C++ Builder 程序设计实用教程

张　晋　　编著

五洋工作室供稿

前　言

C++ Builder（以下简称 CBuilder）是由著名的 Borland 公司开发的开始化程序开发工具，它的基础语言为面向对象的 C++语言，同时它继承了 C++语言的优点，如代码结构清晰、可读性好和代码执行效率高等，所以 CBuilder 一经推出，迅速得到用户的好评。总之，CBuilder 是开发 Windows 应用程序最为强大的工具之一，无论是 Windows 程序设计的初学者，还是有经验的 Windows 程序员，利用 CBuilder 都可以迅速地开发出自己满意的应用程序。

如同大多数计算机软件一样，Borland 公司一直在试图完善这个令人刮目相看的开发软件，所以 CBuilder 一直在人们的期待中更新版本。1999 年底最新发布的 CBuilder 5.0 版，一方面保持了良好的向下兼容性，另一方面，随着版本的提高，CBuilder 程序设计语言在各个方面都有了不同程度的提高，如数据库的体系结构、ActiveX 控件的开发、Web 应用程序和安全进程等，从而使得 CBuilder 的功能更加强大，使用起来也更加方便和灵活，提高了应用程序的开发效率。

为了满足广大读者的愿望，迅速掌握 Windows 环境下的 CBuilder 5 编程方法，作者编写了本书，全书各章的结构安排如下所示：

第一章介绍了利用 CBuilder 5 编制应用程序的一些基础知识，以及如何利用 CBuilder 5 编制简单的 Windows 应用程序；

第二章介绍了 CBuilder 5 中基本的用途和使用方法，熟练地利用 ActiveX 控件开发应用程序是每一个读者应该必备的本领；

第三章介绍了 CBuilder 5 中菜单、工具栏和对话框设计；

第四章介绍了 CBuilder 5 中进行简单的图形绘制和图像处理的基本方法；

第五章集中介绍了在 CBuilder 5 中进行文件处理操作的方法；

第六章集中介绍了 CBuilder 5 中有关用户窗体的界面开发和程序设计；

第七章介绍了 CBuilder 5 中非常吸引人的网络与数据库开发方法。

第八章介绍了多媒体程序开发的常用方法，在这一章中还设计了诸如音频播放器和视频播放器等实例程序；

第九章为最后一章，我们介绍了有关 Windows 信息共享技术的编程，这一应用在 CBuilder 5 中是独挡一面的，希望读者仔细学习。

书中融合了利用 CBuilder 5 开发 Windows 下应用程序的理论和实践，全面和深入地介绍了利用 CBuilder 5 开发应用程序的常用方法和技巧，由于书中附带了很多的程序开发实例，所以实用性很强。需要说明的是，书中示例中所含程序代码、实例名称、图形图片、数据信息等，内容如有雷同，纯属巧合。

最后，由于作者水平有限，编著此书的时间比较仓促，所以书中的错误和不足之处在所难免，望广大读者提出批评和指正，特此感谢！

第一章　C++ Builder 程序设计概述

随着编程概念的更新，VCL（可视化）编程已经成为人们关注的焦点，由 Borland C++系列发展而来的C++ Builder程序设计语言就是一种典型的可视化编程语言，由于它继承了 Borland C++程序设计语言中的优越的全功能平台，弥补了可视化编程与全功能平台之间的障碍，所以编程效率高、代码质量好、更加面向对象，而且解决了诸如 Visual C++、Visual Basic 等可视化编程语言的代码执行效率低下等问题。

注意：

> ✍ Borland 公司也将 C++ Builder 简称为 CBuilder，我们在本书的后面内容中将部分使用这个称呼。

本章，我们首先来学习使用 CBuilder 5 进行 Windows 编程基础知识。

1.1 CBuilder 5 的主要特性

到今天为止，C++ Builder 程序设计语言已经发展到了 5.0 版（即通常所说的 C++ Builder 5），无论是程序设计的初学者，还是大型应用程序的开发人员，利用 CBuilder 5 都可以开发出满意的程序，下面就来介绍一下 CBuilder 5 的主要特性。

1. ActiveX 增强功能

CBuilder 5 增强了许多新的 ActiveX 功能，比如：自动化向导可以生成支持各种自动化服务器对象事件的代码，数据绑定 ActiveX 控件可以和 VCL 数据集通信，新的 COM（资源对象模块）对象向导提供了创建简单的 COM 对象的功能，类库编辑器支持 DLL（动态链接库）和 OOC（面向对象的 C）。

2. 对 XML 的支持

CBuilder 5 提供了对 XML 的支持，从而简化了数据分布，优化了数据交换。用户可以方便地创建高效率的 Internet 程序，来把数据迅速发布到 Internet 上。新版 CBuilder 对 XML 数据的支持使得开发人员能够快速建立起具有可移植性和扩展性的系统，以迎接 Internet 正面临着的又一次冲击波——电子商务。

3. 对 HTML 4 的支持

CBuilder 5 支持 HTML 4，使用户能在 Web 应用中创建各种类型的动态“瘦”客户端程序。把 HTML 4 和 XML 结合起来使用，从而创建动态的极瘦客户端程序，用来满足 Internet 上各种任务的需求。

4. ADO（ActiveX 数据对象）

ADO（ActiveX Data Objects）是微软提供的一项技术。通过 ADO，我们可以方便地访问各种类型的数据库，特别是 OLE DB 数据库。ADO 已成为访问数据库的新的标准接口。CBuilder 5 增添了对 ADO 的支持，是为了让用户能迅速实现对终端用户用来做商业决策的数据的一致性访问，结合 CBuilder 本身的开放式数据元件结构，用户可以很快地建立应用程序，用来把自己的商业数据通过 Internet 发送给客户、最终用户以及整个销售环节。通过 ADO，CBuilder 5 能让用户快速访问关系型或非关系型数据库以及 E-mail 和文件系统。

5. 改善 IDE

CBuilder 5 中的集成开发环境（IDE）有了很大改进，能极大地提高开发效率，它主要依靠简化读写和浏览代码的操作来提高开发效率。

代码编辑器让浏览本单元或相关单元的内容变得容易，工程管理器中的拖放支持使得从已打开的工程中或资源管理器的对话框中选择文件并增加到工程的文件中变得更简单，使用工程管理器中多个工程管理的功能让用户能同时编辑多个工程的文件，应用 CBuilder 5 的可视化窗口设计让用户能够轻松地从模板中选择创建 Internet 程序、分布式计算及 Windows 程序等等。

与 CBuilder 4 相比，CBuilder 5 的 IDE 新特点主要包括：

- 桌面的用户化设置增强了用户对开发环境的控制；
- 编辑键映射功能可以让编辑器按照用户的习惯来工作；
- 工程浏览可以帮助用户更好地理解代码，操作 VCL；
- 带有树视图和数据图表视图（Data Diagram View）的 DataModule 设计器可以帮助用户充分理解程序中的数据；
- 资源文件与工程管理器和编译器集成在一起；

- 动作列表保存了开发计划；
- 控制面板向导可以帮助用户定制应用程序的属性。

6. 增强 VCL 控件

VCL（Visual Component Library——可视化组件库）是 CBuilder 实现代码重用的基本工具。在 CBuilder 5 中，用户可以利用面向对象设计的强大功能开发出稳定、可靠、高效的程序，可以利用现有的面向对象的控件创建自己的控件。

CBuilder 5 企业版的 VCL 中一共包含有 200 多个控件，利用这些控件，用户可以奇迹般地加快应用程序从开始开发到推向市场的速度。无论是开发 Windows 程序还是 Internet 应用，CBuilder 5 基于控件的开发模式都能大大降低开发任务的难度。

CBuilder 5 的 VCL 的新特点包括：

- 提供用于创建和重用复合控件的帧架构（Frames）；
- 增加了能将 HTML 浏览功能集成到应用程序的 WebBrowser 控件；
- 增加了 Microsoft Office 自动化组件集，以便把 Word、Excel 和 Outlook 等 Office 程序快速集成到应用程序中；
- 属性编辑器（Property Editors）支持自画（Owner Draw）功能，简化了属性选择；
- 增强了对高级自画（Advanced Custom Draw）函数的支持，以加强对 Windows API 的控制。

7. 加快复杂工程的开发速度

CBuilder 5 进一步增强了调试功能，即使是非常复杂的工程，查找并修改错误都变得异常简单。CBuilder 5 改进的调试器能帮助用户理解并控制自己编写的代码。用户利用断点可以在需要的地方深入代码进行调试，而现在，用户甚至可以在断点处设置触发断点后要采取的动作，并把断点进行分组集中在一起，以便快速调试自己感兴趣的代码段。

CBuilder 5 的代码调试器包含以下一些新特点：

- 各种调试窗口和编辑器之间紧密集成，拥有一致的用户界面，支持剪贴板和拖动操作；
- 增加了断点提示条（Breakpoint ToolTips），以加快浏览和定位源代码；
- 新增的断点动作（Breakpoint Actions）功能加强了对调试进程的控制；
- 断点集合（Breakpoint Groups）方便了对多条断点的控制；
- 在 DCUs 和调试符号查找路径的帮助下，调试工作变得更简单；
- FPU 调试窗口支持 MMX 指令，从而增强了底层调试的能力；
- 进行多进程调试时，增加了临时进程选项，并支持调试子进程；
- 支持跨越进程边界的调试；
- 能够挂接到正在运行的进程上进行调用。
- CBuilder 5 发程序调试器支持远程调试、多线程调试，并且可以观察 CPU 调试窗口。

上面的介绍仅仅提及了 CBuilder 5 的一些主要的新特点，它还有更多的改进，有待于我们去挖掘。

可以说，CBuilder 5 是目前创建 Web 和 Windows 应用程序最快速、最有效的开发工具之一。这颗新星到底会发出多么耀眼的光芒呢？且让我们拭目以待。

1.2 CBuilder 5 的运行和环境

首先需要注意的是，要想在您的计算机上安装 CBuilder 5，必须在您的计算机上先安装相应的硬件和软件系统，这些系统要求包括：

- Windows 98、Windows 2000 或者 Windows NT 4.X 版本以上操作系统；
- 80586 或更高级的微处理器；
- 一个 CD-ROM 驱动器；
- Microsoft Windows 支持的 VGA 或分辨率更高的监视器；
- 至少 32MB RAM 内存，强烈推荐 64MB RAM 以上；
- 254MB 以上的硬盘剩余空间。

安装完成后，用鼠标单击“开始”菜单中的“程序”/Borland C++ Builder 5 中的 C++ Builder 5 选项就会激活 CBuilder 5。 CBuilder 启动后，在它的缺省画面（软件开发界面）中，包括有主窗口、对象查看器、窗体窗口和代码窗口等，如图 1-1 所示。

提示：

✍ 为避免隐藏在 CBuilder 后的 Program Manager 和曾经运行过的其它程序扰乱版面，分散您的注意力，不妨在启动 CBuilder 前关掉其它应用程序；启动 CBuilder 后，再最小化隐藏在后面的 CBuilder 5 程序组。这样屏幕上就只留下 CBuilder 窗口可见了。

首次加载 CBuilder，屏幕上会出现四个窗口：

- 标题为“CBuilder-Project1”的 CBuilder 主窗口；
- Object Inspector 窗口；
- 标题为“Form1”的窗体（Form）窗口；
- 标题为“Unit1.cpp”的代码编辑窗口。刚启动时这一窗口的大部分被“Form1”窗体所掩盖。将“Form1”窗体移开，或单击 Form1 窗体下方的状态行，可以使其全部可见。在“Form1”窗体的任意可见位置单击鼠标，可以恢复主窗体可见。

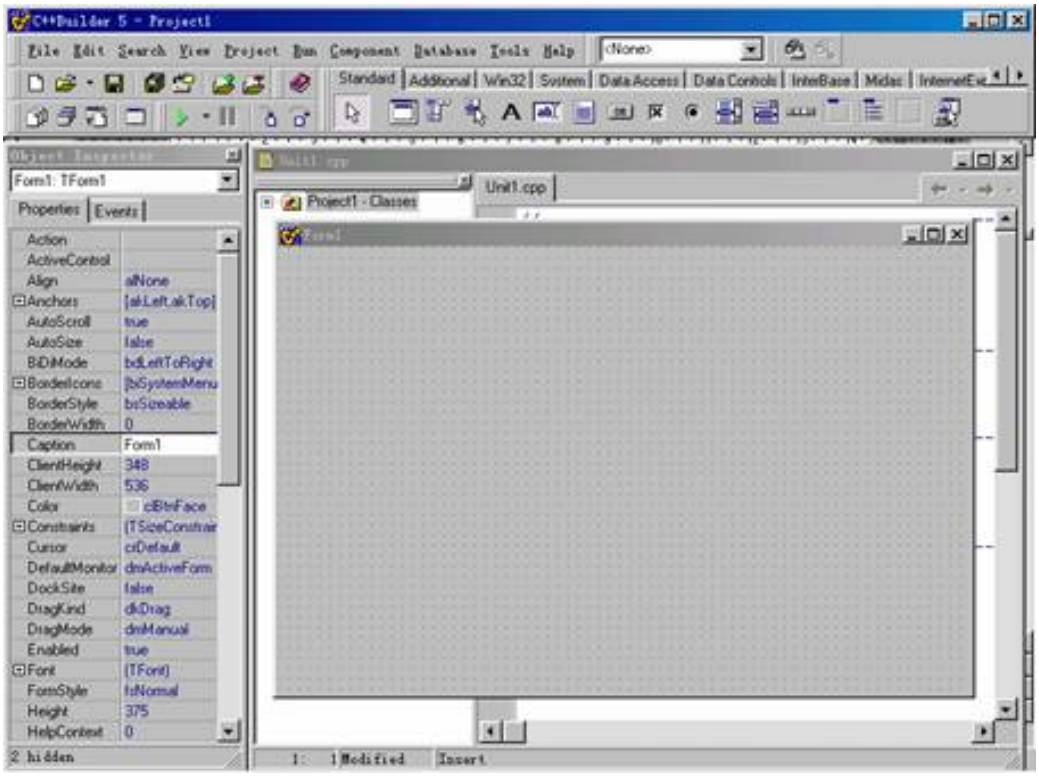


图 1-1 C++ Builder 开发界面

1.2.1 主窗口

CBuilder 的主窗口位于屏幕的上端，包括 Menu（菜单）、Speed Bar（加速条）和 Component Panel（部件选项板）：

- Menu 是下拉式主菜单；

- Speed Bar 位于主窗口的左下端，由两排共 14 个加速按钮组成。这些按钮是菜单功能的快捷方式，各种图标直观地表示了它能执行的动作；
- Component Panel 由一行、若干页对象按钮所组成，利用它来选择需要的部件并将它放到窗体中去。

图 1-2　CBuilder 主窗口

1.2.2　对象检视器

Object Inspector 窗口含有两页：Properties 页显示窗体中当前被选择部件的属性信息，并允许改变对象的属性；Events 页列出了当前部件可以响应的事件。

按动 Object Inspector 下端的“Events”页标签，使得 Events 页可见，这一定的事件后边的空白处，可以定义对象接受到相应事件时执行的动作。

首次启动时，Object Inspector 窗口显示的是当前窗体 Form1 的属性。Object Inspector 根据对象属性的多少，决定是否有滚行显示。移动滚行条，可以查看当前对象的全部属性。

此外，Object Inspector 上还有 Object Selector（对象选择器），位于 Object Inspector 上方的下拉式菜单中。它显示了窗体上所有部件的名称和类型，也包含窗体本身。

可以用 Object Selector 很容易地在窗体的各个部件之间切换，也可以快速地回到窗体本身。当窗体中含有较多的对象时，会发现这是切换对象尤其是回到窗体的最快捷途径。

图 1-3　CBuilder 对象检测器

提示：

✍　想使 Object Inspector 一直可见，可将鼠标移到 Object Inspector 上，按动右键，以启动 Object Inspector 的弹出式菜单，将其设置为 Stay On Top。这对初学者常是一个很重要的设置方式。

1.2.3 窗体窗口

Forms 窗口是开展大部分设计的工作区域。首次启动 CBuilder 5 时显示的是窗体 Form1。可以把部件放在窗体中。
通过移动位置、改变尺寸等操作随心所欲地安排它们，以此来开发应用程序的用户界面。可以把窗体想象成一个可以放置其它部件的容器。
窗体上有栅格(Grids)，供放置部件时对齐位置用，在程序运行时 Grids 是不可见的。如图 1-4 所示。

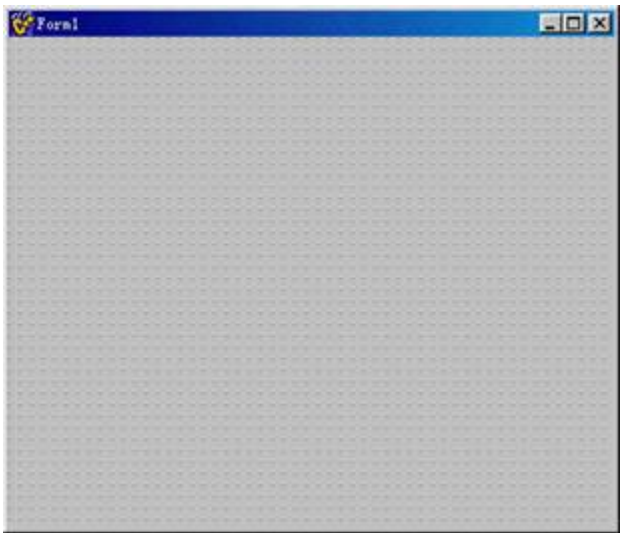


图 1-4　CBuilder 窗体窗口
一个真正的应用程序可能有不止一个窗口，您可以选用不同的窗体进行设计。其它窗体可以是对话框(Dialog Box)、数据录入框等。

