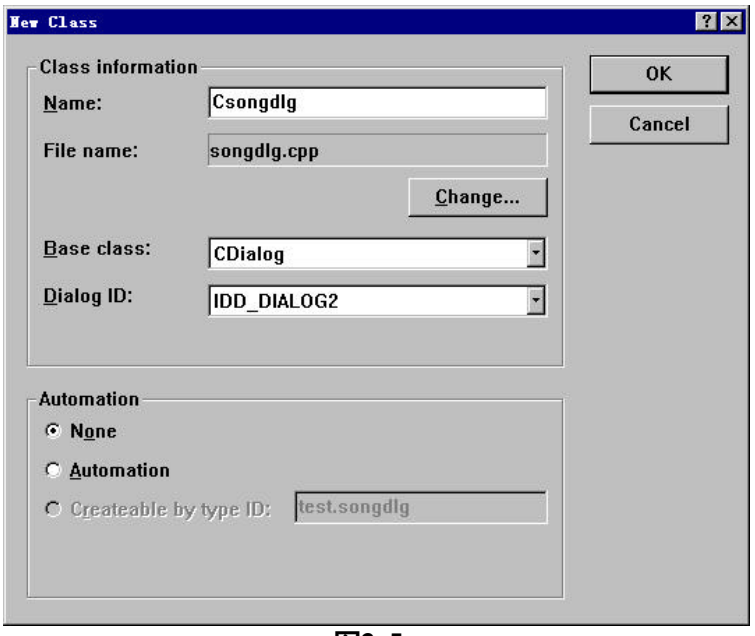


图3.5

单击图3.5中所示CHANGE按钮，在HEADER FILE和IMPLEMENTATION FILE当中分别敲入新类的声明和定义分别写在哪个文件当中，单击OK按钮确认，这样我们就完成了对新的对话框类的定义。单击OK 按钮，CLASS WIZARD将按照我们刚才的要求进行对话框类定义的工作。打开WORKSPACE，选择FILE VIEW一页，在SOURCE FILES和HEADER FILES组当中到CLASS WIZARD已经新建了两个文件，并将它们加入了工程当中。SongDlg.h当中内容是CSongDlg这个类的声明，SongDlg.cpp这个文件当中的内容是这个类的实现。但是目前的程序只是包含了实现一个对话框的最基本功能的代码，调用这个对话框类的DoModal函数之后可以运行它。但是用户通过对话框进行的所有的输入工作都不会被接受。

下面，我们就着手完成实现对话框接受用户输入功能的工作。这里核心的工作就是实现对布置在对话框当中的控件的控制。控制又可以分两种类型：第一种是与界面上的控件交换数据，在对话框中的某些响应函数当中编写取出用户在对话框当中输入的数据。比方说在用户单击了确认输入的按钮，触发了该按钮的单击事件的时候，我们就要从输入新歌的编辑框当中取出曲目字符串保存到数据库当中，并将其显示在曲目列表当中。

我们可以使用MFC提供的一种叫做对话框数据交换（DDX）的机制来从编辑控件当中取出数据。在MFC的对话框类CDialog中已经封装了这种机制。它的工作原理就是在对话框资源中的编辑框和对话框类的一个成员变量之间建立连接。然后由MFC自动地完成在成员变量和控件之间的数据交换工作。首先打开CLASSWIZARD，选中MEMBER VARIABLE这一页，在CLASS NAME列表框当中选择CSongDlg，选择曲目编辑框的ID号IDC_EDIT1，单击ADD VARIABLE按钮。

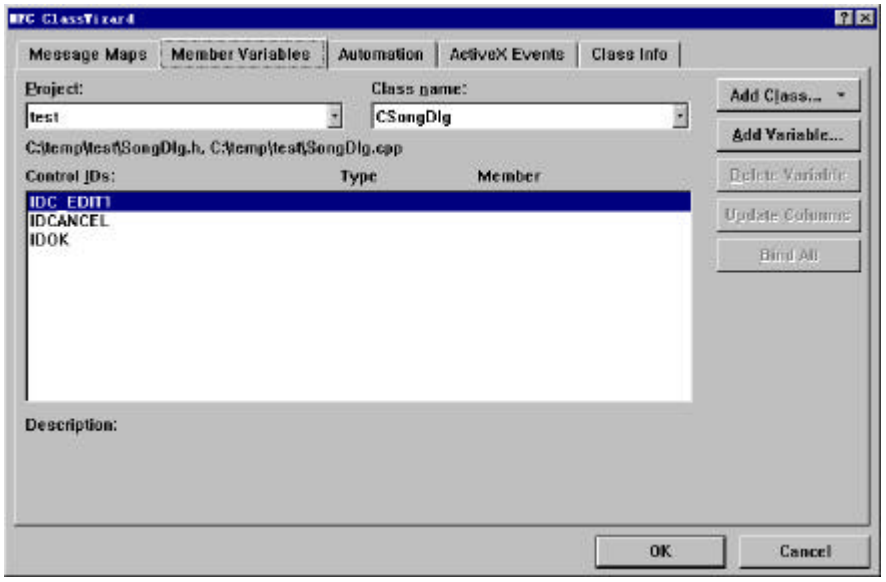


图3.6

在MEMBER VARIABLE NAME 一栏当中敲入变量的名字，在CATEGORY列表框当中可以选择变量的类型，VALUE表示生成一个数据变量，CONTROL类型的变量可以被用来对控件资源进行另一种类型的控制。它的类型依赖于前面选中的控件资源，比方说如果为一个编辑框控件成一个CONTROL类型的成员变量，那么它只能是CEdit类型的。我们将在后面的内容当中具体地介绍如何使用CONTROL类型的成员变量。

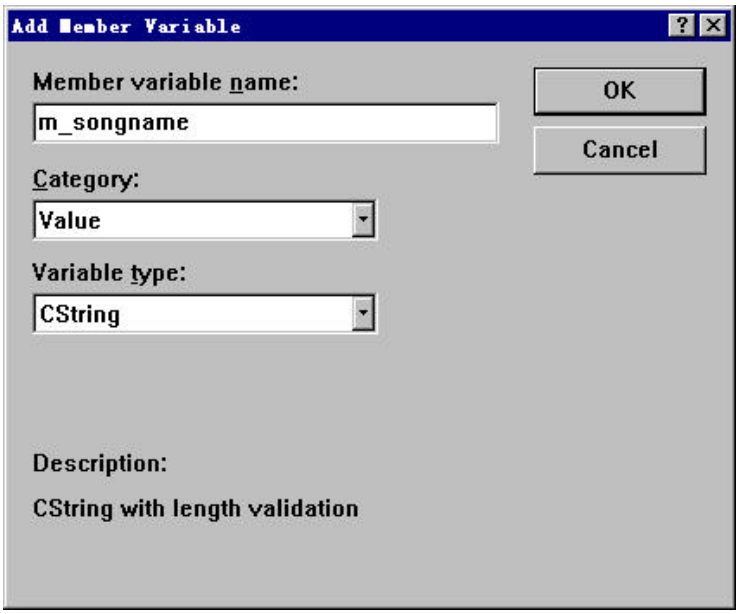


图3.7

生成一个VALUE变量，它的数据类型是字符串。单击OK按钮。这时WIZARD就自动地添加了进行对话框数据交换所有的代码。打开对话框类的头文件和实现文件，我们发现当中增加了一个CString类型的成员变量：

```
// Dialog Data
    //{{AFX_DATA(CSongDlg)
    enum { IDD = IDD_DIALOG2 };
    CString    m_songname;
    //}}AFX_DATA
```

并且在建构函数当中对这个变量进行了初始化：

```
CSongDlg::CSongDlg(CWnd* pParent /*=NULL*/)
```

```
    : CDialog(CSongDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSongDlg)
    m_songname = _T("");
    //}}AFX_DATA_INIT
}
```

在新生成的对话框类CSongDlg 中有如下一个虚函数：

```
virtual void DoDataExchange(CDataExchange* pDX);
                                    //DDX/DDV support
```

DoDataExchange函数就是对话框类和对话框资源进行DDX数据交换的函数。在对话框初始化的时候或者在程序中调用UpdateData()函数的时候，这个函数将会被调用。DDX_TEXT这个函数可以处理多种类型的数据成员变量与控件资源之间的数据交换。这中间包括int,uint,long,DWORD,CString,float,double等。PDX这个参数是一个指向一个CDataExchange对象的指针通过它我们可以设置进行数据交换的方法。比方说：数据交换的方向。这段代码就可以通过PDX的这个标志志判断数据交换的方向是从变量到控件还是从控件到变量，然后进行不同的处理。进行数据交换之后，程序当中就可以通过成员变量来使用用户输入的数据了。

对控件资源的另外一种类型的控制就是要操纵界面控件的外观。比方说，我们可以通过生成一个CONTROL类型的成员变量来控制对话框当中的列表控件。和VALUE类型变量的添加方法一样，我们可以使用CLASSWIZARD生成一个CListControl 类型的对象，在DoDataExchange当中增加了这样的代码：

```
DDX_Control(pDX, IDC_LIST1, m_listCtrl1);
```

DDX_CONTROL也是对话框数据交换机制提供的一个函数，它的作用和DDX_TEXT大致一样。使用刚才定义的控件对象m_listCtrl1,就可以对列表框资源进行操纵了。

当对话框开始运行的时候，我们需要从数据库当中取出已经入库的曲目的名字将其显示在曲目列表框当中。这个工作应该在对话框响应WM_INITDIALOG消息的时候来做。使用CLASS WIZARD来添加这个消息响应函数。在左边的列表框当中选定CSongDlg这个类，在消息列表框当中选定对话框初始化消息，单击ADD FUNCTION按钮，WIZARD就自动地在这个类的声明当中重载了基类的这个成员函数并且在实现文件当中加入了函数体。单击EDIT CODE 按钮，就可以在函数体当中加入我们自己的代码了。

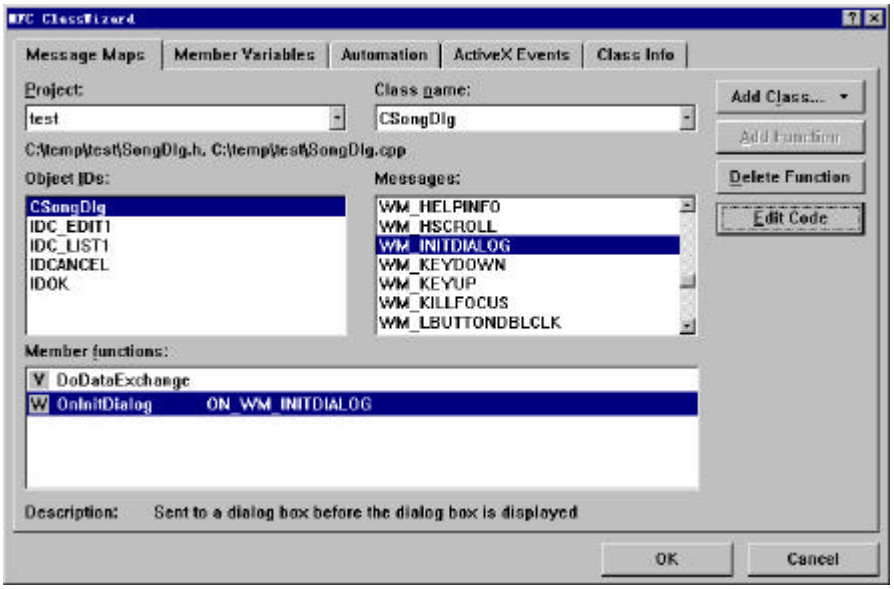


图3.8

在响应WM_INITDIALOG消息的处理函数CSongDlg::OnInitDialog中添加如下一段代码：

```
COleVariant          var;
LV_ITEM              lvitem;
CString              Name("song_name");
char           str[50];
lvitem.iItem = 0;
if(globalRS_>IsOpen())
globalRS_>Close();
CString strQuery = _T("Select * from ");
strQuery += "SONGS";
globalRS_>Open(dbOpenDynaset,strQuery);
globalRS_>m_bCheckCacheForDirtyFields = FALSE;

if(globalRS_>IsOpen())
{
     if(!globalRS_>GetRecordCount())
          return 0;
     globalRS_>MoveFirst();
     while(!globalRS_>IsEOF())
     {
           var =globalRS_>GetFieldValue(_T("[")
                                    + Name + _T("]"));

     lvitem.mask = LVIF_TEXT | LVIF_IMAGE| LVIF_DI_SETITEM;
     lvitem.iItem++ ;
     lvitem.iSubItem = 0;
     strcpy(str , (LPCTSTR)CString(V_BSTRT(&var)));
     lvitem.pszText =str;
     lvitem.iImage = 0;
     m_originsonglist.InsertItem(&lvitem);
     globalRS_>MoveNext();
     }
}
if(globalRS_>IsOpen())
globalRS_>Close();
```

这里使用了DAO技术来访问数据库并使用读出的字符串向列表控件当中添加条目。关于DAO技术的使用方法，我们在其他的章节当中会有详细地介绍。我们关心的是下面这段代码：

```
     lvitem.mask = LVIF_TEXT | LVIF_IMAGE| LVIF_DI_SETITEM;
     lvitem.iItem++ ;
     lvitem.iSubItem = 0;
     strcpy(str , (LPCTSTR)CString(V_BSTRT(&var)));
     lvitem.pszText =str;
     lvitem.iImage = 0;
     m_originsonglist.InsertItem(&lvitem);
```

它执行了对列表控件添加条目的操作。这里需要用到WIN32提供的一个结构：LV_ITEM。我们可以从VC的HELP中找到其定义：

```
typedef struct _LV_ITEM {
UINT   mask;
int    iItem;
```

```
int    iSubItem;
UINT   state;
UINT   stateMask;
LPTSTR  pszText;
int    cchTextMax;
int    iImage;       // index of the list view item担 icon
LPARAM lParam;      // 32_bit value to associate with item
} LV_ITEM;
```

为了添加一个条目，我们首先在在这个结构当中填写条目的信息，然后把它传给列表对象的添加条目函数InsertItem就可以了。

接下来的这段代码位于响应中曲目列表框当中删去选定曲目的按钮单击事件当中。要实现从列表控件当中删去的条目的操作，只要把需要删去的条目的索引号传递给列表对象的删去条目函数DeleteItem就可以了。

```
int totalNum;
totalNum = m_selsonglist.GetItemCount();
int step = 0;
LV_ITEM         lvitem;

lvitem.mask = LVIF_TEXT | LVIF_IMAGE| LVIF_DI_SETITEM;
lvitem.iSubItem = 0;
lvitem.iImage = 0;
while(step <= totalNum)
{
     if(m_selsonglist.GetItemState(step,LVIS_SELECTED))
     {

          m_selsonglist.DeleteItem(step);
     }
     step++;
}
```

我们在这总结一下对对话框上面的控件进行控制所需的工作：

首先在对话框类当中定义成员变量，然后调用对话框的成员变量的函数来操纵界面控件。

2 有关屏幕输出

2．1 设备上下文工作原理

在绝大多数的WINDOWS应用都需要在屏幕上显示自己的数据。由于WINDOWS是一个设备无关的操作系统，所以任何向屏幕上进行输出的功能都要间接地通一个叫做设备上下文(device context)的对象来完成。我们向设备上下文提出屏幕输出的要求，然后由WINDOWS自己来调用具体的输出设备的驱动程序来完成实际的输出工作。围绕设备上下文，MFC提供了一系列与其配合使用的绘图对象，这其中包括画笔对象、刷子对象以及字体对象等等。它们的工作模型是这样的：首先对设备上下文对象——我们简称为DC对象——进行设置，然后选择进行屏幕输出所需要的工具，最后用DC对象的输出函数绘制图形。屏幕输出的目标一般都是窗口的客户区，它是一个万能的输出区域，可以接受无论是文本、位图、还是其他类型的数据（比方说OLE对象）。

2．2 实例绘图原理剖析

在有关用户输入部分的内容当中，我们曾经介绍过一个实例，它访问一个保存歌曲曲目的数据库，用户可以通过对话框让用户定义一个曲目表，并选定一张背景图，然后在一个没有系统菜单的窗口客户区上滚动显示曲目的名字。我们已经介绍了通过对话框介绍用户输入的方法，接下来就着重介绍如何把用户输入的信息在屏幕上显示出来。

我们将用户选定的字符串在一张背景图上面滚动输出。前面已经介绍了使用设备上下文进行工作的基本模型。即：首先选择绘图的工具，然后调用DC的绘图函数来进行绘制。在WINDOWS当中，每次窗口的外观发生改变的时候都会发出一个WM_PAINT消息，窗口的重绘工作都是在响应这个消息的处理函数当中进行的。可以使用CLASSWIZARD来添加这个消息响应函数。之后，就可以在这个函数当中进行屏幕输出了。还有什么时候会触发重绘事件呢?在程序中调用CWND的UpdateWindow 和RedrawWindow数的时候都会触发重绘事件。我们还可以直接使用SendMessage函数向一个指定的窗口送出重绘消息。另外调用CWND的Invalidate函数可以指示重绘的时候是否需要擦去背景，如果使用InvalidateRect函数还可以设置客户区的无效区域，系统重绘的时候将只把该区域的内容重新绘制，我们首先在窗口的客户区上帖一张位图，然后滚动输出文本。如何实现滚动输出呢我们的方法是在程序中设置一个定时器，在定时器计时已满事件WM_TIMER触发的时候来，调用REDRAWWINDOW函数，触发重绘事件，我们只要在它的消息响应函数ONPAINT当中重新绘制背景，擦去原来的文本，然后不断的改变文本输出的位置就可以达到目的了。您可能会重绘整个背景的做法会很耗费了过多的系统时间并且可能产背景的闪烁。这种担心是必要的。在WINDOWS95当中，系统对重绘的机制进行了优化，在我们没有指定无效区域的情况之下，系统自己会选择一个最小的无效区域，只对这一区域进行重绘。

2．3 绘图操作实现

下面介绍绘图操作的源程序使您对设备上下文的使用有一个大致的了解。

首先生成一个设备上下文。CPaintDC是MFC提供的一个从CDC继承出来的类。使用它有什么好处呢?如果直接使用CDC的话，我们需要首先调用CWnd的BeginPaint函数为重绘工作做一些准备工作，在完成绘制之还要用EndPaint函数表示结束绘制工作。所有的绘图操作都必须在这两个函数之间完成。CPaintDC封装了这两个函数，自动地对它们进行调用，使用者无须再去进行这些调用。

```
CPaintDC dc(this);
BITMAP bm;
m_bitmap_>GetBitmap(&bm);

CDC dcImage;
if (!dcImage.CreateCompatibleDC(&dc))
     return;
CBitmap* pOldBitmap = dcImage.SelectObject(m_bitmap);
dc.BitBlt(0, 0, bm.bmWidth, bm.bmHeight,
          &dcImage, 0, 0, SRCCOPY);
```

以上这段代码完成向屏幕上面输出位图的工作。首先根据资源生成一个位图对象，然后生成成一个和CPaintDC一致的内存DC对象，在内存DC当中选择这个位图。BITMAP是一个WIN32提供的位图结构，我们将这幅位图的信息保存在这个结构当中。这样做的原因是由于在使用到位图的位置及大小信息。BITMAP结构的定义如下：

```
typedef struct tagBITMAP {  /* bm */
    int     bmType;
    int     bmWidth;
    int     bmHeight;
    int     bmWidthBytes;
```

```
    BYTE    bmPlanes;
    BYTE    bmBitsPixel;
    LPVOID  bmBits;
} BITMAP;
```

作好这些准备工作之后。调用DC对象的BitBlt函数把位图从内存DC当中贴到绘图DC当中来。前四个参数指示了位图在目的DC上的位置和大小。第五个参数是原来保存位图的内存DC的地址。接下来的两个参数是从位图的哪一点开始进行拷。最后这个参数设置该位图和屏幕上当前内容的相互关系，SRCCOPY的意思是拷贝过来覆盖原来的内容。这个参数还有其他的许多选择比方说取反操作或者异或操作，设置不同的参数可以获得丰富的效果。

下面介绍如何输出文本。首先对DC对象进行设置：

```
dc.SetBkMode(TRANSPARENT);
dc.SetTextColor(RGB(0 , 155 , nowX*nowX));
```

这里把文本输出的背景置为透明，然后设置输出文本的前景颜色。

下面这段程序的意思是为将要输出的文本选择字体。

```
LOGFONT logfont;
memset(&logfont, 0, sizeof(logfont));
logfont.lfWeight = 50;
logfont.lfHeight = 50;
lstrcpy(logfont.lfFaceName, "黑体");
nowFont.CreateFontIndirect(&logfont);
dc.SelectObject(&nowFont);
```

首先声一个WIN32提供的字体信息结构。为其分配内存空间。再按照我们的要求填写这个字体结构，设置字体的宽度，设置字体的高度，选择字体种类，根据这个结构生成一个CFONT字体对象，让DC对象选中这个字体对象，最后使用DC的文本输出函数来输出一个字符串。

总而言之，进行屏幕输出的规则如下：

第一 必须通过CDC对象进行屏幕输出；

第二 设置DC对象的输出属性；

第三 选择绘图工具

第四 用CDC对象的绘图函数。

有关屏幕输出的内容就介绍到这里。

2．4 有关屏幕映射方式

在一般的情况之下，我们都以像素作为绘图的单位，我们称之为设备坐标。我们在进行绘图操作的时候，不可避免的要用到设备坐标系。

WINDOWS 提供了几种映射方式，或称坐标系。可以通过它们来和设备上下文相联系。比方说不管是什么样的显示设备，如果我们需要在上面显示一个 2 英寸高，2 英寸宽的矩形，该怎样处理呢？这就要依赖于我们所设定的坐标系。如果我们指定了 MM_TEXT 方式，这时坐标原点就位于屏幕的左上角，X 轴和 Y 轴的方向分别指向我们面对屏幕的右方和下方，它的绘图单位是像素，如果一英寸对应 72 个像素的话，我们就需要这样绘制这个矩形：

```
DC.Rectangle(CRect( 0,0,72*2,72*2));
```

所以我们如果我们指定了 MM_LOENGLISH 方式，那么一个绘图单位就是百分之一英寸，坐标原点仍然位于屏幕的左上角，但是 X 轴和 Y 轴 的方向恰好和 MM_TEXT 方式下的轴方向相反，同样完成绘制上面提到的矩形的工作，我们就需要写出这样的代码：

```
DC.Rectangle(CRect(0,0,200,_200));
```

可见，坐标系的选择对我们编写程序有很大的影响。

此外，在有些时候，我们需要在几个不同的坐标系下面工作，那么还需要在进行在这些坐标系之间的转换工作。所以 ，我们有必要在这里详细介绍以下 WINDOWS 的坐标映射方法。

一般来说，最常用的就是 WM_TEXT 方式。在 WM_TEXT 坐标方式下面，坐标被映射到了像素，X 的值向右方递增，Y 的值向下递增，并且可以通过调用 CDC 的 SetViewpotOrg 函数来改变坐标原点。下面的代码把屏幕映射方式设为 MM_TEXT 方式，并且把坐标原点设在（300，300）处：

```
DC.SetMapMode(MM_TEXT);
DC.SetViewportOrg(CPoint(300,300));
```

另外，WINDOWS 提供了一组非常重要的比例固定的映射方式，在这些映射方式下面，我们可以改变它的坐标原点，却无法改变它的比例因子。对于 MM_LOENGLISH 映射方式，我们已经知道它的 X 值是向右递减的，Y 的值是向下递减的，所有的固定比例的映射方式都遵循这一原则。它们的比例因子也各不相同。我们列表如下：

映射方式	逻辑单位
MM_LOENGLISH	0.01 英寸
MM_HIENGLISH	0.001 英寸
MM_LOMETRIC	0.1 毫米
MM_HIMETRIC	0.01 毫米
MM_TWIPS	1/1440 英寸

最后一种映射方式 MM_TWIPS 常常用语打印机，一个’twip’单位相当于 1/20 个点（一点近似与 1/72）英寸。例如，如果指定的 MM_TWIPS 一个社单位，那么对于 12 点大小的字模来说，字符的高度为 12x20，即 240 个 twip。

除了固定比例的映射方式，WINDOWS 还提供了比例可变的映射方式，在这种映射方式下面，我们除了可以改变它们比例因子之外还可以改变比例因子。借助于这样的映射方式，当用户改变窗口的尺寸的时候，绘制的图形的大小也可以根据比例发生相应的变化；同样，当我们翻转某个轴的时候，他们所绘制的图像，也以另外的一个轴为轴心进行翻转。这样的映射方式有两种：MM_ISOTROPIC 和 MM_ANIOTROPIC。

在 MM_ISOTROPIC 方式下，纵横的比例为 1：1，换句话说，无论比例因子如何变化，我们画出的图形不会改变自己的形状。但是在 MM_ANIOSTROPIC 方式下面，X 和 Y 的比例因子可以独立地变化，图形的形状可以发生变化。

我们分析下面这段程序：

```
void CAView::OnDraw(CDC *pDC)
{
     CRect clientDC;
     GetClientRect(clientRect);
     pDC_>SetMapMode(MM_ANISOTROPIC);
     pDC_>SetWindowExt(1000,1000);
     pDC_>SetViewportExt(clientRect.right,_clientRect.bottom);
     pDC_>SetViewportOrg(clientRect.right/2, clientRect.bottom/2);
     pDC_>Ellipse(CRect(_500, _500, 500, 500));
}
```

这段代码的功能是这样的，首先取得窗口客户区矩形的大小，然后用 SetWindowExt 和 SetViewportExt 函数设定比例，结果窗口尺寸的大小被设为 1000 个逻辑单位高和 1000 个逻辑单位宽，坐标原点被设为窗口的中心，在这样的设置之下，绘制出一个半径为 500 个逻辑单位的椭圆。

在这里如果将映射方式改变为 MM_ISOTROPIC 那么就将画出一个圆。圆的直径是窗口举行宽和高的最小值。

下面我们给出逻辑单位到设备单位的公式：

X 比例因子 = X 视口范围/X 窗口范围

Y 比例因子 = Y 视口范围/Y 窗口范围

设备 X = 逻辑 X *X 比例因子 + X 坐标原点偏移
设备 Y = 逻辑 Y *Y 比例因子 + Y 坐标原点偏移

当我们设定了设备上下文的映射方式之后，就可以直接使用逻辑坐标作为其参数了，但是从 WM_MOUSEMOVE 消息所获得的鼠标的坐标值是设备坐标。许多其他的 MFC 库函数，尤其是类 CRect 的成员函数，只接受设备坐标。所以我们有时要利用 CDC 的 LPtoDP 和 DPtoLP 在逻辑坐标和设备坐标之间进行转换的工作。

下面我们列出进行坐标映射工作的时候所要遵循的一些规则：

- 可以认为 CDC 的所有成员函数都以逻辑坐标作为参数，但和 CRect 有关的函数例外。
- 可以认为 CWnd 的成员函数都以设备坐标作为参数。
- 所有的 HIT_TEST 操作都应该考虑设备坐标。
- 以逻辑坐标的形式来保存数据，否则用户对窗口进行滚动操作的时候，这个数据就不再有效了。