1.2.4 代码窗口

代码窗口一开始处于窗体窗口之下。因为在 CBuilder 中，设计用户界面直接在窗体中进行，运行结果和设计样板完全一致。
当部件被放到窗体上时，CBuilder 会自动生成大部分的用户界面代码。您所应做的只是在它为您生成的框架中加入完成所需功能的程序段而已。点动 Form1 的状态行使代码窗口可见。

如图 1-5 所示，显示了空窗体 Form1 的代码窗口。

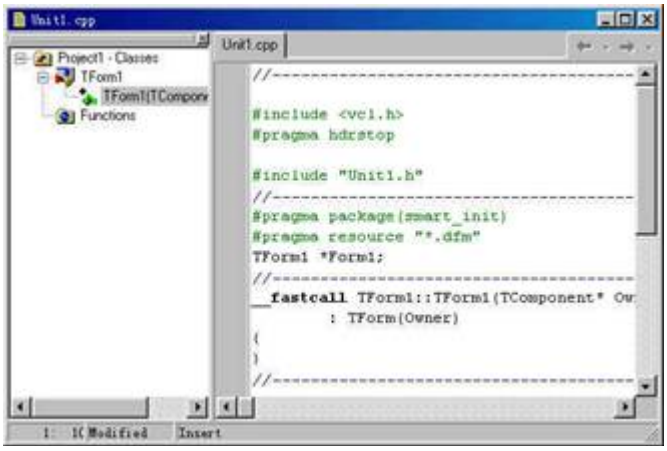


图 1-5　CBuilder 代码窗口

1.3　第一个简单的示例

下面就以一个简单的示例程序来说明，在 CBuilder 5 中进行应用程序开发的一般步骤和基本的方法，为以后的程序设计打一个基础，编制示例程序的基本步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，在 Memo 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 控件，接着向窗体上添加两个 Button 控件，添加控件后的窗体如图 1-6 所示。

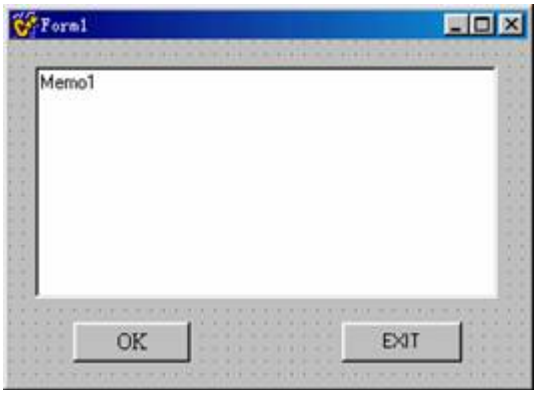


图 1-6　添加控件后的窗体

其中 Memo 控件用于显示文本，而 OK 按钮用于显示一个对话框，EXIT 按钮实现退出程序。窗体和控件的属性设置如下所示：

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 332
   Height = 242
   Caption = 'Form1'
   Color = clBtnFace
   Font.Charset = DEFAULT_CHARSET
```

```
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Button1: TButton
    Left = 40
    Top = 176
    Width = 73
    Height = 25
    Caption = 'OK'
    Font.Charset = ANSI_CHARSET
    Font.Color = clWindowText
    Font.Height = -14
    Font.Name = 'Times New Roman'
    Font.Style = []
    ParentFont = False
    TabOrder = 0
    OnClick = Button1Click
  end
  object Memo1: TMemo
    Left = 16
    Top = 16
    Width = 289
    Height = 145
    Lines.Strings = ( 'Memo1')
    TabOrder = 1
  end
  object Button2: TButton
    Left = 208
    Top = 176
    Width = 75
    Height = 25
    Caption = 'EXIT'
    TabOrder = 2
  end
end
```

2. 程序的初始化

程序的初始化过程，实际上就是对窗体 FormCreate()事件的初始化，在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Memo1->Clear();
//清除文本框
Form1->Memo1->Lines->Text="单击下面的 OK 按钮，您将会看到.....";
//设置 Memo 控件中的文本内容
}
```

程序说明：

首先执行窗体 FormCreate()事件中的代码，即通过语句 Form1->Memo1->Clear();语句来清空文本框中的内容，最后通过语句 Form1->Memo1->Lines->Text="单击下面的 OK 按钮，您将会看到.....";来设置文本框 Memo 中的文本信息，并且显示在文本控件中。

3. 添加代码

在程序的设计过程中，用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口。在程序的设计阶段，把光标移动到代码窗口的 Button1Click()事件处理过程中，并且添加如下所示的按钮响应代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
ShowMessage("Hello World!");
//显示一个“Hello World!”对话框
}
```

在程序运行的过程中，用户用鼠标的左键单击按钮时，那么程序就自动激活按钮控件的 Button1Click()事件，然后通过语句 ShowMessage("Hello World!");来显示一个如图 1-7 所示的对话框。

图 1-7　显示对话框信息

4. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，就会弹出一个如图 1-8 所示的保存文件对话框。

图 1-8　保存文件对话框

在弹出的对话框中选择合适的文件名保存文件。保存程序文件后，按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，文本控件显示一段友好的文本信息，单击 OK 按钮控件，在窗体上就会弹出一个对话框，在其中显示字符串信息“Hello World!”，单击 EXIT 按钮程序结束退出。程序运行的结果如图 1-9 所示。

图 1-9　程序运行结果

程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
```

```
#include <vcl.h>
#pragma hdrstop

#include "Hello.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
ShowMessage("Hello World!");
//显示一个“Hello World!”对话框
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Memo1->Clear();
//清除文本框
Form1->Memo1->Lines->Text="单击下面的 OK 按钮，您将会看到.....";
//设置 Memo 控件中的文本内容
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
  {
Application->Terminate();
//程序运行结束
}
//---------------------------------------------------------------------
```

1.4 快速利用向导

在 CBuilder 5 中系统为用户提供了许多个向导应用程序，利用这些向导程序可以节省很多的劳动，可以少用甚至不用添加代码就可以生成一个应用程序，下面就利用 Application Wizard 向导来生成一个简单的应用程序，具体的程序设计步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New 项，在 CBuilder 5 的集成开发环境中就会弹出一个如图 1-10 标题为 New Item 的对话框，用户在其中可以选择各种向导程序和窗体形式等。

图 1-10　New Items 对话框

在 New Item 对话框中有九个选项——Projects、Data Modules、Business、New、ActiveX、Multitier、Project1、Forms 和 Dialogs，九个选项下对应着相应的选项，如下所示：

- New 选项：Application、Batch File、Component、Data Module、DLL、Form、Package、Project Group、Report、Resource DLL Wizard、Service、Service Application、Text、Thread Object、Unit 和 Web Server Application 等；
- ActiveX 选项：ActiveForm、ActiveX Control、ActiveX Library、Automation Object、COM Object、Property Page 和 Type Library 等；
- Multitier 选项：CORBA Data Module、CORBA Object、MTS Data Module、MTS Object 和 Remote Data Module 等；
- Project1 选项：MainForm 等；
- Forms 选项：About box、Dual list box、QuickReport Labels、QuickReport List、QuickReport Master/Detail 和 Tabbed pages 等；
- Dialogs 选项：Dialog with Help、Dialog Wizard、Password Dialog、Reconcile Error Dialog 和 Standard Dialog 等；
- Projects 选项：Application Wizard、MDI Application、SDI Application 和 Win95/Win98Logo Application 等；
- Data Modules 选项：Customer Data 等；
- Business 选项：Database Form Wizard、DB Web Applicati...、Decision Cube Sample、QuickReport Wizard 和 TeeChart Wizard 等；

选择 Projects 选项下的 Application Wizard，单击 OK 按钮进入下一步。

2. 选择菜单

在由向导生成的应用程序中，用户可以在如图 1-11 所示的菜单选择对话框中选择标准菜单的类型。

图 1-11　菜单选择窗口

在菜单选择对话框中，用户可以选择 File 菜单、Edit 菜单、Windows 菜单和 Help 菜单。相应的菜单设计结构如图 1-12 所示。

在菜单选择对话框中选中四个菜单，然后用鼠标单击 Next 按钮进入下一步。

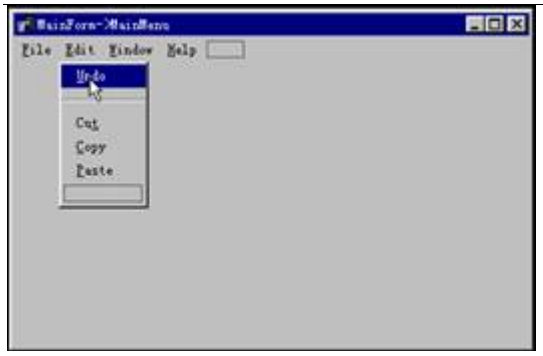


图 1-12　对应的设计菜单

3. 选择工具栏

在应用程序中，我们经常可以看到作为菜单加速控制的工具栏，在 CBuilder 5 的向导程序中，用户也可以对工具栏进行选择操作，如图 1-13 即为一个设计工具栏的对话框，单击 Insert 按钮即可添加相应的加速按钮。

图 1-13　设计工具栏上的加速按钮

设计完成的工具栏如图 1-14 所示。

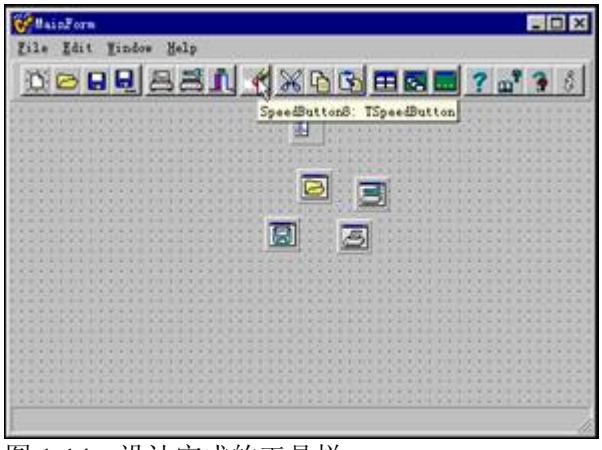


图 1-14　设计完成的工具栏

在工具栏设计对话框中用鼠标单击 Next 按钮进入下一步。

4. 完成设置

在向导程序的最后一个对话框中，用户可以为生成的应用程序选择一个有效的驱动器路径和一个应用程序的文件名，如图 1-15 所示。

图 1-15　选择程序路径和设置

在这个对话框中不但可以选择程序的路径和存储文件名，还可以对程序的特性进行设置，如是否作为一个 MDI 程序，是否建立一个状态条，在状态条中是否显示状态信息等。

在对话框中选择好应用程序的路径和文件名后，在 Options 选项卡下选中三个选项，单击 Finish 按钮就完成了应用程序的设计工作。

5. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序的运行的结果如图 1-16 所示。

图 1-16　程序运行结果

但是在这个应用程序中，用于还没有添加各个事件的响应代码，所以它的程序功能还不是很完善，但是它为我们生成了一个标准的程序框架，用户可以在此基础上添加自己的代码。

由向导生成的程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include "Main.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TMainForm *MainForm;
//---------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
```

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
	Application->OnHint = ShowHint;
}
//---------------------------------------------------------------------

void __fastcall TMainForm::ShowHint(TObject *Sender)
{
	StatusLine->SimpleText = Application->Hint;
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FileNew(TObject *Sender)
{
	//--- Add code to create a new file ---
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FileOpen(TObject *Sender)
{
	if (OpenDialog->Execute())
	{
		//---- Add code to open OpenDialog->FileName ----
	}
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FileSave(TObject *Sender)
{
	//---- Add code to save current file under current name ----
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FileSaveAs(TObject *Sender)
{
	if (SaveDialog->Execute())
	{
		//--- Add code to save current file under SaveDialog->FileName ---
	}
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FilePrint(TObject *Sender)
{
	if (PrintDialog->Execute())
	{
		//---- Add code to print current file ----
	}
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FilePrintSetup(TObject *Sender)
{
	PrintSetupDialog->Execute();
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FileExit(TObject *Sender)
{
	Close();
```

```
}
//---------------------------------------------------------------------
void __fastcall TMainForm::EditUndo(TObject *Sender)
{
    //---- Add code to perform Edit Undo ----
}
//---------------------------------------------------------------------

void __fastcall TMainForm::EditCut(TObject *Sender)
{
    //---- Add code to perform Edit Cut ----
}
//---------------------------------------------------------------------

void __fastcall TMainForm::EditCopy(TObject *Sender)
{
    //--- Add code to perform Edit Copy ----
}
//---------------------------------------------------------------------

void __fastcall TMainForm::EditPaste(TObject *Sender)
{
    //---- Add code to perform Edit Paste ----
}
//---------------------------------------------------------------------

void __fastcall TMainForm::WindowTile(TObject *Sender)
{
    Tile();
}
//---------------------------------------------------------------------

void __fastcall TMainForm::WindowCascade(TObject *Sender)
{
    Cascade();
}
//---------------------------------------------------------------------

void __fastcall TMainForm::WindowArrange(TObject *Sender)
{
    ArrangeIcons();
}
//---------------------------------------------------------------------

void __fastcall TMainForm::HelpContents(TObject *Sender)
{
    Application->HelpCommand(HELP_CONTENTS, 0);
}
//---------------------------------------------------------------------

void __fastcall TMainForm::HelpSearch(TObject *Sender)
{
    Application->HelpCommand(HELP_PARTIALKEY, Longint(""));
}
//---------------------------------------------------------------------

void __fastcall TMainForm::HelpHowToUse(TObject *Sender)
{
    Application->HelpCommand(HELP_HELPONHELP, 0);
}
```

```
//---------------------------------------------------------------------

void __fastcall TMainForm::HelpAbout(TObject *Sender)
{
      //---- Add code to show program's About Box ----
}
//---------------------------------------------------------------------
```

1.5 小　结

在本章中，通过两个示例应用程序，使得用户对 CBuilder 5 下的可视化编程有一个基本的认识。

通过本章的学习，读者可以对 CBuilder 5 编程特性有一个大致的概念，但是有一些东西还是需要读者在编程的实践中自己去体会。第二章，我们将学习如何使用 Cbuilder 中常用的对话框控件开发程序。

第二章　常用的可视化组件

在 CBuilder 5 中包括很多的组件，同时又有几个最为基本的组件，也是最为常用的组件，它们是 Button 组件、Bitbtn 组件、Memo 组件、CheckBox 组件、RadioButton 组件、Calendar 组件、RichEdit 组件、ListView 组件以及 ColorGrid 组件，能够熟练的运用这些组件，是设计大应用程序的基础，所以在本章中，将以几个示例程序的形式来向读者说明各个组件的使用方法。

2.1 通用按钮组件

计算器是我们进行科学计算的基本工具之一，在 Windows 操作系统的“附件”中就带有一个“计算器”应用程序，在 Windows 的“开始”菜单中，选择“程序”/“附件”中的“计算器”选项，在屏幕上就会弹出一个如图 2-1 所示的“计算器”。

图 2-1 Windows 中的计算器

读者可以看到，计算器上有众多的按钮，点击这些按钮就可以实现简单的运算了。下面就利用按钮组件（Button 组件）来制作一个“计算器”应用程序，它能够实现整数的简单的加减乘除运算，具体的程序设计步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Button 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮组件，接着向窗体上添加 15 个 Button 组件和一个 Edit 组件，各个组件的功能将在后面分别的加以介绍。

添加组件后的窗体如图 2-2 所示。

图 2-2　添加组件后的窗体

在上面所添加的 16 个 Button 组件中，可以大致的分为三类——字符类、运算类和清零类，下面分别的加以介绍。

2. 字符类 Button 组件

所谓的字符类 Button 组件，即十个 Caption 属性设置为从 0—9 的 Button 组件，它们的作用是接收用户在程序运行过程中的输入，它们的属性设置如下所示：

```
object Button1: TButton
   Caption = '1'
end
object Button2: TButton
   Caption = '2'
end
object Button3: TButton
   Caption = '3'
end
object Button4: TButton
   Caption = '4'
end
object Button5: TButton
   Caption = '5'
end
object Button6: TButton
   Caption = '6'
end
object Button7: TButton
   Caption = '7'
end
object Button8: TButton
   Caption = '8'
end
object Button9: TButton
   Caption = '9'
end
object Button10: TButton
   Caption = '0'
end
```

其实在它们的属性设置中，还应该包括如 Left、Top、Width 和 Height 等属性的设置，但是由于篇幅的原因，在这里就不多加介绍了，读者可以从添加组件后的窗体中看出它们之间的相对位置关系。

3. 运算类 Button 组件

所谓的运算类 Button 组件指的是五个分别能够完成加、减、乘、除、输出运算结果的 Button 组件，它们的属性设置如下所示：

```
  object Button11: TButton
    Caption = '='
  end
  object Button13: TButton
    Caption = '+'
  end
  object Button14: TButton
    Caption = '_'
  end
  object Button15: TButton
    Caption = '*'
  end
  object Button16: TButton
    Caption = '/'
  end
```

与列举字符类 Button 组件相同，操作类 Button 组件的属性也只列举了 Caption 属性，如果读者有什么不清楚的地方，请参看添加组件后的窗体。

4. 程序的初始化

由于在程序运行的过程中，要进行诸如存储数据等操作，所以在窗体的声明段中要声明三个变量 temp1、temp2 和 flag。

```
TForm1 *Form1;
int temp1;
float temp2;
int flag;
```

其中变量 temp1、temp2 用做中间变量，即临时存储数据，而整型变量 flag 则用于在程序运行的过程中判断用户选择的运算类型，即用户到底选择的是加法运算还是减法运算等。

在程序的初始化过程中，还应该包括窗体 FormCreate()事件的初始化，在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Form1->Edit1->Clear();
 //清除文本框中的内容
 temp1=0;
 temp2=0;
 flag=0;
 //变量赋初值
}
```

窗体 FormCreate()事件中的语句 form1->Edit1->Clear();用来在程序运行的初期清空文本框，以接收用户的输入。

5. 响应字符输入

在程序运行的过程中，用户在字符类 Button 组件上按下鼠标的左键时，在文本框中就应该能够随时的显示出用户的输入，为了实现这一功能，需要在十个字符类 Button 组件的事件响应中添加如下代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button1->Caption;
 //单击 1 按钮
}
……                          //略去
void __fastcall TForm1::Button10Click(TObject *Sender)
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button10->Caption;
 //单击 0 按钮
}
```

由于篇幅的关系，在这里仅仅列举了字符类组件 Button1 和 Button10 的事件响应代码，其它几个组件的响应代码请读者参见附后的源程序代码。

6. 运算符响应

在本示例程序中，共有四种运算符——“+、-、*、/”，在程序运行的过程中，单击这四个运算符，就可以实现相应的加减乘除运算，下面以加法运算为例来说明在程序中如何实现简单的数学运算。

```
void __fastcall TForm1::Button13Click(TObject *Sender)
{
flag=1;
//设置加法标志
temp1=StrToInt(Form1->Edit1->Text);
//为变量 temp1 赋值
Form1->Edit1->Clear();
//清空文本框
}
```

如果用户在程序运行的过程中，用鼠标的左键单击“+”，程序首先会设置标志变量 flag 的值为 1，也就表示程序将要进行的数学运算为加法运算，然后通过语句 temp1=StrToInt(Form1->Edit1->Text);把当前文本框中的数据存储在临时变量 temp1 中，最后清空文本框。

其它几个运算符的代码实现过程请读者参看附后的源程序。

7. 输出运算结果

最后程序运算的结果要输出到文本框中，但是其中的一个难点就在于如何判断用户所选择的算法类型，在本程序中，标志变量 flag 中存储着用户的选择类型，所以在“=”按钮的响应代码中通过一个选择语句来得到程序的运算结果：

```
void __fastcall TForm1::Button11Click(TObject *Sender)
{
 switch (flag)
 {
       case 1: temp2=StrToFloat(Edit1->Text)+StrToFloat(temp1);
                  //进行加法运算
                  break;
       case 2: temp2=StrToFloat(temp1)-StrToFloat(Edit1->Text);
                  //进行减法运算
                  break;
       case 3: temp2=StrToFloat(Edit1->Text)*StrToFloat(temp1);
                  //进行乘法运算
                  break;
       case 4: temp2=StrToFloat(temp1)/StrToFloat(Edit1->Text);
                  //进行除法运算
 }
```

```
 Edit1->Text=FloatToStr(temp2);
 //显示运算结果
}
```

值得注意的是，临时变量 temp2 中存储的程序运算结果为 Float 类型，而文本框的输出类型为 String 类型，所以在程序中要调用一个 floattostr()函数进行数据类型的转换，最后通过语句 edit1->Text=FloatToStr(temp2) ;来输出程序的运算结果。

8. 运行程序

按照附后的源程序，添加剩余的程序代码后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的初始画面如图 2-3 所示。

图 2-3　程序运行的初始画面

在程序运行的过程中，用鼠标单击窗体上的数字按钮，输入数据 520，然后单击运算符“*”，再次输入数据 1314，单击“=”，程序运行结果如图 2-4 所示。

图 2-4　程序运算的结果

在这个示例应用程序中，不但可以实现加法运算，还可以实现减法、乘法和除法的运算。而单击“清零”按钮，就可以清除文本框中所有输入的数据，重新开始计算。

程序完整的源代码如下所示：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#include "U_Calc.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
int temp1;
```

```
float temp2;
int flag;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Form1->Edit1->Clear();
 //清除文本框中的内容
 temp1=0;
 temp2=0;
 flag=0;
 //变量赋初值
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button1->Caption;
 //单击 1 按钮
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button2->Caption;
 //单击 2 按钮
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button3->Caption;
 //单击 3 按钮
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button4->Caption;
 //单击 4 按钮
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button5->Caption;
 //单击 5 按钮
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
 Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button6->Caption;
 //单击 6 按钮
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button7Click(TObject *Sender)
{
  Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button7->Caption;
  //单击 7 按钮
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button8Click(TObject *Sender)
{
  Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button8->Caption;
  //单击 8 按钮
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button9Click(TObject *Sender)
{
  Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button9->Caption;
  //单击 9 按钮
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button10Click(TObject *Sender)
{
  Form1->Edit1->Text=Form1->Edit1->Text+Form1->Button10->Caption;
  //单击 0 按钮
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button13Click(TObject *Sender)
{
flag=1;
//设置加法标志
temp1=StrToInt(Form1->Edit1->Text);
//为变量 temp1 赋值
Form1->Edit1->Clear();
//清空文本框
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button14Click(TObject *Sender)
{
flag=2;
//设置减法标志
temp1=StrToInt(Form1->Edit1->Text);
//为变量 temp1 赋值
Form1->Edit1->Clear();
//清空文本框
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button15Click(TObject *Sender)
{
flag=3;
//设置乘法标志变量值
temp1=StrToInt(Form1->Edit1->Text);
//为变量 temp1 赋值
Form1->Edit1->Clear();
//清空文本框
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button16Click(TObject *Sender)
{
flag=4;
//设置除法标志
temp1=StrToInt(Form1->Edit1->Text);
//为变量 temp1 赋值
Form1->Edit1->Clear();
//清空文本框
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button11Click(TObject *Sender)
{
 switch (flag)
 {
        case 1: temp2=StrToFloat(Edit1->Text)+StrToFloat(temp1);
                //进行加法运算
                break;
        case 2: temp2=StrToFloat(temp1)-StrToFloat(Edit1->Text);
                //进行减法运算
                break;
        case 3: temp2=StrToFloat(Edit1->Text)*StrToFloat(temp1);
                //进行乘法运算
                break;
        case 4: temp2=StrToFloat(temp1)/StrToFloat(Edit1->Text);
                //进行除法运算
 }
 Edit1->Text=FloatToStr(temp2);
 //显示运算结果
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button12Click(TObject *Sender)
{
 Form1->Edit1->Clear();
 //清除文本框中的内容
 temp1=0;
 temp2=0;
 flag=0;
 //变量赋初值
}
//---------------------------------------------------------------------------
```

2.2 颜色调节组件

在图形图象处理软件中，经常可以见到类似于调色板的颜色调节程序，利用这些颜色调节程序，可以很方便的在程序的运行过程中改变系统的颜色设置。
在 CBuilder 5 中，开发工具为我们提供了很好的颜色调节组件，下面就在 CBuilder 5 中制作一个颜色调节程序，它能够实现动态调节文本输入框的前景色和背景色。
具体的程序制作步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Memo 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 组件。
添加 Memo 组件后，把鼠标移动到组件工具栏上的其它选项上，向窗体上添加一个 CColorGrid 组件、两个 CheckBox 组件和两个 Button 组件，各个组件的功能在后面将予以详细的介绍。添加组件后的窗体如图 2-5 所示。

图 2-5 添加组件后的窗体
其中窗体中各个组件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 363
  Height = 345
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object CheckBox2: TCheckBox
    Left = 256
    Top = 272
    Width = 89
    Height = 25
    Caption = '随机背景色'
    Font.Charset = ANSI_CHARSET
    Font.Color = clWindowText
    Font.Height = -14
```

```
    Font.Name = 'Times New Roman'
    Font.Style = []
    ParentFont = False
    TabOrder = 0
    OnClick = CheckBox2Click
  end
  object CheckBox1: TCheckBox
    Left = 256
    Top = 232
    Width = 97
    Height = 17
    Caption = '随机前景色'
    Font.Charset = ANSI_CHARSET
    Font.Color = clWindowText
    Font.Height = -14
    Font.Name = 'Times New Roman'
    Font.Style = []
    ParentFont = False
    TabOrder = 1
    OnClick = CheckBox1Click
  end
  object Button2: TButton
    Left = 136
    Top = 280
    Width = 97
    Height = 25
    Caption = '背 景 选 定'
    Font.Charset = ANSI_CHARSET
    Font.Color = clWindowText
    Font.Height = -14
    Font.Name = 'Times New Roman'
    Font.Style = []
    ParentFont = False
    TabOrder = 2
    OnClick = Button2Click
  end
  object Button1: TButton
    Left = 136
    Top = 232
    Width = 97
    Height = 25
    Caption = '前 景 选 定'
    Font.Charset = ANSI_CHARSET
    Font.Color = clWindowText
    Font.Height = -14
    Font.Name = 'Times New Roman'
    Font.Style = []
    ParentFont = False
    TabOrder = 3
    OnClick = Button1Click
  end
  object Memo1: TMemo
    Left = 8
    Top = 8
    Width = 337
    Height = 217
    Font.Charset = ANSI_CHARSET
    Font.Color = clWindowText
    Font.Height = -14
    Font.Name = 'Times New Roman'
```

```
    Font.Style = []
    Lines.Strings = (
      'Memo1')
    ParentFont = False
    TabOrder = 4
  end
  object CColorGrid1: TCColorGrid
    Left = 0
    Top = 232
    Width = 128
    Height = 84
    TabOrder = 5
    OnChange = CColorGrid1Change
  end
end
```

2. 响应按钮事件

在窗体上的两个按钮组件的功能如下所示：

- “前景选定”按钮（组件 Button1）：在程序运行的过程中动态的设置文本输入框的前景色，即组件中文本的显示颜色；
- “背景选定”按钮（组件 Button2）：在程序的运行过程中动态的设置文本输入框的背景色。

能够实现以上功能的按钮响应代码如下所示：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->Memo1->Font->Color=Form1->CColorGrid1->ForegroundColor;
//设置文本框的前景色
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->Memo1->Color=Form1->CColorGrid1->BackgroundColor;
//设置文本框的背景色
}
//---------------------------------------------------------------------------
```

在程序运行的过程中，当用户单击这两个按钮时，程序就会激活相对应的语句来设置 Memo1 组件的文本显示前景色和背景色。

3. 响应 CColorGrid 事件

CColorGrid 组件的功能是在程序运行的过程中，为文本框组件提供调节颜色的容器，在程序设计的过程中，用鼠标的左键双击 CColorGrid 组件，在弹出的代码窗口中添加如下所示的 CColorGrid 组件响应代码：

```
void __fastcall TForm1::CColorGrid1Change(TObject *Sender)
{
Form1->Memo1->Font->Color=Form1->CColorGrid1->ForegroundColor;
Form1->Memo1->Color=Form1->CColorGrid1->BackgroundColor;
//随时更改文本框的颜色设置
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，当用户用鼠标单击 CColorGrid 组件时，就会激活 CColorGrid1Change 事件，CColorGrid 组件会自动判断用户单击鼠标左键还是鼠标右键，如果是鼠标左键单击

CColorGrid 组件，那么就通过 Form1->Memo1->Font->Color =Form1->CColorGrid1->ForegroundColor;语句来设置 Memo1 组件的文本显示前景色，如果是鼠标右键单击，那么就通过语句 Form1->Memo1->Color=Form1->CColorGrid1-> BackgroundColor；来设置 Memo1 组件的背景色。

4. 响应 CheckBox 组件

CheckBox 组件的功能是在程序运行的过程中，为 Memo1 组件提供一个随机的文本显示前景色和背景色，为此在程序设计的过程中，用鼠标的左键双击相应的 CheckBox 组件，在弹出的代码窗口中添加如下所示的响应代码：

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
if (Form1->CheckBox1->Checked)
    Form1->Memo1->Font->Color=RGB(random(255),random(255),random(255));
    //在 Memo 组件中以随机前景色显示文本
}
//---------------------------------------------------------------------

void __fastcall TForm1::CheckBox2Click(TObject *Sender)
{
if (Form1->CheckBox2->Checked)
    Form1->Memo1->Color=RGB(random(255),random(255),random(255));
    //在 Memo 组件中显示随机背景色
}
//---------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，如果用户单击 CheckBox 组件，就会激活相应的 CheckBox1Click 事件或 CheckBox2Click 组件。

然后程序会自动判断 CheckBox 组件的选中状态，如果处于选中状态，那么就会通过语句 Form1->Memo1->Font->Color=RGB(random(255)，random(255)，random(255))；在 Memo 组件中以随机前景色显示文本或通过语句 Form1->Memo1->Color=RGB(random(255)，random(255)，random(255))；在 Memo 组件中显示随机背景色。

5. 运行程序

按照附后的程序清单添加代码。

做完以上各步的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件。

然后按键盘上的功能键【F9】运行程序，程序运行的初始画面如图 2-6 所示。

图 2-6　程序运行的初始画面

在程序的运行的过程中，可以在文本输入框中输入文本后，用鼠标的左键和右键在 CColorGrid 组件上选择文本框的前景色和背景色，还可以用随机颜色来设置 Memo1 组件的参数。

程序运行结果如图 2-7 所示。

图 2-7　程序运行结果

注意在程序运行的过程中，在 CColorGrid 组件上用鼠标左键单击可以选择前景色，用鼠标的右键单击可以选择背景色。

程序的完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <stdlib.h>
#include "Color.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CGRID"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->Memo1->Font->Color=Form1->CColorGrid1->ForegroundColor;
//设置文本框的前景色
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->Memo1->Color=Form1->CColorGrid1->BackgroundColor;
//设置文本框的背景色
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
if (Form1->CheckBox1->Checked)
    Form1->Memo1->Font->Color=RGB(random(255),random(255),random(255));
    //在 Memo 组件中以随机前景色显示文本
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CheckBox2Click(TObject *Sender)
{
if (Form1->CheckBox2->Checked)
    Form1->Memo1->Color=RGB(random(255),random(255),random(255));
    //在 Memo 组件中显示随机背景色
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CColorGrid1Change(TObject *Sender)
{
Form1->Memo1->Font->Color=Form1->CColorGrid1->ForegroundColor;
Form1->Memo1->Color=Form1->CColorGrid1->BackgroundColor;
//随时更改文本框的颜色设置
}
//---------------------------------------------------------------------------
```

2.3 列表框组件

在进行文本处理的过程中，我们经常要与字体打交道，在这里所讲的字体不是指字的大小，也不是指字体的颜色，而是指“宋体”、“隶书”等字体形式，常见的字体选择对话框如图 2-8 所示。

图 2-8　字体选择对话框

在本节中，我们就利用 ListBox 组件和 ComboBox 组件来制作一个应用程序，在其中可以为文本框中的文字选择一个合适的字体，同时可以通过键盘上的字符来选择合适的字体，具体的程序制作步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Memo 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 组件，接着向窗体上添加一个 ComboBox 组件和一个 ListBox 组件，以及两个用于说明的标签组件。各个组件的功能在后面再加以介绍，各个组件的属性设置如下所示：

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 391
   Height = 269
   Caption = 'Form1'
   Color = clBtnFace
   OnCreate = FormCreate
   object ComboBox1: TComboBox
      Left = 16
      Top = 16
      Width = 169
      Height = 24
      OnClick = ComboBox1Click
   end
   object ListBox1: TListBox
      Left = 16
      Top = 48
      Width = 169
      Height = 177
      OnClick = ListBox1Click
      OnDrawItem = ListBox1DrawItem
      OnMeasureItem = ListBox1MeasureItem
```

```
  end
  object Memo1: TMemo
    Left = 192
    Top = 16
    Width = 169
    Height = 209
  end
end
```

添加组件后的窗体如图 2-9 所示。

图 2-9　添加组件后的窗体

窗体上添加的组件作用如下所示：

- ListBox1 组件：显示系统的字体；
- ComboBox1 组件：显示系统的字体；
- Memo1 组件：显示文本文件。

2. 程序的初始化

程序的初始化过程，也就是对窗体 FormCreate()事件的初始化。在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Form1->ComboBox1->Items=Screen->Fonts;
 Form1->ListBox1->Items=Screen->Fonts;
 //设置列表框控件的列表项
 Form1->Memo1->Lines->LoadFromFile("d:\\my music\\蝴蝶花.txt");
 //设置文本框显示内容
}
//---------------------------------------------------------------------------
```

在程序运行的初期，首先激活窗体 FormCreate()事件中的代码，执行语句 Form1->ComboBox1->Items=Screen->Fonts;和 Form1->ListBox1->Items=Screen->Fonts;在组件 Listbox1 和 ComboBox1 中显示系统当前所有字体，然后通过语句 Form1->Memo1->Lines->LoadFromFile("d:\\my music\\蝴蝶花.txt");在文本组件中显示一个歌词文本的内容。

3. 添加字体选择代码

在程序运行的过程中，当用户在 ComboBox 组件中选择一个字体时，在 ListBox 组件中应该显示出当前选中的字体，同时在 Memo 组件中可以应用当前选中的字体显示指定的文本文件，为了实现以上的功能，在程序设计过程中添加下列代码：

```
void __fastcall TForm1::ListBox1Click(TObject *Sender)
{
   Memo1->Font->Name=ListBox1->Items->Strings[ListBox1->ItemIndex];
   //为 Memo 控件应用选中的字体
   ComboBox1->ItemIndex=ListBox1->ItemIndex;
   //显示相应的关联选项
}
//---------------------------------------------------------------------