3 文件处理

几乎所有的软件都需要将程序当中的信息保存在磁盘存储器上面。这些信息可能是程序运行的初始化数据，或者是程序中计算得到的结果，还可能是程序经常用到的资料。从磁盘存储器上存取数据的工作往往是通过文件操作或者数据库操作来完成的。关于数据库操作的内容，我们将在后面的章节当中进行详细的介绍，在下面的内容中，我们主要讨论VC如何实现一般意义上的数据存取工作。

VC是面向对象的开发平台，在使用MFC编写的程序中，我们定义和生成了各种各样的对象，通过它们之间的协同工作完成程序的功能。所以在MFC中，程序的存取工作的核心内容就是如何实现这些对象的持续化。一个可以实现持续化的对象知道如何保存和载入它们自己的数据。比方说，在程序使用MFC文档/视图结构的时候，如果为文档对象提供正确的持续化功能，它将在您需要的时候自动地保存和恢复自己的数据，并且保持用户最新的修改结果。需要注意的是对象的持续化同样是将数据保存到磁盘文件中去，它的好处在于MFC对二进制文件的存取过程进行了封装，使用来实现对象存取的程序代码得到了简化。

当然，如果您更喜欢直接操作文件进行数据的存取工作，MFC也提供了更为直接的渠道。CFile这个类封装了所有对文件的常用操作，使用CFile对象处理文件会比使用API函数简单得多。

3．1 对象持续化简述

在MFC当中，对象的持续化功能主要是通过文档/视图结构当中文档对象的序列化机制来实现的。下面，我们将详细介绍如何使用序列化机制来实现对象的持续化。

序列化，简单地说就是向一个持久性的存储媒体——如磁盘文件保存对象或读取对象的过程。可以实现序列化的类——即从CObject继承而来的类，有一个叫做Serialize的成员函数，序列化工作主要是在这个函数当中进行的。我们使用一张示意图来说明序列化的原理。

MFC使用一个类型为CArchive的归档对象充当磁盘文件与程序中的对象的中介。归档对象总是与一个CFile对象相关联，从中获得一些进行序列化所需要的信息，比如说文件名，以及存取标志等。对象的持续化的主要工作就是将自己的成员变量或者当前状态保存起来。我们可以使用经过重载的流入和流出操作符直接向归档对象中保存或者取出变量成员的值，而将这些数据保存到磁盘文件中的工作由CArchive对象指示CFile对象来完成。当用户在打开或保存拥有文档对象数据的文件或者使用文档对象的Open 、

Save、Save As菜单命令时，MFC便会自动调用Serialize函数。

使类实现序列化，需要五个步骤：

1、从Cobject 类或其派生类派生用户类；

2、在类声明中使用DECLARE_SERIAL宏；

3、重载Serialize 函数；

4、定义不用变量的构造函数；

5、在类实现文件中使用宏IMPLEMENT_SERIAL。

在下一节中，我们将以一个具体的实例来详细说明如何实现类的序列化。

3．2 实例分析

在描述了类序列化的基本原理之后，让我们来看一个具体的序列化实例DrawLine。启动这个实例。

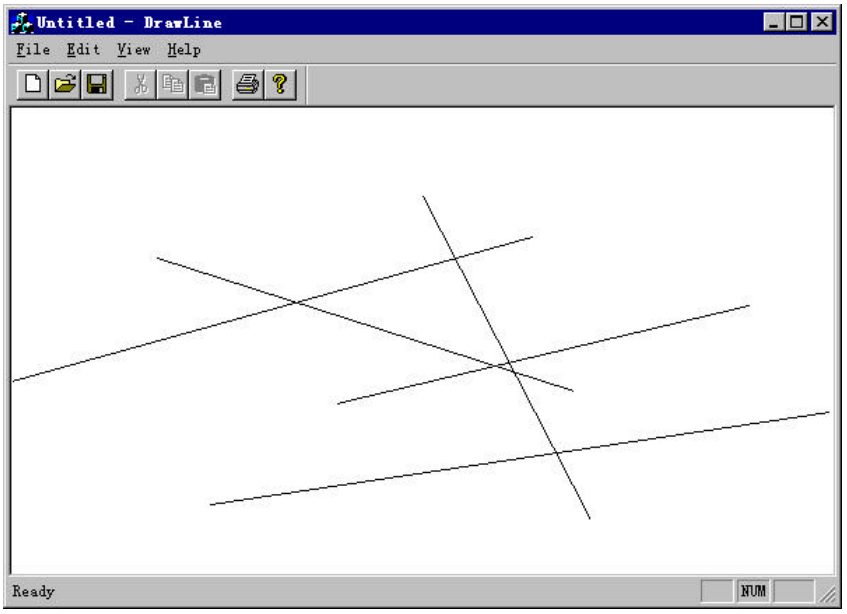


图3.9

在这个实例中，我们将完成对直线的简单绘制。在视图按下鼠标左键，然后拖动鼠标到一个新位置松开鼠标，程序就画出一条直线。再来看看这个程序的基本构成。

除了基本的文档类和视类之外，我们还有一个直线类CLine，它有四个int成员变量m_x1,m_y1,m_x2,m_y2，用来记录直线两个端点的X轴和Y轴方向坐标，此外有一个Draw成员函数,Draw是根据直线的以上四个成员变量，在视图客户区中绘出直线。在WorkSpace的ClassView中双击类CLine的Draw函数，则可以看到Draw的实现。

```
void CLine::Draw (CDC *PDC)
{
   PDC_>MoveTo (m_x1, m_y1);
   PDC_>LineTo (m_x2, m_y2);
}
```

我们对视类CDrawLineView的OnLButtonDown,OnMouseMove,OnLButtonUp 三条消息进行了处理，在WorkSpace的ClassView中同样可看到它们的实现。

此外我们在文档类CDrawLineDoc中有一个成员变量m_LineArray,它是用来记录我们在视图客户区所画的直线。函数GetLine是根据索引取得m_LineArray中的一条直线，GetNumLines则是取得直线的总数的。

在了解了基本的程序结构之后，下面对直线对象进行序列化处理。

1 从 Object 类中派生并使用宏 DECLARE_SERIAL

打开定义CLine这个类的头文件line.h，可以看到这个类是从CObject 类派生出来的。要对CLine类实现序列化，需要在类的声明中加入宏DECLARE_SERIAL的调用，并在类的实现文件中，加入宏IMPLEMENT_SERIAL 的调用。CObject 类拥有基本的序列化功能，通过对此类的继承实现可以获得这些功能，此外一个无参数的构造函数是不可缺少的。

我们打开Line.h后，在CLine类定义中第一句就可以是DECLARE_SERIAL(CLine),这个宏不需要加分号。

2 重载 Serialize 成员函数

我们要实现序列化，先对其进行改造，在WorkSpace的ClassView中选择CLine类，单击鼠标右键，选择Add Member Function增加一个成员函数：

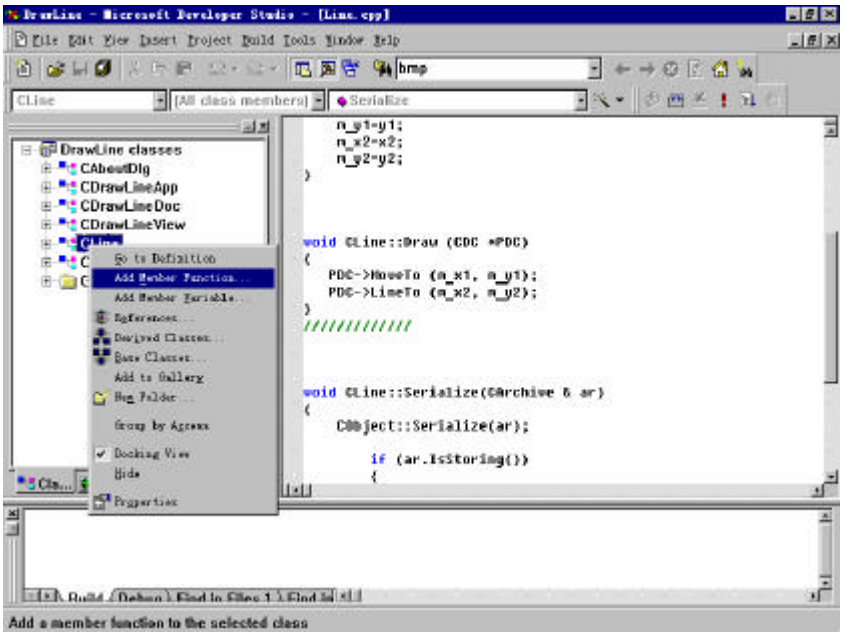


图3.10

VC将会跳出如3.11所示下添加函数的对话框：

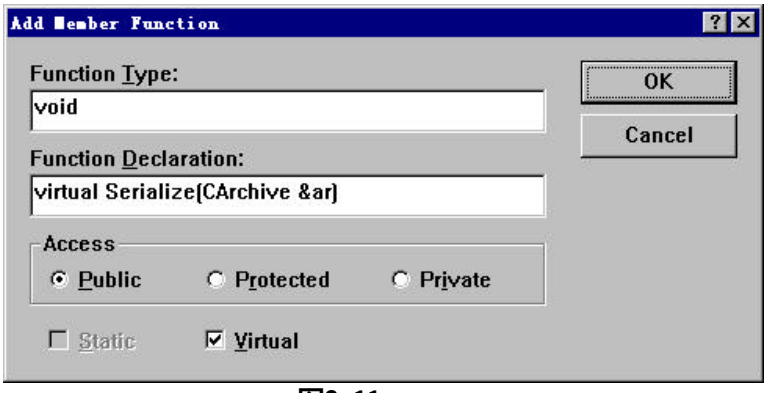


图3.11

在Function Type输入void,在Function Declaretion输入 Serialize(CArchive& ar),然后选择Virtual，按OK即可。然后在ClassView中可以看到这个函数。

下面我们编辑这个函数，双击WorkSpace显示的CLine类的Serialize函数，则转到Line.cpp中其实现处。这个函数的实现如下：

```
void CLine::Serialize(CArchive & ar)
{
    CObject::Serialize(ar);

        if (ar.IsStoring())
        {
          ar<<m_x1<<m_y1;
          ar<<m_x2<<m_y2;
         }
         else
         {
           ar>>m_x1>>m_y1;
           ar>>m_x2>>m_y2;
          }

}
```

首先调用基类的Serialize函数，CObject::Serialize(ar);然后判断是保存数据还是载入数据，然后再根据判断的结果进行实际的存取工作。这里ar就是框架程序传递给序列化函数的归档对象指针。

当调用完基类的序列化函数后，判断ar的状态，当ar.IsStoring()返回真时，这时进行数据保存；当ar.IsStoring()返回非真时，这时CArchive 对象要求读取数据。

3 使用操作符存取数据

在上面的代码中我们用到了>>和<<，在这里对它们作一个介绍，>>和<<是一种操作符，用来指示向CArchive对象读取还是保存数据，必要时我们可以重载重定向符。如ar>>m_x1>>m_y1;这一句，其中>>表示从ar中读出数据m_x1,m_y2,这个符号及>>可以连用，亦可以分开来用，如ar>>m_x1;ar>>m_y1;同样ar<<m_x1<<m_y1;中的<<是把数据存入ar中。

4 文档对象序列化

在对直线对象进行序列化函数处理之后，接下来，我们对文档类进行改写，首先实现文档类的序列化函数。双击WorkSpace中CDrawLineDoc类Serialize函数打开它，文档类的Serialize函数是个虚拟成员函数的，其缺省实现是不做任何工作的。

我们再来看文档类的序列化函数，CDrawLineDoc类的序列化函数主要是对CLine类对象的序列化函数的调用，其大体情况是这样的：当进行保存时，我们先得到直线的总数——调用GetNumLines函数，然后用一个循环对每一条直线对象调用序列化函数；当进行读取数据时，亦先得到直线总条数——从文件中读出，然后同样用一个循环，每次读出一条直线——调用对象的序列化函数，然后把它加入到文档类的成员变量m_LineArray中去，直到直线读完。这个函数的源代码如下：

```
void CDrawLineDoc::Serialize(CArchive& ar)
{
     int linenum=GetNumLines();
     if (ar.IsStoring())
     {
          ar << linenum;
          for(int i=0;i<linenum;i++)
               m_LineArray.GetAt(i)_>Serialize(ar);
     }
     else
     {
          m_LineArray.RemoveAll();
          ar >> linenum;
          CLine Line;
          for(int i=0;i<linenum;i++)
          {
               CLine *PLine = new CLine ();
               PLine_>Serialize(ar);
               m_LineArray.Add (PLine);
          }
          UpdateAllViews(NULL);
     }
}
```

5 连接视图

接着，我们对文档类和视类进行一些必须的处理：在ClassWizard 中，对CDrawLineDoc增加两个成员函数：OnNewDocument和OnOpenDocument。对OnNewDocument 的调用是当新生成一个文档时，所以此时需要处理文档类的成员变量m_LineArray，我们将它里面的直线全部除去，同时调用UpdateAllViews(NULL),更新视图，我们对这个函数的代码作如下改动：

```
BOOL CDrawLineDoc::OnNewDocument()
{
     if (!CDocument::OnNewDocument())
          return FALSE;
     // TODO: add reinitialization code here
     // (SDI documents will reuse this document)
     m_LineArray.RemoveAll();
     UpdateAllViews(NULL);
```

```
    return TRUE;
}
```

框架程序对OnOpenDocument的调用是在我们打开文件时，所以这个时候，在我们读取数据之前，同样要清除m_LineArray所有的直线，我们对这个函数的代码作如下改动：

```
BOOL CDrawLineDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    m_LineArray.RemoveAll();
    UpdateAllViews(NULL);
    if (!CDocument::OnOpenDocument(lpszPathName))
        return FALSE;

    // TODO: Add your specialized creation code here

    return TRUE;
}
```

我们还需接着对视类的OnDraw进行改写，让它可以实现重画，在这里我们只是简单地对每一个直线类对象调用Draw成员函数，对每一条直线进行重画，我们对这个函数的代码作如下改动：

```
void CDrawLineView::OnDraw(CDC* pDC)
{
    CDrawLineDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: add draw code for native data here
    for (int i = 0; i<pDoc_>GetNumLines(); i++)
    {
        (pDoc_>GetLine(i)) _>Draw(pDC);
    }
}
```

至此我们对程序的序列化已经完成，对程序编译运行即可。

3．3 与文件处理关系密切的类 CFile

CFile类用来处理正常文件的I/O操作，它直接供无缓冲的、二进制磁盘输入/输出服务，并且通过其派生类间接支持文本文件和内存文件。因为CFile类是基本上封装在CArchive类之中了，所以我们只对这个类作简单介绍。

CFile类有三个构造函数，其原型如图所示。

```
virtual BOOL Open( LPCTSTR lpszFileName, UINT nOpenFlags,
                   CFileException* pError = NULL )
```

其中：hFile为一个已打开文件的句柄。LpszFileName指定所想要文件的路径的字符串。路径可以是相对的或绝对的。NOpenFlags指共享和存取方式，对于这个标志的说明，我们留到后面专门说明。

CFile类用Open来创建和打开文件。使用Open创建新文件，必须有一个文件名，并且选择一定的打开方式：

CFile对磁盘文件的定点读和写是通过函数Read ,Write和Seek进行的。

```
virtual UINT Read( void* lpBuf, UINT nCount );

virtual void Write( const void* lpBuf, UINT nCount );
```

```
virtual LONG Seek( LONG lOff, UINT nFrom );
```

函数Read：返回传输给缓冲区的字节数。如果读到文件尾，返回值可能小于nCount值。LpBuf：指向用户定义的缓冲区的指针，用来接收数据；nCount：要从文件中读出的最大字节数；函数Write:写缓冲区数据到文件中；Seek用于定位文件指针位置，如果所请求的位置合法,则返回距离文件头的新字节偏移。

文件的打开和关闭是相对的，打开一个文件之后，必须把它关闭，文件的关闭是相当简单的，用CFile对象调用Close函数即可。

下面描述文件共享和存取标志。下列标志指定打开文件时可执行的动作。可以用OR来组合下面所列的选项。一个存取许可和一个共享选项是必需的；modeCreate 和modeNoInberit方式是可选的。

- modeCreate指示构造函数创建一个新文件，如果该文件已存在，则该文件截短为0。
- modeRead 打开文件用于读。
- modeReadWrite 打开文件用于读写。
- modeWrite 打开文件用于只写。
- modeNoInberit 阻止文件被子进程继承。
- shareDenyNone 打开文件，不允许其它进程读或写该文件。如果该文件已由其它任何进程用兼容方式打开，则Create将失败。
- shareDenyWrite 打开文件，不允许其它进程写该文件。如果该文件已由其它任何进程用兼容方式或写方式打开，则Create将失败。
- shareDenyRead 打开文件，不允许其它进程读该文件。如果该文件已由其它任何进程用兼容方式或读方式打开，则Create将失败。
- shareExclusive 以独占方式打开文件，不允许其它进程写该文件。如果该文件已用其它读或写方式打开，即使是当前进程打开，则构造也将失败。
- shareCompat 以兼容方式打开文件，允许给定机器上任何进程打开该文件任意次。如果该文件已用其它任何共享方式打开，构造将失败。
- typeText 设置文本方式，对回车换行进行特殊处理，它只用于派生类。
- typeBinary 设置二进制方式，它只用于派生类。

4 DAO 技术

4．1 DAO 与 ODBC

在WINDOWS环境下进行数据库访问工作您有两种选择：使用DAO技术或者使用ODBC技术。ODBC（OPEN DATABASE CONNECTIVITY）即开放式数据库互联，作为WINDOWS开放性准结构的一个重要部分已经为很多的WINDOWS程序员所熟悉。DAO（DATA ACCESS OBJECTS）即数据访问对象集（DATA ACCESS OBJECTS）是MICROSOFT提供的基于一个数据库对象集合的访问技术。它们都是WINDOWS API的一个部分，可以独立于DBMS进行数据库访问。那么ODBC和DAO的区别在哪里呢？ODBC和DAO访问数据库的机制是完全不同的。ODBC的工作依赖于数据库制造商提供的驱动程序，使用ODBC API的时候，WINDOWS的ODBC管理程序，把数据库访问的请求传递给正确的驱动程序，驱动程序再使用SQL语句指示DBMS完成数据库访问工作。DAO则绕开了中间环节，直接使用MICROSOFT提供的数据库引擎（MICROSOFT JET DATABASE ENGINE）提供的数据库访问对象集进行工作。速度比ODBC快。数据库引擎目前已经达到了3.0版本。它是DAO、MS ACCESS、MS VISUAL BASIC等等WINDOWS应用进行数据库访问的基础。引擎本身的数据库格式为MDB，也支持对目前流行的绝大多数数据库格式的访问，当然MDB是数据库引擎中效率最高的数据库。

如果您使用客户机/服务器模型的话，建议您使用ODBC方案；如果您希望采用MDB格式的数据库，或者利用数据库引擎的速度，那么DAO是更好的选择。

4．2 使用 MFC 实现 DAO 技术

MFC对所有的DAO对象都进行了封装。使用MFC进行DAO编程，首先要为每一个打开的数据库文件提供一个数据库对象——CDaoDatabase，由这个对象管理数据库的连接。然生成记录集对象——CDaoRecordset，通过它来进行查询、操作、更新等等的工作。如果需要在程序中管理数据库的结构，则需要使用DAO当中的表结构信息对象CDaoTableInfo及字段定义对象CDaoFieldInfo来进行获得或者改变数据库表结构的工作。CDaoDatabase、CDaoRecordset、CDaoTableDefInfo、CDaoFieldInfo是使用MFC进行DAO编程的最基本也是最常用的类。

下面，我们通过一个实例来介绍如何使用MFC的DAO类来进行数据库访问的工作。在这个实例当中，我们将在程序当中建立一个学生档案管理数据库，并通过对话框来添加、删除和浏览记录。我们首先看以下程序运行的情况。

我们将针对程序的功能依次介绍如何生成和使用数据库对象、记录集对象以及如何通过记录集来操纵数据库。我们将通过解释对数据库进行操作的源程序来介绍如何用MFC来实现DAO技术。

下面介绍如何建库：

首先新建一个数据库对象。

```
newDatabase =  new CDaoDatabase;
newDatabase_>Create(_T("stdfile.mdb"),
                        dbLangGeneral, dbVersion30);
```

利用数据库引擎在磁盘上建立一个MDB格式的数据库文件。

stdfile.mdb是在磁盘上面建立的数据库文件的名字，

dbLangGeneral 是语言选项。

dbVersion30这是数据库引擎版本选项。

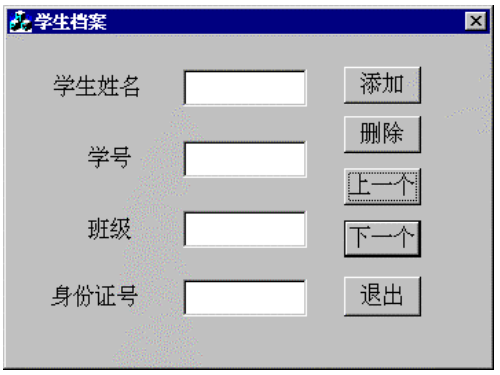


图3.12

然后新建一个数据库表定义信息对象。

```
CDaoTableDef  *TableInfo;
TableInfo = new CDaoTableDef(newDatabase);
TableInfo_>Create(_T("student"));
```

新建一个字段定义信息对象。

按要求填写字段定义信息对象。

定义字段名称：

```
FieldInfo_>m_strName = CString("studentName");
```

定义字段类型：

```
FieldInfo_>m_nType = dbText;
```

定义字段所占的字节数大小：

```
FieldInfo_>m_lSize = 10;
```

定义字段特性：

```
FieldInfo_>m_lAttributes = dbVariableField | dbUpdatableField;
```

dbVariableField参数的意思是该字段的所占的字节数是可变的。

dbUpdatableField参数的意思是该字段的值是可变的。

根据字段定义对象在数据库表对象当中生成字段。

```
TableInfo_>CreateField(*FieldInfo);
```

在生成了所有的字段之后，将新的数据库表的定义填加到数据库对象当中去。

```
TableInfo_>Append();
```

下面介绍如何进行数据库操作：

首先生成记录集对象：

```
Recordset = new CDaoRecordset(newDatabase);
```

然后使用SQL语句打开记录集对象。首先把SQL语句记入一个字符串：

```
CString    strQuery = _T("Select * from student");
```

使用这个字符串打开记录集。

```
Recordset_>Open(dbOpenDynaset , strQuery);
```

dbOpenDynaset参数的意思是表示记录集打开的类型。dbOpenDynaset的意思是打开一个可以双向滚动的动态记录集。这个记录集中的记录是使用我们定义的SQL语句对数据库进行查询得到的。这个参数还有另外的两种选择：

dbOpenTable参数指示打开一个数据表类型的记录集，使用这种类型的记录集只能对单一的数据库中的记录进行操纵。

如果使用dbOpenSnapshot参数表示打开的是映像记录集，它实际上是所选择的记录集的一个静态的拷贝，在只需要进行查询操作或者希望制作报表的时候，使用这种记录集比较合适，它不会对数据库中的数据进行修改。

接下来对记录集当中的一个标志位赋值，说明是否要求自动地标记出 CACHE 当中经改变的记录。使用记录集的时候是 DAO 把被检索出的记录读入 CACHE，所有的操纵都是针对 CACHE 中的记录进行的，要实现对数据库当中的记录更新必须把 CACHE 记录中被改变的字

段的值写回到数据库文件当中去。这个标志位的作用就是当 CACHE 中的数据改变的时候，是否需要自动的标记出记录中那些应该被写回的字段。

下面介绍如何填加一个记录。

```
m_Recordset _>AddNew();
m_Recordset_>Update();
```

使用AddNew()这个函数可以在数据表记录集或者是动态记录集当中添加新的记录，调用AddNew()之后必须接着调用Update()来确认这个添加动作，将新的记录保存到数据库文件当中去。新的记录在数据库当中的位置取决于当前记录集的类型：如果是动态记录集，新记录都将被插入到记录集的末尾。如果是数据表记录集的话，当数据库表中定义了主键的时候新记录将按照库表的排序规则插入到合适的地方；如果没有定义主键那么新记录也会被插入到记录集的末尾。

用AddNew()会改变记录集的当前记录。只有将当前记录定位在新记录上，才能填写它的数据。所以我们使用MoveLast函数使刚刚添加的记录成为当前记录，然后调用Edit函数对新记录进行编辑。

```
m_Recordset_>MoveLast();
m_Recordset_>Edit();
```

依次给新记录的字段进行赋值：

```
COleVariant          var1(m_Name , VT_BSTRT);
m_Recordset_>SetFieldValue(_T("studentName") , var1);
COleVariant          var2(m_ID , VT_BSTRT);
m_Recordset_>SetFieldValue(_T("studentID") , var2);
COleVariant          var3(m_Class , VT_BSTRT);
m_Recordset_>SetFieldValue(_T("studentClass") , var3);
COleVariant          var4(m_SID , VT_BSTRT);
m_Recordset_>SetFieldValue(_T("studentSID") , var4);
```

COleVariant 这个类封装了WIN32提供的VARIANT这个结构以及对它的操作。这个类当中可以存储多种类型的数据。需要注意的是这种包容能力是通过C语言当中的UNION提供的，就是说一个COleVariant 对象只能保存一种类型的数据。我们先把字段的值装入OLE变体对象，再使用这个变体对象对记录中的字段进行赋值。VT_BSTRT参数的作用是在生成OLE变体对象的时候指示将要封入的数据的类型为字符串。当对所有的字段都结束赋值后，调用Update 函数来保存刚才的修改。

```
m_Recordset_>Update();
```

注意，在调用Update函数之前,如果进行了改变当前记录的操作,那么前面进行的所有的赋值工作都将丢失，而且不会给出任何的警告。

这段代码从记录集中取出一个记录的值,这里同样要用到OLE变体对象。记录集的GetFieldValue将返回一个变体对象，我们首先取得这个变体对象，然后从中取出需要的值。

这里V_BSTRT指示从变体对象当中取出字符串类型的数据。

如何从数据库中删去一个记录呢？首先要使该记录成为当前记录。然后使用Delete函数来执行删除操作。

```
m_Recordset_>Delete();
```

删除之后，我们必须把当前记录更改为其他的记录，以确认这个删除动作。

以上就是在MFC中使用DAO进行数据库操作的方法。

了解了前面的内容，相信您对MFC类库已经有了比较深入的认识，可以使用MFC编写出不错的程序了。下面，我们将向您介绍如何在VISUAL C++集成开发环境之下调试自己的程序。