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
   ListBox1->ItemIndex=ComboBox1->ItemIndex;
   //显示相应的关联选项
   Memo1->Font->Name=ListBox1->Items->Strings[ListBox1->ItemIndex];
   //为 Memo 控件应用选中的字体
}
//---------------------------------------------------------------------
```

4. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序。
在程序运行的过程中，首先文本区域会显示一段歌词的文本，如图 2-10 所示。

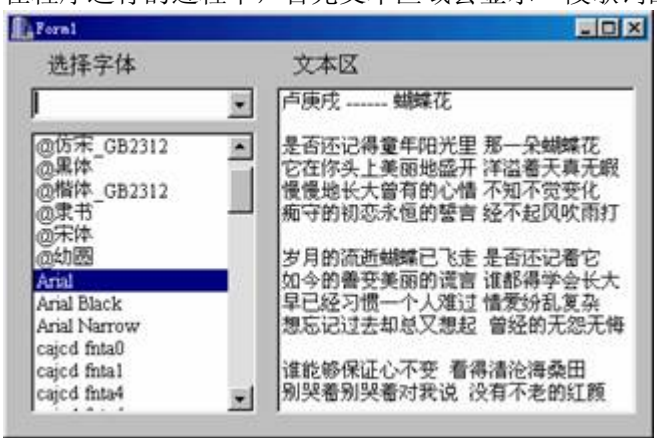


图 2-10　程序运行结果
通过键盘或者鼠标在列表框中选择一种字体，这时的文本框中的文字将会以选定的字体显示，运行结果如图 2-11 所示。

图 2-11　程序运行结果

附程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "List.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Form1->ComboBox1->Items=Screen->Fonts;
  Form1->ListBox1->Items=Screen->Fonts;
  //设置列表框控件的列表项
  Form1->Memo1->Lines->LoadFromFile("d:\\my music\\蝴蝶花.txt");
  //设置文本框显示内容
}
//---------------------------------------------------------------------

void __fastcall TForm1::ListBox1Click(TObject *Sender)
{
```

```
    Memo1->Font->Name=ListBox1->Items->Strings[ListBox1->ItemIndex];
    //为 Memo 控件应用选中的字体
    ComboBox1->ItemIndex=ListBox1->ItemIndex;
    //显示相应的关联选项
}
//---------------------------------------------------------------------

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
    ListBox1->ItemIndex=ComboBox1->ItemIndex;
    //显示相应的关联选项
    Memo1->Font->Name=ListBox1->Items->Strings[ListBox1->ItemIndex];
    //为 Memo 控件应用选中的字体
}
//---------------------------------------------------------------------
```

2.4 文本备忘录组件

在 CBuilder 5 中 Memo 组件的主要作用是显示指定的文本，在使用的过程中，它可以用做文本显示的容器，用于接收用户的输入，同时还可以显示指定的文本文件，所以在一定的程度上，Memo 组件还可以用做文本文件的处理，它的图标如图 2-12 所示。

图 2-12　Memo 组件图标

下面我们就以 Memo 组件为例来说明在 CBuilder 5 中如何进行文本文件的简单处理操作，这个示例程序所能够实现的功能如下所示：

- 可以打开一个典型的文本文件，用户可以在程序运行的过程中选择文件的路径和文件名；
- 进行简单的文本编辑后，可以把修改结果保存到一个文本文件中；
- 可以为 Memo 组件中的显示文本选择合适的字体；
- 可以为 Memo 组件选择合适的背景色。

一个典型的颜色设置对话框如图 2-13 所示。

图 2-13　颜色设置对话框

能够实现以上功能的具体程序制作步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Memo 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 组件。

接着向窗体上添加四个 Button 组件、一个 OpenDialog 组件、一个 SaveDialog 组件、一个 FontDialog 组件和一个 ColorDialog，添加组件后的窗体如图 2-14 所示。

图 2-14　添加组件后的窗体

窗体中各个组件的属性设置如下所示：

```
object Form1: TForm1
   Left = 191
   Top = 107
   Width = 544
   Height = 375
   Caption = 'Form1'
   object Memo1: TMemo
      Left = 40
      Top = 96
      Width = 337
      Height = 201
      Lines.Strings = ( 'Memo1')
      ScrollBars = ssBoth
   end
   object Button1: TButton
      Left = 392
      Top = 96
```

```
    Width = 97
    Height = 33
    Caption = '打开文件'
  end
  object Button2: TButton
    Left = 392
    Top = 152
    Width = 97
    Height = 33
    Caption = '存储文件'
  end
  object Button3: TButton
    Left = 392
    Top = 208
    Width = 97
    Height = 33
    Caption = '字体设置'
  end
  object Button4: TButton
    Left = 392
    Top = 264
    Width = 97
    Height = 33
    Caption = '颜色设置'
  end
  object OpenDialog1: TOpenDialog
    Left = 144
    Top = 56
  end
  object SaveDialog1: TSaveDialog
    Left = 112
    Top = 56
  end
  object FontDialog1: TFontDialog
    Left = 176
    Top = 56
  end
  object ColorDialog1: TColorDialog
    Left = 208
    Top = 56
  end
end
```

2. 程序的初始化

程序的初始化过程，也就是对窗体 FormCreate()事件的初始化。
在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Memo1->Clear();
//清空文本框
Form1->OpenDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->OpenDialog1->InitialDir="c:\\Windows";
//设置缺省文件目录
Form1->OpenDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
```

```
//设置文件过滤条件
Form1->OpenDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->SaveDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->SaveDialog1->InitialDir="c:\\Windows";
//设置缺省文件目录
Form1->SaveDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
//设置文件过滤条件
Form1->SaveDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->Button2->Enabled=false;
//设置按钮有效状态
Form1->FontDialog1->Font=Form1->Memo1->Font;
Form1->ColorDialog1->Color=Form1->Memo1->Color;
//对话框的初始化
}
//---------------------------------------------------------------------
```

程序说明：

在程序运行的初期，程序首先清空文本框中的文本显示内容，这为用户输入文字和打开文件作好了准备工作。

由于程序刚刚开始运行时，还没有打开文件，所以“存储文件”按钮应该处于无效的状态，这是通过语句 Form1->Button2->Enabled=false;来实现的。

在程序的初始化过程中，还包括对 OpenDialog 组件的初始化，通过语句一条 Form1->OpenDialog1->InitialDir:='c:\\windows'；来设置组件所显示对话框的缺省路径为 c:\windows。

同时设置的文件显示过滤器为'All Files(*.*)|*.*|Text Files(*.txt)|*.txt'，这也就意味着，在程序运行的过程中，“打开”对话框显示的文件只有两种选择——要么显示所有的文件，要么只显示以*.txt 为后缀的文本文件。

与 OpenDialog 组件的初始化相同，SaveDialog 组件的初始化过程也是由两条语句：

form1->SaveDialog1->InitialDir:='c:\\windows;

form1->SaveDialog1->Filter:='All Files(*.*)|*.*|Text Files(*.txt)|*.txt;它们的功能在 OpenDialog 组件的初始化过程中已经讲述过了。

在窗体 FormCreate()事件的最后通过两条语句

Form1->FontDialog1->Font=Form1->Memo1->Font;

Form1->ColorDialog1->Color=Form1->Memo1->Color;

完成对“字体”对话框和“颜色”对话框的初始化，把它们的初始状态设置与 Memo 组件相一致。

至此，程序的初始化工作就完成了。

3. 响应“打开文件”按钮

在程序的设计阶段，用鼠标的左键双击窗体上的“打开文件”按钮，在弹出的如图 2-15 所示代码窗口的左侧子窗口中选择 Tform1/Published/Button1Click 选项。

在代码窗口的右侧子窗口中就会显示出组件“打开文件”的代码响应过程。

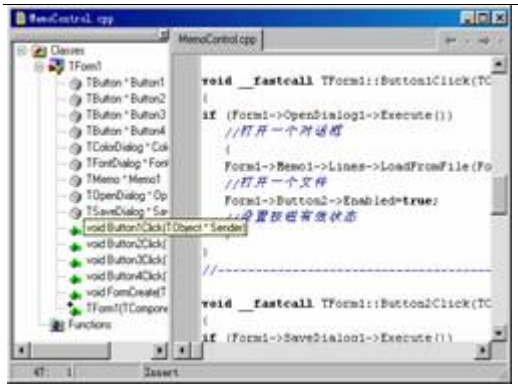


图 2-15　弹出的代码窗口

在其中添加如下所示的按钮响应代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //打开一个对话框
    {
    Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
    //打开一个文件
    Form1->Button2->Enabled=true;
    //设置按钮有效状态
    }
}
//---------------------------------------------------------------------
```

程序说明：

在所添加的代码段中，特别值得注意的是语句 Form1->OpenDialog1->Execute；和语句 Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->Filename)，其中前者的作用是显示一个“打开文件”对话框。用户可以在其中选择一个文本文件，后者的作用是打开用户在“打开文件”对话框中所选择的文件。

当用户打开一个文本文件后，窗体中的“存储文件”按钮就应该变为有效的状态，这一功能是通过语句 Form1->Button2->Enabled=true;来实现的。一个典型的打开文件的对话框如图 2-16 所示。

图 2-16　“打开”对话框

相应的，在“存储文件”按钮的事件响应中添加的代码如下所示：

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (Form1->SaveDialog1->Execute())
    //显示一个对话框
    {
    Form1->Memo1->Lines->SaveToFile(Form1->SaveDialog1->FileName);
    //保存文件
    }
}
//---------------------------------------------------------------------
```

4. 响应属性设置按钮事件

在窗体中放置有一个 FontDialog 组件和一个 ColorDialog 组件，它们的作用是分别为文本框组件设置显示字体和背景色。
下面以 FontDialog 组件为例来说明在程序中如何添加响应属性设置按钮事件的代码。在程序的设计阶段，用鼠标的左键双击组件 FontDialog1，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中。
添加如下所示的按钮事件响应代码：

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
if (Form1->FontDialog1->Execute())
      //打开一个对话框
      {
      Form1->Memo1->Font=Form1->FontDialog1->Font;
      //设置控件中文本显示字体
      }
}
//---------------------------------------------------------------------------
```

程序说明：
在 FontDialog 组件的响应代码中，语句 Form1->FontDialog1->Execute;的作用是显示一个对话框，用户可以在其中为 Memo 组件中的文本设置字体、大小、风格以及颜色等，而 Form1->Memo1->Font=Form1->FontDialog1->Font；语句的作用是将用户在“字体”对话框中的选择传送给 Memo 组件。
一个典型的字体设置对话框如图 2-17 所示。

图 2-17 “字体”对话框

5. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的初始画面如图 2-18 所示。

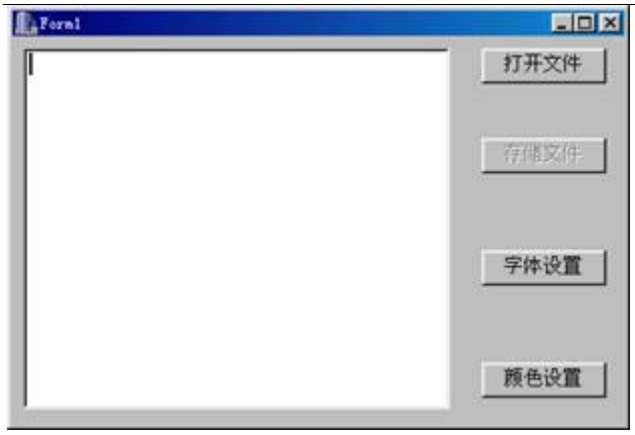


图 2-18　程序运行的初始画面

首先用鼠标的左键单击按钮“打开文件”，在屏幕上就会弹出一个对话框，提示用户选择一个有效的文本文件，单击“打开”按钮，在 Memo 组件中就会显示出文本文件中的内容，如图 2-19 所示。

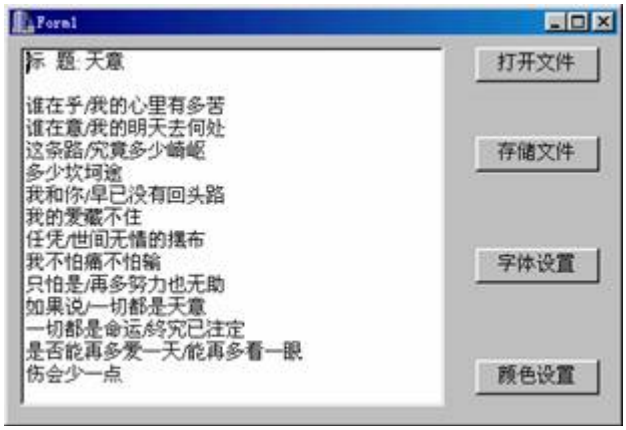


图 2-19　一个打开的文本文件

在程序运行的过程中，用户可以通过单击“颜色设置”按钮和“字体设置”按钮来改变系统的设置，如图 2-20 所示为改变了字体和颜色后的效果（注明：字体为小 6 号，颜色为淡红）。

图 2-20　字体和颜色效果

编辑文本后，单击“存储文件”按钮，在窗体上就会弹出一个“另存为”对话框。在“另存为…”对话框中选择一个有效的路径和文件名，单击“保存”按钮就完成了文本文件的保存工作。

附程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "MemoControl.h"
//---------------------------------------------------------------------
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Memo1->Clear();
//清空文本框
Form1->OpenDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->OpenDialog1->InitialDir="c:\\PWin98";
//设置缺省文件目录
Form1->OpenDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
//设置文件过滤条件
Form1->OpenDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->SaveDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->SaveDialog1->InitialDir="c:\\PWin98";
//设置缺省文件目录
Form1->SaveDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
//设置文件过滤条件
Form1->SaveDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->Button2->Enabled=false;
//设置按钮有效状态
Form1->FontDialog1->Font=Form1->Memo1->Font;
Form1->ColorDialog1->Color=Form1->Memo1->Color;
//对话框的初始化
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //打开一个对话框
    {
    Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
    //打开一个文件
    Form1->Button2->Enabled=true;
    //设置按钮有效状态
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (Form1->SaveDialog1->Execute())
    //显示一个对话框
    {
    Form1->Memo1->Lines->SaveToFile(Form1->SaveDialog1->FileName);
    //保存文件
```

```
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
if (Form1->FontDialog1->Execute())
      //打开一个对话框
      {
      Form1->Memo1->Font=Form1->FontDialog1->Font;
      //设置控件中文本显示字体
      }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
if (Form1->ColorDialog1->Execute())
    //打开一个对话框
    {
    Form1->Memo1->Color=Form1->ColorDialog1->Color;
    //设置控件背景色
    }
}
//---------------------------------------------------------------------------
```

2.5 日期时间组件

在这个时间日期动态查询程序中，用户可以查询从公元 1 年到公元 9999 年的任何一天是星期几，同时还可以动态显示系统当前的时间，其实利用 CBuilder 5 所提供的组件制作这个程序是一件轻而易举的事，它所利用到的组件有 CCalendar 组件、CSpinButton 组件、Button 组件、Edit 组件、Timer 组件和 Label 组件，程序运行时的显示情况如图 2-21 所示。

图 2-21　日期查询程序

在程序的窗体中的各个组件的作用如下所示：

- 按钮组件：完成查询的动作；
- 文本框组件：显示当前的日期和固定的提示文本；
- CCalendar 组件：充当程序的运算工具，程序运行时的数据运算都是在 Calendar 组件的内部完成的；

- CSpinButton 组件：用于微调相应的年、月、日；
- Label 组件：在程序运行的过程中用于动态显示系统当前时间的容器；
- Timer 组件：为程序能够实现动态的时间日期查询提供一个计时器。

好了，下面我们就开始制作这个日期查询程序吧，具体的程序设计过程如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Button 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮组件。
添加按钮组件后，把鼠标移动到组件工具栏上的其它选项上，向窗体上添加 1 个 CCalendar 组件、3 个 CSpinButton 组件、1 个 Timer 组件、1 个 Label 组件和 4 个 Edit 组件，各个组件的功能在前面已经介绍过了。添加组件后的窗体如图 2-22 所示。

图 2-22　添加组件后的窗体

用户可以通过打开相应的窗体文件查看这些属性的设置情况，如果用户自己手动的改变了窗体文件中的属性设置，改动的结果将会直接的反映到程序设计的窗体中。

2. 程序的初始化

在程序的设计过程中，用鼠标的左键双击窗体中的空白处，在屏幕上就会弹出一个代码窗口，光标的缺省位置是在窗体 FormCreate()事件的处理过程中，在光标的当前位置添加如下代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Edit1->Text="请输入年份";
Form1->Edit2->Text="请输入月份";
Form1->Edit3->Text="请输入日期";
//文本组件的初始化
Form1->CCalendar1->Year=1977;
Form1->CCalendar1->Month=1;
Form1->CCalendar1->Day=19;
//CCalendar 组件的初始化
Form1->CCalendar1->Enabled=false;
Form1->Edit1->Enabled=false;
Form1->Edit2->Enabled=false;
Form1->Edit3->Enabled=false;
Form1->CSpinButton1->Enabled=false;
Form1->CSpinButton2->Enabled=false;
Form1->CSpinButton3->Enabled=false;
Form1->Timer1->Enabled=false;
}
//---------------------------------------------------------------------------
```

在程序运行时，首先执行窗体 FormCreate()事件中的代码，通过前面三条语句可以完成对文

本框组件的初始化工作，设置第一个文本框显示“请输入年份”，设置第二个文本框显示“请输入月份”，设置第三个文本框显示“请输入日期”；在对 CCalendar 组件进行初始化的过程中，把当前的日期显示设置为 1977 年 1 月 19 日，最后通过一系列语句完成了对组件有效状态的设置。经过这样初始化的 CCalendar 组件如图 2-23 所示。

图 2-23　CCalendar 组件的初始化

3. 响应 CCalendar 事件

在程序设计的过程中用鼠标的左键单击 CCalendar 组件的事件列表框，激活事件列表框后，双击其中的 OnChange 事件，在屏幕上就会弹出一个 CCalendar 组件的 CCalendar1Change() 事件处理过程，在其中添加如下的响应代码：

```
void __fastcall TForm1::CCalendar1Change(TObject *Sender)
{
Form1->Edit1->Text=IntToStr(Form1->CCalendar1->Year);
Form1->Edit2->Text=IntToStr(Form1->CCalendar1->Month);
Form1->Edit3->Text=IntToStr(Form1->CCalendar1->Day);
}
//---------------------------------------------------------------------------
```

在程序运行过程中，当用户手动改变 CCalendar 组件的日期时，就会激活组件的 CCalendar1Change()事件，然后通过调用 Form1->Edit1->Text=IntToStr(Form1->CCalendar1->Year);等代码段使得文本框能够跟踪用户的选择，随时的显示当前选中的日期。

CCalendar 组件的常用属性如下：

表 2-1　CCalendar 组件的常用属性

Align	Anchors	BorderStyle	Color
Constraints	Ctl3D	Cursor	Day
DragCursor	DragKind	DragMode	Enabled
Font	GridLineWidth1	Height	HelpContext

续　表

Hint	Left	Month	Name
ParentColor	ParentFont	ParentShowHinTrue	PopupMenu
ReadOnly	ShowHint	StartOfWeek	TabOrder
TabStop	Tag	Top	UseCurrentDatTrue
Visible	Width	Year	

4. 响应文本框事件

在程序的运行过程中，当用户在文本框中输入一个有效的时间变量时（即在第一个文本框中输入值应该在 1—9999 之间，而第二个文本框中的输入值应该在 1—12 之间，在第三个文本框中的输入值应该在 1—31 之间），程序应该能够在 CCalendar 组件中判断出指定日期到底是星期几。

为了实现以上的功能，在程序的设计过程中用鼠标的左键双击窗体的文本框组件，在屏幕上就会弹出对应的代码窗口，在其中就可以添加响应文本框事件的代码，三个文本框中的代码如下所示：

```
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
Form1->CCalendar1->Year=StrToInt(Form1->Edit1->Text);
//年数改变
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit2Change(TObject *Sender)
{
Form1->CCalendar1->Month=StrToInt(Form1->Edit2->Text);
//月份改变
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit3Change(TObject *Sender)
{
Form1->CCalendar1->Day=StrToInt(Form1->Edit3->Text);
//日期改变
}
//---------------------------------------------------------------------------
```

5. 响应 CSpinButton 事件

窗体上 CSpinButton 组件的作用是在程序运行的过程中微调对应文本框中的数值，这里的一个难点就在于，文本框中的显示数值为 String；类型，而 CSpinButton 组件能够处理的变量类型为整型，所以在程序中调用了两个变量类型转换函数 StrToInt()和 IntToStr()，它们能够实现字符串类型变量和整型变量之间的转换。

在 CSpinButton 组件中有两个微调按钮，一个是带有向上箭头图案的按钮，另外一个是带有向下箭头图案的按钮，所以在添加代码时要对这两个微调按钮分别处理，三个 CSpinButton 组件的代码添加过程在这里就不多加叙述了，如果读者有不清楚的地方，请参看附后的源程序代码。

组件 CSpinButton1 的代码如下所示：

```
void __fastcall TForm1::CSpinButton1DownClick(TObject *Sender)
{
if (StrToInt(Form1->Edit1->Text)>1)
Form1->Edit1->Text=IntToStr(StrToInt(Form1->Edit1->Text)-1);
//年数减一
/*或者用下面的方法：
  Form1->CCalendar1->PrevYear;*/
}
//---------------------------------------------------------------------------
void __fastcall TForm1::CSpinButton1UpClick(TObject *Sender)
{
if (StrToInt(Form1->Edit1->Text)<9999)
Form1->Edit1->Text=IntToStr(StrToInt(Form1->Edit1->Text)+1);
//年数加一
/*或者用下面的方法：
  Form1->CCalendar1->NextYear;*/
}
//---------------------------------------------------------------------------
```

6. 响应按钮事件

窗体上按钮的功能是，在程序运行的过程中，当用户在三个文本框中输入要查询的日期时，单击按钮就可以立刻在组件 Calendar1 中显示出指定日期是星期几，能够实现以上功能的按钮响应代码如下所示：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->CCalendar1->Enabled=true;
Form1->Edit1->Enabled=true;
Form1->Edit2->Enabled=true;
Form1->Edit3->Enabled=true;
Form1->CSpinButton1->Enabled=true;
Form1->CSpinButton2->Enabled=true;
Form1->CSpinButton3->Enabled=true;
Form1->Timer1->Enabled=true;
//设置组件的有效状态
Form1->CCalendar1->Year=StrToInt(Form1->Edit1->Text);
Form1->CCalendar1->Month=StrToInt(Form1->Edit2->Text);
Form1->CCalendar1->Day=StrToInt(Form1->Edit3->Text);
//查询文本框中显示的日期
}
//---------------------------------------------------------------------------
```

7. 运行程序

按照附后的程序清单添加代码。做完以上各步的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，可以在文本框中输入一个日期，然后单击“执行查询”按钮，程序运行的结果如图 2-24 所示。

图 2-24　程序运行结果

我们可以在三个文本框中依次的输入如下日期——1978、10、20，查询的结果表明 1978 年 10 月 20 日为星期五。同时我们还可以发现，在窗体的标签组件中会动态显示系统当前的时间。

附程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Week.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CCALENDR"
```

```
#pragma link "CSPIN"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Edit1->Text="请输入年份";
Form1->Edit2->Text="请输入月份";
Form1->Edit3->Text="请输入日期";
//文本组件的初始化
Form1->CCalendar1->Year=1977;
Form1->CCalendar1->Month=1;
Form1->CCalendar1->Day=19;
//CCalendar 组件的初始化
Form1->CCalendar1->Enabled=false;
Form1->Edit1->Enabled=false;
Form1->Edit2->Enabled=false;
Form1->Edit3->Enabled=false;
Form1->CSpinButton1->Enabled=false;
Form1->CSpinButton2->Enabled=false;
Form1->CSpinButton3->Enabled=false;
Form1->Timer1->Enabled=false;
//设置组件的有效状态
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->CCalendar1->Enabled=true;
Form1->Edit1->Enabled=true;
Form1->Edit2->Enabled=true;
Form1->Edit3->Enabled=true;
Form1->CSpinButton1->Enabled=true;
Form1->CSpinButton2->Enabled=true;
Form1->CSpinButton3->Enabled=true;
Form1->Timer1->Enabled=true;
//设置组件的有效状态
Form1->CCalendar1->Year=StrToInt(Form1->Edit1->Text);
Form1->CCalendar1->Month=StrToInt(Form1->Edit2->Text);
Form1->CCalendar1->Day=StrToInt(Form1->Edit3->Text);
//查询文本框中显示的日期
}
//---------------------------------------------------------------------------
void __fastcall TForm1::CCalendar1Change(TObject *Sender)
{
Form1->Edit1->Text=IntToStr(Form1->CCalendar1->Year);
Form1->Edit2->Text=IntToStr(Form1->CCalendar1->Month);
Form1->Edit3->Text=IntToStr(Form1->CCalendar1->Day);
//在文本框中显示日期信息
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CSpinButton1DownClick(TObject *Sender)
```

```
{
if (StrToInt(Form1->Edit1->Text)>1)
Form1->Edit1->Text=IntToStr(StrToInt(Form1->Edit1->Text)-1);
//年数减一
/*或者用下面的方法：
  Form1->CCalendar1->PrevYear;*/
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CSpinButton1UpClick(TObject *Sender)
{
if (StrToInt(Form1->Edit1->Text)<9999)
Form1->Edit1->Text=IntToStr(StrToInt(Form1->Edit1->Text)+1);
//年数加一
/*或者用下面的方法：
  Form1->CCalendar1->NextYear;*/
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CSpinButton2DownClick(TObject *Sender)
{
if (StrToInt(Form1->Edit2->Text)>1)
Form1->Edit2->Text=IntToStr(StrToInt(Form1->Edit2->Text)-1);
//月份减一
/*或者用下面的方法
  Form1->CCalendar1->PrevMonth;*/
}
//---------------------------------------------------------------------------
void __fastcall TForm1::CSpinButton2UpClick(TObject *Sender)
{
if (StrToInt(Form1->Edit2->Text)<12)
Form1->Edit2->Text=IntToStr(StrToInt(Form1->Edit2->Text)+1);
//月份加一
/*或者用下面的方法
  Form1->CCalendar1->NextMonth;*/
}
//---------------------------------------------------------------------------
void __fastcall TForm1::CSpinButton3DownClick(TObject *Sender)
{
if (StrToInt(Form1->Edit3->Text)>1)
Form1->Edit3->Text=IntToStr(StrToInt(Form1->Edit3->Text)-1);
//日期减一
}
//---------------------------------------------------------------------------
void __fastcall TForm1::CSpinButton3UpClick(TObject *Sender)
{
if (StrToInt(Form1->Edit3->Text)<31)
Form1->Edit3->Text=IntToStr(StrToInt(Form1->Edit3->Text)+1);
//日期加一
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->Label1->Caption=TimeToStr(Time());
//动态显示系统时间
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit1Change(TObject *Sender)
```

```
{
Form1->CCalendar1->Year=StrToInt(Form1->Edit1->Text);
//年数改变
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit2Change(TObject *Sender)
{
Form1->CCalendar1->Month=StrToInt(Form1->Edit2->Text);
//月份改变
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit3Change(TObject *Sender)
{
Form1->CCalendar1->Day=StrToInt(Form1->Edit3->Text);
//日期改变
}
//---------------------------------------------------------------------------
```

2.6 状态条组件

在编程过程中，我们经常需要知道当前光标的位置，特别是大的应用程序，光标位置的定位就显得尤其重要。另外，在工作的过程中，怎样才能够随时的获得当前的时间呢，在 Windows 操作系统中，以上两个问题通过状态条都可以轻而易举的完成，一个典型的状态条如图 2-25 所示。

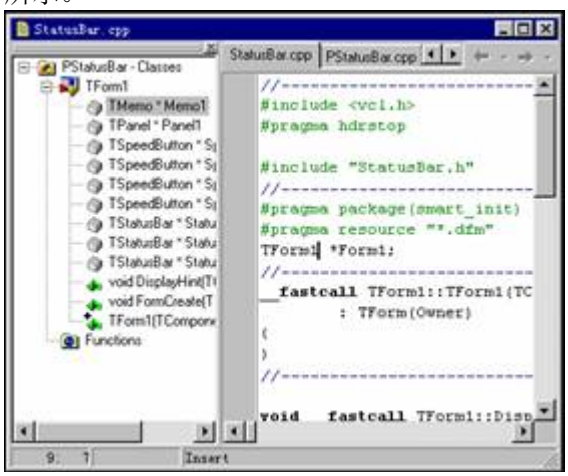


图 2-25　典型的系统状态条

在如图 2-25 所示的状态条中，我们可以直观的得到如下的信息：当前的光标位于代码窗口的第 9 行第 7 列，当前的键盘输入状态为 Insert 状态等。下面就以一个制作状态条应用程序的实例来说明 CBuilder 5 中的 Panel 组件、SpeedButton 组件和 StatusBar 组件的使用方法，具体的程序制作步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Memo 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 组件。添加 Memo 组件后，把鼠标移动到组件工具栏上的其它选项上，向窗体上添加 1 个 Panel 组件、4 个 SpeedButton 组件、3 个 StatusBar 组件，各个组件的功能在后面将予以详细的介绍。
添加组件后的窗体如图 2-26 所示。

图 2-26　添加组件后的窗体
其中各个组件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 434
  Height = 336
  Caption = 'Form1'
  OnCreate = FormCreate
  object Panel1: TPanel
    Left = 0
    Top = 0
    Width = 426
    Height = 49
    Align = alTop
    object SpeedButton1: TSpeedButton
      Left = 8
      Top = 8
      Width = 33
      Height = 33
      Hint = '新建文件'
      ParentShowHint = False
      ShowHint = True
    end
    object SpeedButton2: TSpeedButton
      Left = 48
      Top = 8
      Width = 33
      Height = 33
      Hint = '打开文件'
      ParentShowHint = False
      ShowHint = True
    end
    object SpeedButton3: TSpeedButton
      Left = 88
      Top = 8
```

```
      Width = 33
      Height = 33
      Hint = '存储文件'
      ParentShowHint = False
      ShowHint = True
    end
    object SpeedButton4: TSpeedButton
      Left = 128
      Top = 8
      Width = 33
      Height = 33
      Hint = '打印文件'
      ParentShowHint = False
      ShowHint = True
    end
  end
  object Memo1: TMemo
    Left = 0
    Top = 56
    Width = 425
    Height = 217
    Hint = '文档窗口'
    ParentShowHint = False
    ShowHint = True
  end
  object StatusBar1: TStatusBar
    Left = 0
    Top = 280
    Width = 145
    Height = 28
    Align = alNone
    Panels = <>
    SimplePanel = True
  end
  object StatusBar2: TStatusBar
    Left = 136
    Top = 280
    Width = 145
    Height = 28
    Align = alNone
    Panels = <>
    SimplePanel = True
  end
  object StatusBar3: TStatusBar
    Left = 280
    Top = 280
    Width = 145
    Height = 28
    Align = alNone
    Panels = <>
    SimplePanel = True
  end
end
```

在本程序中用到了 1 个 Panel 组件、3 个 SpeedButton 组件、3 个 StatusBar 组件和 1 个 Memo 组件，它们的作用如下所示：

- Panel 组件：用做三个加速按钮(SpeedButton 组件)的容器，使得三个加速按钮组成一个集合；
- SpeedButton 组件：为状态条显示提示文本提供信息；
- StatusBar 组件：显示应用程序中的提示文本和当前系统的时间和日期；

- Memo 组件：作用与 SpeedButton 组件相同。

2. 创建一个新的过程

由于在本程序中要在状态条中显示应用程序中各个组件的提示信息，但是在系统并没有提供现成的过程，所以在程序的设计过程中我们要自己创建一个新的过程，在键盘上按下功能键 Alt+F12，在屏幕上就会弹出一个如图 2-27 所示的对话框。

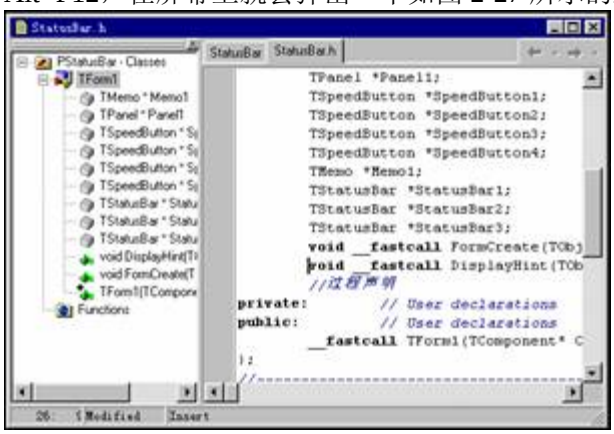


图 2-27　弹出的代码窗口

把光标移动到声明编辑代码中，并且添加如下所示的代码：

```
//---------------------------------------------------------------------------
#ifndef StatusBarH
#define StatusBarH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:   // IDE-managed Components
        TPanel *Panel1;
        TSpeedButton *SpeedButton1;
        TSpeedButton *SpeedButton2;
        TSpeedButton *SpeedButton3;
        TSpeedButton *SpeedButton4;
        TMemo *Memo1;
        TStatusBar *StatusBar1;
        TStatusBar *StatusBar2;
        TStatusBar *StatusBar3;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall DisplayHint(TObject *Sender);
        //过程声明
private:   // User declarations
public:    // User declarations
        __fastcall TForm1(TComponent* Owner);
};
```

```
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

在声明段中 void __fastcall DisplayHint(TObject *Sender);即为刚刚声明的新过程。

3. 为新建过程添加代码

在前一步中我们声明了一个新的过程，接下来就应该为这个新建的过程添加程序处理代码，在代码窗口的程序段中自己用键盘输入如下的过程处理代码：

```
void __fastcall TForm1::DisplayHint(TObject *Sender)
{
  StatusBar1->SimpleText = GetLongHint(Application->Hint);
  //显示提示文本
  Form1->StatusBar2->SimpleText=TimeToStr(Time());
  //显示当前的时间
  Form1->StatusBar3->SimpleText=DateToStr(Date());
  //显示当前的日期
}
//---------------------------------------------------------------------------
```

值得注意的是，以上这段代码的框架无法通过 CBuilder 5 生成，只能够自己输入，第一条语句：StatusBar1->SimpleText = GetLongHint(Application->Hint);用于在第一个状态条中显示普通的提示文本，而通过第三条和第五条语句则可以在相应的状态条中显示系统当前的时间和日期。

以上程序段的难点就在于，通过 Time 和 Date 所取得的系统信息（时间和日期）的变量类型与 StatusBar 组件中 SimpleText:属性的类型不一致，这就需要调用两个类型转化函数 TimeToStr()和 DateToStr()来完成。

4. 调用新建过程

以上已经完成了新建过程的各种处理工作，接下来就要在程序中来调用它，在代码窗口中将光标的位置定位在 FormCreate()过程的事件处理代码中，并且添加如下所示的新建过程调用代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Application->OnHint = DisplayHint;
}
//---------------------------------------------------------------------------
```

5. 运行程序

做完以上各步的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行结果如图 2-28 所示。

图 2-28　程序运行结果

在程序的运行过程中，把鼠标移动到不同的组件上，在状态条上就会显示出相应的信息，同时在窗体底部其余的两个状态条中将会随时的显示系统当前的时间和日期。
程序的完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "StatusBar.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::DisplayHint(TObject *Sender)
{
  StatusBar1->SimpleText = GetLongHint(Application->Hint);
  //显示提示文本
  Form1->StatusBar2->SimpleText=TimeToStr(Time());
  //显示当前的时间
  Form1->StatusBar3->SimpleText=DateToStr(Date());
  //显示当前的日期
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Application->OnHint = DisplayHint;
}
//---------------------------------------------------------------------------
```

2.7 小　结

在本章中，通过几个示例程序的具体制作，向用户说明了在 CBuilder 5 中几个基本组件——Button 组件、Bitbtn 组件、Memo 组件、CheckBox 组件、RadioButton 组件、Panel 组件、ScrollBar 组件、Calendar 组件、ListView 组件和 ColorGrid 组件的使用方法，熟练的运用这些组件，是设计一个好的应用程序的基础，希望读者能够仔细的体会本章示例程序中对基本组件的处理过程，并且为以后的学习打下良好的基础。