5 打印

打印功能是现在几乎所有的应用程序所必须具备的一项基本功能，这一点从 APPWizard 生成的应用程序框架的菜单中缺省地包括打印和打印预览功能中可以看出来。但是打印功能的实现如果只依赖于以前的 Windows API 调用来实现的话，是一件非常繁杂的事情。相比之下，Microsoft 的基本类库应用框架则大大简化了打印功能的实现，并且还提供了打印预览功能。

5.1 打印和显示

以前我们进行的输出工作都是向屏幕上的一块窗口区域中进行，而打印则是打印机向打印纸上输出一些东西。的确，这两者之间有很大的相似性，比如，它们都能输出文本，也都能输出一些图形。正是基于这些相似性，在 Windows 中，用设备上下文将它们之间的共性统一起来了。你在进行打印和输出时，你可以用相同的输出函数（如 TextOut）来往屏幕或打印纸上输出。Windows 会在不同的情况下，将输出联系到相应的设备上。但是这并不意味着我们可以完全不考虑这两者之间的差别，而认为我们只要实现了屏幕输出功能就自动地实现了相应的打印功能。毕竟，它们之间有一些无法统一的差别：打印时，有页和分页的概念，即数据或输出是有条理地组织在一张张有一定大小的纸上；而屏幕输出时，则没有页和分页的功能，同时，可以认为屏幕输出的输出区域是没有大小限制的，超出窗口范围的，我们可以用滚动条来滚动。

CView 有这样三个虚拟函数：OnPaint(),OnDraw(CDC *pDC),OnPrint()。OnPaint()是当视类窗口需要在屏幕上输出时被调用，负责完成窗口的屏幕输出显示工作；OnPrint()是当打印一页时被调用，负责完成向某页打印纸上打印。这两个函数的缺省实现中包含了对 OnDraw(CDC *pDC)的调用。我们以前的程序中，并不考虑打印问题，所以我们被告之在 OnDraw 中考虑如何完成屏幕输出工作。但现在不一样了，我们在 OnDraw 中添加代码时，必须小心这有可能是在进行打印，并不是向一个窗口中输出。如果这两者在程序中不能统一，就必须分开考虑。分分开考虑有两种方法：一是分别在 OnPaint 和 OnPrint 中完成屏幕输出和打印输出工作，而不必依赖于 OnDraw。二是在 OnDraw 中，我们可以调用 pDC_>IsPrinting() 来识别目前是在进行哪种输出工作。如果是在进行打印，pDC_>IsPrinting()返回 true。这样我们就可以在 OnDraw 中区别对待了。

5.2 打印分页

前面我们讲过打印输出一个很特殊的地方在于它的分页这一点上。要输出的的内容是被安排在不同的页上的。这样，当我们有超过一页的内容要显示时，我们就必须考虑分页，要能计算出显示到哪时必须换到下一页上去输出。这种拆页计算是有一定的麻烦的，我们会在后面结合给出的例子来讨论。现在这里要说明的是 OnPrint 的每一次调用对应着打印某一页。OnPrint(CDC *pDC,CPrintInfo *pInfo)中的第二个参数中包含了我们感兴趣的一些关于页的信息。CPrintInfo 类型的对象有一个 m_nCurPage 的公共成员属性，它告诉了我们目前是在打印哪一页。我们计算拆页时就显然要利用它。

应用程序框架在每一次调用 OnPrint 打印一页之前，都会调用 OnPrepareDC(CDC *pDC，CPrintInfo *pinfo) 这个虚函数。这个虚函数在每次屏幕输出前也会被调用来允许做一些设置。在后一种情况下， 第二个参数 pinfo 等于 NULL。打印时，我们可以在这里做一些针对某一页做一些设备上下文的设置工作。同时还有另一项重要的设置工作，我们可以设置打印工作的结束，即告诉应用框架，所有的页都已打印完了，打印可以结束了。这是通过设置 pinfo 的公共成员属性 m_bContinuePrinting 为 false 来完成的。做此设置后，应用框架便不会再调用 OnPrint,从而结束打印工作。

5.3 打印工作的开始和结束

前面我们讨论了每一页打印的准备工作，在每一次打印任务之前和之后，应用框架也允许我们进行一些设置工作。在每次打印工作开始时，应用程序会跳出一个打印对话框，如图 3.13。

而应用框架在跳出这个对话框之前，会调用 OnPreparePrinting(CPrintInfo *pinfo),我们可以通过调用 pinfo 的 SetMaxPage 和 SetMinPage 来设置出现在对话框中的最大最小页。

当关闭这个对话框之后，OnBeginPrinting 会被调用。

当打印工作结束后，OnEndPrinting 会被调用。

图 3.13

5.4 打印程序实例

在这里，我们制作了一个包含了打印功能的例子。它从一个学生信息数据库中抽取所有的记录，然后显示在一个 CScrollView 中，当用户要将它们打印出来时，程序按每页打印五条记录将这些记录打印出来。这个例子中用到了 DAO 来访问数据库，在本书的前面章节中已介绍过，读者可以先复习一下，在本章我们不会详细解释相应的语句。

在这个例子中，我们先利用 APPWizard 生成一个应用程序框架。注意在生成的第六步中将 CPrintView 的基类从 CView 改成

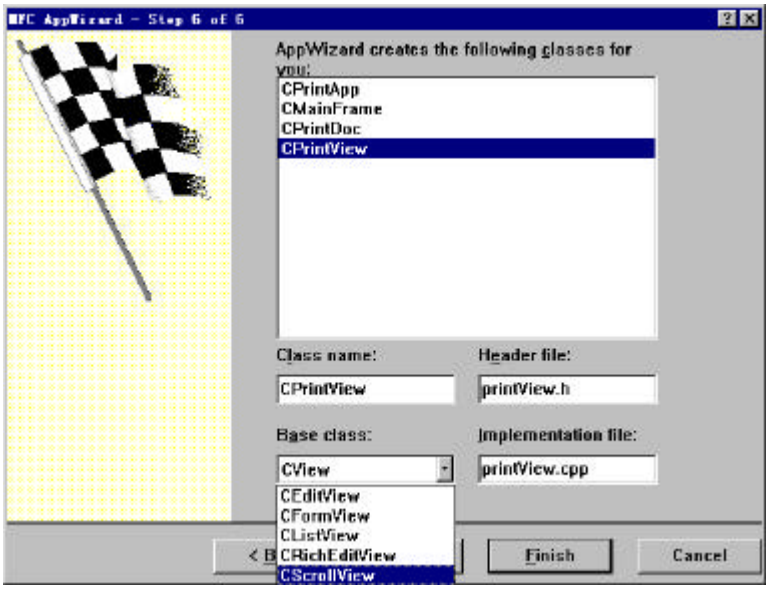


图 3.14

CScrollView,如图 3.14 所示。

在这个例子中，我们就分开考虑打印和显示输出，即在 OnPrint 中考虑打印，在 OnPaint 中考虑显示输出。而在 OnDraw 不做任何工作。

在 CPrintView 中，我们添加一个虚函数：OnPreparePrinting,其代码如下：

```
BOOL CPrintView::OnPreparePrinting(CPrintInfo* pInfo)
{
     // default preparation
    pInfo_>SetMaxPage((Recnumber+4)/5);
     return DoPreparePrinting(pInfo);
}
```

这其中 pInfo_>SetMaxPage((Recnumber+4)/5)用来设置打印的最大页数。Recnumber 是我们通过访问数据库计算出来的记录条数。我们前面说过，每五条记录分一页，（Recnumber+4）/5 就可以得到页数。

下面是 OnBeginPrinting 中的代码

```
m_bPrintEnd=false;
StuRecSet_>MoveFirst();
if (StuRecSet_>IsEOF())
    m_bPrintEnd=true;
```

m_bPrintEnd 是我们往 CPrintView 类中添加的一个用来指示是否该结束打印的成员。

```
void CPrintView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
     CScrollView::OnPrepareDC(pDC, pInfo);
     // TODO: Add your specialized code here and/or call the base class
     if (pInfo!=NULL)
       pInfo_>m_bContinuePrinting=!m_bPrintEnd;
}
```

上面是虚函数 OnPrepareDC 中的代码，注意 if （pInfo!=NULL）的判断，我们要识别是否是为打印做准备。

下面是最重要的 OnPrint 中的代码：

```
void CPrintView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
    // TODO: Add your specialized code here and/or
    //call the base class
    int Curpage=pInfo_>m_nCurPage;
    StuRecSet_>SetAbsolutePosition((Curpage_1)*5);

    //Assume to print 5 records every page
        TEXTMETRIC tm;
    int nColumnWidth[4];
    int i,j;
     pDC_>SetMapMode(MM_TWIPS);
    pDC_>GetTextMetrics(&tm);

    nheight=tm.tmHeight+tm.tmExternalLeading;

        for (i=0;i<4;i++)
        nColumnWidth[i]=(pDC_>GetTextExtent(ColumnName[i])).cx;
        RECT r;
         r.right=r.left=720;
        r.top=_720;
         r.bottom=(r.top_nheight);
```

```
 //print the column headers
    for (i=0;i<4;i++)
{
       r.right=r.left+nColumnWidth[i];
       pDC_>ExtTextOut(r.left,r.top,ETO_CLIPPED,
                          &r,ColumnName[i],NULL);
     r.left=r.right;
}
 r.top_=nheight;
 r.bottom_=nheight;

 //print next 20 student records
 for (j=0;j<5;j++)
 {
   r.right=r.left=720;
   for (i=0;i<4;i++)
    {
       r.right=r.left+nColumnWidth[i];
       pDC_>ExtTextOut(r.left,r.top,ETO_CLIPPED,&r,
           CString(V_BSTRT(&(StuRecSet_>GetFieldValue(i)))),
            NULL);
        r.left=r.right;
    }

       //set for next record printing
    r.top_=nheight;
    r.bottom_=nheight;
    StuRecSet_>MoveNext();
    if (StuRecSet_>IsEOF())
    {
       m_bPrintEnd=true;
         break;
       }
 }
 CScrollView::OnPrint(pDC, pInfo);
}

 pDC_>SetMapMode(MM_TWIPS);
    pDC_>GetTextMetrics(&tm);
    nheight=tm.tmHeight+tm.tmExternalLeading;
```

这几句语句的作用是设置输出的映射方式，然后获取有关文本输出的一些参数，以便后面进行计算输出。

```
 //print the column headers
for (i=0;i<4;i++)
 {
       r.right=r.left+nColumnWidth[i];
       pDC_>ExtTextOut(r.left,r.top,ETO_CLIPPED,
                              &r,ColumnName[i],NULL);
     r.left=r.right;
```

```
 }
```

以上这几句是在每一页上打印记录各个域的名称，以便下面输出各条记录。

接下来就应该是输出五条记录（如果还有五条的话），它被包含在 for(j=0;j<5;j++) 这个循环中。

```
for (i=0;i<4;i++)
    {
      r.right=r.left+nColumnWidth[i];
       pDC_>ExtTextOut(r.left,r.top,
        ETO_CLIPPED,&r,
       CString(V_BSTRT(&(StuRecSet_>GetFieldValue(i)))),NULL);
        r.left=r.right;
    }
```

这个 for 循环是嵌在前面那个循环之中的，它负责将一条记录的每个域（共 4 个）输出在某一行上。

```
   StuRecSet_>MoveNext();
    if (StuRecSet_>IsEOF())
    {
       m_bPrintEnd=true;
         break;
       }
```

然后移向数据库的下一条记录，并判断是否已到了数据库中最后一条记录，如是，则跳出循环，并置 m_bPrintEnd 为 true,以便下一次在 OnPrepareDC 中能正确地结束打印。

在我们给出的这个例子中，还是简化了很多的考虑的，特别是在拆页的计算上，我们简单地指定每五条记录一页，而一个真正的实用程序肯定是要通过获知纸张的大小（这可以通过 CDC 类的 GetDeviceCaps 来进行），然后计算得出一页上可以输出多少条记录。

四、VC 程序调试

在开发程序的过程中，经常需要查找程序中的错误，这就需要利用调试工具来帮助你进行程序的调试，当然目前有许多调试工具，而集成在 VC 中的调试工具以其强大的功能，一定使你爱不释手。下面我们先来介绍 VC 中的调试工具的使用。

1 VC 调试工具

1．1 调试环境的建立

在 VC 中每当建立一个工程(Project)时,VC 都会自动建立两个版本：Release 版本，和 Debug 版本,正如其字面意思所说的，Release 版本是当程序完成后,准备发行时用来编译的版本,而 Debug 版本是用在开发过程中进行调试时所用的版本。

DEBUG 版本当中，包含着 MICROSOFT 格式的调试信息，不进行任何代码优化，而在 RELEASE 版本对可执行程序的二进制代码进行了优化，但是其中不包含任何的调试信息。

在新建立的工程中，你所看到是 DEBUG 版本，若要选择 RELEASE 版本，可以选择菜单 PROJECT 中的 SETTING 命令，这时屏幕上面弹出 PROJECT SETTEING 对话框，在 SETTING FOR 下拉列表中选择 RELEASE，按 OK 退出,如图 4.1。

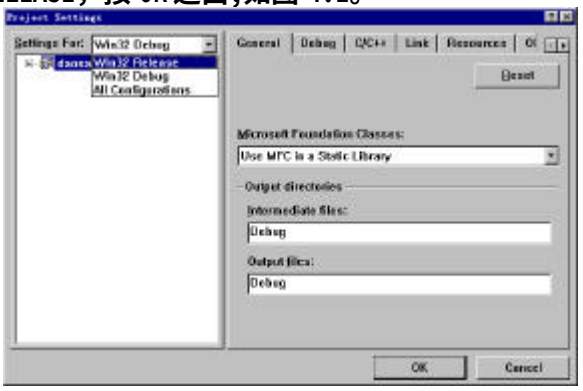


图 4.1

在调试程序的时候必须使用 DEBUG 版本，我们可以在 Project Setting 对话框的 C/C++页中设置调试选项。

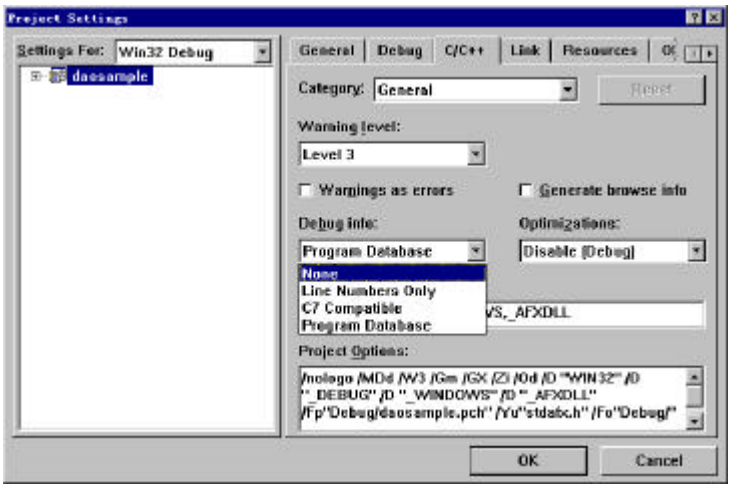


图 4.2

各个选项的含意如下：

- Program Database 表示产生一个存储程序信息的数据文件(.PDB),它包含了类型信息和符号化的调试信息；
- Line Numbers Only 表示程序经过编译和链接产生的.OBJ 或.EXE 文件仅仅包含全局和外部符号以及行号信息；
- C7 Compatible 表示产生一个.OBJ 或.EXE 文件行号信息以及符号化的调试信息；
- None 表示不产生任何调试信息。

1．2 调试的一般过程

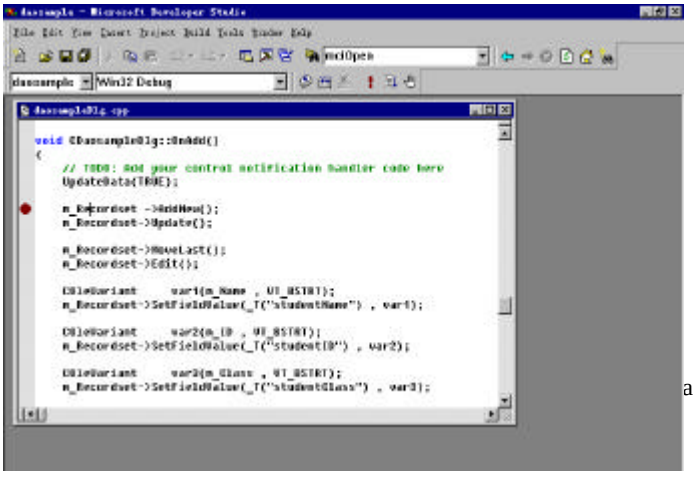
调试，说到底就是在程序的运行过程的某一阶段观测程序的状态,而在一般情况下程序是连续运行的，所以我们必须使程序在某一地点停下来。所以我们所做的第一项工作就是设立断点。其次,再运行程序,当程序在设立断点处停下来时,再利用各种工具观察程序的状态。程序在断点停下来后，有时我们需要按我们的要求控制程序的运行，以进一步观测程序的流向，所以下面我们依次来介绍断点的设置，如何控制程序的运行以及各种观察工具的利用。

1．3 如何设置断点

在 VC 中，你可以设置多种类型的断点，我们可以根据断点起作用的方式把这些断点分为三类:1、与位置有关的断点；2、与逻辑条件有关的断点 3、与 WINDOWS 消息有关的断点下面我们分别介绍这三类断点。

首先我们介绍与位置有关的断点。

1、 最简单的是设置一般位置断点，你只要把光标移到你要设断点的位置，当然这一行必须包含一条有效语句的；然后按工具条上的 add/remove breakpoint 按钮或



按快捷键 F9；这时你将会在屏幕上看到在这一行的左边出现一个红色的圆点表示这二设 立了一个断点。

图 4.3

2 、有的时候你可能并不需要程序每次运行到这儿都停下来，而是在满足一定条件的情况下才停下来，这时你就需要设置一种与位置有关的逻辑断点。要设置这种断点我们只需要从 EDIT 菜单中选中 breakpoint 命令，这时 Breakpoint 对话框将会出现在屏幕上。选中 Breakpoint 对话框中的 LOCATION 标签，使 LOCATION 页面弹出，如图 4.4

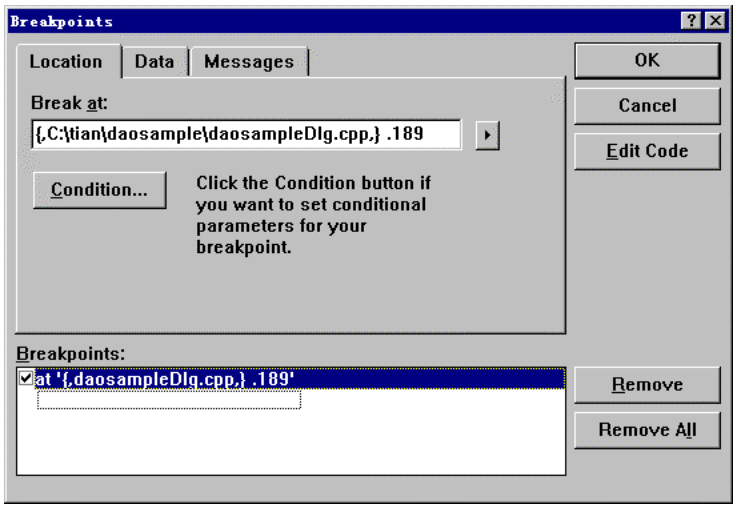


图 4.4

单击 condition 按钮，弹出 Breakpoint 对话框，在 Expression 编辑框中写出你的逻辑表达式，如 X>=3 或 a+b>25,最后按 OK 返回。

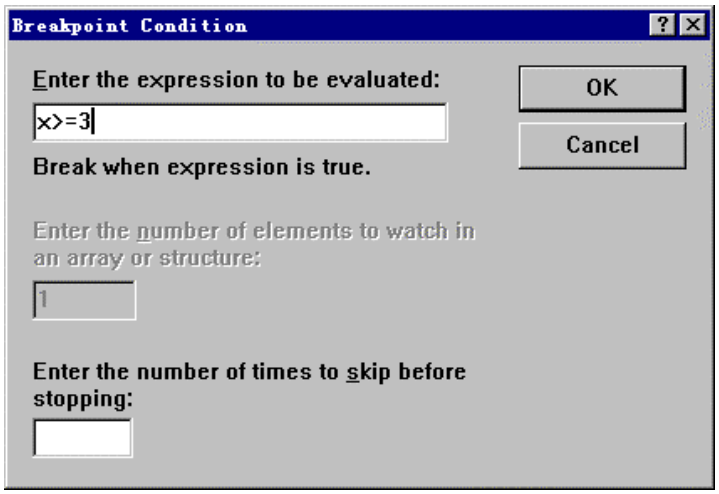


图 4.5

这种断点主要是由其位置发生作用的，但也结合了逻辑条件，使之更灵活。

3、有时我们需要更深入地调试程序，我们需要进入程序的汇编代码，因此我们需要在在汇编代码上设立断点：要设立这种断点我们只需从 View 菜单中选 Debug window 命令 ，

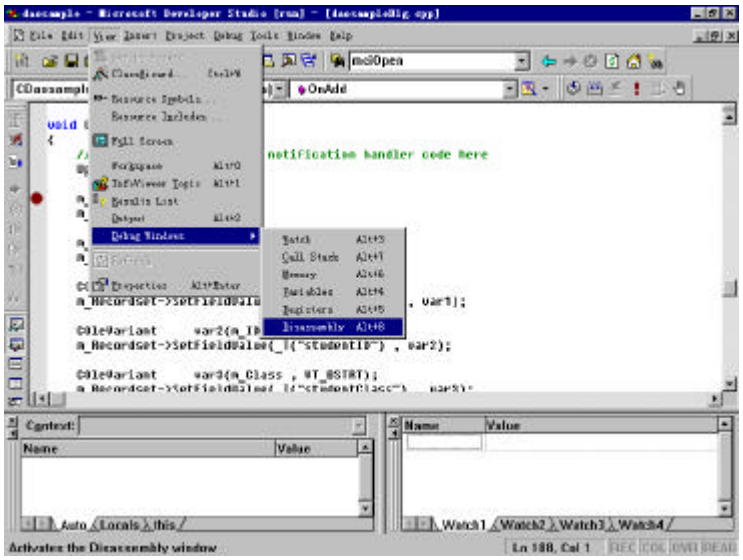


图 4.6

再选 Disassembly 子命令，这时汇编窗口将会出现在屏幕上。

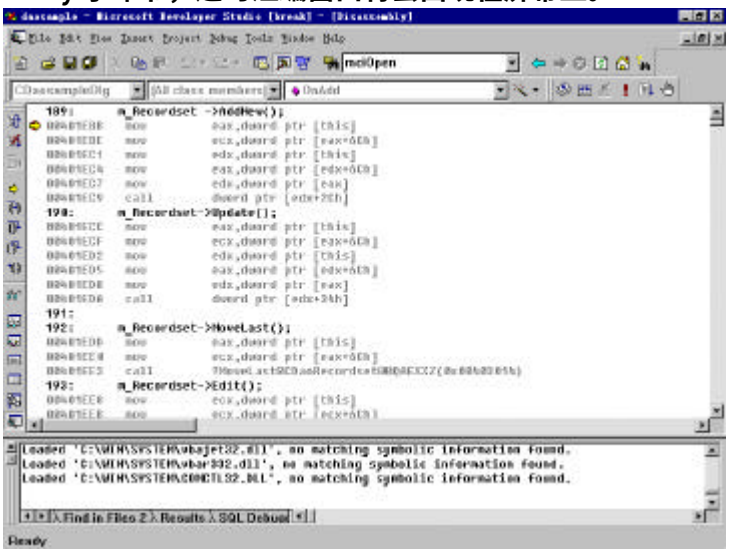


图 4.7

在图 4.7 中的汇编窗口中你将看到对应于源程序的汇编代码，其中源程序是用黑体字显示，下面是且对应的汇编代码。要设立断点，我们只需将光标移到你想设断点处然后点击工具条上的 Insert/Remove Breakpoints 按钮，此后你将会看到一个红圆点出现在该汇编代码的右边。

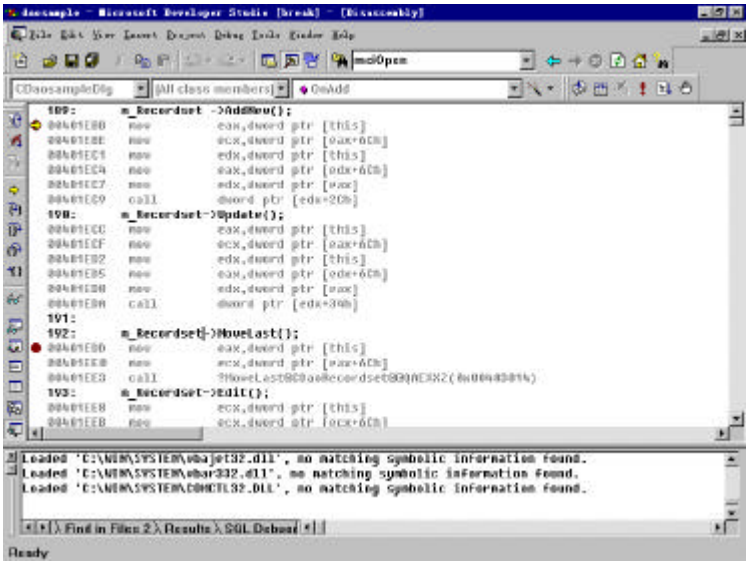


图 4.8

上面所讲的断点主要是由于其位置发挥作用的，即当程序运行到设立断点的地方时程序将会停下来。但有时我们设立只与逻辑条件有关的断点，而与位置无关。所以下面介绍一下与逻辑条件有关的断点。