第三章　界面交互功能的使用

对话框和菜单是可视化编程的特色之一，从某种意义上讲，也是衡量应用程序功能的一个方面，合理的利用菜单和对话框等 Windows 界面，可以大大的提高应用程序的人机交互功能，增加用户与程序之间的交流机会。
在本章中，我们仍然以几个比较常用的示例，将对对话框和菜单这两种人机交互界面进行比较详细的讲解。

3.1 对话框设计和使用

对话框是 Windows 操作系统中程序与用户沟通的一种常见交互方式，对话框可以向用户提供当前程序的运行状况，也可以接受用户输入的信息，在 CBuilder 中，对话框的程序设计又有两种方式——对话框函数和标准标准对话框，下面将分别的予以介绍和说明。

3.1.1 对话框函数简介

在 CBuilder 中的对话框函数大体上可以分为两种——输入对话框函数和输出对话框函数，输入对话框函数用于接收用户在程序运行过程中输入的信息，其中包括 ShowMessage()函数、MessageDlg()函数，而输出对话框函数则用于显示一个对话框窗体，向用户报告当前程序的运行状态等信息，它包括 InputBox()函数下面就对各个函数分别的加以介绍。

1. ShowMessage()函数

对话框函数中的 ShowMessage()函数用于在程序运行的过程中显示一个包含一个字符串信息的对话框，用户只有对这个对话框进行正确的响应之后，才能够关闭对话框，进行下一步的工作。
ShowMessage()函数的语法结构如下所示：
ShowMessage(const Msg: string);
下面的示例程序就是用于在程序运行中产生一个对话框，在其中显示一个字符串，具体的步骤如下所示：

2. 开始工作

首先其中一个新的项目，在屏幕上就会弹出一个空白的窗体，向窗体上添加一个按钮控件，控件的标题设置为“Hello”;

3. 添加代码

在程序的设计过程中用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口，在其中添加如下所示的按钮响应代码：

```
char *string;
//定义字符串变量
void __fastcall TForm1::Button1Click(TObject *Sender)
{
str="欢迎来到 C++ Builder 世界,在这里将向您介绍 showmessage 函数的基本用法";
//为字符串变量赋值
ShowMessage(str);
//显示对话框
}
```

4. 运行程序

完成以上的工作后，选择菜单“File”中的“Save All”选项，在弹出的窗口中选择一个合适的文件名存储文件，然后按键盘上的功能键 F9 运行程序。
在程序的运行过程中，用鼠标的左键单击窗体中按钮“Hello”，在窗体上就会弹出一个对话框，如图 3-1 所示。

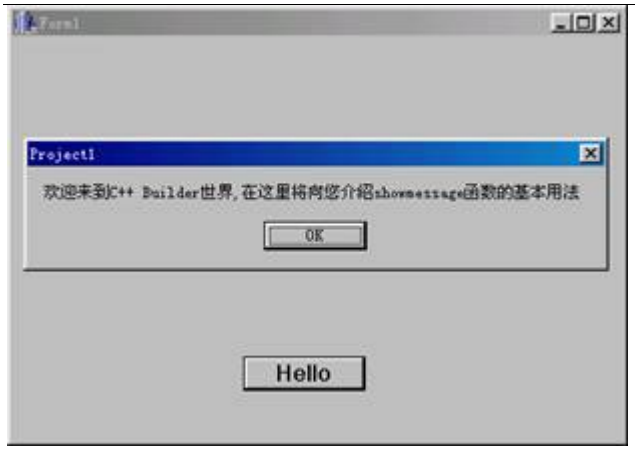


图 3-1　ShowMessage()函数输出对话框

5. MessageDlg()函数

对话框函数中的 MessageDlg()函数用于在程序运行的过程中显示一个包含一个字符串、位图和按钮信息的对话框，用户只有对这个对话框进行正确的响应之后，才能够关闭对话框，进行下一步的工作。

MessageDlg()函数的语法结构如下所示：

function MessageDlg(const Msg: string; AType: TMsgDlgType;AButtons: TMsgDlgButtons; HelpCtx: Longint): Word;

下面的示例程序就是用于在程序运行中产生一个对话框，在其中显示一个字符串，具体的步骤如下所示：

6. 开始工作

首先其中一个新的项目，在屏幕上就会弹出一个空白的窗体，向窗体上添加一个按钮控件，控件的标题设置为“Welcome”。

7. 添加代码

在程序的设计过程中用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口，在其中添加如下所示的按钮响应代码：

```
char *string;
//定义字符串变量

void __fastcall TForm1::Button1Click(TObject *Sender)
{
str="欢迎来到 C++ Builder 世界,在这里将向您介绍 showmessage 函数的基本用法";
//为字符串变量赋值
MessageDlg(str, mtConfirmation, TMsgDlgButtons()<<mbYes<<mbNo,0);
//显示对话框
}
```

8. 运行程序

在程序的运行过程中，用鼠标的左键单击窗体中按钮“Welcome”，在窗体上就会弹出一个对话框，如图 3-2 所示。

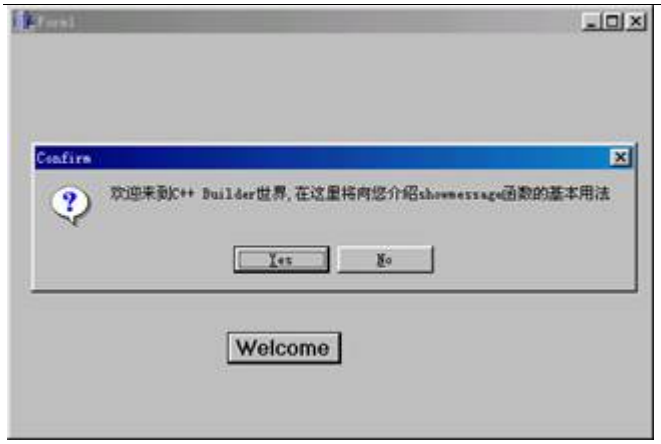


图 3-2　MessageDlg()函数输出对话框

9. InputBox()函数

对话框函数中的 InputBox()函数用于在程序运行的过程中显示一个包含一个字符串和按钮信息的输入对话框，用户只有对这个对话框进行正确的响应之后，才能够关闭对话框，进行下一步的工作。
它的语法结构如下所示：
function InputBox(const ACaption, APrompt, ADefault: string): string;
下面以一个示例来说明 InputBox()函数的用法，具体的设计步骤如下：

10. 开始工作

在程序的设计阶段，向空白的窗体上放置一个按钮控件、一个标签和一个输入文本框，如图 3-3 所示。

图 3-3　输入对话框
控件的作用如下：

- 按钮空间 Button1：用来显示一个数字输入框；
- 文本框空间 Edit1：用来显示信息输入框中输入数字的平方值。

11. 添加代码

并且在按钮的响应事件中添加如下代码：

```
AnsiString str;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
str=InputBox("平方计算器","请输入原始数据：","");
//显示一个输入对话框
Edit1->Text=FloatToStr(StrToFloat(str)*StrToFloat(str));
//显示输入数据的平方值
}
```

12. 运行程序

存储文件，运行程序，如图 3-4（左）所示。然后单击“数据数据”按钮上，在窗体的上方就会出现一个如图 3-4（右）所示的输入对话框。

图 3-4　输入对话框

在输入对话框中输入一个正整数 122，单击“OK”按钮，在程序运行的文本框中就会显示出 122 的平方值，结果如图 3-5 所示。

图 3-5　程序运行结果

3.1.2　标准对话框控件的应用

在 CBuilder 中为了方便用户的开发工作，提供了 10 种标准的对话框控件——打开文件对话框、存储文件对话框、打开图片对话框、存储文件对话框、打印对话框、打印设置对话框、字体设置对话框、颜色设置对话框、查找对话框和替换对话框。

以上这十种标准对话框的属性和用法在这里就不多加叙述了，下面通过一个示例程序来说明打开文件对话框、字体设置对话框和颜色设置对话框的使用方法和各自所能够实现的功能。

注意：

✍　在下面的这个示例程序中，用户在程序的运行过程中，可以有选择的打开一个以*.txt 为文件后缀的文本文件，并且在程序的运行过程中，还可以改变文本的显示形式，如字体和颜色的设置等。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Memo 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 组件。

接着向窗体上添加 4 个 Button 组件、一个 OpenDialog 组件、一个 SaveDialog 组件、一个 FontDialog 组件和一个 ColorDialog，添加组件后的窗体如图 3-6 所示。

图 3-6　添加组件后的窗体

窗体中各个组件的属性设置如下所示：

```
object Form1: TForm1
   Left = 191
   Top = 107
   Width = 544
   Height = 375
   Caption = 'Form1'
   object Memo1: TMemo
      Left = 40
      Top = 96
      Width = 337
      Height = 201
      Lines.Strings = ( 'Memo1')
      ScrollBars = ssBoth
   end
   object Button1: TButton
      Left = 392
      Top = 96
      Width = 97
      Height = 33
      Caption = '打开文件'
   end
   object Button2: TButton
      Left = 392
      Top = 152
      Width = 97
      Height = 33
      Caption = '存储文件'
   end
   object Button3: TButton
      Left = 392
      Top = 208
      Width = 97
      Height = 33
      Caption = '字体设置'
   end
   object Button4: TButton
      Left = 392
      Top = 264
      Width = 97
      Height = 33
      Caption = '颜色设置'
   end
   object OpenDialog1: TOpenDialog
      Left = 144
      Top = 56
```

```
  end
  object SaveDialog1: TSaveDialog
    Left = 112
    Top = 56
  end
  object FontDialog1: TFontDialog
    Left = 176
    Top = 56
  end
  object ColorDialog1: TColorDialog
    Left = 208
    Top = 56
  end
end
```

2. 程序的初始化

程序的初始化过程，也就是对窗体 FormCreate()事件的初始化。
在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Memo1->Clear();
//清空文本框
Form1->OpenDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->OpenDialog1->InitialDir="c:\\Windows";
//设置缺省文件目录
Form1->OpenDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
//设置文件过滤条件
Form1->OpenDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->SaveDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->SaveDialog1->InitialDir="c:\\Windows";
//设置缺省文件目录
Form1->SaveDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
//设置文件过滤条件
Form1->SaveDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->Button2->Enabled=false;
//设置按钮有效状态
Form1->FontDialog1->Font=Form1->Memo1->Font;
Form1->ColorDialog1->Color=Form1->Memo1->Color;
//对话框的初始化
}
//---------------------------------------------------------------------------
```

程序说明：
在程序运行的初期，程序首先清空文本框中的文本显示内容，这为用户输入文字和打开文件作好了准备工作，由于程序刚刚开始运行时，还没有打开文件，所以“存储文件”按钮应该处于无效的状态，这是通过语句 Form1->Button2->Enabled=false;来实现的。
在程序的初始化过程中，还包括对 OpenDialog 组件的初始化，通过语句一条 Form1->OpenDialog1->InitialDir:='c:\\windows';来设置组件所显示对话框的缺省路径为 c:\windows，同时设置的文件显示过滤器为'All Files(*.*)|*.*|Text Files(*.txt)|*.txt'，这也就意味着，在程序运行的过程中，“打开”对话框显示的文件只有两种选择——要么显示所有的

文件，要么只显示以*.txt 为后缀的文本文件。与 OpenDialog 组件的初始化相同，SaveDialog 组件的初始化过程也是由两条语句：

form1->SaveDialog1->InitialDir:='c:\\windows';

form1->SaveDialog1->Filter:='All Files(*.*)|*.*|Text Files(*.txt)|*.txt';

它们的功能在 OpenDialog 组件的初始化过程中已经讲述过了。

在窗体 FormCreate()事件的最后通过两条语句

Form1->FontDialog1->Font=Form1->Memo1->Font;

Form1->ColorDialog1->Color=Form1->Memo1->Color;

完成对“字体”对话框和“颜色”对话框的初始化，把它们的初始状态设置与 Memo 组件相一致。

至此，程序的初始化工作就完成了。

3. 响应“打开文件”按钮

在程序的设计阶段，用鼠标的左键双击窗体上的“打开文件”按钮，在弹出的如图 3-7 所示代码窗口的左侧子窗口中选择 Tform1/Published/Button1Click 选项，在代码窗口的右侧子窗口中就会显示出组件“打开文件”的代码响应过程。

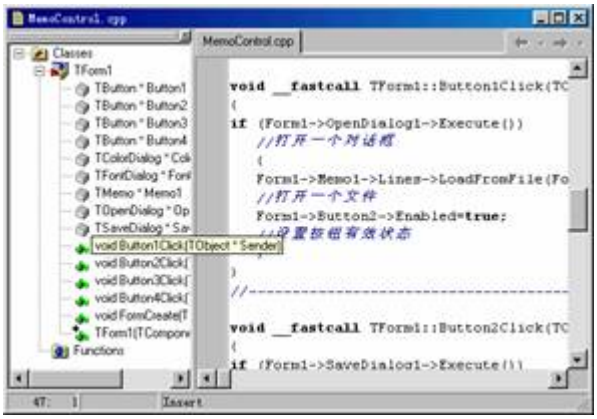


图 3-7　弹出的代码窗口

在其中添加如下所示的按钮响应代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //打开一个对话框
    {
    Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
    //打开一个文件
    Form1->Button2->Enabled=true;
    //设置按钮有效状态
    }
}
//---------------------------------------------------------------------
```

在所添加的代码段中，特别值得注意的是语句 Form1->OpenDialog1->Execute;和语句 Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->Filename);，其中前者的作用是显示一个“打开文件”对话框，用户可以在其中选择一个文本文件，后者的作用是打开用户在“打开文件”对话框中所选择的文件。

当用户打开文本文件后，窗体中的“存储文件”按钮就应该变为有效的状态，这一功能是通过语句 Form1->Button2->Enabled=true;来实现的。典型的“打开文”话框如图 3-8 所示。

图 3-8 “打开”对话框

相应的，在“存储文件”按钮的事件响应中添加的代码如下所示：

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (Form1->SaveDialog1->Execute())
    //显示一个对话框
    {
    Form1->Memo1->Lines->SaveToFile(Form1->SaveDialog1->FileName);
    //保存文件
    }
}
```

4. 响应属性设置按钮事件

在窗体中放置有一个 FontDialog 组件和一个 ColorDialog 组件，它们的作用是分别为文本框组件设置显示字体和背景色。下面以 FontDialog 组件为例来说明在程序中如何添加响应属性设置按钮事件的代码，具体的设计步骤如下：

在程序的设计阶段，用鼠标的左键双击组件 FontDialog1，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中，并且添加如下所示的按钮事件响应代码：

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
if (Form1->FontDialog1->Execute())
     //打开一个对话框
     {
     Form1->Memo1->Font=Form1->FontDialog1->Font;
     //设置控件中文本显示字体
     }
}
```

在 FontDialog 组件的响应代码中，语句 Form1->FontDialog1->Execute;的作用是显示一个对话框，用户可以在其中为 Memo 组件中的文本设置字体、大小、风格以及颜色等，而 Form1->Memo1->Font=Form1->FontDialog1->Font;语句的作用是将用户在“字体”对话框中的选择传送给 Memo 组件。一个典型的字体设置对话框如图 3-9 所示。

图 3-9 “字体”对话框

5. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的初始画面如图 3-10 所示。

图 3-10　程序运行的初始画面

首先用鼠标的左键单击按钮“打开文件”，在屏幕上就会弹出一个对话框，提示用户选择一个有效的文本文件，单击“打开”按钮，在 Memo 组件中就会显示出文本文件中的内容，如图 3-11 所示。

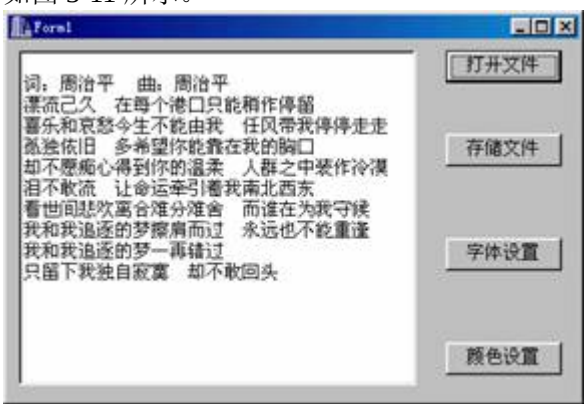


图 3-11　一个打开的文本文件

在程序运行的过程中，用户可以通过单击“颜色设置”按钮和“字体设置”按钮来改变系统的设置，如图 3-12 所示为打开的“颜色”对话框，您可以十分方便的为文本设置系统颜色。

图 3-12　设置颜色

如图 3-13 所示，为改变了字体和颜色后的效果（注明：字体为小 6 号，颜色为淡蓝）。

图 3-13　字体和颜色效果

编辑文本后，单击“存储文件”按钮，在窗体上就会弹出一个“另存为”对话框。在“另存

为…”对话框中选择一个有效的路径和文件名，单击“保存”按钮就完成了文本文件的保存工作。

附程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "MemoControl.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Memo1->Clear();
//清空文本框
Form1->OpenDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->OpenDialog1->InitialDir="c:\\PWin98";
//设置缺省文件目录
Form1->OpenDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
//设置文件过滤条件
Form1->OpenDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->SaveDialog1->Title="请选择一个文本文件：";
//设置对话框标题
Form1->SaveDialog1->InitialDir="c:\\PWin98";
//设置缺省文件目录
Form1->SaveDialog1->Filter="All Files(*.*)|*.*|Text Files(*.txt)|*.txt";
//设置文件过滤条件
Form1->SaveDialog1->DefaultExt=String("TXT");
//设置缺省扩展名
Form1->Button2->Enabled=false;
//设置按钮有效状态
Form1->FontDialog1->Font=Form1->Memo1->Font;
Form1->ColorDialog1->Color=Form1->Memo1->Color;
//对话框的初始化
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //打开一个对话框
    {
    Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
    //打开一个文件
    Form1->Button2->Enabled=true;
    //设置按钮有效状态
```