（1）逻辑条件触发断点的设置：

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会出现 Breakpoint 对话框。

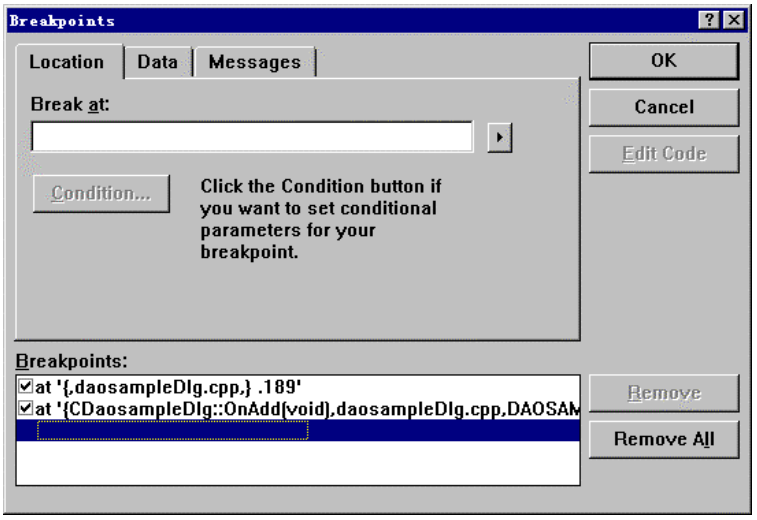


图 4.9

- 选中 Breakpoint 对话框中的 DATA 标签，对应的页面将会弹出

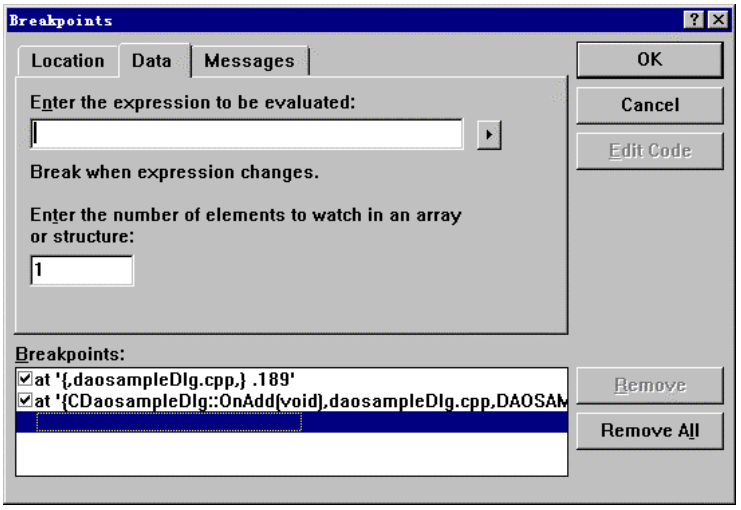


图 4.10

- 在图 4.10 的 DATA 页面中的 Expression 编辑框中写出你的逻辑表达式，如 (X==3);

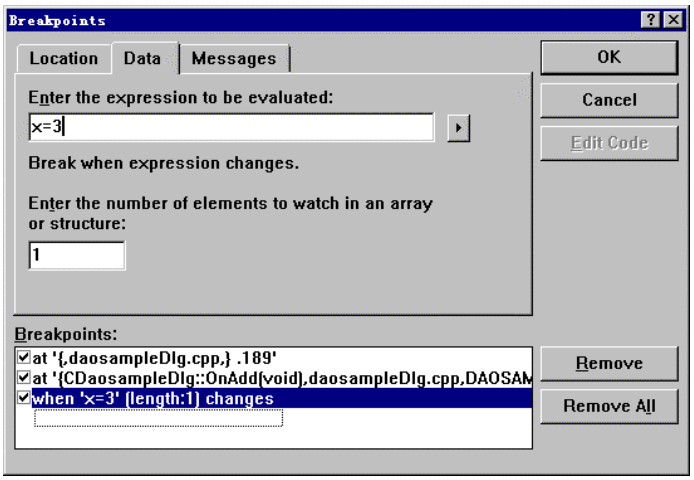


图 4.11

- 最后按 OK 返回。

其他几种断点的设置的方法都与之类似。我们一一加以说明。

(2) 监视表达式发生变化断点：

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会出现 Breakpoint 对话框。
- 选中 Breakpoint 对话框中的 DATA 标签，对应的页面将会弹出
- 在 Expression 编辑框中写出你需要监视的表达式
- 最后按 OK 键返回。

(3) 监视数组发生变化的断点：

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会 出现 Breakpoint 对话框。
- 选中 Breakpoint 对话框中的 DATA 标签，对应的页面将会弹出
- 在 Expression 编辑框中写出你需要监视数组名；
- 在 Number of Elements 编辑框输入你需要监视数组元素的个数；
- 按 OK 键返回。

(4) 监视由指针指向的数组发生变化的断点：

- 从 EDIT 菜单中选中 breakpoint 命令，这时在屏幕上将会出现 Breakpoint 对话框。

- 选中 Breakpoint 对话框中的 DATA 标签；
- 在 Expression 编辑框中输入形如*pointname,其中*pointname 为指针变量名；
- 在 Number of Elements 编辑框输入你需要监视数组元素的个数；
- 按 OK 键返回。

(5) 监视外部变量发生变化的断点：

- 从 EDIT 菜单中选中 breakpoint 命令这时屏幕上将会出现 Breakpoint 对话框；
- 选中 Breakpoint 对话框中的 DATA 标签；
- 在 Expression 编辑框中输入变量名；
- 点击在 Expression 编辑框的右边的下拉键头；
- 选取 Advanced 选项，这时 Advanced Breakpoint 对话框出现；
- 在 context 框中输入对应的函数名和(如果需要的话)文件名；
- 按 OK 键关闭 Advanced Breakpoint 对话框。
- 按 OK 键关闭 Breakpoints 对话框。

(6) 在讲了位置断点和逻辑断点之后我们再讲一下与 WINDOWS 消息有关的断点。

注意：此类断点只能工作在 x86 或 Pentium 系统上。

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会出现 Breakpoint 对话框；
- 选中 Breakpoint 对话框中的 MESSAGE 标签，对应的页面将会弹出；
- 在 Break At WndProc 编辑框中输入 Windows 函数的名称；
- 在 Set One Breakpoint From Each Message To Watch 下拉列表框中选择对应的消息；
- 按 OK 返回。

1．4 控制程序的运行

上面我们讲了如何设置各类断点，下面我们来介绍如何控制程序的运行。当我们从菜单 Build 到子菜单 Start Debuging 选择 Go 程序开始运行在 Debug 状态下，程序会由于断点而停顿下来后，可以看到有一个小箭头，它指向即将执行的代码。

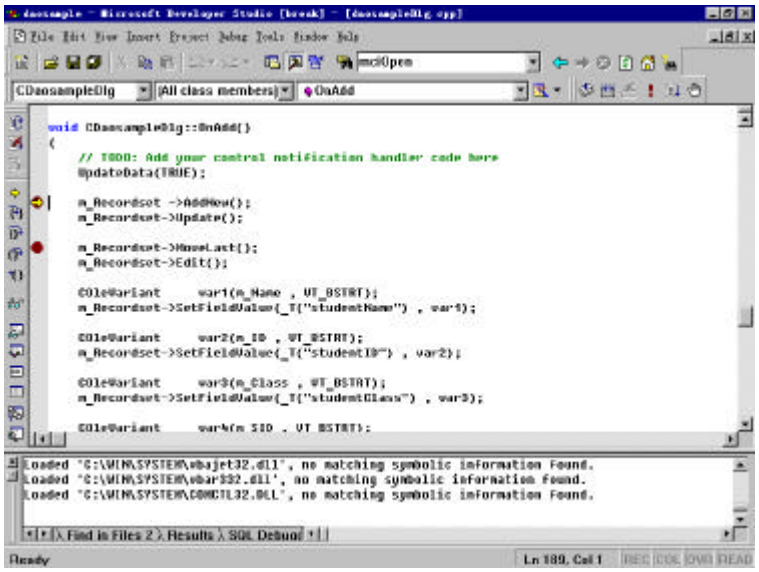


图 4.12

随后，我们就可以按要求来控制程序的运行：其中有四条命令：Step over，step Into ，Step Out ,Run to Cursor。

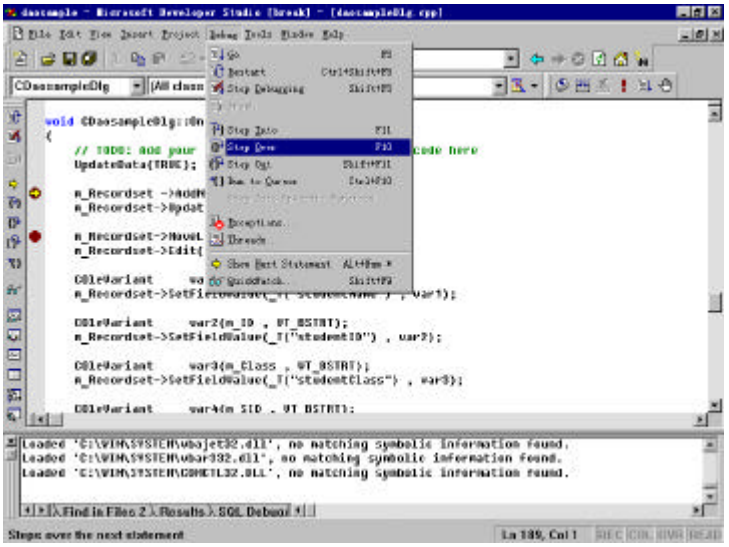


图 4.13

在图 4.13 中:

Step over 的功能是运行当前箭头指向的代码(只运行一条代码)。

Step Into 的功能是如果当前箭头所指的代码是一个函数的调用，则用 Step Into 进入该函数进行单步执行。

Step Out 的功能是如当前箭头所指向的代码是在某一函数内，用它使程序运行至函数返回处。

Run to Cursor 的功能是使程序运行至光标所指的代码处。

1．5 查看工具的使用

调试过程中最重要的是要观察程序在运行过程中的状态，这样我们才能找出程序的错误之处。这里所说的状态包括各变量的值，寄存中的值,内存中的值，堆栈中的值 ，为此我们需要利用各种工具来帮助我们察看程序的状态。

- 弹出式调试信息泡泡(Data Tips Pop_up Information)。

当程序在断点停下来后，要观察一个变量或表达式的值的最容易的方法是利用调试信息泡泡。要看一个变量的值，只需在源程序窗口中，将鼠标放到该变量上,你将会看到一个信息泡泡弹出，其中显示出该变量的值。

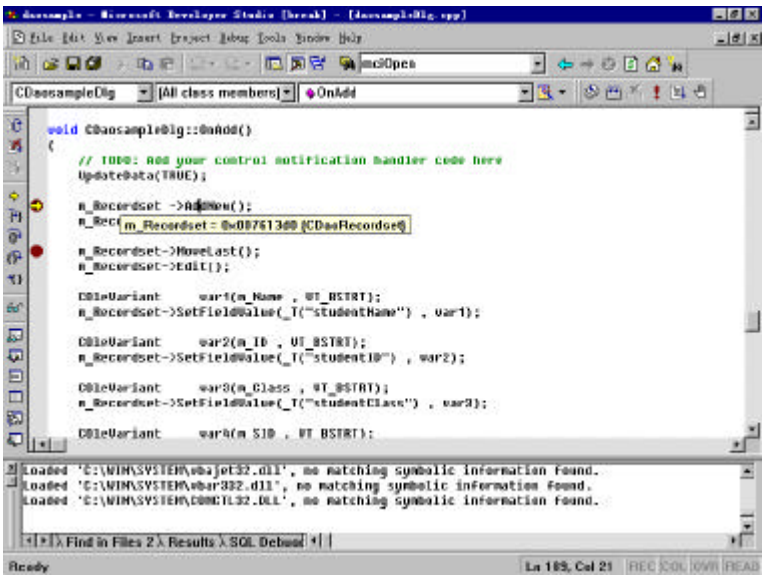


图 4.14

要查看一个表达式的值，先选中该表达式，仍后将鼠标放到选中的表达式上，同样会看到一个信息泡泡弹出以显示该表达式的值如图 4.15 所示。

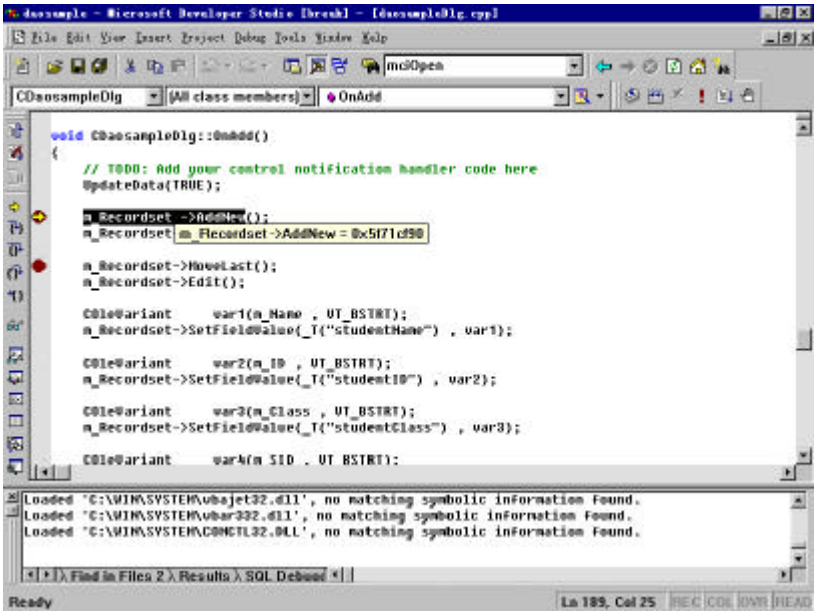


图 4.15

♦ 变量窗口(VARIABLE WINDOW)。

在 VIEW 菜单，Debug window 选 Variables window；变量窗口将出现在屏幕上。其中显示着变量名及其对应的值。你将会看到在变量观察窗口的下部有三个标签：AUTO ,LOCAL,THIS 选中不同的标签，不同类型的变量将会显示在该窗口中。

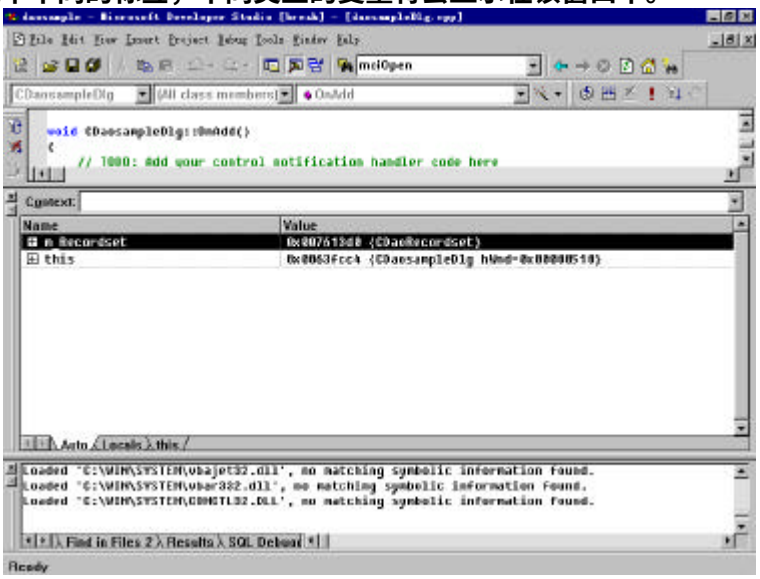


图 4.16

♦ 观察窗口(WATCH WINDOW)：

在 VIEW 菜单，选择 Debug window 命令，Watch window 子命令。这时变量窗口将出现在屏幕上。

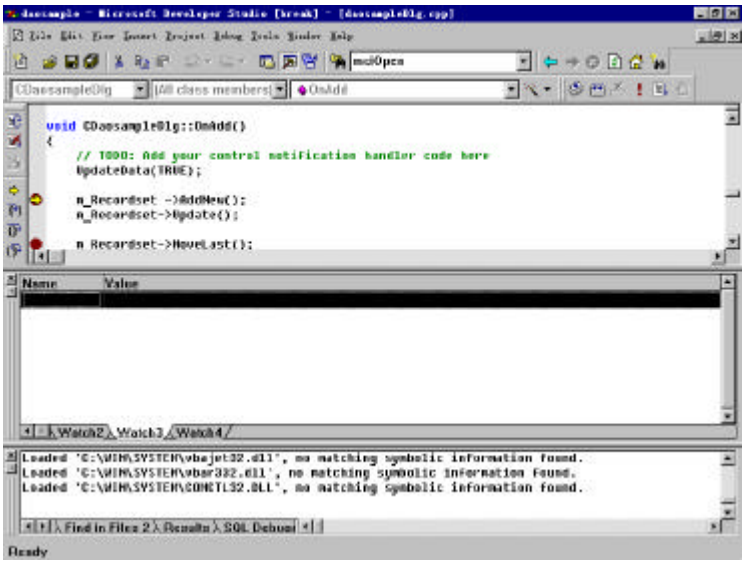


图 4.17

在图 4.17 的观察窗口中双击 Name 栏的某一空行，输入你要查看的变量名或表达式。

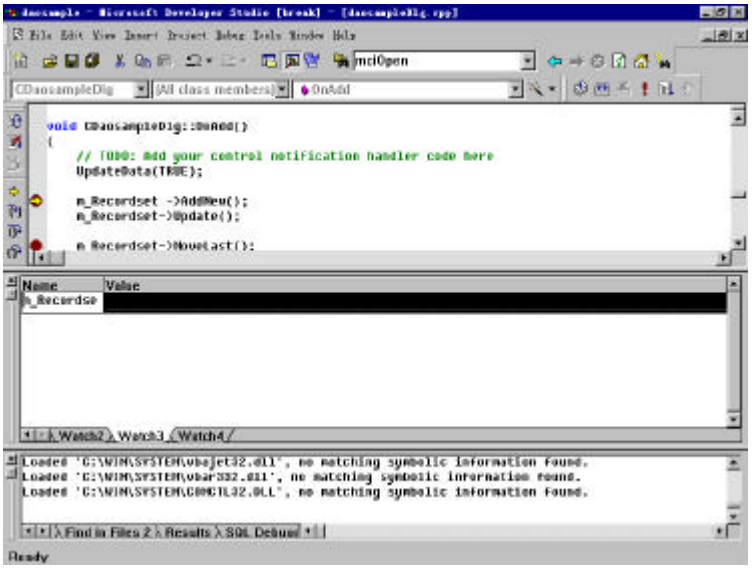


图 4.18

回车后你将会看到对应的值。观察窗口可有多页，分别对应于标签 Watch1,Watch2,Watch3 等等。假如你输入的表达式是一个结构或是一个对象，你可以用鼠标点取表达式右边的形如 + ，以进一步观察其中的成员变量的值如图 4.19。

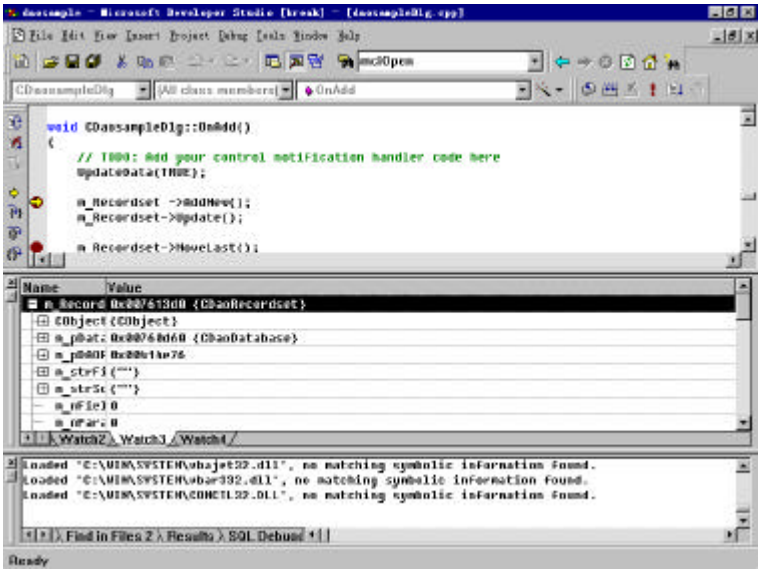


图 4.19

♦ 快速查看变量对话框(quick watch)；

在快速查看变量对话框中你可以象利用观察窗口一样来查看变量或表达式的值。但我们还可以利用它来该变运行过程中的变量，具体操作如下：

(1) 在 Debug 菜单，选择 Quick Watch 命令，这时屏幕上将会出现 Quick Watch 对话框；

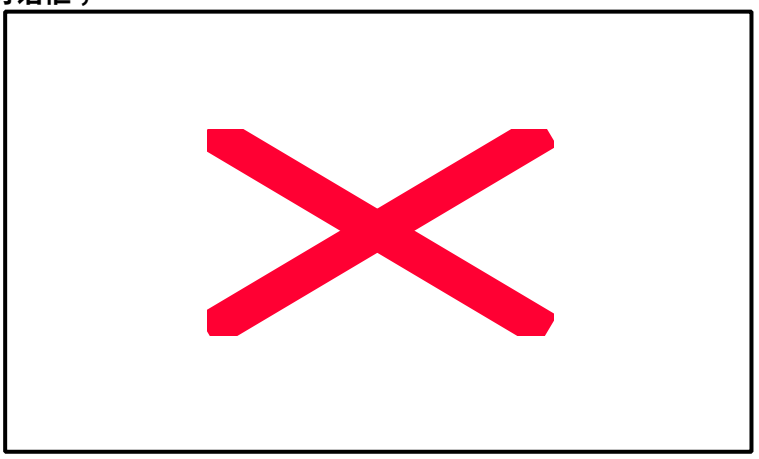


图 4.20

(2) 在 Expression 编辑框中输入变量名，按回车；

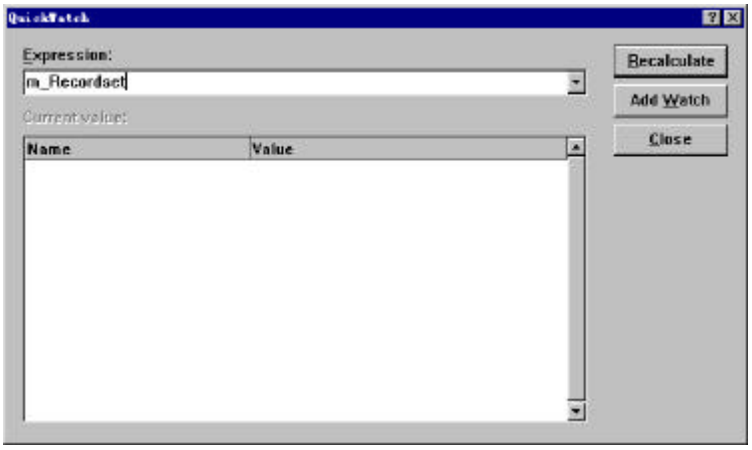


图 4.21

(3) 在 Current Value 格子中将出现变量名及其当前对应的值如图 4.22:

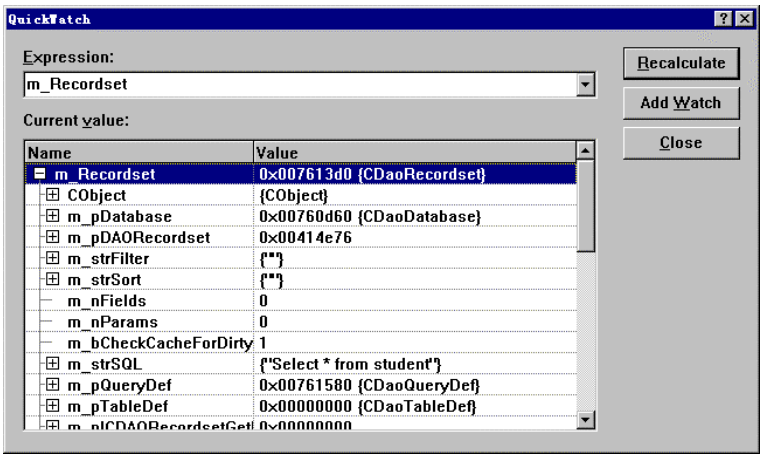


图 4.22

(4) 如要改变该变量的值只需双击该变量对应的 Name 栏，输入你要改变的值；

(5) 如要把该变量加入到观察窗口中，点击 Add watch 按钮；

(6) 点击 Close 按钮返回；

♦ 我们还可以直接查看内存中的值

(1) 从 View 菜单中选取 Debug windows 及 Memory 子命令。Memory Window 出现；

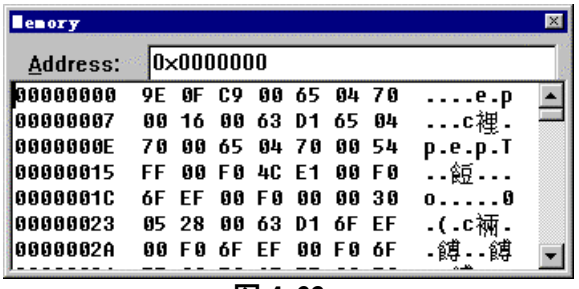


图 4.23

(2) 在 Address 编辑框中输入你要查看的内存地址，回车。对应内存地址中的值将显示在 Memory window 的窗口中。

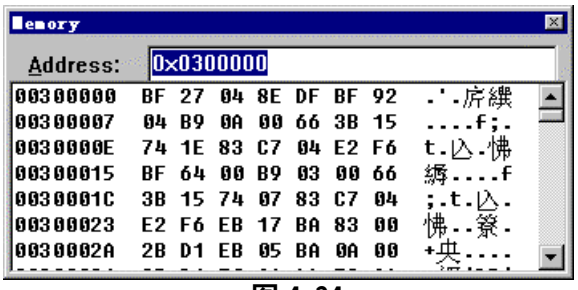


图 4.24

♦ 在调试过程中，有时我们需要查看或改寄存器中的值。我们只需：

(1) 从 View 菜单中选取 Debug window 及 Registers 子选项。Registers 窗口出现。在 Registers 窗口中，信息以 Register = Value 的形式显示，其中 Register 代表寄存器的名字，Value 代表寄存器中的值。

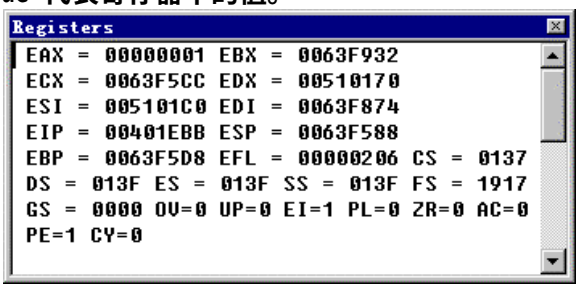


图 4.25

(2) 如果你要修改某一个寄存器的值，用 TAB 键，或鼠标将光标移到你想改变的值的右边，然后输入你想要的值。回车返回。

在寄存器中，有一类特殊的寄存器称为标志寄存器，其中有八个标志位：

OV 是溢出标志；

UP 是方向标志；

EI 是中断使能标志；

Sign 是符号标志，

Zero 是零标志。

Parity 是奇偶较验标志。

Carry 是进位标志。

2 高级调试技术

前面我们讲了调试工具的使用，利用它可以就进行常规的调试，即使程序在某处停下来，再观察程序的当前壮态。而且这些工具在且它调试器中也有。但我们知道我们知道在 VC 程序的开发过程中，光有这些工具是不够的。为了更快更好地开发程序，我们还需要利用更高级的调试工具。我们知道，在利用 VC 开发过程中，利用 MFC 将会极大地方便应用程序的开发，所以开发人员往往是利用 MFC 来开发应用程序，正是这个原因 Microsoft 公司在 MFC 中提供了一些特性来帮助你进行程序的调试。

我们知道在 MFC 中，绝大多数类都是从一个叫做 Cobject 的类继承过来的，虽然这是一个虚基类，但它定义了许多成员函数，其中许多成员函数是用来支持程序的调试的，如 Dump ,Assertvalid 等成员函数。另外他们都支持如 TRACE,ASSERT 等宏，并支持内存漏洞的检查等等。我们知道，为了支持调试，类库肯定在在性能上有所损失，为此 Microsoft 公司提供了两个不同的版本的类库：Win32 Debug 版本和 Win32 Release 版本。在前面我们已经提到，每当我们建立一个工程时，我们也有对应的两个版本。在你的 DEBUG 版本的工程中，编译器连接 DEBUG 版本的 MFC 类库；在你的 RELEASE 版本的工程中编译器连接 RELEASE 版本的 MFC 类库以获得尽可能快的速度。下面我们来介绍这些工具的利用。

2.1 TRACE 宏的利用

TRACE 宏有点象我们以前在 C 语言中用的 Printf 函数，使程序在运行过程中输出一些调试信息，使我们能了解程序的一些状态。但有一点不同的是：TRACE 宏只有在调试状态下才有所输出，而以前用的 Printf 函数在任何情况下都有输出。和 Printf 函数一样，TRACE 函数可以接受多个参数如：

```
int x = 1;
int y = 16;
float z = 32.0;
TRACE( "This is a TRACE statement\n" );
TRACE( "The value of x is %d\n", x );
TRACE( "x = %d and y = %d\n", x, y );
TRACE( "x = %d and y = %x and z = %f\n", x, y, z );
```

要注意的是 TRACE 宏只对 Debug 版本的工程产生作用，在 Release 版本的工程中，TRACE 宏将被忽略。

2.2 ASSERT 宏的利用

在开发过程中我们可以假设只要程序运行正确，某一条件肯定成立。如不成立 ，那么我们可以断言程序肯定出错。在这种情况下我们可以利用 ASSERT 来设定断言。ASSERT 宏的参数是一个逻辑表达式，在程序运行过程中，若该逻辑表达式为真，则不会发生任何动作，若此表达式为假，系统将弹出一个对话框警告你，并停止程序的执行。同时要求你作出选择：Abort，Ignore，Retry。若你选择 Abort，系统将停止程序的执行;若你选择 Ignore 系统将忽略该错误，并继续执行程序；若你选择 Retry ，系统将重新计算该表达式，并激活调试器。同 TRACE 宏一样，ASSERT 宏只 DEBUG 版本中起作用，在 RELEASE 版本中 ASSERT 宏将被忽略。

2.3 ASSERT_VALID 宏的利用以及类的 AssertValid()成员函的重载

ASSERT_VALID 宏用来在运行时检查一个对象的内部合法性，比如说现在 有一个学生对象，我们知道每个学生的年龄一定大于零，若年龄小于零，则该学生对象肯定有问题。事实上，ASSERT_VALID 宏就是转化为对象的成员函数 AssertValid()的调用，只是这种方法更安全。它的参数是一个对象指针，通过这个指针来调用它的 AssertValid()成员函数。

与此相配套，每当我们创建从 Cobject 类继承而来的一个新的类时，我们可以重载该成员函数，以执行特定的合法性检查。

2.4 对象的 DUMP 函数的利用

Dump 函数用来按指定的格式输出一个对象的成员变量，来帮助你诊断一个对象的内部情况。与 AssertValid 成员函数一样，Dump 也是 Cobject 类的成员函数。Dump 函数的参数是一个 CdumpContext 对象，你可以象利用流一样往向这个对象中输入数据。当你创建一个 Cobject 继承而来的 新类时，你可以按如下步骤重载你自己的 Dump 函数：

(1) 调用基类的 Dump 函数，以输出基类的内容;

(2) 向 Cdumpcontest 对象输出该类的数据.

例如,典型的 Dump 函数定义如下:

```
#ifdef _DEBUG
void CPerson::Dump( CDumpContext& dc ) const
{
    // call base class function first
    CObject::Dump( dc );

    // now do the stuff for our specific class
    dc << "last name: " << m_lastName << "\n"
        << "first name: " << m_firstName << "\n";
}
#endif
```

你可能已经注意到整个函数的定义都包含在#ifdef _DEBUG 和#endif 中，这使得 Dump 成员函数只在 DEBUG 版本中发生作用，而对 RELEASE 版本不发生作用。

3 内存漏洞的检查

也许你已经知道，在 C++和 C 语言中指针问题也就是内存申请与释放是一个令人头疼的事情，假如你申请了内存，但没有释放，并且你的程序需要长时间地运行，那么，系统的资源将逐渐减少，当系统的资源全部被用完时，系统将会崩溃。所以在开发程序的过程中一定要保证资源的完全释放。下面我们来介绍内存漏洞的检查。

也许你会问，系统是怎样支持内存漏洞的检查的？其实在你的 Debug 版本中所有的有关内存分配的函数都是被重载过的，具体过程是这样的，当你的程序申请内存时，它首先调用一般的内存分配函数分配一块稍大的内存块。在这一内存块中分为四个小块：Heap Information，buffer ，User memory block，buffer。第一块为有关堆的信息，比如，申请该内存的地点(文件名，行号)，此内存块的类型(如整型，浮点，或某一类的对象)等

等。第二块是一个缓冲区,用于截获用户对其申请内存使用越界的情况。第三块是真正给用户的内存，返回的指针也是指向这儿。第四块也是一个缓冲区，作用同第二块。

当你申请的内存均被记录在案后，要检查内存漏洞就比较容易了，粗略地说，假如你要检查某一程序段是否有内存漏洞，你只需在这一程序 段的开始要求系统为你做一个内存使用情况的映象，记录下程序开始时的内存使用情况，然后在程序段的末尾再使系统为你做一次内存映象，比较两次映象，以检查是否有没释放的内存，假如有未释放的内存，根据这一块中有关分配情况的信息来告诉用户在那儿申请的内存没释放。

具体地讲检查内存漏洞需要以下几个步骤：

- 在你所检测的程序段的开始处建立一个 CmemoryState 对象，调用其成员函数 Checkpoint，以取得当前内存使用情况的快照；
- 在你所检测的程序段的末尾处再建立一个 CmemoryState 对象，调用其成员函数 Checkpoint ,以取得当前内存使用情况的快照；
- 再建立第三个 CmemoryState 对象，调用其成员函数 Difference,把第一个 CmemoryState 对象和第二个 CmemeoryState 对象作为其参数.,如果两次内存快照不相同，则该函数返回非零，说明此程序 段中有内存漏洞。下面我们来看一个典型的例子：

```
      // Declare the variables needed
      #ifdef _DEBUG
      CMemoryState oldMemState, newMemState, diffMemState;
      OldMemState.Checkpoint();
      #endif
    // do your memory allocations and deallocations...
      CString s = "This is a frame variable";
      // the next object is a heap object
      CPerson* p = new CPerson( "Smith", "Alan", "581_0215" );
      #ifdef _DEBUG
      newMemState.Checkpoint();
      if( diffMemState.Difference( oldMemState, newMemState ) )
      {
        TRACE( "Memory leaked!\n" );
      }
#endif
```

五 Visual C++与多媒体

对于一般的应用程序来说，Visual C++ 可以说是包罗万象，然而令人遗憾的是，几乎没有听说过Visual C++ 对多媒体提供过什么支持，甚至有人说Visual C++不适合多媒体编程。若是我们完全使用Visual C++的类库而不想点花招的话，恐怕连最一个简单的RPG游戏都编不出来。对于一个需要大量动画、声音的多媒体应用程序来说，Visual C++ 最多提供了一个外壳，而编制一个优秀的声音、动画引擎的任务，就落到了程序员的身上。

在以后各章节中，将陆续介绍大家如何开发这个引擎。

1 对声音的处理

1．1 媒体控制接口

MCI(Media Control Interface)媒体控制接口是MircroSoft提供的一组多媒体设备和文件的标准接口，它的好处是可以方便地控制绝大多数多媒体设备包括音频、视频、影碟、录像等多媒体设备，而不需要知道它们的内部工作状况。但是古人云：成也萧何，败也萧何。MCI虽然看上去高大全，但对于一些高级应用来说，它是远远不够的。好比Visual C++虽然看上去无所不能，却需要程序员自己开发多媒体引擎一样。对于MCI指令集，我们将只作简单介绍，重点放在后面的波形文件混音器上。

MCI的控制方式：

一般说来，程序员使用两个函数就可以与MCI打交道了：

```
MCIERROR mciSendCommand(MCIDEVICEID wDeviceID, UINT uMsg,
                        DWORD dwFlags, DWORD dwParam );
```

命令字符串方式，用接近于日常生活用语的方式发送控制命令，适用于高级编程如VB、TOOLBOOK等。

```
MCIERROR mciSendString(LPCTSTR lpszCommand,LPTSTR lpszReturnStr
                       ing, UINT cchReturn, HANDLE hwndCallback
                       );
```

命令消息方式，用专业语法发送控制消息，适用于VC等语言编程，此方式直接与MCI设备打交道。

对于mciSendCommand，第一个参数指定了设备标识，这个标识会在程序员打开MCI设备时由系统提供。第二个参数指定将如何控制设备，详细请查阅后面“MCI指令”一栏。第三个参数为访问标识，第四个参数一般是一个数据结构，标识程序在访问MCI时要的一些信息。有关详细资料，请查阅本光盘配套书。

对于mciSendString，第一个参数为一串控制字符串，返回信息由系统填入第二个参数，第三个参数指明返回信息的最大长度，若对MCI装置设定了"notify"标志则需要在第四个参数填上返回窗口句柄。

举例：

```
mciSendCommand(DeviceID,MCI_CLOSE,NULL,NULL);//关闭一个MCI设备；
mciSendString("open aaa.avi",0,0,0);  //打开文件"aaa.avi";
```

MCI的设备类型：

MCI的设备类型有：

设备描述	描述字符串	说明
MCI_ALL_DEVICE_ID		所有设备
MCI_DEVTYPE_ANIMATION	Animation	动画设备
MCI_DEVTYPE_CD_AUDIO	Cdaudio	CD音频
MCI_DEVTYPE_DAT	Dat	数字音频
MCI_DEVTYPE_DIGITAL_VIDEO	Digitalvideo	数字视频
MCI_DEVTYPE_OTHER	Other	未定义设备
MCI_DEVTYPE_OVERLAY	Overlay	重叠视频
MCI_DEVTYPE_SCANNER	Scanner	扫描仪
MCI_DEVTYPE_SEQUENCER	Sequencer MIDI	序列器
MCI_DEVTYPE_VCR	Vcr	合式录像机
MCI_DEVTYPE_VIDEODISC	Videodisc	激光视盘
MCI_DEVTYPE_WAVEFORM_AUDIO	waveaudio Wave	音频

对于未在上面定义的MCI设备，用户可查看system.ini文件中[mci]部分，例如：

```
[mci]
cdaudio=mcicda.drv
sequencer=mciseq.drv
waveaudio=mciwave.drv
avivideo=mciavi.drv
videodisc=mcipionr.drv
vcr=mcivisca.drv
ActiveMovie=mciqtz.drv
QTWVideo=mciqtw.drv
MPEGVideo=C:\PROGRA~1\XING\XINGMP~1\xmdrv95.dll
```

其中最后两句分别指明了Apple的QuickTime设备，设备名为"QTWVidio"、MPEG影像设备，设备名为"MPEGVideo"。

在MCI编程中，既可以将设备描述当设备名，也可以将描述字符串当设备名，一个极端偷懒的办法是程序员不要在程序中指定设备名，Windows将自动根据文件扩展名识别设备类型。

举个例子来说，打开一个多媒体文件有以下三种方式：

[1]：自动识别：打开一个"WAV"文件

```
MCI_OPEN_PARMS mciOpen;
mciOpen.lpstrDeviceType=0;
mciOpen.lpstrElementName="aaa.wav";
mciSendCommand(NULL,MCI_OPEN,    MCI_OPEN_ELEMENT,
                (DWORD)&mciOpen);
```

[2]：指定设备描述：打开CD播放器

```
MCI_OPEN_PARMS  mciOpen;
mciOpen.lpstrDeviceType= (LPSTR)MCI_DEVTYPE_CD_AUDIO ;
mciSendCommand(NULL,MCI_OPEN,MCI_OPEN_TYPE | MCI_OPEN_TYPE_ID,
               (DWORD)&mciOpen);
```

[3]：指定描述字符串： 打开一个AVI文件

```
MCI_OPEN_PARMS  mciOpen;
mciOpen.lpstrDeviceType="avivideo";
mciOpen.lpstrElementName="aaa.avi";
mciSendCommand(NULL,MCI_OPEN,MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                (DWORD)&mciOpen);
```

注意三种打开方式中，函数第三个参数的区别，后面会讲到这种区别。

MCI指令

MCI使用如下指令：

MCI_BREAK	*设置中断键，缺省是"CTRL+BREAK"*
MCI_CAPTURE	抓取当前帧并存入指定文件，仅用于数字视频
MCI_CLOSE	关闭设备
MCI_CONFIGURE	弹出配置对话框，仅用于数字视频
MCI_COPY	拷贝数据至剪贴板
MCI_CUE	延时播放或录音
MCI_CUT	删除数据
MCI_DELETE	删除数据
MCI_ESCAPE	仅用于激光视频
MCI_FREEZE	将显示定格
MCI_GETDEVCAPS	获取设备信息
MCI_INDEX	当前屏幕显示与否，仅用于VCR设备
MCI_INFO	获取字符串信息
MCI_LIST	获取输入设备数量，支持数字视频和VCR设备
MCI_LOAD	装入一个文件
MCI_MARK	取消或做一个记号，与MCI_SEEK配套
MCI_MARK	取消或做一个记号，与MCI_SEEK配套
MCI_MONITOR	为数字视频指定报告设备
MCI_OPEN	打开设备
MCI_PASTE	粘帖数据
MCI_PAUSE	暂停当前动作
MCI_PLAY	播放
MCI_PUT	设置源、目的和边框矩形
MCI_QUALITY	定义设备缺省质量
MCI_RECORD	开始录制
MCI_RESERVE	分配硬盘空间
MCI_RESTORE	拷贝一个bmp文件至帧缓冲
MCI_RESUME	使一个暂停设备重新启动
MCI_SAVE	保存数据
MCI_SEEK	更改媒体位置
MCI_SET	设置设备信息
MCI_SETAUDIO	设置音量
MCI_SETTIMECODE	启用或取消VCR设备的时间码
MCI_SETTUNER	设置VCR设备频道
MCI_SETVIDEO	设置video参数
MCI_SIGNAL	在工作区上设置指定空间
MCI_STATUS	获取设备信息
MCI_STEP	使播放设备跳帧
MCI_STOP	停止播放
MCI_SYSINFO	返回MCI设备信息
MCI_UNDO	取消操作
MCI_UNFREEZE	使使用MCI_UNFREEZE的视频缓冲区恢复运动
MCI_UPDATE	更新显示区域
MCI_WHERE	获取设备裁减矩形
MCI_WINDOW	指定图形设备窗口和窗口特性

其中比较常用的指令有MCI_OPEN、MCI_CLOSE、MCI_PLAY、MCI_STOP、MCI_PAUSE、MCI_STATUS等等。

实例分析

在Visual C++ 5.0菜单上点取new，单击projects，选取MFC AppWizard(exe)，创建一个新的名为"mcitest"的工程文件，OK确定。

注意在应用程序类型中选择"Dialog based"，然后Finish完成。这是一个基于对话框的应用程序，为了完成MCI测试的任务，我们要更改一下对话框资源。点取"Resource View"，在"Dialog"下选取"IDD_MCITEST_DIALOG"对话框，依次添加Button如图所示。

完成对话框的修改。右键单击mcitest files、选取Add Files To Project，加入配套光盘中提供的"commci.cpp"和"commci.h"文件。打开ClassWizard，在Class Name下选择CMcitestDlg，加入所有的按键消息处理函数。

在"cmcitestDlg"类中,分别用"COMMCI"定义Wav、Midi、Avi三个成员变量,在按钮响应过程中分别写上相应处理函数open( )、play( )、close( )、pause( )、stop( )。

在"projoct"菜单下单击setting,弹出设置对话框,在"link"下"object/library modules"下加入"winmm.lib"，编译并运行程序:

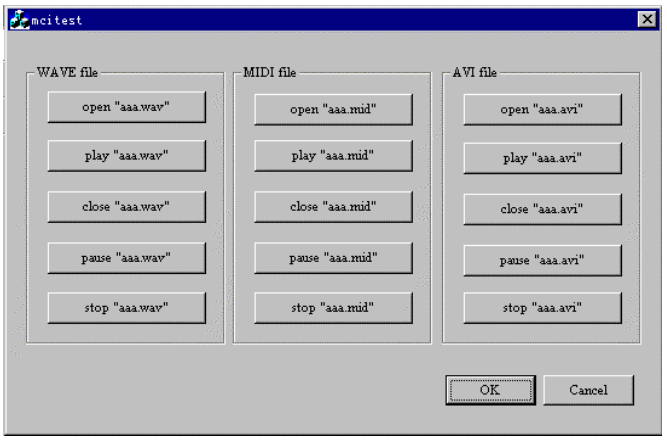


图5.1

源程序介绍

```
// commci.h: interface for the commci class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_COMMCI_H__90CEFD04_CC96_11D1_94F8_0000B431BBA1__INCLUDED_)
#define AFX_COMMCI_H__90CEFD04_CC96_11D1_94F8_0000B431BBA1__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

//#include <windows.h>
#include <mmsystem.h>

class COMMCI
```

```
{
private:
     HWND                hOwer;                      //窗口的拥有者
     MCI_OPEN_PARMS      mciOpen;

public:
     COMMCI();
     ~COMMCI()           {Close();  }
     MCIERROR   Open(LPCSTR DeviceType,LPCSTR filename);
          //通过描述字符串打开设备
     MCIERROR   Open(int DeviceType,LPCSTR filename); //通过设备类型打开设备
     MCIERROR   Open(LPCSTR filename);               //自动检测设备
     void  Play(HWND hWnd);                       //播放MCI，hWnd为回调窗口句柄
     void  Close(void);                              //关闭设备
     void  Stop(void);                               //停止设备
     void    Pause(void);                            //暂停设备
     DWORD Query();                                  //检测设备
};

////////////////////////////////////////////////////////
//  COMMCI.CPP  中使用到的结构
////////////////////////////////////////////////////////
//typedef struct tagMCI_OPEN_PARMS {
//    DWORD       dwCallback;
//  MCIDEVICEID wDeviceID;
//  WORD        wReserved0;
//  LPCSTR      lpstrDeviceType;
//  LPCSTR      lpstrElementName;
//  LPCSTR      lpstrAlias;
//} MCI_OPEN_PARMS, FAR *LPMCI_OPEN_PARMS;

//typedef struct tagMCI_PLAY_PARMS {
//  DWORD   dwCallback;
//  DWORD   dwFrom;
//  DWORD   dwTo;
//} MCI_PLAY_PARMS, *PMCI_PLAY_PARMS, FAR *LPMCI_PLAY_PARMS;

//typedef struct tagMCI_STATUS_PARMS {
//  DWORD   dwCallback;
//  DWORD   dwReturn;
//    DWORD   dwItem;
//  DWORD   dwTrack;
//} MCI_STATUS_PARMS, *PMCI_STATUS_PARMS,
               FAR * LPMCI_STATUS_PARMS;


////////////////////////////////////////////////////////
//    mci 初始化方式
////////////////////////////////////////////////////////
//COMMCI.Open("waveaudio",filename);   wave ; *.wav ,
```

```
//COMMCI.Open("sequencer",filename);   midi ; *.mid , *.rmi
//COMMCI.Open("reelmagic",filename);   vcd  ; *.mpg ,
//COMMCI.Open("avivideo",filename);    avi  ; *.avi ,


//////////////////////////////////////////////////////
//   mci 状态返回值
//////////////////////////////////////////////////////
//    case MCI_MODE_NOT_READY:
//    case MCI_MODE_STOP:
//    case MCI_MODE_PLAY:
//    case MCI_MODE_RECORD:
//    case MCI_MODE_SEEK:
//    case MCI_MODE_PAUSE:
//    case MCI_MODE_OPEN:

#endif // !defined(AFX_COMMCI_H__90CEFD04_CC96_11D1_94F8_0000B431BBA1__INCLUDED
_)


// commci.cpp: implementation of the commci class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
//#include "mcitest.h"
#include "commci.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

COMMCI::COMMCI()
{
     memset(this,0,sizeof(COMMCI));
}

MCIERROR COMMCI::Open(LPCSTR DeviceType,LPCSTR filename)
{
     //如果有打开的设备就关闭
     if (mciOpen.wDeviceID) Close();
     //初始化MCI_OPEN_PARMS结构
     mciOpen.lpstrDeviceType=DeviceType;
     mciOpen.lpstrElementName=filename;
     //除了打开设备设备代码为0，下面的任何mciSendCommand语句都要指定设
```

```
    //备代码。
    if ( mciSendCommand(NULL,MCI_OPEN,
                    MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                                    (DWORD)&mciOpen))
         return FALSE;
    return TRUE;
}

MCIERROR COMMCI::Open(LPCSTR filename)
{
    if (mciOpen.wDeviceID) Close();
    mciOpen.lpstrElementName=filename;
    if ( mciSendCommand(NULL,MCI_OPEN,
                    /*MCI_OPEN_TYPE |*/ MCI_OPEN_ELEMENT,
                                      (DWORD)&mciOpen))
         return FALSE;
    return TRUE;
}

MCIERROR COMMCI::Open(int DeviceType,LPCSTR filename)
{
    if (mciOpen.wDeviceID) Close();
    mciOpen.lpstrDeviceType=(LPCSTR)DeviceType;
    mciOpen.lpstrElementName=filename;
    return  mciSendCommand(NULL,MCI_OPEN,
                    MCI_OPEN_TYPE | MCI_OPEN_TYPE_ID ,(DWORD)&mciOpen);
}

void COMMCI::Play(HWND hWnd)
{
    MCI_PLAY_PARMS       mciPlay;
    hOwer=hWnd;                        //回调窗口句柄
    //MCI_PLAY_PARMS结构只需要设定回调窗口句柄
    mciPlay.dwCallback=(WORD)hOwer;
    mciSendCommand(mciOpen.wDeviceID,MCI_PLAY,MCI_NOTIFY,
                                          (DWORD)&mciPlay);
}

void COMMCI::Close(void)
{
    if (mciOpen.wDeviceID)
         mciSendCommand(mciOpen.wDeviceID,MCI_CLOSE,NULL,NULL);
    memset(this,0,sizeof(COMMCI));
}

void COMMCI::Stop(void)
{
    if (mciOpen.wDeviceID)
         mciSendCommand(mciOpen.wDeviceID,MCI_STOP,NULL,NULL);
}
```

```
void COMMCI::Pause(void)
{
     if (mciOpen.wDeviceID)
          mciSendCommand(mciOpen.wDeviceID,MCI_PAUSE,NULL,NULL);
}


DWORD COMMCI::Query()
{
     MCI_STATUS_PARMSmciStatus;
     mciStatus.dwItem=MCI_STATUS_MODE;
     mciSendCommand(mciOpen.wDeviceID,MCI_STATUS,
                    MCI_STATUS_ITEM,(LPARAM)&mciStatus);
     return mciStatus.dwReturn;
};
```

对于类COMMCI定义了如下几个成员函数：

- 一个构造函数和一个析构函数。
- 一个打开函数，其参数分别是要打开设备的类型和文件名。
- 播放函数，其参数是回调函数句柄。
- 关闭函数。
- 停止函数。
- 暂停函数。
- 状态检测函数。

在一个MCI的处理过程中，必须使用以下流程：

- 打开一个MCI设备。
- 然后Open播放MCI设备，其间可以暂停和停止播放，
- 最后关闭MCI设备。

在以上任何步骤中，都可以用状态检测函数检测工作状态。

下面我们看一下MCI的实现过程：

1．OPEN MCI

首先，我们 初始化一个MCI_OPEN_PARMS的结构，其中要用到两个值。其中mciOpen.lpstrDeviceType指定了要打开的设备类型，这些设备类型可从前面的“MCI设备类型”选取。可以是标识或描述字符串，例如语句mciOpen.LpstrDeviceType=MCI_DEVTYPEVCR与语句mciopen.LpstrDeviceType="Vcr"是等价的。若不指定类型则计算机将根据文件名自动识别设备，接下来mciOpen.LpstrElimmentName指定了要打开的文件名，最后调用MciSendComand指定计算机将在结构的wDeviceID中填入打开的设备代码；以后应用程序将根据此设备代码访问MCI设备。

这里谈一下三种打开方式的区别：

[1]：自动识别：打开一个"WAV"文件

```
MCI_OPEN_PARMS mciOpen;
mciOpen.lpstrDeviceType=0;
mciOpen.lpstrElementName="aaa.wav";
mciSendCommand(NULL,MCI_OPEN,    MCI_OPEN_ELEMENT,
                                        (DWORD)&mciOpen);
```

[2]：指定设备描述：打开CD播放器

```
MCI_OPEN_PARMS  mciOpen;
mciOpen.lpstrDeviceType= (LPSTR)MCI_DEVTYPE_CD_AUDIO ;
mciSendCommand(NULL,MCI_OPEN,MCI_OPEN_TYPE | MCI_OPEN_TYPE_ID,
                                        (DWORD)&mciOpen);
```

[3]：指定描述字符串： 打开一个AVI文件

```
          MCI_OPEN_PARMS  mciOpen;
mciOpen.lpstrDeviceType="avivideo";
mciOpen.lpstrElementName="aaa.avi";
mciSendCommand(NULL,MCI_OPEN,MCI_OPEN_TYPE
             | MCI_OPEN_ELEMENT,
              (DWORD)&mciOpen);
```

请注意mciSendCommand函数第三个参数的区别：

MCI_OPEN_TYPE：表示要使用MCI_OPEN_PARMS结构中的LpstrDiviceType参数，这可区分指定设备打开方式和自动识别方式之间的区别。在自动方式中，不需使用LpstrDeviceType参数。因此，也不需指定MCI_OPEN_TYPE。

MCI_OPEN_ELEMENT：表示LpstrDeviceType参数中的是设备表述字符串。

MCI_OPEN_TYPE_ID：表示LpstrDeviceType参数中的是设备描述。

2 PlayMci：

在play函数中，需要一个返回窗口句柄，以便应用程序在播放结束后向此窗口发送一个消息，告诉窗口已经播放结束。我们首先初始化一个MCI_PLAY_PARMS的数据结构：将其中dwCallback参数赋与窗口句柄。然后调用mciSendCommend，当然发送的指令是MCI_PLAY,告诉系统开始播放，另外第三个参数指定MCI_NOTIFY，告诉系统播放完后要通知自己。

QueryMci：

要想检测MCI播放状态，就要发送指MCI_STATUS，并标志MCI_STATUS_ ITEM，返回值在结构MCI_STATUS_PARMS的dwReturn上。

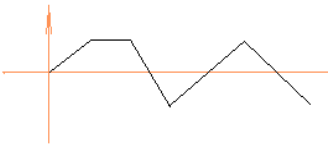
关于MCI的进一步详细情况，本章节将不再讲述，有关MCI中用到的命令详解、 数据结构，请大家自己查阅本书配套图书。