```
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (Form1->SaveDialog1->Execute())
    //显示一个对话框
    {
    Form1->Memo1->Lines->SaveToFile(Form1->SaveDialog1->FileName);
    //保存文件
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
if (Form1->FontDialog1->Execute())
      //打开一个对话框
      {
      Form1->Memo1->Font=Form1->FontDialog1->Font;
      //设置控件中文本显示字体
      }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
if (Form1->ColorDialog1->Execute())
    //打开一个对话框
    {
    Form1->Memo1->Color=Form1->ColorDialog1->Color;
    //设置控件背景色
    }
}
//---------------------------------------------------------------------
```

3.2 菜单设计和使用

在 Windows 操作系统（无论是在 Windows 98/2000，还是在 Windows NT）中，大量的采用了菜单技术。

一个典型的菜单就是 Windows 资源管理器中的系统菜单和弹出式菜单，如图 3-14 所示，用户在其中可以方便的使用各种选项进行操作。

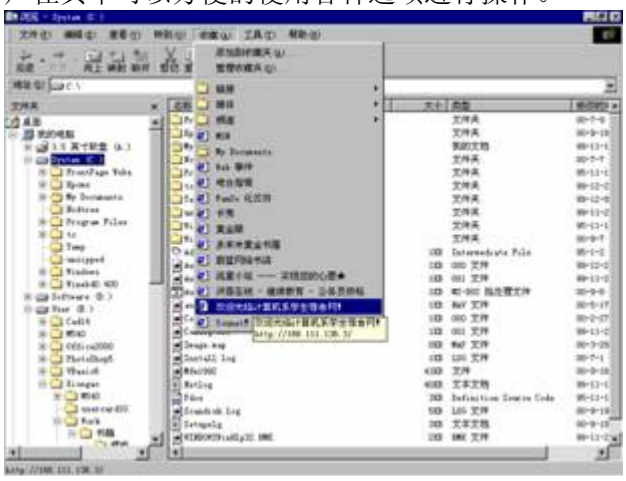


图 3-14　资源管理器中的菜单

在 CBuilder 中有两种菜单设计器——MainMenu Designer(主菜单设计器)和 PopMenu Designer(弹出菜单设计器)，它们都是不可见控件。

注意：

✍　所谓的不可见，指的是在程序运行的过程中不可见，并不是程序设计过程中的不可见。

下面就来分别的讲述一下如何利用主菜单设计器和弹出菜单设计器设计系统菜单和弹出式菜单。

3.2.1 菜单及其概述

主菜单设计器位于 CBuilder 控件工具栏中 Standard 选项下的第二项，它的主要功能是设计系统菜单，或者叫做下拉式菜单。

1. 创建菜单项

下面将会通过一个示例来说明如何在 CBuilder 中创建一个菜单项，在这个菜单项中，有一个主菜单，而这个主菜单又由五个子菜单项组成，但是它们目前还不能够实现任何功能，具体的步骤如下：

2. 添加控件

按照前面的方法新建立一个程序项目（Application Project），首先从控件工具栏上选择 MainMenu（主菜单设计器），然后把它添加到空白的窗体上，在实际的操作过程中，可以通过两种方法向窗体上添加“主菜单设计器”：

- 在控件工具栏上单击鼠标的左键，然后把鼠标移动到空白的窗体上，按下的同时移动鼠标，在窗体上就会绘制出一个“主菜单设计器”；
- 在控件工具栏上用鼠标的左键双击“主菜单设计器”的图标，系统就会自动的在空白的窗体上放置一个“主菜单设计器”，但是这样做通常需要用户手动的调节控件的位置。

如图 3-15 所示即为放置了一个“主菜单设计器”的窗体。

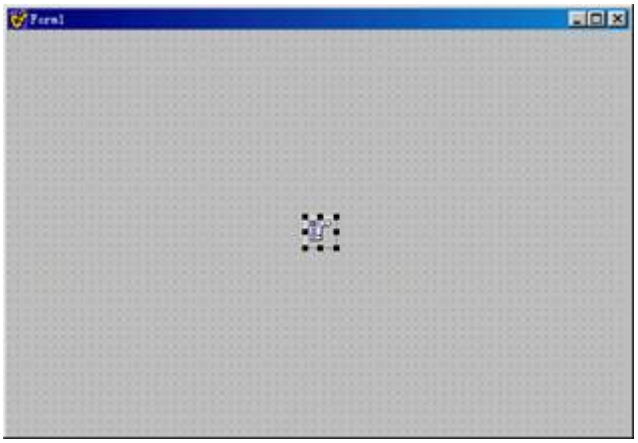


图 3-15 放置了“主菜单设计器“的窗体

虽然在窗体上放置了“主菜单设计器”，但是只有激活了“主菜单设计器”才能够对主菜单进行设计等操作，激活“主菜单设计器”的方法也有两种：

- 用鼠标的左键在“主菜单设计器”上双击，在屏幕上就会弹出一个空白的主菜单设计窗口；
- 在控件的属性列表中用鼠标的左键单击 Items 属性右侧属性输入框中的…按钮也可以打开主菜单设计器。

3. 添加菜单项

在刚添加的“主菜单设计器”上双击鼠标的左键，就会弹出一个如图 3-16 所示的主菜单设计窗口。

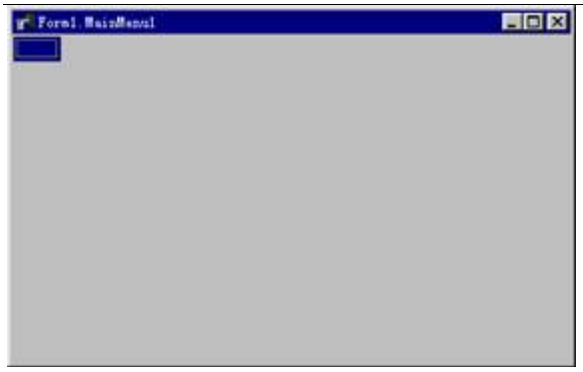


图 3-16　打开的主菜单设计器

在主菜单设计器中对主菜单及其子菜单项的属性设置如下所示，读者可以打开项目文件所对应的窗体文件（以*.dfm 为文件后缀）来观察程序中对窗体及各个控件的属性设置，由于“主菜单设计器”也属于控件类，所以相应菜单项的属性设置也可以在窗体文件中观察到。

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 544
   Height = 375
   Caption = 'Form1'
   Menu = MainMenu1
   object MainMenu1: TMainMenu
      Left = 464
      Top = 264
      object File1: TMenuItem
         Caption = 'File'
         object New1: TMenuItem
            Caption = 'New'
         end
         object Open1: TMenuItem
            Caption = 'Open'
         end
         object Close1: TMenuItem
            Caption = 'Close'
         end
         object Save1: TMenuItem
            Caption = 'Save'
         end
         object Exit1: TMenuItem
            Caption = 'Exit'
         end
      end
   end
end
```

经过以上设置的菜单如图 3-17 所示。

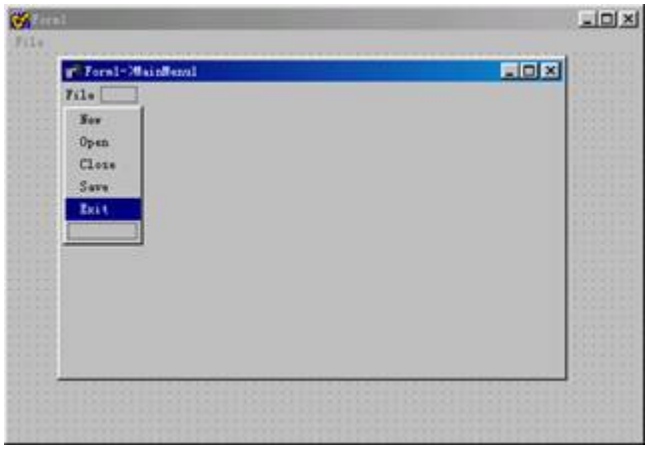


图 3-17　设置属性值后的主菜单

经过这样设置的“主菜单设计器”拥有一个主菜单项“File”，在这个菜单项下有 5 个子菜单项——“New、Open、Close、Save 和 Exit”。

4. 修改菜单项

值得注意的是，以上的设计菜单的方法只是一种通用的方法，我们也可以通过在程序设计过程中修改项目文件的窗体文件来达到设计菜单的目的，如上例所示，在主菜单“File”下有 5 个子菜单项，我们可以首先打开程序的窗体文件，然后在子菜单项 Save 和 Exit 之间添加如下内容：

```
object Save2: TMenuItem
Caption = 'Save as'
end
object About1: TMenuItem
Caption = 'About'
end
```

保存窗体文件，重新打开项目文件，在程序的设计阶段用鼠标的左键单击菜单项“File”，结果如图 3-18 所示。

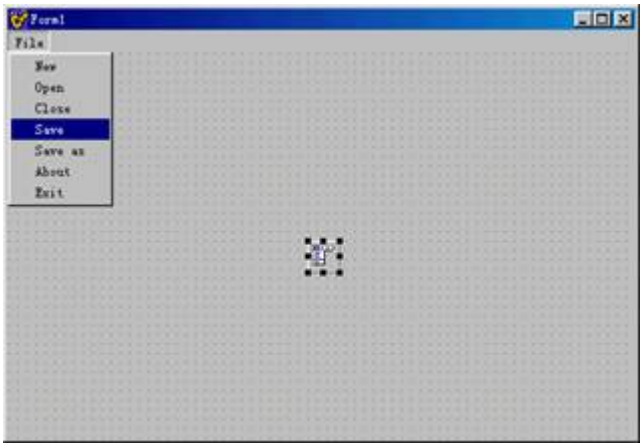


图 3-18　修改后的菜单项

5. 设置分隔条

在菜单系统的设计中，经常可以看到所谓的将菜单项分组的技术，即在同一个菜单项下将一个几个子菜单项组合成一个集合，从而用分隔条相互的隔离开来，如图 3-19 所示即为一个典型的分隔条设置示例。

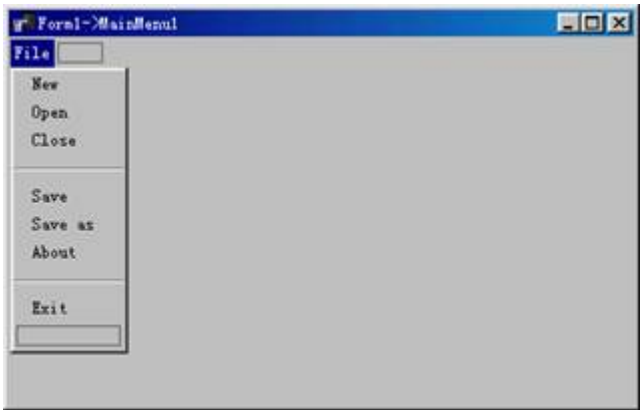


图 3-19　菜单项中的分隔条

其实分隔条的设计也是相当简单的，只要在菜单项的 Caption 属性中设置为“-”即可，如图 3-19 所示的菜单项分隔条属性设置如下：

```
object New1: TMenuItem
Caption = 'New'
end
object Open1: TMenuItem
Caption = 'Open'
end
object Close1: TMenuItem
Caption = 'Close'
end
object N1: TMenuItem
Caption = '-'
End                    //设置分割条
object Save1: TMenuItem
Caption = 'Save'
end
object Save2: TMenuItem
Caption = 'Save as'
end
object About1: TMenuItem
Caption = 'About'
end
object N2: TMenuItem
Caption = '-'
End                    //设置分割条
object Exit1: TMenuItem
Caption = 'Exit'
end
```

6. 设置快捷键

一般来说，常用的菜单项都有自己的快捷键（加速键或者热键），为菜单项设置一个合适的快捷键，可以提高用户访问命令和使用菜单项的效率，如图 3-20 所示为 Word 中“编辑”菜单，基本上每一个菜单项都有自己的快捷键方式。

图 3-20　有快捷键的菜单项

在 CBuilder 中可以为每一个子菜单项设置快捷键：

在 CBuilder 中，通过设置子菜单项的 Caption 属性可以为子菜单项设置加速键，在设置 Caption 属性时，在需要设置为加速键的字母前面加一个“&”符号，这样菜单项中的该字母就会自动的加上一个下划线，在程序运行过程中用户按下 Alt 键和该字母的组合键就可以实现选中子菜单项的功能。

如图 3-21 所示即为设置了加速键的子菜单项——New，在程序运行的过程中，用户随时的按下 Alt+N 键就可以访问子菜单项 New。

图 3-21　设置加速键

当为子菜单项设置热键时，操作方法与以上略有不同。

首先用鼠标选中想要设置热键的子菜单项，然后用鼠标的左键单击这个菜单项，返回到子菜单项的属性列表中，单击属性列表中 ShortCut 属性输入框右侧的下拉按钮，在弹出的热键列表中选择一个合适的热键即可。

如图 3-22 所示为设置了加速键和热键的菜单效果。

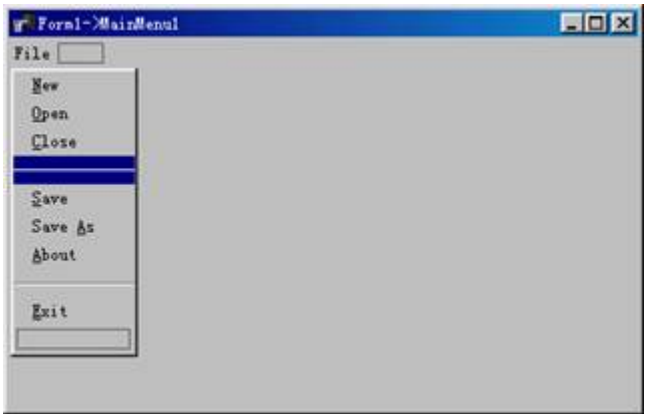


图 3-22　设置了加速键和热键的菜单项

其中各个菜单项的属性设置如下所示：

```
object File1: TMenuItem
Caption = 'File'
object New1: TMenuItem
Caption = '&New'
end
object Open1: TMenuItem
Caption = '&Open'
end
object Close1: TMenuItem
Caption = '&Close'
end
object N1: TMenuItem
Caption = '-'
end
object Save1: TMenuItem
Caption = '&Save'
end
object Save2: TMenuItem
Caption = 'save &As'
end
object About1: TMenuItem
Caption = '&About'
end
object N2: TMenuItem
```

```
Caption = '-'
end
object Exit1: TMenuItem
Caption = '&Exit'
end
end
```

7. 设置复选标志

所谓的复选标志，指的是当用户选择菜单项中的某一个子菜单项时，在相应的菜单项的前面就会显示一个选中的标记符号“√”，而当用户再次单击这个菜单项时，复选标志又会消失。复选标志可以在程序的设计阶段通过子菜单项的属性来加以设置，方法是在程序的设计阶段首先用鼠标选中子菜单项。

然后把鼠标移动到菜单项对应的属性列表框中，把 Checked 属性设置为 True。

如图 3-23 所示。那么在程序的运行过程中，该菜单项的前面就会显示一个复选标志。

图 3-23　设置菜单的复选标志

当然，我们也可以在程序的运行过程中通过代码来动态的设置菜单项的复选标志，首先在窗体的 FormCreate()事件中加入对菜单项的初始化代码如下所示：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Open1->Checked=true;
//初始化菜单项的状态
}
```

以上的代码在程序运行的初期就把子菜单项 New 设置为复选状态，然后在程序的设计阶段用鼠标的左键单击子菜单项 New，在弹出的代码窗口中添加如下所示的相应菜单事件的代码：

```
void __fastcall TForm1::New1Click(TObject *Sender)
{
Form1->Open1->Checked=false;
Form1->New1->Checked=true;
//鼠标单击后改变菜单项的状态
}
```

这段代码能够实现子菜单项复选状态之间的互换，即用鼠标单击子菜单项会改变菜单项的选中状态。以上代码的运行结果如图 3-24 所示。

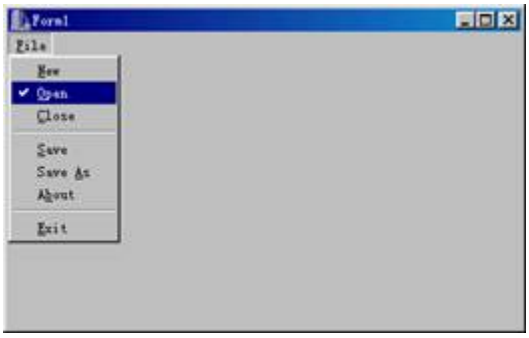


图 3-24　动态改变菜单的复选状态

8. 菜单项有效状态设置

在其它的应用程序中，经常的看到在一些菜单中，程序可以动态的设置菜单的有效状态，即在一定的条件下，某个菜单处于无效的状态，在另外的条件下，菜单项又会变为有效的状态，一个十分明显的例子就是编辑菜单中 Paste 子菜单项，当剪贴板中没有数据时，Paste 就会处于无效的状态，一旦剪贴板上有数据，那么 Paste 就会立刻的变为有效的状态，在 CBuilder 中也可以实现这种效果。

下面就讲述一下如何通过代码来在程序的运行过程中动态的改变菜单项的有效状态。

首先按照如图 3-25，按照前面所讲的方法设计一个比较完整的系统下拉式菜单，包括另外两个主菜单 Edit 和 Help。

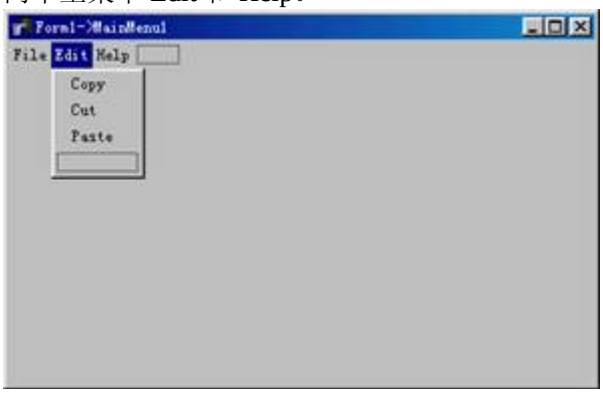


图 3-25　设计完成的下拉式菜单

然后，在程序的设计过程中，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中的 FormCreate()事件处理过程中，并且添加如下所示的程序初始化代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Copy1->Enabled=true;
Form1->Paste1->Enabled=false;
//初始化菜单项的有效状态
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的初期，在剪贴板上还没有可以利用的数据，所以要通过语句 Form1->Paste1->Enabled=false;把子菜单项 Paste 设置为无效的状态。

在程序运行的过程中，当用户用鼠标的左键单击子菜单项 Copy 时，系统的剪贴板就会带有数据，所以子菜单项 Paste 就要变为有效的状态。

在代码窗口中找到子菜单项的响应事件 Copy1Click()，并且添加如下的菜单响应代码：