1．2 波形混音器

假如你是一位游戏爱好者，不知道有没有注意过不同的游戏音乐，音效实现上的区别。比较早的游戏一般采用midi加上单个wav文件的方式，例如《天使帝国》，早期的“三国志”等等，这些游戏一般以midi作背景音乐。遇到人物走路、打架或天灾人祸时用一些波形文件衬托一下气氛，只是Midi文件在早期声卡上表现实在单薄，以一个发烧友的眼光来看实在不堪入耳。后来采用CD+单个wav的声音方式，音质有了明显改善，例如《仙剑奇侠传》。但由于wav在同一时刻只能播放一个，毕竟单薄。很难想象两人打架只有一人发出“嘿嘿”的声音。因此多波形混音成了游戏编程主流，大家看“红色警报”中坦克的轰鸣声，人物的哭叫声，飞机的咆啸声，电塔的滋滋声与出色的背景音乐溶合一起，令人心旷神怡情不自禁。想要编出这样的程序，就要掌握波形混音的原理。

什么是Wav文件：

Wav文件直接反映了一个声音在每个时刻的大小值，比如说以下一段波形：



我们按每人0.1秒取一点，得到的wav文件数值就是0,1,1,-1,0,1。因此，假如我们能把许多Wav文件的数据直接相加，你听到的就是所有的声音，这就是混音器的原理。

下面我们分析一下Wav文件结构：

我们可以打开一个Wav文件直接看其二进制码：

C:\user\wave\22.wav

```
00000000 5249 4646 9CB6 1E00 5741 5645 666D 7420
00000010 1000 0000 0100 0200 2256 0000 44AC 0000
00000020 0200 0800 6461 7461 78B6 1E00 7F7F 7F7F
00000030.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
00000040.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
00000050.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
00000060.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
00000070.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
00000080.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
00000090.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
000000A0.7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F 7F7F
```

大家可以看到wav文件存储格式如下：

```
"RIFF"
××××        文件大小
"WAVE"
"fmt"
××××        PCMWAWFORMAT——数据结构大小
××××
 ……         数据结构“PCMWAVEFORMAT”
××××
data
××××        数据大小
：           数据
```

首先是字符串“RIFF”，表示此文件遵循一种标准格式名为“资源互换文件格式”（Resource Intercharge Format）。后面紧跟四个字节指明文件大小。其次是字符串“WAVE”和“fmt”，后面紧跟一个名为“PCMWAVEFORMAT”的结构，最后是字符串“data”，紧跟数据大小及所有数据。PIFF文件为一种树状结构，基本构成单位是“块”，图5.2中是wav文件中“块”关系图。

图5.2

如图所示，wav文件结构为两层，由父块“RIFF”和两个子块“fmt”、“data”组成。“fmt”块包含wav文件格式信息，“data”块包含数据信息。

2 多媒体文件 I/O

多媒体文件I/O与普通文件I/O相似，但支持多媒体“RIFF”格式，并提供了缓冲和非缓冲文件I/O。

所有的多媒体文件I/O函数名前缀为mmio，消息名前缀为MMIO。

低级波形音频函数：

低级音频服务允许用户直接与音频设备驱动程序打交道，直接控制音频设备如波形，Midi的播放与记录，低级音频函数是一个设备无关接口。

低级音频函数前缀均为wave，按输入函数、输出函数区分为WaveIn××××和WaveOut××××。

波形音频的重放过程。

首先，我们要调用多媒体文件I/O函数mmIO××××(　)，并根据多媒体文件I/O生成在wave重放中需要的结构和数据，并将这些结构和数据用waveOut××××(　)函数重放。

用户可以根据加密的需要将wave文件篡改，去掉文件头和“fmt”块，只保留数据块。 或者将其压缩，只要重放时能在内存中还原出数据文件。并记得文件音频格式和大小，就能重放音频。

常用mmio函数及实现过程简介：

```
mmioOpen(  )          打开一个RIFF文件
mmioDescend (  )      进入块
mmioRead(  );         该取RIFF文件
mmioAscend (  );      跳出块
mmioClose(  );        关闭PIFF文件
```

对于块来说，进入块和跳出块是配对的。

读取WAV文件的读取过程：

```
mmioOpen(  )          打开文件
      ↓
mmioDescend ("WAVE")  进入"fmt"块
      ↓
mmioRead(  )          读取WAVE文件格式信息
      ↓
mmioAscend (  )       跳出"fmt"块
      ↓
mmioDescend ("data")  进入"data"块
      ↓
mmioRead(  )          读取WAVE数据信息
      ↓
mmioClose(  )         关闭文件。
```

输出WAV文件的过程：

```
WaveOutOpen ()              打开一个输出设备
      ↓
WaveOutPrepareHeader()      准备WAVE数据头。
      ↓
WaveOutWrite()              将数据写入设备并开始播放
      ↓
WaveOutReset()              停止播放并重置管理器
      ↓
WaveOutClose()              并闭播放设备
      ↓
```

WaveOutUnpareHeader()　　清理用WaveOutPrepareHeader准备的Wave。

实例:

这个实例实现的功能是首先打开背景音乐,然后每五秒钟加入一段配音。背景音乐放完后将停止播放，大家可以听一下背景+配音+配音+……+配音产生的实际效果。为了实现这个功能，我们封装了一个类。大家可以在光盘上找到这两个文件“wavemix..h”和“wavemix.cpp”.

首先，我们看一下“wavmix.h”,这里定义了一个称为“mwave”的类，

其中成员函数有：构造、析构函数、open(打开文件)、play(播放文件)、Add（往缓冲中加混音文件）、Stop（停止播出）、close（关闭输出设备，类重新初始化）。

现在我们照前面方法建一个基于对话框的程序，在“File View”中加入上述两个文件。打开“WavemixDlg.h”,在前面加上“#include "wavemix.h"”,

在类class cWavemixDlg中加入私有数据“MWAVE mWave”，在类“cwavemixDlg”中加入成员OnlnitDialog( )和OnTime()，,

在此函数中加入一个定时器，打开并播放背景音乐。在定时器的向应过程OnTime()中加入函数mwave.Add("2.waw".)使之每五秒钟向缓冲加入一个配音。编译并运行之。

源程序分析

```
// wavemix.h : main header file for the WAVEMIX application
//
#if !defined(AFX_WAVEMIX_H__54F4AF66_CE37_11D1_94F8_0000B431BBA1__INCLUDED_)
#define AFX_WAVEMIX_H__54F4AF66_CE37_11D1_94F8_0000B431BBA1__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
     #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"      // main symbols
/////////////////////////////////////////////////////////////////////////////
#include "windows.h"
#include "mmsystem.h"
#ifdef WIN32
#define WAVDATA BYTE
#else
#define WAVDATA BYTE _huge
#define WAVEFORMATEX PCMWAVEFORMAT
#endif

#ifdef WIN32
#define WAV16DATA     WORD
#else
#define WAV16DATA     WORD _huge
#endif

class MWAVE
```

```
    {
        private:
            BOOL            OpenFlage;
            DWORD           DataSize;
            HGLOBAL         hData;
            WAVDATA*        lpData;

            PCMWAVEFORMAT   pFormat;
            WAVEHDR         WaveHead;
            HWAVEOUT        hWaveOut;

        public:
            MWAVE(){memset(this,0,sizeof(MWAVE));};
            ~MWAVE(){Close();};
            int Open(char*);      //打开一个WAV文件
            int Play(HWND);       //播放一个WAV文件
            int Add(char*);       //往正在播放的WAV设备中添加WAV
                                  文件
            int Stop();           //停止播放
            int Close();          //关闭设备
    };

#endif // !defined(AFX_WAVEMIX_H__54F4AF66_CE37_11D1_94F8_0000B431BBA1__INCLUDE
D_)

// wavemix.cpp:Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "wavemix.h"
#include "wavemixDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// wavemix Class
/////////////////////////////////////////////////////////////////////////////

int MWAVE::Open(char* name)
    {
        HMMIO               hMmio;
        MMCKINFO pinfo;
        MMCKINFO cinfo;

        if(hMmio)Close();

        //打开WAV文件，返回一个HMMIO句柄
```

```
            hMmio=mmioOpen(name,NULL,MMIO_READ);
            if(!hMmio)return FALSE;
            OpenFlage=1;

            //查找父块"wave";
            pinfo.fccType=mmioFOURCC('W','A','V','E');
            if(mmioDescend(hMmio,&pinfo,NULL,MMIO_FINDRIFF))goto FALSE_END;

            //查找子块"fmt"  parent"riff";
            cinfo.ckid=mmioFOURCC('f','m','t',' ');
            if(mmioDescend(hMmio,&cinfo,&pinfo,MMIO_FINDCHUNK))
              goto FALSE_END;

            mmioRead(hMmio,(LPSTR)&pFormat,sizeof(PCMWAVEFORMAT));//cinfo.cksiz
e);
            if(pFormat.wf.wFormatTag!=WAVE_FORMAT_PCM)
              goto FALSE_END;

            //跳入块"FMT"
            mmioAscend(hMmio,&cinfo,0);

            //查找数据块
            cinfo.ckid=mmioFOURCC('d','a','t','a');
            if(mmioDescend(hMmio,&cinfo,&pinfo,MMIO_FINDCHUNK))
              goto FALSE_END;
            DataSize=cinfo.cksize;

            //读取数据
            hData=GlobalAlloc(GMEM_MOVEABLE
                        | GMEM_SHARE,DataSize);
            lpData=(WAVDATA*)GlobalLock(hData);
            if( !hData || !lpData ) goto FALSE_END;
            if(mmioRead(hMmio,(HPSTR)lpData,DataSize)
                                      !=(LRESULT)DataSize)
              goto FALSE_END;

            //close and return
            mmioClose(hMmio,MMIO_FHOPEN);
            return TRUE;

FALSE_END:
            if(hMmio)mmioClose(hMmio,MMIO_FHOPEN);
            if(lpData)LocalUnlock(hData);
            if(hData)GlobalFree(hData);
            memset(this,0,sizeof(MWAVE));
            return 0;
      }

int MWAVE::Play(HWND hP)
      {
            if(!OpenFlage)return FALSE;
```

```
            //检测系统播放功能
            if(waveOutOpen(NULL,WAVE_MAPPER,
                        (WAVEFORMATEX*)&pFormat,NULL,
                         NULL,WAVE_FORMAT_QUERY))
             return Close();

            if(waveOutOpen(&hWaveOut,WAVE_MAPPER,
                  ( WAVEFORMATEX*)&pFormat,(DWORD)hP,
                                  0,CALLBACK_WINDOW))
             return Close();

            WaveHead.lpData=(LPSTR)lpData;
            WaveHead.dwBufferLength=DataSize;
            WaveHead.dwFlags=0L;
            WaveHead.dwLoops=0L;

            往WAV设备中添加数据
            if(waveOutPrepareHeader(hWaveOut,&WaveHead,
                    sizeof(WAVEHDR)))
            return Close();

            if(waveOutWrite(hWaveOut,&WaveHead,sizeof(WAVEHDR)))
            return Close();

            return TRUE;
       }

//#define min(a, b)  (((a) < (b)) ? (a) : (b))

int MWAVE::Add(char* name)
       {
            register int x;
            if(!OpenFlage)return Open(name);

            MWAVE wav;
            if(!wav.Open(name))return FALSE;

            MMTIME time;
            //获得WAV文件当前播放位置
            time.wType=TIME_BYTES;
            if(waveOutGetPosition(hWaveOut,&time,sizeof(MMTIME)))
                 time.u.cb=0;
            DWORD start=((time.u.cb>>1)<<1);
            DWORD end=min(DataSize_start,wav.DataSize);

            register WAVDATA* lpd=lpData+start;
            for(register DWORD i=0;i<end;i++)
                 {
                      //将两组WAV文件数据相加，并检测数据大小是否合法，如果//数
                      据大小越界，则分别取最大值和最小值
```

```
                    x=(((*(lpd+i))+(*(wav.lpData+i))))_128;
                    if(x<0)x=0;
                    if(x>255)x=255;
                    *(lpd+i)=(BYTE)(x);
               }

          return TRUE;
     }

int MWAVE::Stop()
     {return !waveOutReset(hWaveOut);}

int MWAVE::Close()
     {
          if(hWaveOut)
               {
                    waveOutReset(hWaveOut);
                    waveOutClose(hWaveOut);
                    waveOutUnprepareHeader(hWaveOut,&WaveHead,
                                           sizeof(WAVEHDR));
               }
          if(lpData)LocalUnlock(hData);
          if(hData)GlobalFree(hData);
          memset(this,0,sizeof(MWAVE));
          return 0;
     }
```

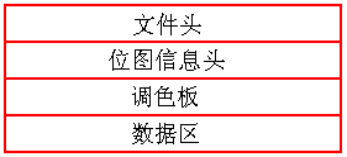
3 多媒体图形图像技术

现在我们讲述windows多媒体中最重要，最核心的技术——图形技术。对于Windows的图形图像技术来说，包括基本的GDI绘制对像如点、线、矩形、图像和位图文件，引而广之所有的动画文件都要利用到windows图像图形技术编程。

本章节我们主要讲述Bmp文件实现过程、调色板应用，及一些Bmp图像合成技术，例如：透空技术、Bmp动画技术等。

Bmp文件结构

Bmp文件由以下几个部分组成：文件头、位图信息头、调色板、数据区。下面这张图显示了Bmp文件结构：



Windows位图显示的必要条件：

我们分析一下一个Windows API函数：

```
int SetDIBitsToDevice(hdc, uXDest, uYDest, uWidth, uHeight, uXSrc, uYSre,
                      uStartScan, cScanLines, lpvBits, lpbmi, fuColorUse)
```

请查看这个函数的第十、十一个参数lpvBits,lpbmi。其中lpvBits指明了指向内存中BMP数据的地址，lpbmi指向一个BITMAPINFO的数据结构，只要有了这两个参数，一个BMP位

图就被确定了。大家可以看到绝大多数的Windows图形图像浏览软件所能分析的文件格式，例如jpg、gif、pcx、tif等等，都是先在内存中建一个数据缓冲区，再根据图形图像格式建立一个BITMAPINFO的数据结构，再利用SetDIBitsToDevice函数写到屏幕上。我们下面几个实例，也基本上采用这个方式。

Windows的调色板

Windows的显示设备可以显示出成千上万种颜色，但是，在同一个屏幕上能同时显示的颜色数并不是成千上万种，我们把显示设备分为单色、十六色、二百五十六色、增强色、真彩色，或者是按照显示位区分成1bit，4bit，8bit，16bit，24bit。由于Windows的理论显示数和实际显示数并不相符，因此需要一个方案来解决这个问题，这就要用到调色板这个概念。为何要使用调色板我们在此并不作详细讨论，只是有一点要弄明白，只有十六色和二百五十六色位图才需要调色板。

下面是使用调色板的方法

首先要读入一个位图文件，再判断这个位图文件的颜色数，假如这个文件是十六色或是二百五六色，那么这个文件中必会有一个调色板信息，读取这个调色板信息，将这个调色板信息转为一个LOGPALETTE的结构，根据这个结构生成逻辑调色板，然后每次在要显示位图前，使用SelectPalette函数将逻辑调色板选入设备描述表hDC，再使用RealizePalette函数实现这个调色板，当显示完成后再使用SelectPalette函数，将旧的调色板选择回来，一个调色板调用过程就完成了。

构造windows图像处理类库

我们假设这个类库包含以下功能：

1.处理调色板；
  a.生成逻辑调色板；
  b.实现调色板；

  2.处理BMP文件
    a.读取BMP图像；
    d.显示图像；
    c.实现以上“处理调色板”的功能；
  3.处理FLC动画
    a.读取FLC动画文件；
    b.播放参数设置；
    c.显示图像；
    d.实现以上“处理调色板”的功能；

通过以上假设我们看到要设计的类库有一些公共特点：

1.都要处理调色板；
2.除了“1”以外，“2”和“3”均包含了windows图像显示的必要条件：结构
  BITMAPINFO和一块bitmap数据区。

为此我们首先构造一个处理调色板的类，这个类的实现上述“处理调色板”的功能。为什么要单独处理调色板而不让它附属于“处理BMP文件”或是“处理FLC动画”，其原因是：在大规模多媒体编程中，往往有几十或几百个BMP图像或动画，却只需要一两个调色板，过度泛滥的调色板往往造成版面切换过程中的闪烁，并浪费内存空间，因此要单独处理调色板。

其次我们构造一个处理DIB图像的类：在这个类里，核心为两个参数（结构BITMAPINFO和一块bitmap数据区）和一个调用API函数SetDIBitsToDevice（）的显示函数Show()。这个类继承于调色板类，因为处理图像文件必须处理调色板。并成为其他图像文件的基类，因为要处理图像文件必须要有这两个结构和函数。

有了这两个基类后，我们再在这两个基类的基础上构造其他类。

```
// MyBmp.h: interface for the MyBmp class.
```

```
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_MYBMP_H__34151075_C57B_11D1_94F8_0000B431BBA1__INCLUDED_)
#define AFX_MYBMP_H__34151075_C57B_11D1_94F8_0000B431BBA1__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define FALSERETURN {_lclose(hFile);return 0;}
#define IMAGECOLORS(x,y)        (1<<((x)*(y)))
//单行BMP数据的字节数
#define DWORD_WBYTES(x)               ((((x)+31UL)>>5)<<2)

#define min(a, b)  (((a) < (b)) ? (a) : (b))

#ifdef WIN32
#define BMPDATA BYTE
#else
#define BMPDATA BYTE _huge
#endif
// #include "test.cpp"

#define PALVERSION      0x0300
#define DIB_STORAGEWIDTH(x) ((x + 3) & ~3)
#define WIDTHBYTES(i) ((i+31)/32*4)
#define MAXPALETTE    256     /* max. # supported palette entries */
#define MAXLOADFLCNUMBER   255

class PALETTE
{
private:
     HPALETTE             hOldPal;
protected:
     HPALETTE                    hPal;

public:
     PALETTE(){memset(this,0,sizeof(PALETTE));}
     ~PALETTE(){if(hPal)DeleteObject(hPal);};
     HPALETTE CreatePal(BITMAPINFO*);
     HPALETTE CreatePal(LPLOGPALETTE Pal);
     HPALETTE GetPal(){return hPal;}
     UINT SetPal(HDC);
    void ResetPal(HDC);

     };


class MYDIB : public PALETTE
{
```

```
protected:
     HGLOBAL                         hData;
     struct
     {
          BITMAPINFOHEADERb;
          RGBQUAD                    r[256];
     }p;

public:
     LPBITMAPINFO              lpInfo;
     LPBITMAPINFOHEADER    lpInfoHead;

     MYDIB();
     ~MYDIB();
     int   Show(HDC,int,int,int,int,int,int,int,int,DWORD);
     inline int      Show(HDC hDC,int x1,int y1,int x2,int y2,
                              int x3,int y3,int x4,int y4)
          {    return Show(hDC,x1,y1,x2,y2,x3,y3,x4,y4,SRCCOPY);}
     inline int      Show(HDC hDC,int x,int y)
          {    return Show(hDC,x,y,0,0,0,0,0,0,SRCCOPY);}
     inline int      Show(HDC hDC)
          {    return Show(hDC,0,0,0,0,0,0,0,0,SRCCOPY);}

     Show(MYDIB*,int,int,int,int,int,int,BYTE,BYTE,BYTE,
                                             BYTE,BYTE,BYTE);
     Show(MYDIB*,int,int,int,int,int,int,register
                                        BYTE,register BYTE);

};

class BMP : public MYDIB
{

public:
     int             Open(LPCSTR,int);
     int inline      Open(LPCSTR name){return Open(name,0);}
};

#endif // !defined(AFX_MYBMP_H__34151075_C57B_11D1_94F8_0000B431BBA1__INCLUDED
_)

// MyBmp.cpp: implementation of the MyBmp class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "MyBmp.h"

#ifdef _DEBUG
#undef THIS_FILE
```

```
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
//class PALETTE
//////////////////////////////////////////////////////////////////////

UINT PALETTE::SetPal(HDC hDC)
     {
          if(hPal)
               {
                    SelectPalette(hDC,hPal,0);
                    return RealizePalette(hDC);
               }
          else return FALSE;
     }

void PALETTE::ResetPal(HDC hDC)
     {
          if(hOldPal)
               {
                    SelectPalette(hDC,hOldPal,0);
            hOldPal=0;
               }
   }

HPALETTE PALETTE::CreatePal(LPLOGPALETTE Pal)
{
     if(hPal)DeleteObject(hPal);
     if(Pal_>palNumEntries<=256)hPal=CreatePalette(Pal);
     return hPal;
}

//////////////////////////////////////////////////////////////////////
//class DIBPALETTE
//////////////////////////////////////////////////////////////////////

HPALETTE PALETTE::CreatePal(BITMAPINFO* info)
{

     struct
          {
               WORD         palVersion;
               WORD         palNumEntries;
               PALETTEENTRY palPalEntry[256];
          } p;
```

```
     LPLOGPALETTE Pal=(LPLOGPALETTE)&p;

     Pal_->palVersion=0x300;
     Pal_->palNumEntries=
          /*min*/((WORD)IMAGECOLORS(info_->bmiHeader.biBitCount,1));//,info_->bm
iHeader.biClrUsed);

     for(int i=0;i<Pal_->palNumEntries;i++)
     {
          Pal_->palPalEntry[i].peRed=info_->bmiColors[i].rgbRed;
          Pal_->palPalEntry[i].peGreen=
                      info_->bmiColors[i].rgbGreen;
          Pal_->palPalEntry[i].peBlue=info_->bmiColors[i].rgbBlue;
          Pal_->palPalEntry[i].peFlags     =PC_NOCOLLAPSE;
     }

     return PALETTE::CreatePal(Pal);
}

////////////////////////////////////////////////////////////////////////
// MYDIB Class
////////////////////////////////////////////////////////////////////////

MYDIB::MYDIB()
{
     memset(this,0,sizeof(BMP));
     lpInfo=(LPBITMAPINFO)&p;
     lpInfoHead=(LPBITMAPINFOHEADER)&p;
}

MYDIB::~MYDIB()
{
     if(hData)GlobalFree(hData);
}

int MYDIB::Show(HDC hDC,int x1,int y1,int x2,int y2,int x3,
                           int y3,int x4,int y4,DWORD Rop)
     {
          if(x2<=0)x2=(int)lpInfoHead_->biWidth+x2;
          if(y2<=0)y2=(int)lpInfoHead_->biHeight+y2;
          if(x4<=0)x4=(int)lpInfoHead_->biWidth+x4;
          if(y4<=0)y4=(int)lpInfoHead_->biHeight+y4;
//        if(w_hp)SetPalette(hDC);
          BMPDATA* Data=(BMPDATA*)GlobalLock(hData);
//        int i=StretchDIBits(hDC,x1,y1,x2,y2,x3,y3,x4,y4,
                              Data,Info,DIB_RGB_COLORS,Rop);
          int i=StretchDIBits(hDC,x1,y1,x2,y2,x3,
       (int)lpInfoHead_->biHeight_y3_y4,x4,y4,Data,lpInfo,
                                        DIB_RGB_COLORS,Rop);
          GlobalUnlock(hData);
          return i;
```

```
    }

int MYDIB::Show(MYDIB* dib,int x1,int y1,int x2,int y2,int x3,int y3,
                    BYTE r1,BYTE g1,BYTE b1,BYTE r2,BYTE g2,BYTE b2)
{
    register DWORD c1=(((DWORD)r1)<<16)+(((DWORD)g1)<<8)+b1;
    register DWORD c2=(((DWORD)r2)<<16)+(((DWORD)g2)<<8)+b2;
    register DWORD c;
    register DWORD *rq=(DWORD*)p.r;
    if(!dib_>hData)
    {
        memcpy(dib,this,sizeof(MYDIB));
        dib_>lpInfo=(LPBITMAPINFO)&(dib_>p);
        dib_>lpInfoHead=(LPBITMAPINFOHEADER)&(dib_>p);
        dib_>lpInfoHead_>biWidth   =x2+x1;
        dib_>lpInfoHead_>biHeight  =y2+y1;
        DWORD Size=dib_>lpInfoHead_>biWidth
            *dib_>lpInfoHead_>biHeight
            *IMAGECOLORS(dib_>lpInfoHead_>biBitCount,1)/8;
        dib_>hData=GlobalAlloc(GMEM_FIXED,Size);
    }

    x2=min(x2,(int)dib_>lpInfoHead_>biWidth _x1);
    y2=min(y2,(int)dib_>lpInfoHead_>biHeight_y1);

    BMPDATA *Data1=(BMPDATA*)GlobalLock(hData);
    BMPDATA *Data2=(BMPDATA*)GlobalLock(dib_>hData);
    DWORD w1=DWORD_WBYTES(lpInfoHead_>biWidth
                    *lpInfoHead_>biBitCount);
    DWORD w2=DWORD_WBYTES(dib_>lpInfoHead_>biWidth
                    *dib_>lpInfoHead_>biBitCount);
    Data1+=(w1*(lpInfoHead_>biHeight_y3_y2)
                       +x3*lpInfoHead_>biBitCount/8);
    Data2+=(w2*(dib_>lpInfoHead_>biHeight_y1_y2)
                    +x1*dib_>lpInfoHead_>biBitCount/8);
    for(int j=0;j<y2;j++)
    {
        for(register int i=0;i<x2;i++)
        {
            c=*(rq+*(Data1+i));
            if(   (c<c1)||   (c>c2)||((WORD)c<(WORD)c1)
                  ||   ((WORD)c>(WORD)c2)
                  ||   ((BYTE)c<(BYTE)c1)
            ||((BYTE)c>(BYTE)c2)) *(Data2+i)=*(Data1+i);
        }
        Data1+=w1;
        Data2+=w2;
    }

    GlobalUnlock(hData);
    GlobalUnlock(dib_>hData);
```

```
    return TRUE;
}

int MYDIB::Show(MYDIB* dib,int x1,int y1,int x2,int y2,int x3,
                    int y3,register BYTE x,register BYTE y)
{
    register BMPDATA d;

    if(!dib_>hData)
    {
        memcpy(dib,this,sizeof(MYDIB));
        dib_>lpInfo=(LPBITMAPINFO)&(dib_>p);
        dib_>lpInfoHead=(LPBITMAPINFOHEADER)&(dib_>p);
        dib_>lpInfoHead_>biWidth   =x2+x1;
        dib_>lpInfoHead_>biHeight  =y2+y1;
        DWORD Size=(dib_>lpInfoHead_>biWidth+1)
              *dib_>lpInfoHead_>biHeight
              *dib_>lpInfoHead_>biBitCount/8;
            //IMAGECOLORS(dib_>lpInfoHead_>biBitCount,1)/8;
        dib_>hData=GlobalAlloc(GMEM_FIXED,Size);
    }

    x2=min(x2,(int)dib_>lpInfoHead_>biWidth _x1);
    y2=min(y2,(int)dib_>lpInfoHead_>biHeight_y1);
    BMPDATA *Data1=(BMPDATA*)GlobalLock(hData);
    BMPDATA *Data2=(BMPDATA*)GlobalLock(dib_>hData);
    DWORD w1=DWORD_WBYTES(lpInfoHead_>biWidth
                          *lpInfoHead_>biBitCount);
    DWORD w2=DWORD_WBYTES(dib_>lpInfoHead_>biWidth
                          *dib_>lpInfoHead_>biBitCount);
    Data1+=(w1*(lpInfoHead_>biHeight_y3_y2)
            +x3*lpInfoHead_>biBitCount/8);
    Data2+=(w2*(dib_>lpInfoHead_>biHeight_y1_y2)
            +x1*dib_>lpInfoHead_>biBitCount/8);

    for(int j=0;j<y2;j++)
    {
        for(register int i=0;i<x2;i++)
        {
            d=*(Data1+i);
            if((d<x)||(d>y))*(Data2+i)=d;
        }
        Data1+=w1;
        Data2+=w2;
    }

    GlobalUnlock(hData);
    GlobalUnlock(dib_>hData);
    return TRUE;
}
```

```
//////////////////////////////////////////////////////////////////////
//class bmp
//////////////////////////////////////////////////////////////////////

int BMP::Open(LPCSTR File,int sty)  //sty:   0:enable hpal  1:disnnable hpal
{
     BITMAPFILEHEADER     FileHead;
     int                        i;//,j,k;
     HFILE                      hFile;
     DWORD                      uBytes;
     DWORD                      Size;

     hFile=_lopen(File,OF_READ);
     if(hFile==HFILE_ERROR)return 0;

     //读取文件头
     i=_lread(hFile,&FileHead,sizeof(BITMAPFILEHEADER));
     if(i==HFILE_ERROR) FALSERETURN;//goto BMP_FALSE_END;
//   Type=FileHead.bfType;

     //读取信息头
     i=sizeof(BITMAPINFOHEADER)+sizeof(RGBQUAD)*256;
     _lread(hFile,lpInfoHead,i);

     //建立调色板
     if(!sty)CreatePal(lpInfo);

     uBytes=_llseek(hFile,0,2);
     if ((FileHead.bfSize)>uBytes) FALSERETURN;//goto BMP_FALSE_END;


     if(hData)GlobalFree(hData);
     Size=uBytes_FileHead.bfOffBits;
     hData=GlobalAlloc(GMEM_FIXED,Size);
     BMPDATA * Data=(BMPDATA*)GlobalLock(hData);

     //读取数据
     _llseek(hFile,FileHead.bfOffBits,0);

     for(;_lread(hFile,Data,RBLOCK)==RBLOCK;Data+=RBLOCK);
     GlobalUnlock(hData);
     _lclose(hFile);
     return TRUE;
}
```

在MyBmp.h文件中为了构造BMP框架，我们定义了三个类：class PALETTE、class DIBPALETTE 、class MYDIB 。其中PALETTE是基类。DIBPALETTE继承了类PALETTE，而类MYDIB又继承了类DIBPALETTE。

- class PALETTE：

在此类中，我们定义了两个成员变量hPal和hOldPal、七个成员函数。这七个成员函数功能如下：其中两个分别是构造函数和析构函数，两个函数CreatePal

(BITMAPINFO*)、CreatePal(LPLOGPALETTE)根据指定参数完成构造调色板的作用，一个函数SetPal(HDC)实现调色板，一个函数ResetPal(HDC)恢复调色板，一个函数GetPal()获得调色板。

- class MYDIB：

在此类中，我们定义了两个核心成员变量p和hData,其中p是一个自定义结构，它包含一个BMPINFO头信息和一个调色板，hData是指向内存中一块数据的句柄，另外两个参数lpInfo和lpInfoHead实际上是指向结构p的指针。四个成员函数Show()的内核是API函数SetDIBitsToDevice()。它们的功能是根据结构p和句柄hData把图像显示到屏幕上。

- class BMP：

在此类中，我们只定义了两个成员函数Open，它们的功能是打开一个BMP文件，并将文件内容填入其基类的参数中。前面我们提到过在多媒体编程中需要用公共调色板，但有时也需要用私有调色板，因此在Open函数中第二个参数指定了这个区别，若参数为0则构造自己的hPal，否则自己的hPal无效。

实例分析

在这个实例中，我们将调入一个BMP文件，并把它显示到屏幕上，程序过程如下：
建立一个对话框属性的应用程序，然后是加入光盘中提供的源文件“mybmp.h”和“mybmp.cpp”，在文件“showbmpdlg.h”头加上“#include "mybmp.h"”，在类“CShowbmpDlg”定义中加入成员变量“BMP bmp”，在类“CShowbmpDlg”的成员函数OnInitDialog()中用“bmp.Open("1.bmp")”读取文件“1.bmp”，在OnPaint()中加入以下调色板实现函数和显示函数

```
bmp.SetPal(dc.m_hDC)
bmp.Show(dc.m_hDC)
bmp.ResetPal(dc.m_hDC)
```

注意SetPal()和ResetPal()要配对使用，SetPal()用在Show()前，ResetPal()用在Show()后。编译并运行之。

4 图像合成

假如要实现一个动画例如一只老鼠从屏幕左边往右边跑过去，一般的书上是这么介绍的：首先做一个老鼠的画片，再画一张黑白老鼠掩模图片。首先用掩模图处理屏幕，再用掩模处理老鼠图片，最后把处理过的老鼠贴到屏幕上，前后要处理三个BitBlt函数。而且这样处理过程会使屏幕出现明显闪烁。要想制止闪烁还要先做一个兼容DC，先把屏幕图片拷贝至兼容DC，再在兼容DC上处理完老鼠后在再拷贝回屏幕。前后要用到五个BitBlt函数。图片比较小还好，若是图片很大，那速度简直就是“去年今日此电脑，人面老鼠相映红，人面不知何处去，老鼠还在慢慢爬”。是否有其他的解决方法呢？

实现透明位图的方式

1. 最愚蠢的办法：直接在屏幕上逐点用SetPixel函数画图。
2. 一般的方法：用BitBlt函数作至少三个运算。
3. 最快的方法：直接写屏。
4. 性价比最高的办法：直接写数据缓冲区。

四种方式的讨论：

1. 对于初搞图像处理的程序员来说，似乎逐点画过去的办法是第一选择，然而事实证明这是世界上速度最慢的方法，用它实现的最理想的动画效果是“去年一滴相思泪，今日方流到嘴边”。

2. BitBlt：前面已介绍过，此处不再介绍。
3. 直接写屏：Dos下这种方式用得比较多，在windows环境下编程，windows3.1环境下只能加挂WinG才能实现，在win95或NT下可用函数CreateDIBSection（）或是使用MicroSoftDirectX函数实现。这种方式我们将出专著介绍。此处不作讨论。
4. 写数据缓冲区：这个方法对环境要求较小，不需外挂软件，兼容于3.1和95、NT，速度也还可以，它的原理与直接写屏相似，而且可以方便地移植到直接写屏方式中去。我们将在此介绍此方法。

读写数据缓冲区：

大家可能还记得在前面介绍的class MYDIB，里面有两个参数，一个是bmp信息头，一个是bmp数据区，大家是否能想象得到假如修改了bmp数据区的数据，再显示图像会有什么结果？这块数据区，就是我们要使用的数据缓冲区。

透明位图

要想实现透明位图，首先要有两张图片，一张作为源位图，一张作为目的位图，程序员要把源位图贴到目的位图上，并且要指明什么颜色要屏蔽掉，为此，我们在class MYDIB上增加了一个函数Show(MYDIB* dib,int x1,int y1,int x2,int y2,int x3,int y3,BYTE r1, BYTE g1,BYTE b1,BYTE r2,BYTE g2,BYTE b2)，这个函数的用法有点类似于BitBlt函数，它的意思为：把己方缓冲区内的数据拷贝到类dib的缓冲区中去，其中从RGB(r1,g1,b1)至RGB(r2,g2,b2)的颜色为透明色，x1、y1、x2、y2、x3、y3为目标坐标、拷贝范围、源坐标，其意义与BitBlt相同。在Show函数的实现过程中，我们首先算出要改变的源数据、目标数据地址，然后分析要拷贝的数据颜色是否属于屏蔽色，假如是屏蔽色，则不拷贝数据，否则拷贝。

另外一种透明位图方式

透明色固然是一种比较简单的实现方式，但是有的时候也需要另外一种实现方式，这就是直接指定颜色索引方式，我们可以指定在调色板中某某号至某某号为透明色。因此，在class MYDIB中再增加一个函数Show(MYDIB* dib,int x1,int y1,int x2,int y2,int x3,int y3,register BYTE x,register BYTE y)，这个函数的原理与前一种方式差不多，只是比前一种方式少了四个参数，由以颜色指定透明色改成以颜色索引指定透明色。

透明位图的刷新速度

到底更改数据缓冲区方式的速度快，还是BitBlt速度快？要是BitBlt速度快的话，以前的一番心血岂非成了滚滚长江东逝水，为此我们要用实例分析一下，建立一个名为Tp的基于对话框的程序，加入源程序mybmp.cpp和mybmp.h，在tpdlg.h文件头中加入#include "mybmp.h"，在类CTPDlg中加入两个成员变量bmp1和bmp2。在窗口初始化时设置定时器，打开文件“1.bmp”、“2.bmp”，在定时器消息响应过程中完成拷贝和刷新过程，编译并运行程序。我们可以看到一个“AllTime”参数，它显示刷新256张位图需要大约20_21秒左右。现在注释掉定时器消息响应过程中的透底函数bmp1.Show((MYDIB*)&bmp2,0,0,640,480,0,0, 0,0,0,i,i,i)，再看刷新256张位图大约需要15_16秒，这是单纯使用函数StretchDIBits所需的时间。可见此处一个透明位图完成时间相当于一点四个BitBlt时间，比照BitBlt方式的三个BitBlt时间（差效果）、五个BitBlt时间（好效果）要好得多。当然，这与直接写屏比又差得多了。

现在再将透底函数换成bmp1.Show((MYDIB*)&bmp2,0,0,640,480,0,0, 0,i)，我们不由惊喜地看到现在刷新256张位图的时间为16_17秒，几乎可以认为，缓冲区读写时间已经

可以忽略不计。

图5.3

实例分析

在这个实例中，我们要实现一个动画，背景是一位绝代佳人，前面有一只狗牵着它的宠物跑来跑去。素材需要五张图片，其中背景一张，动画四张。我们分析一下它的实现方式：

在类CMovieDlg中，我们首先用语句BMP bmp[5]定义了五张图片，然后用语句MYDIB temp定义了一个临时图片。在对话框初始化过程函数中分别读入五张位图，设定定时器为一百毫秒，在定时器响应函数中操作过程如下：首先将背景写入临时图片，再将小狗透去白色写入临时图片，最后将临时图片写上屏幕。

图5.4

5 FLC 动画

FLC和FLI动画同属于AutoDesk公司的产品，它们采用帧与帧之间求差及单帧RLE编码的方式，其特点是易于解码和编码。当然，它们没有为音频提供便利限制了它的应用范围。在FLC和FLI之间，FLI由于大小仅局限于320x200，调色板仅64色，最多4000帧长度，使它已差不多退出了历史舞台。因此，我们在此不再讲述FLI，只介绍FLC动画原理及实现方式。

FLC文件的结构

FLC文件的结构图示如下：

FLC文件头数据结构如下：

偏移	大小	名称	说明
0	4	文件大小	整数，表示文件的字节长度
4	2	标识符	AF12H，FLI 的
6	2	帧新数目	文件中的帧数目，最大为 4000
8	2	宽度	屏幕宽度(像素)
10	2	高度	屏幕高度(像素)
12	2	位数/底色	8
14	2	标志了	表明文件已关好
16	4	速度	每帧延时的微秒数
20	2	未用	0
22	4	文件建立日期	采用 MS—DOS 的格式
26	4	文件建立者	用于创建动画的 Aimator Pro 程序序列号；
30	4	文件更新日期	文件是最近更新日期，为 MS—DOS 格式
34	4	文件更新者	形式与文件建立者相似只是改为最近更新者
38	2	AspectX	纵横尺寸比 X，Y 和 X 部分
40	2	AspectY	纵横尺寸比 X，Y 和 Y 部分
42	38	保留位	全为 0
80	4	偏移 1	从文件起始处到第一个帧块的偏移
84	4	偏移 2	从文件起始处到第两个帧块的偏移，留作循环用
88	40	保留位	全为 0

帧头数据结构：

偏移	大小	名称	说明
0	4	帧大小	包括帧头在内的帧大小(对 FLI，小于 64K 字节)
4	2	ID	帧的标识符，F1FAH
6	2	块数目	帧中的块数目，0 表示相同的帧
8	8	保留住	全为 0

块头数据结构：

偏移	大小	名称	说明
0	4	块大小	包含块头在内的块大小(字数)
4	2	类型代码	块的类型

块的类型及解释：

代码	名称	代码所指的块类型
4	FLI_COLOR_256	8 位/底色的调色板的数据
7	FLI_WORLD_LC	2 字节的 RLE 压缩像素数据
11	FLI_COLOR	6 位/底色的调色板的数据
12	FLI_LC	行压缩(FLC 中通常不用)
13	FIL_BLACK	将整个屏幕的色彩置为色彩 0(只在第一帧中出现)
15	FLI_BRUN	字节方向的运行长度压缩—只在第一帧中有
16	FLI_COPY	未压缩数据(64K 字节)
18	FLI_PREVIEW	预显图像数据

FLC动画播放源程序简介

我们分析一下FLC动画源程序：打开文件flcw.h，可以看见文件中定义了FLC文件头、帧头、块头及块类型的宏。另外我们可以发现类FLCW是从类MYDIB中派生来的，这是因为不管FLC动画本身如何复杂，只要是往屏幕上画图，最后都是要在内存中转换成位图形式。FLC动画实现过程中，在Open阶段就生成了一个大小与FLC动画相同的内存位图，以后每次读取下一帧，就把数据写入内存位图，再将内存位图贴到屏幕上。

```
// FLCW.h: interface for the FLCW class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FLCW_H__2A3B58A3_C964_11D1_94F8_0000B431BBA1__INCLUDED_)
#define AFX_FLCW_H__2A3B58A3_C964_11D1_94F8_0000B431BBA1__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifdef     __cplusplus
extern "C" {
#endif

/*#ifndef       GlobalPtrHandle
#define     GlobalPtrHandle(lp)         \
                ((HGLOBAL)LOWORD(GlobalHandle(FP_SEG(lp))))
#endif

#define    IsOverSegAlians(lp,off)    \
              (((DWORD)FP_OFF(lp)+(WORD)off)>0XFFFFL?1:0)

#include "mybmp.h"
```

```
//#define PALVERSION      0x300
#define DIB_STORAGEWIDTH(x) ((x + 3) & ~3)
#define WIDTHBYTES(i) ((i+31)/32*4)
#define MAXPALETTE    256     /* max. # supported palette entries */

#pragma pack(2)


     /* Flic Header */
typedef struct{
     long       size; /* Size of flic including this header. */
     WORD       type; /* Either FLI_TYPE or FLC_TYPE below. */
     WORD       frames;    /* Number of frames in flic. */
     WORD       width;     /* Flic width in pixels. */
     WORD       height;    /* Flic height in pixels. */
     WORD       depth;     /* Bits per pixel.  (Always 8 now.) */
     WORD       flags;     /* FLI_FINISHED | FLI_LOOPED ideally. */
     long  speed;     /* Delay between frames. */
     short reserved1; /* Set to zero. */
     DWORD created;   /* Date of flic creation. (FLC only.) */
     DWORD creator;   /* Serial # of flic creator.(FLC only.)*/
     DWORD updated;   /* Date of flic update. (FLC only.) */
     DWORD updater;   /* Serial # of flic updater.(FLC only.)*/
     WORD       aspect_dx; /* Width of square rectangle. (FLC only.) */
     WORD       aspect_dy; /* Height of square rectangle. (FLC only.) */
     char       reserved2[38];/* Set to zero. */
     long       oframe1;   /* Offset to frame 1. (FLC only.) */
     long       oframe2;   /* Offset to frame 2. (FLC only.) */
     char       reserved3[40];/* Set to zero. */
}    FLICHEAD;

typedef FLICHEAD  *   LPFLICHEAD ;
     /* Values for FlicHead.type */
#define FLI_TYPE 0xAF11u   /* 320x200 .FLI type ID */
#define FLC_TYPE 0xAF12u   /* Variable rez .FLC type ID */
     /* Values for FlicHead.flags */
#define FLI_FINISHED 0x0001
#define FLI_LOOPED     0x0002

     /* Optional Prefix Header */
typedef struct{
     long  size;/* Size of prefix including header. */
     WORD       type;      /* Always PREFIX_TYPE. */
     short      chunks;    /* Number of subchunks in prefix. */
     char  reserved[8];/* Always 0. */
}    PrefixHead;

     /* Value for PrefixHead.type */
#define PREFIX_TYPE  0xF100u
```

```
     /* Frame Header */
typedef struct{
     long size;      /* Size of frame including header. */
     WORD type;      /* Always FRAME_TYPE */
     short chunks;   /* Number of chunks in frame. */
     char reserved[8];/* Always 0. */
}    FRAMEHEAD;

typedef FRAMEHEAD  *  LPFRAMEHEAD ;

     /* Value for FrameHead.type */
#define FRAME_TYPE 0xF1FAu

     /* Chunk Header */
typedef struct{
     LONG size;      /* Size of chunk including header. */
     WORD type;      /* Value from ChunkTypes below. */
}    CHUNKHEAD;

typedef CHUNKHEAD  *  LPCHUNKHEAD ;
#pragma pack(4)

enum ChunkTypes
     {
     COLOR_256 = 4,  /* 256 level color pallette info. (FLC only.) */
     DELTA_FLC = 7,  /* Word_oriented delta compression. (FLC only.) */
     COLOR_64 = 11,  /* 64 level color pallette info. */
     DELTA_FLI = 12, /* Byte_oriented delta compression. */
     BLACK = 13,     /* whole frame is color 0 */
     BYTE_RUN = 15,  /* Byte run_length compression. */
     LITERAL = 16,   /* Uncompressed pixels. */
     PSTAMP = 18,    /* "Postage stamp" chunk. (FLC only.) */
     };

typedef struct {
     BYTE  r ;
     BYTE  g ;
     BYTE  b ;
}    COLOR ;
typedef COLOR  *LPCOLOR ;

typedef struct {
     WORD  x ;
     WORD  y ;
     WORD  width ;
     WORD  height ;
}    WRYRECT ;


//define load method
```

```
#define FLC_NONE        0
#define    FLC_LOADINMEM   1
//#define FLC_SKIPFIRST   2
#define FLC_AUTO     4

#define    FLC_LOOPS  0

//   define     error message
#define FLC_ERR_OPEN  _1
#define    FLC_ERR_HEADER  _2
#define    FLC_ERR_FRAME   _3
#define    FLC_ERR_CHUNK   _4
#define    FLC_ERR_COLOR   _5

#define    FLC_SUCCESS          0

//define   flc play status
#define    FLC_STOPPED          0
#define    FLC_PLAYING          1
#define    FLC_PAUSED      2
#define    FLC_NOTREADY    3



typedef union tagWPT{
     short * w ;
     BYTE * ub ;
     char *     b ;
}    WPT ;

//if fps=NULL,this function is GetFlcSpeed
// return is frame per second nulli is error;



#ifdef __cplusplus
}
#endif

#define    SUCCESSFUL           0
#define    FLCERR_READHEAD      _1
#define    FLCERR_NOTFLC        _2
#define    FLCERR_NOMEMORY      _3
#define    FLCERR_READFILE      _4
#define    FLCERR_READFRAME_5
#define    FLCERR_READCHUNK_6
#define    SKIPFRAME            _7


class FLCW : public MYDIB
{
```

```
private:
     PALETTE* pal;   //0:havn't palette  !0:have palette
     WORD  wm_Notify ;
     WORD width ;
     WORD height ;
     long  size ;
     WORD frames ;
     DWORD speed;
     WORD loops ;
     WORD curr_frame ;
     WORD curr_loop ;
     short flags ;
     long  oframe2 ;
     WORD StartFrame;
     DWORD dwNextTime;
     HFILE hFile ;
     HWND  hPlayWnd ;
     short fStatus ;
     WORD  xOrg ;
     WORD  yOrg ;
     WRYRECT    rcChange ;

     void DecodeColor(LPSTR Body) ;
     void DecodeBRun(LPSTR Body) ;
     void DecodeCopy(LPSTR Body) ;
     void DecodeBlack() ;
     void DecodeDeltaFli(LPSTR Body) ;
     void DecodeDeltaFlc(LPSTR Body) ;
public:
//   FLCW(){memset(this,0,sizeof(FLCW));}
     FLCW()
     {
          lpInfoHead_>biSize = sizeof(BITMAPINFOHEADER) ;
          lpInfoHead_>biPlanes=1 ;
          lpInfoHead_>biBitCount=8 ;
          lpInfoHead_>biCompression=BI_RGB;
     }

     ~FLCW(){CloseFlc();}

//   BOOL SDIFlcMessageLoop(HWND hMainWnd,HACCEL hAccel,
                                             MSG* msg);
//   BOOL MDIFlcMessageLoop(HWND hFrameWnd,HWND hClientWnd,
                                       HACCEL hAccel,MSG* msg);

     int Open(LPCSTR ,PALETTE* );
     BOOL CloseFlc() ;

     void PlayFlc(HWND hwnd,WORD xOrg,WORD yOrg,
                                          WORD wm_notify) ;
     void StopFlc() ;
```

```
    void PauseFlc() ;
    short QueryFlc() ;
    int Loop(WORD wLoops);
    float Speed(float fps);

    WORD  GetTotalFrame();
    WORD  GetCurFrame();

    BOOL ShowFlc(HDC hdc) ;

    BOOL FlcIdle() ;

    short ReadFrame() ;

    void PlayFlcAFrame();

};

#endif // !defined(AFX_FLCW_H__2A3B58A3_C964_11D1_94F8_0000B431BBA1__INCLUDED_)

// FLCW.cpp: implementation of the FLCW class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

#include "flcw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//#include "test.cpp"
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

//modefy in 1998.4.4.10:50  TJ

//////////////////////////////////////////////////////////////////////
// Class FLCW
//////////////////////////////////////////////////////////////////////
/*
void FLCW::DecodeColor(LPSTR Body)
{
    short     start=0 ;
    BYTE *    cbuf=(BYTE *)(Body+2) ;
    short     ops =*(short *)Body ;
    WORD      count ;
```

```
	short	number=0 ;
	short	i ;

	while(__ops >= 0)
	{
		start += *cbuf++ ;
		if((count = *cbuf++)==0)count = 256 ;
		number += count ;
		for(i = start;i < (start+count);i ++)
		{
			lpInfo_>bmiColors[i].rgbRed = *cbuf++ ;
			lpInfo_>bmiColors[i].rgbGreen = *cbuf++ ;
			lpInfo_>bmiColors[i].rgbBlue = *cbuf++ ;
			lpInfo_>bmiColors[i].rgbReserved = 0;
		}
		start+=count ;
	}
	lpInfo_>bmiHeader.biBitCount=8;
	lpInfoHead_>biClrUsed = number ;
	if(pal)CreatePal(lpInfo);

}
*/
void FLCW::DecodeColor(LPSTR Body)
{
	int		point=0 ;
	BYTE *		cbuf=(BYTE *)(Body+2) ;
	short	ops =*(short *)Body ;
	WORD	count ;
	int		i;

	while(__ops >= 0)
	{
		point += *cbuf++ ;
		if((count = *cbuf++)==0)count = 256 ;
		i=point+count;
		for(;point < i;point ++)
		{
			lpInfo_>bmiColors[point].rgbRed = *cbuf++ ;
			lpInfo_>bmiColors[point].rgbGreen = *cbuf++ ;
			lpInfo_>bmiColors[point].rgbBlue = *cbuf++ ;
			lpInfo_>bmiColors[point].rgbReserved = 0/*PC_NOCOLLAPSE*/ ;
		}
	}
	lpInfo_>bmiHeader.biBitCount=8;
	lpInfoHead_>biClrUsed = point ;
	if(!pal)CreatePal(lpInfo);
}

void FLCW::DecodeBRun(LPSTR Body)
{
```

```
     short x,y=0 ;
     WORD  bytewidth ;
     char  psize ;
     LPSTR lpDib ;
     LPSTR lpTemp ;
    LPSTR  t ;
     LPSTR t1 ;

     bytewidth = DIB_STORAGEWIDTH(width) ;

     lpDib = (LPSTR)GlobalLock(hData) ;
     if(!lpDib) return  ;

     lpTemp= lpDib;//(LPSTR)MoveToDibData(lpDib);
     t1=Body ;
//   t1++ ;
     t=lpTemp+(unsigned long)(height_1)*(unsigned long)bytewidth ;
     while(y<height){
          x=0 ;
          t1++ ;
          psize = 0 ;
          while(x<width){
               psize = *t1++ ;
               if(psize>=0){
//             short j ;
                    memset(t,*t1++,psize) ;
                    t += psize ;

               }
               else{
                    psize = _psize ;
                    memcpy(t,t1,psize);
                    t+=psize ;
                    t1+=psize ;
               }
               x+=psize ;
          }
          y++ ;
          t=lpTemp+(unsigned long)(height_y_1)*
                         (unsigned long)bytewidth ;
     }
     GlobalUnlock(hData) ;
     rcChange.x = 0 ;
     rcChange.y = 0 ;
     rcChange.width = width ;
     rcChange.height=height ;

}

void FLCW::DecodeDeltaFlc(LPSTR Body)
{
```

```
    short lp_count,i ;
    short opcount ;
    short min_x,min_y,max_x,max_y ;
    WORD  bytewidth ;
    LPSTR temp ;
    LPSTR t,t1 ;
    LPSTR lpDib ;
    WPT       wpt ;
    short     j ;

    short     psize ;
    signed    curr_x,curr_y ;

    min_x = 900 ;
    min_y = 900;
    max_x = 0;
    max_y = 0 ;

    curr_x=0 ;
    curr_y=height_1 ;

    bytewidth = WIDTHBYTES(width*8) ;
    lpDib = (LPSTR)GlobalLock(hData) ;
    if(!lpDib) return ;
    temp=lpDib;
    temp = temp+(DWORD)(height_1)*bytewidth ;
    t = temp ;
    i = 0 ;

    wpt.ub =(BYTE *)Body ;
    lp_count = *wpt.w++ ;
    while(i<=lp_count){
        if((opcount=*wpt.w++)>=0){
            i++ ;
            for(j = 0; j<opcount;j++){
                curr_x += *wpt.ub++ ;
                psize = *wpt.b ;
                t = temp+curr_x ;
                if(curr_x!=*(wpt.ub_1)||*wpt.ub!=1
                    ||  *t!=*(wpt.ub+1)
                    ||*(t+1)!=*(wpt.ub+2)){
            if(min_x>curr_x)min_x = curr_x ;
                }
            wpt.b++ ;
                if((psize+=psize)>=0){
                    memcpy(t,wpt.ub,psize);
                    curr_x+=psize ;
                    t += psize ;
                    wpt.ub+=psize ;
                    if (max_x<curr_x)    max_x=curr_x ;
                }
```

```
                    else{
                    BYTE  j ;
                         psize = _psize ;
                      curr_x+=psize ;
                         psize=psize>>1 ;
                         for(j=0;j<psize ;j++)
                         {
                              *(short *)t=*wpt.w ;
                              t++;
                       t++ ;
                    }
                         wpt.w++ ;
                         if (max_x<curr_x)     max_x=curr_x ;
                     }
                }
                if(i==lp_count) break ;
                curr_x=0 ;
                curr_y__ ;
                if((WORD)min_y>(height_curr_y_1))
                     min_y = height_curr_y_2 ;
                temp = temp_(DWORD)bytewidth ;
                continue ;
           }
           if(((WORD)opcount)&0x4000){//skip lines
                opcount = _opcount ;
                curr_y _= opcount ;
                temp = temp_(DWORD)opcount*(DWORD)bytewidth ;
                if((WORD)min_y>(height_curr_y_1))
                     min_y = height_curr_y_2 ;
                curr_x = 0 ;
            continue ;
           }
           t = temp+(DWORD)bytewidth_1L ;
           t1 = (LPSTR)(wpt.w_1) ;
           *t=*t1 ;
           max_x=width ;
        continue ;
      }
      GlobalUnlock(hData);
      max_y = height_curr_y_1 ;
      if(min_y==900)  min_y = 0 ;
      rcChange.x = min_x ;
      rcChange.y=min_y ;
      rcChange.width = max_x_min_x ;
      rcChange.height=max_y_min_y+1 ;
      return ;
}

void FLCW::DecodeDeltaFli(LPSTR Body)
{
      short           *w ;
```

```
    BYTE *          cpt ;
    short      lines ;
    BYTE       opcount ;
    char       psize ;
    LPSTR      lpDib ;
    LPSTR      t,temp ;
    WORD       min_x,min_y,max_x,max_y ;
    WORD       curr_x,curr_y ;
    short      i ;
    short      bytewidth ;

    bytewidth=WIDTHBYTES(width*8) ;

    min_x = 900 ;
    min_y = 900 ;
    max_x = 0;
    max_y = 0 ;

    w = (short *)Body ;
    cpt = (BYTE *)(w+2);
    lpDib=(LPSTR)GlobalLock(hData) ;
    if(!lpDib) return ;

    curr_y = *w++ ;
    lines = *w ;
    temp=lpDib+(DWORD)(height_1_curr_y)*bytewidth ;

    curr_y = height_curr_y_1 ;
    if((unsigned)min_y>(height_curr_y_1))
         min_y=height_curr_y_2  ;
    if((unsigned)max_y<(height_curr_y_1))
         max_y=height_curr_y_1 ;
    for(i=0;i<lines;i++){
         curr_x = 0 ;
         opcount = *cpt++ ;
         while(opcount>0){
              curr_x+=*cpt++ ;
              if((WORD)min_x>curr_x)     min_x = curr_x ;
              if((WORD)max_x<curr_x)     max_x = curr_x ;
//            t = lpDib+(Ulong)curr_y*(Ulong)bytewidth+curr_x ;
              t=temp+curr_x ;
              psize = *cpt++ ;
              if(psize<0){
                   psize = _psize ;
                   memset(t,*cpt++,psize);
                   curr_x += psize ;
                   t+=psize ;
              }
              else{
                   memcpy(t,cpt,psize) ;
                   t+=psize ;
```

```
                    cpt+=psize ;
                    curr_x+=psize ;
             }
                if(min_x>curr_x)min_x=curr_x ;
                if(max_x<curr_x)max_x = curr_x ;
             opcount__ ;
            }
           curr_y__ ;
         temp_=bytewidth ;
           if(min_y>(height_curr_y_1))min_y = height_curr_y_2;
           if(max_y<(height_curr_y_1)) max_y=height_curr_y_2 ;
    }
     GlobalUnlock(hData) ;
     rcChange.x = min_x ;
     rcChange.y=min_y ;
     rcChange.width=max_x_min_x ;
     rcChange.height=max_y_min_y+1 ;
    return ;
}

void FLCW::DecodeBlack()
{
     DWORD s=GlobalSize(hData);
     LPSTR p=(LPSTR)GlobalLock(hData) ;
     memset(p,0,s);
     GlobalUnlock(hData);

     rcChange.x = 0 ;
     rcChange.y = 0 ;
     rcChange.width = width ;
    rcChange.height = height ;
}

void FLCW::DecodeCopy( LPSTR Body)
{
     short bytewidth ;
     LPSTR lpDib ;
     LPSTR t ;
     WORD       i ;

     lpDib = (LPSTR)GlobalLock(hData) ;
     if(!lpDib)return ;
     bytewidth = WIDTHBYTES(width*8) ;

     t=lpDib;
     t=t+(DWORD)(height_1)*bytewidth ;
     for(i=0;i<height;i++){
          memcpy(t,Body,bytewidth) ;
          t_=bytewidth ;
          Body+=bytewidth ;
     }
```

```
    GlobalUnlock(hData) ;
    rcChange.x = 0 ;
    rcChange.y = 0 ;
    rcChange.width = width ;
    rcChange.height = height ;
    return;
}

int FLCW::Open(LPCSTR lpszFile,PALETTE* p)
{
    ////////////////////////////
    //func:
    //1:read FlicHead;
    ////////////////////////////

    FLICHEAD   flicHead ;
    long       oframe1;
    long        dibSize;

    if(hFile)_lclose(hFile);
    hFile = _lopen(lpszFile,OF_READ) ;
    if(hFile==HFILE_ERROR)     return 0 ;

    if(_hread(hFile,(LPSTR)&flicHead,sizeof(FLICHEAD))!=sizeof(FLICHEAD)){
        _lclose(hFile);
        return     FALSE;//FLCERR_READHEAD ;
    }
    if(flicHead.type!=FLC_TYPE){
        _lclose(hFile);
        return FALSE;//FLCERR_NOTFLC ;
    }

    width=flicHead.width ;
    height=flicHead.height;
    size = flicHead.size_flicHead.oframe2 ;
    frames=flicHead.frames ;
    speed=flicHead.speed ;

    loops = 1 ;
    curr_frame = 0 ;
    curr_loop = 0 ;
    pal=p;

//  flags = nFlag ;
    oframe1 = flicHead.oframe1 ;
    oframe2=flicHead.oframe2;
    fStatus = FLC_NOTREADY ;

    dibSize=(LONG)(WIDTHBYTES(width*8)*(LONG)height);
    if(hData)GlobalFree(hData);
    hData=GlobalAlloc(GHND,dibSize) ;
```

```
    if(!hData)return FLCERR_NOMEMORY ;
    lpInfoHead_>biWidth=width ;
    lpInfoHead_>biHeight=height;
    lpInfoHead_>biSizeImage=dibSize ;

    _llseek(hFile,oframe1,0);
    return TRUE ;
}

BOOL FLCW::CloseFlc()
{
    _lclose(hFile) ;

    return    TRUE ;
}

void FLCW::PlayFlcAFrame()
{
    HDC       hdc ;

    curr_frame++;
    dwNextTime=speed+GetTickCount();

    if(ReadFrame()==SUCCESSFUL)
    {
        hdc = GetDC(hPlayWnd) ;
        if(pal)pal_>SetPal(hdc);
        else SetPal(hdc);
        Show(hdc,xOrg+rcChange.x,yOrg+rcChange.y,
                rcChange.width,rcChange.height,
                    rcChange.x,rcChange.y,rcChange.width,
                                    rcChange.height);
        if(pal)pal_>ResetPal(hdc);
        else  ResetPal(hdc);
        ReleaseDC(hPlayWnd,hdc) ;
    }

    if(curr_frame>frames)
    {
        curr_frame=1;
        _llseek(hFile,oframe2,0);
        curr_loop++;
    }

    return ;
}

short FLCW::ReadFrame()
{
    HGLOBAL    hBody ;
    LPSTR lpBody ;
```

```
//    LPSTR lpBuf ;
      WORD  nChunkNumber ;
      WORD  i ;
      CHUNKHEAD  ChunkHead ;
      WORD       type ;
//    long  li ;
      long  size ;

      FRAMEHEAD  FrameHead ;
      if(_hread(hFile,&FrameHead,sizeof(FRAMEHEAD))
              !=sizeof(FRAMEHEAD))return FLCERR_READFRAME ;
      if(FrameHead.type!=FRAME_TYPE)
           return FLCERR_READFRAME ;

      nChunkNumber=FrameHead.chunks ;

      for(i=0;i<nChunkNumber;i++){
           {
                if(_hread(hFile,&ChunkHead,sizeof(CHUNKHEAD))
                                        !=sizeof(CHUNKHEAD)){
                     return FLCERR_READFRAME ;
                }
                type=ChunkHead.type ;
                size = ChunkHead.size_sizeof(CHUNKHEAD) ;
                #ifdef _MN_CDROM
                dwReadSize+=size;
                #endif
           }
           if(type==LITERAL)
                size+=2 ;
           if(size){
                LPSTR lpTemp ;
                hBody=GlobalAlloc(GMEM_MOVEABLE,size);
                lpBody=(LPSTR)GlobalLock(hBody) ;
                lpTemp = lpBody ;
{
                     if(_hread(hFile,(LPSTR)lpBody,size)!=size){
                          GlobalUnlock(hBody) ;
                          GlobalFree(hBody) ;
                          return FLCERR_READCHUNK ;
                     }
                }
           }
           switch(type){
                case COLOR_256 ://4,FLI_COLOR_256
                     DecodeColor(lpBody) ;
                     if(nChunkNumber==1)
                    memset(&rcChange,0,sizeof(RECT));
                     break ;
                case DELTA_FLC ://7,FLI_WORLD_LC
                     DecodeDeltaFlc(lpBody) ;
```

```
                    break ;
               case DELTA_FLI :  //12,FLI_LC
                    DecodeDeltaFli(lpBody) ;
                    break ;
               case BLACK :           //13,FLI_BALCK
                    DecodeBlack() ;
                    break ;
               case BYTE_RUN :        //15,FLI_BRUN
                    DecodeBRun(lpBody) ;
                    break ;
               case LITERAL :         //16,FLI_COPY
                    DecodeCopy(lpBody) ;
                    break;
               case PSTAMP :          //18,FLI_PREVIEW
               break ;
          }
          GlobalUnlock(hBody) ;
          GlobalFree(hBody) ;
     }
     if(nChunkNumber==0)
     {
          rcChange.x=0;
          rcChange.y=0;
          rcChange.width=0;
          rcChange.height=0 ;
     }

     return SUCCESSFUL ;
}


void FLCW::PlayFlc(HWND hwnd,WORD xorg,WORD yorg,WORD wm_notify)
{
     wm_Notify=wm_notify ;
     xOrg=xorg ;
     yOrg=yorg ;
     hPlayWnd = hwnd ;

     if(fStatus!=FLC_PLAYING)
     {
          fStatus=FLC_PLAYING;
          StartFrame=curr_frame;
          dwNextTime=GetTickCount();
     }
     return ;
}

void FLCW::StopFlc()
{
     curr_frame=1;
     _llseek(hFile,oframe2,0);
```

```
    fStatus=FLC_STOPPED ;
    PostMessage(hPlayWnd,wm_Notify,
              FLC_STOPPED,MAKELONG(curr_frame,0));
}

void FLCW::PauseFlc()
{

    if(fStatus!=FLC_PAUSED&&fStatus!=FLC_NOTREADY){
         fStatus=FLC_PAUSED ;
         PostMessage(hPlayWnd,wm_Notify,
              FLC_PAUSED,MAKELONG(curr_frame,0));
    }
    return ;
}

BOOL FLCW::ShowFlc(HDC hdc)
{

    if(hPal==0)     return FALSE ;
    if(hData==0)    return FALSE ;
    SetPal(hdc);
    Show(hdc,xOrg,yOrg);
    ResetPal(hdc);
    return TRUE ;
}


short FLCW::QueryFlc()
{
    return fStatus ;
}


WORD FLCW::GetTotalFrame()
{
    return frames;
}

int  FLCW::Loop(WORD wLoops)
{
    int Lastloop=loops;
    if(wLoops>=0)loops=wLoops;
    return    Lastloop;
}

WORD FLCW::GetCurFrame()
{
    return curr_frame;
}
```

```
float FLCW::Speed(float fps)
{
     float oldfps=  (float)(1000000.0/speed);
     if(fps>0)speed=(DWORD)(1000000.0/fps);
     return oldfps;
}

BOOL FLCW::FlcIdle()
{
//    if(GetCapture())return TRUE;

     if(fStatus==FLC_PLAYING)
     {
          if(GetTickCount()>=dwNextTime)PlayFlcAFrame();
          if(curr_loop>=loops)
          {
               curr_loop=0;
               fStatus=FLC_STOPPED;
               PostMessage(hPlayWnd,wm_Notify,fStatus,
                                        curr_frame);
          }
     }
     return FALSE;
}
```

程序员调用FLC流程

1. 用Open(LPCSTR lpszFile,int id)函数打开FLC文件，其参数lpszFile指要打开的文件名，参数id意义与类BMP中Open函数相同，值为零表示使用FLC文件中内建的调色板，否则忽略这个调色板，当在屏幕上播放多个动画或既有动画又有图片时要用到这个参数。程序首先打开指定文件，读入文件头，根据文件头信息申请内存建立内存位图，初始化部分参数，并将文件指针指向文件起始数据区。
2. PlayFlc(HWND hwnd,WORD xorg,WORD yorg,WORD wm_notify)开始播放动画。在类FLCW内部，用参数fStatus决定动画播放与否，只要这个参数值为FLC_PLAYING，就播放动画，我们还可以用这个参数检验当前播放状态，函数QueryFlc()就是返回fStatus值。参数xorg和yorg指定播放位置，假如在播放过程中想移动位置，可以用这个函数重新指定。参数wm_notify指定当动画播放完成后将发送什么消息给父窗口。
3. FlcIdle()函数是使用整个FLC类的关键。由于FLC播放是一个过程，不是一转眼就能完成的事，我们当然不能把整个程序的控制权交给它，让它播放完后再交回来，只好每隔一段时间关照它一下，让它检查一下现在的状态，假如时间到了，可以播放下一帧了，就放下一帧后返回，否则直接返回。因此在程序中需要有一个定时器，不停地调用FlcIdle。在FlcIdle函数中，核心又是if(GetTickCount()>=dwNextTime)PlayFlcAFrame()函数，这个函数的作用是检测是否到了播放下一帧的时间，然后调用PlayFlcAFrame函数播放下一帧。

FLC播放单帧图片机理

1. PlayFlcAFrame()，PlayFlcAFrame()的作用是逐帧播放FLC，它首先调用ReadFrame()函数，如读取成功则使用MYDIB类函数Show将内存位图拷贝至屏幕。
2. ReadFrame()，ReadFrame()函数是整个FLC类的核心所在，它的作用是读取文件数据，再将数据解码至内存中。其操作过程如下：读取帧头，获取当前帧内块数，循环读取

块头、块数据，根据块类型解码数据至内存位图中。块类型FLI_COLOR_256、FLI_WORLD_LC、FLI_LC、FLI_BALCK、FLI_BRUN、FLI_COPY分别对应于函数DecodeColor()、DecodeDeltaFlc()、DecodeDeltaFli()、DecodeBlack()、DecodeBRun()、DecodeCopy()。

实例分析

在这个实例中，我们要在一个窗口中同时播放三个FLC动画，注意这三个动画的调色板是不同的，为了协调这三个动画的颜色，干脆这三个调色板一个都不要，使用一个photoshop内置公用调色板，这个调色板在二百五十六种颜色范围内，采用每种颜色兼顾一点的办法，在实际应用中，要使用抖动算法才能模拟出与原来颜色乱真的效果，在此自己算太累，系统又不提供算法，况且这个算法与我们的主题关系不大，因此你看到的颜色会有些失真，除非系统设置大于二百五十六色。

建立一个基于对话框的应用程序playflc，在类CPlayflcDlg内定义三个FLCW类flc，外加一个BMP类bmp，bmp类的作用是取得一个调色板，我们已事先用photoshop作了一个长宽分别为1比1的图片，当然它包含了公用调色板的信息。在对话框初始化过程函数OnInitDialog()中，我们设定一个定时器，并分别读入文件1.flc、2.flc、3.flc、1.bmp，注意读FLC时第二个参数为(PALETTE*)&bmp，这就是告诉程序使用bmp的调色板。我们打算当用户按下ok键时开始播放动画，因此增加一个按键响应函数OnOK() ，添加三个PlayFlc函数，在定时器响应过程函数中填入三个FlcIdle函数。运行程序。

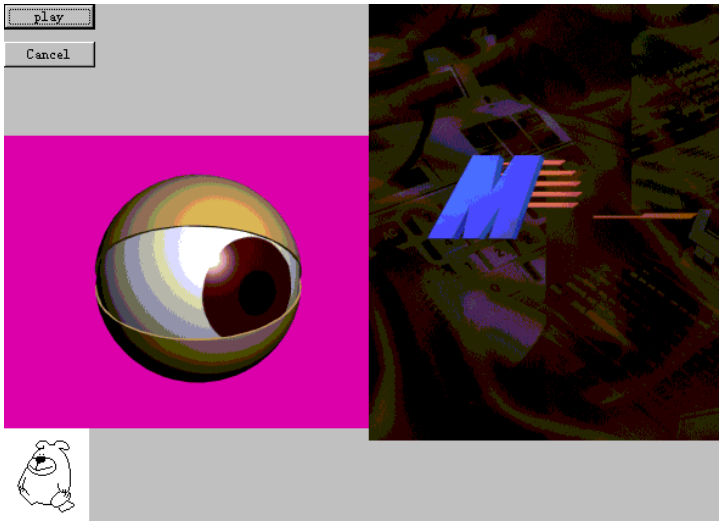


图5.5

热点

所谓热点，是指在指定窗口上设定一些区域，当用户在此区域上有所动作时，会发生一些事情，这些区域就是热点，热点的作用类似于在窗口上开了若干按钮窗口，当然它的功能要比button强大得多，最重要的是热点可以是不规则的，而button必须是矩形。

热点的原理

热点涉及到极复杂的坐标系统，比如一个四边形，需要四个坐标确定它的形状，六边形就需要六个坐标确定形状，若是再复杂一些，比如需在世界地图上区分中国和其它国家，那就不是几个几十个坐标就能解决问题了。假如一个中大型的多媒体程序，涉及到比较多的热点，就算一个热点由四个坐标设定，二十个热点就需要80个坐标。若是这些热点都由手工设定，那么以后的麻烦事就层出不穷。因为二个图像吻合得不好，就算有一个点的误差，人的肉眼也是看得出来的。有没有比较简单的办法可以减少程序员处理坐标的工作量，甚至可以不理会热点坐标？还好Windows提供了处理区域的函数，这就是RGN。

RGN的处理过程如下：程序员向系统提供一串坐标值，系统返回一个HRGN句柄；以后程

序员就可以使用这个句柄管理应用程序。比如说，程序员可以在指定区域范围内画图，而不影响到区域外面的部位。

热点制作工具

我们首先介绍一下热点制作工具，Hospot4.exe这个程序的作用是由用户自定义热点，再将热点写入指定文件。

打开Hospot4.exe文件，读入文件“1.bmp”，然后在菜单上选择“定义”就可以开始定义热点了。读入的bmp文件一般是程序中要用到的背景文件，在上面按键应该已经画好，所以一般只要照着画好的按键轮廓描，当描完后点取左键鼠标，在文件菜单上选择存盘，输入定义的热点名既可。然后又可以定义下一个热点。照此顺序做完所有热点退出，查看上多了一个文件*.dat，这个dat文件就是热点配置文件。

热点类Class Hot

```
#ifndef HOT_H
#define    HOT_H

#include <windows.h>

#define MAX_POINTS    50
#define MAX_HOT       20

typedef struct {
     HRGN       hRGN;                  //handle of hot region
     char  image[14];             //image name
     char  target[32];            //hotspot name
     BOOL       active;        //tag the hotspot enable or disable
     } POLYHOT;

class HOT
{
private:
     HRGN       hOldRgn;

public:
     int              HotNumber;
     POLYHOT    PHot[MAX_HOT];

public:
     HOT(){ memset(this,0,sizeof(HOT)); }
     ~HOT()     { DeleteAll(); }
     void  DeleteAll();
     int        Open(LPCTSTR filename);
     int        CurInHot(POINT pt );
     int        EnableHot(int id ,int statue);
                 //enable or disnable one hotspot

     HRGN  SetRgn(HDC hdc,int id);
     void  ResetRgn(HDC hdc);
};
```

```
#endif

#include "stdafx.h"

#include "hotspot.h"

int HOT::Open(LPCTSTR filename)
{
     HFILE  fh;
     char poly_header[5];

     struct{
          char  image[128];              //image name
          char  target[128];             //hotspot name
          int   num;                     //number of hotspots
          POINT points[MAX_POINTS];      //point array for every hotspot
     } hsr;

     if(HotNumber)DeleteAll();

     fh = _lopen(filename,OF_READ);
     _lread(fh,poly_header,4);
     poly_header[4] = '\0';
     if (strcmp(poly_header,"POLY")){_lclose(fh);return FALSE; }

     for ( ; ; HotNumber++)
          {
               if(_lread(fh,&hsr,sizeof(hsr))!=sizeof(hsr))break;

               PHot[HotNumber].hRGN
            =CreatePolygonRgn(hsr.points,hsr.num, ALTERNATE );
               strcpy(PHot[HotNumber].image, hsr.image);
               strcpy(PHot[HotNumber].target, hsr.target);
               PHot[HotNumber].active = 1;
          }
     _lclose(fh);

     return HotNumber;
}

int  HOT::CurInHot(POINT pt )
{
     for ( int i = 0 ; i < HotNumber;  i ++ )
          if(PtInRegion(PHot[i].hRGN,pt.x,pt.y)&&PHot[i].active)          ret
urn i;
     return _1;
}

void HOT::DeleteAll()
{
```

```
    for(int i=0;i<HotNumber;i++) if(PHot[i].hRGN)DeleteObject( PHot[i].hRGN
 );
    memset(this,0,sizeof(HOT));
}

//if statue<0 return Current active
//else set active=statue and return last active
int HOT::EnableHot(int id ,int statue)
{
    if ( id >= HotNumber || id < 0 )return _1;
    int i=PHot[id].active;
    if (statue<0) return i;
    PHot[id].active = statue ;
    return i;
}

HRGN HOT::SetRgn(HDC hdc,int id)
{
    if ( id >= HotNumber || id < 0 ) return FALSE;

    if(hdc,PHot[id].hRGN)hOldRgn=
                (HRGN)SelectObject(hdc,PHot[id].hRGN);
      return hOldRgn;
}

void HOT::ResetRgn(HDC hdc)
{
    if(hOldRgn)SelectObject(hdc,hOldRgn);
    hOldRgn=0;
}
```

热点类实现将dat文件转化为RGN，并且对RGN进行操作功能。

热点类由以下成员函数组成：

open(LPCTSTR filename)

打开名为 filename的热点文件，并将热点坐标转成HRGN。

CurInHot(point pt)

检查点point是否属于热点区域内，假如是则返回属于哪一个热点。

EnableHot (int id, int statue)

把第id个热点设置成使用或不使用状态，假如Statue是负数，则返回第id个热点的状态。

DeletAll( )

删除所有热点。

SetRgn( )

设置HDC上可使用区域，便将以后对HDC的操作只限制在指定区域范围内，而对范围外的区域无效。

ResetRgn( )返回设置RGN以前的状态。

在程序中使用类HOT

现在我们要编一个安装程序，假设这个安装程序有如下功能：有四个按键，分别是"install"、"uninstall"、"Run"、"Exit"，其中“Run”键为禁止状态，当鼠标移到任何一个按键上，除禁止状态外，按钮都要变成加亮状态，鼠标形状变成手状，当鼠标在按钮上按下时，按钮形状变成按下状态，鼠标形状变成按下的手状。为了编好这个程序，要美工做

出四张图片，第一张为正常状态，第二张为四个按钮全为加亮状态，第三张为四个按钮全部按下状态，第四张为四个按钮全部禁止状态。我们如下构造应用程序：

建立一个基于对话框的应用程序HOT，在类CHotDLg pt增加七个成员变量，其中一个热点类HOT hot，四个bmp 类BMP bmp[4]，分别表示四张位图，二个鼠标HCURSOR hcu[2]。在对话框初始化函数中，分别读入四个bmp位图，一个热点资源文件“1.dat”，两个鼠标资源。注意四个位图文件只有一个有调色板，并且要用SetClass Long函数将窗口类属性设成没有缺省鼠标，否则不能用SetCusor函数设置当前鼠标。然后用Classwizard添加三个成员函数OnMouseMove、OnLButtonDown、OnLButtonUp。处理OnMouseMove函数过程如下：检查鼠标是否被按下。检查鼠标移到了哪个热点内，根据热点及鼠标状态确定当前应该用哪个鼠标。检查当前热点是否刚才热点，如果两个相同，表示状态没有发生改变，那么就没有必要进行进一步的处理，函数就可以返回了。若发生了改变，则要区分进行进一步的处理，函数就可以返回了，若发生了改变，则要区分两种状态：一种是从外面进入热点，那么要重画当前热点，假如鼠标移出热点，那么要重画刚才热点。处理鼠标键按下过程如下：检查当前鼠标在哪个热点内，并重画这个热点，再将鼠标形状设成手指按下状。处理鼠标键放开过程如下：检查当前鼠标在哪个热点，并重画这个热点，并将鼠标设回手指状。编译并运行程序。

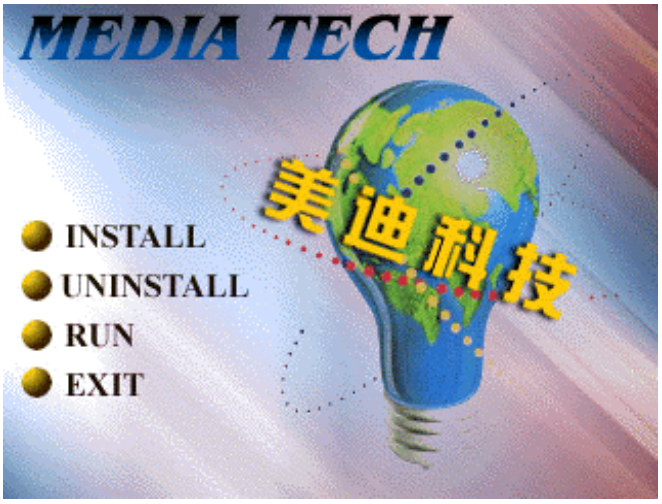


图5.6