```
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
Form1->Copy1->Enabled=true;
Form1->Paste1->Enabled=true;
//鼠标单击后改变菜单项的状态
}
//---------------------------------------------------------------------------
```

做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序，结果如下：

首先 Paste 菜单项显灰色，是无效的（如图 3-26）；但是，单击 Copy 菜单项后，Paste 菜单项就有效了（如图 3-27）。

图 3-26　Paste 菜单项无效

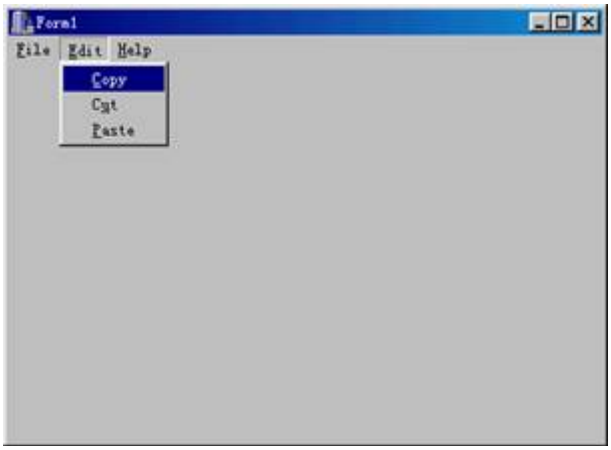


图 3-27　Paste 菜单项有效

3.2.2 菜单程序的设计

在 Windows 操作系统中，有一个简单的文本处理应用程序——“记事本”，在 Windows 的“开始”菜单中选择“程序”/“附件”/“记事本”，就会激活“记事本”应用程序，结果如图 3-28 所示。

图 3-28 “记事本”应用程序

下面将向读者介绍一个简单的文本处理应用程序，在其中可以实现以下的功能：

- 能够新建一个文本文件，并且能够把用户的修改结果保存到文本文件中；
- 能够打开一个已经存在的文本文件，而且在程序运行的过程中能够加以修改；
- 在程序运行的过程中，应用程序能够同剪贴板之间进行简单的数据交换等。

制作能够实现以上功能的文本处理应用程序的具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 MainMenu 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个主菜单设计器，接着向窗体上添加一个 OpenDialog 组件、一个 SaveDialog 组件和一个 Memo 组件，各个组件的功能将在后面分别的加以介绍，添加组件后的窗体如图 3-29 所示。

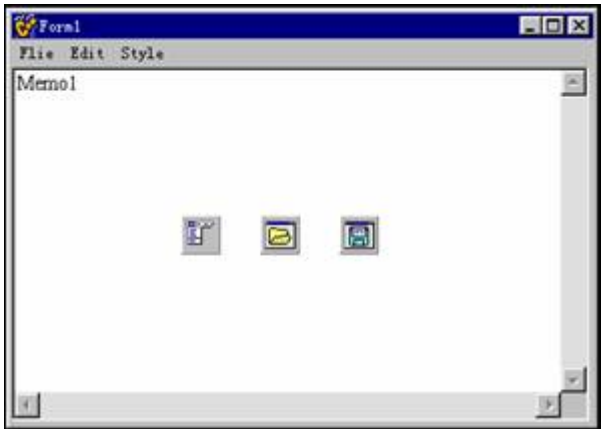


图 3-29 添加组件后的窗体

其中添加到窗体上的各个组件的功能如下所示：

- MainMenu 组件的作用是为应用程序设计一个下拉式主菜单，至于菜单的设计过程在第五章中有详细的介绍；
- 而 OpenDialog 组件的作用是显示一个对话框，用户在其中可以选择一个有效的文本文件；
- SaveDialog 组件的作用是为用户存储文件提供存储的路径和文件名；

■ Memo 组件的作用是为用户在程序运行过程中进行文本编辑提供一个容器。

窗体以及添加到窗体上组件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 420
  Height = 297
  Caption = 'Form1'
  Menu = MainMenu1
  OnCreate = FormCreate
  object Memo1: TMemo
    Left = 0
    Top = 0
    Width = 409
    Height = 249
    ScrollBars = ssBoth
  end
  object MainMenu1: TMainMenu
    Left = 120
    Top = 104
    object Flie1: TMenuItem
      Caption = 'Flie'
      object New1: TMenuItem
        Caption = 'New'
        OnClick = New1Click
      end
      object Open1: TMenuItem
        Caption = 'Open'
        OnClick = Open1Click
      end
      object Save1: TMenuItem
        Caption = 'Save'
        OnClick = Save1Click
      end
      object Exit1: TMenuItem
        Caption = '-'
      end
      object Exit2: TMenuItem
        Caption = 'Exit'
        OnClick = Exit2Click
      end
    end
    object Edit1: TMenuItem
      Caption = 'Edit'
      object Cut1: TMenuItem
        Caption = 'Cut'
        OnClick = Cut1Click
      end
      object Copy1: TMenuItem
        Caption = 'Copy'
        OnClick = Copy1Click
      end
      object Paste1: TMenuItem
        Caption = 'Paste'
        OnClick = Paste1Click
      end
      object Delete1: TMenuItem
        Caption = 'Delete'
        OnClick = Delete1Click
      end
```

```
        object N1: TMenuItem
          Caption = '-'
        end
        object SelectAll1: TMenuItem
          Caption = 'Select All'
          OnClick = SelectAll1Click
        end
      end
      object Style1: TMenuItem
        Caption = 'Style'
        object Left1: TMenuItem
          Caption = 'Left'
          OnClick = Left1Click
        end
        object Center1: TMenuItem
          Caption = 'Center'
          OnClick = Center1Click
        end
        object Right1: TMenuItem
          Caption = 'Right'
          OnClick = Right1Click
        end
        object N2: TMenuItem
          Caption = '-'
        end
        object Bold1: TMenuItem
          Caption = 'Bold'
          OnClick = Bold1Click
        end
        object Italic1: TMenuItem
          Caption = 'Italic'
          OnClick = Italic1Click
        end
        object UnderLine1: TMenuItem
          Caption = 'UnderLine'
          OnClick = UnderLine1Click
        end
        object StrikeOut1: TMenuItem
          Caption = 'StrikeOut'
          OnClick = StrikeOut1Click
        end
      end
    end
    object OpenDialog1: TOpenDialog
      Left = 176
      Top = 104
    end
    object SaveDialog1: TSaveDialog
      Left = 232
      Top = 104
    end
end
```

经过以上的属性设置后的下拉式主菜单如图 3-30 所示。

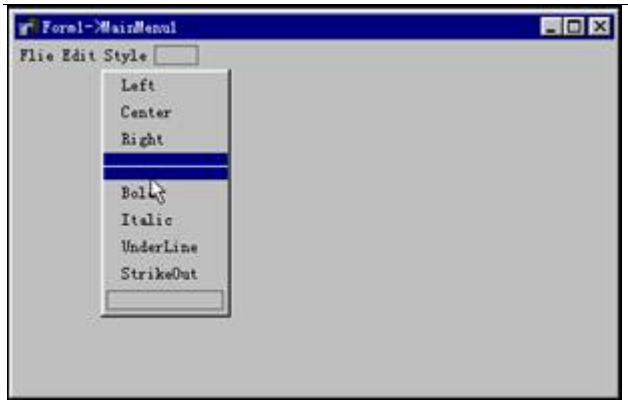


图 3-30　设计完成的下拉式主菜单

在下拉式主菜单中只包括 3 个主菜单项 File、Edit 和 Style，在各个主菜单项的下面又包括有几个子菜单项，菜单的结构如表 3-1 所示。

表 3-1　菜单项的结构表

主菜单项	子菜单项
Flie	New
	Open
	Save
	Exit
Edit	Cut
	Copy
	Paste
	Delete
	Select All
Style	Left
	Center
	Right
	Bold
	Italic
	UnderLine
	StrikeOut

2. 程序的初始化

程序的初始化过程，实际上就是对窗体 FormCreate()事件的初始化，在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Form1->Memo1->Clear();
 //清空文本框
 Form1->OpenDialog1->Title="请选择一个文本文件："
 Form1->SaveDialog1->Title="请选择一个文本文件："
 //设置对话框的标题
 Form1->OpenDialog1->InitialDir="c:\\pwin98";
 Form1->SaveDialog1->InitialDir="c:\\pwin98";
 //设置对话框的缺省路径
```

```
 Form1->OpenDialog1->Filter="Text Files(*.txt)|*.txt";
 Form1->SaveDialog1->Filter="Text Files(*.txt)|*.txt";
 //设置文件过滤器
 Form1->OpenDialog1->DefaultExt="txt";
 Form1->SaveDialog1->DefaultExt="txt";
 //设置缺省文件扩展名
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的初期，首先执行窗体 FormCreate()事件中的代码，即通过语句 Form1->Memo1->Clear();来清空文本框中的显示内容，把两个对话框组件的显示标题都设置为“请选择一个文本文件：”，然后设置 OpenDialog 组件和 SaveDialog 组件的缺省路径为 c:\\pwin98，最后通过两条语句 Form1->OpenDialog1->Filter="Text Files(*.txt)|*.txt"、Form1->SaveDialog1->Filter="Text Files(*.txt)|*.txt";来设置文件的过滤条件为'Text Files(*.txt)|*.txt'，即在两个对话框中只能够显示以*.txt 结尾的文本文件，并且程序的初始化设置了缺省的文件扩展名为*.txt，经过代码初始化后的窗体及其组件如图 3-31 所示。

图 3-31　初始化后的窗体及其组件

3. 响应 File 菜单

在 File 菜单下有四个子菜单项——New、Open、Save 和 Exit，它们的功能如下所示：

- 子菜单 New：新建一个文本文件，同时清空文本框；
- 子菜单 Open：打开一个已经存在的文本文件；
- 子菜单 Save：把当前文本框中的内容存储到一个文本文件中去；
- 子菜单 Exit：结束程序的运行。

下面仅以子菜单 Open 项来说明 File 菜单项的代码添加过程，其余子菜单的代码添加过程请参见附后的源程序代码。

为了能够实现子菜单项 Open 项的功能，在程序的设计过程中用鼠标的左键单击菜单 File/Open，在屏幕上就会弹出一个代码窗口，在其中可以添加对子菜单项 Open 项的响应代码如下所示：

```
void __fastcall TForm1::Open1Click(TObject *Sender)
{
 if (Form1->OpenDialog1->Execute())
       //打开一个对话框
       {
       Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
       //打开文本文件
       }
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击菜单 File/Open 时，就会激活菜单的

Open1Click()事件，然后程序通过语句 Form1->OpenDialog1->Execute()来显示一个对话框，用户可以在其中选择一个有效的文本文件，单击“打开”按钮后，在文本框中就会显示出打开的文本文件的内容，具体的打开文件的操作是通过 Memo 组件来完成的，实现的语句为 Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);。

程序运行过程中的 File 菜单如图 3-32 所示。

图 3-32　程序运行过程中的 File 菜单

4. 响应 Edit 菜单

在 Edit 菜单下有五个子菜单项——Cut、Copy、Paste、Delete 和 Select，它们的功能如下所示：

- 子菜单 Cut：把当前文本框中选中的数据剪贴到剪贴板上；
- 子菜单 Copy：把当前文本框中选中的数据复制到剪贴板上；
- 子菜单 Paste：把当前剪贴板上的数据粘贴到文本框的当前位置上；
- 子菜单 Delect：把当前文本框中的选中数据清除；
- 子菜单 Select：选中当前剪贴板上的所有数据。

下面仅以子菜单 Copy 项来说明 Edit 菜单项的代码添加过程，其余子菜单的代码添加过程请参见附后的源程序代码。

为了能够实现子菜单项 Copy 项的功能，在程序的设计过程中用鼠标的左键单击菜单 Edit/Copy，在屏幕上就会弹出一个代码窗口，在其中可以添加对子菜单项 Copy 项的响应代码如下所示：

```
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
 if (Form1->Memo1->SelText!="")
      //如果选中了文本内容
 {
   Form1->Memo1->CopyToClipboard();
   //将选中的数据复制到剪贴板上
 }
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击菜单 Edit/Copy 时，就会激活菜单的 Copy1Click()事件，如果当前选中的文本非空，程序就通过语句 Form1->Memo1->CopyTo Clipboard();把当前文本框中的选中数据复制到剪贴板上。

程序运行过程中的 Edit 菜单如图 3-33 所示。

图 3-33　程序运行过程中的“Edit”菜单

5. 响应 Style 菜单

在 Style 菜单下有 7 个子菜单项——Left、Center、Right、Bold、Italic、UnderLine 和 StrikeOut，它们的功能如下所示：

- 子菜单 Left：将当前文本框中的文本以左对齐方式显示；
- 子菜单 Center：将当前文本框中的文本以居中对齐方式显示；
- 子菜单 Right：将当前文本框中的文本以右对齐方式显示；
- 子菜单 Bold：当前文本框中的文本以黑体显示；
- 子菜单 Italic：当前文本框中的文本以斜体显示；
- 子菜单 UnderLine：当前文本框中的文本以下划线形式显示；
- 子菜单 StrikeOut：当前文本框中的文本以删除线形式显示。

下面仅以子菜单 Center 和 Bold 项来说明 Style 菜单项的代码添加过程，其余子菜单的代码添加过程请参见附后的源程序代码。

为了能够实现子菜单项 Center 项的功能，在程序的设计过程中用鼠标的左键单击菜单 Style/Center，在屏幕上就会弹出一个代码窗口，在其中可以添加对子菜单项 Center 项的响应代码如下所示：

```
void __fastcall TForm1::Center1Click(TObject *Sender)
{
 Form1->Memo1->Alignment=taCenter;
 //设置文本居中对齐
}
//---------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击菜单 Style/Center 时，就会激活菜单的 Center1Click ()事件，然后程序通过语句 Form1->Memo1->Alignment=taCenter;来设置文本框中的文本为居中对齐方式。

为了能够实现子菜单项 Bold 项的功能，在程序的设计过程中用鼠标的左键单击菜单 Style/Bold，在弹出的代码窗口中添加对子菜单项 Bold 项的响应代码：

```
void __fastcall TForm1::Bold1Click(TObject *Sender)
{
Form1->Bold1->Checked=! Form1->Bold1->Checked;
//设置复选状态
if (Form1->Bold1->Checked)
    //如果选中黑体
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style<<fsBold;
    //当前文本框中的文本以黑体的形式显示
    else
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style>>fsBold;
    //当前文本框中的文本不以黑体的形式显示
}
//---------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击菜单 Style/Bold 时，就会激活菜单的 Bold 1Click ()事件，通过语句 Form1->Bold1->Checked=! Form1->Bold1->Checked;来设置子菜单项 Bold 的复选状态，这样当用鼠标的左键单击子菜单 Bold 时，就会转换子菜单项的复选状态。

然后程序通过语句 Form1->Bold1->Checked 来判断子菜单 Bold 的复选状态，如果处于复选状态，那么通过语句 Form1->Memo1->Font->Style=Form1->Memo1->Font->Style <<fsBold；使得文本框中的文本以黑体显示，否则将通过语句 Form1->Memo1->Font->Style= Form1->Memo1->Font->Style>>fsBold；在文本框中正常的显示文本。

程序运行过程中的 Style 菜单如图 3-34 所示。

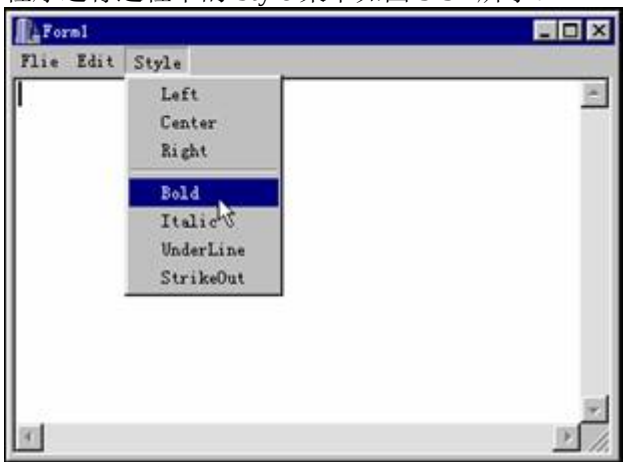


图 3-34　程序运行中的 Style 菜单

6. 运行程序

按照附后的源程序，添加剩余的程序代码后，按键盘上的功能键 F9 运行程序，选择菜单 File/Open，选择一个有效的文本文件后。

单击“打开”按钮，在程序运行的窗体中就显示了这个文本文件的内容，如图 3-35 所示。

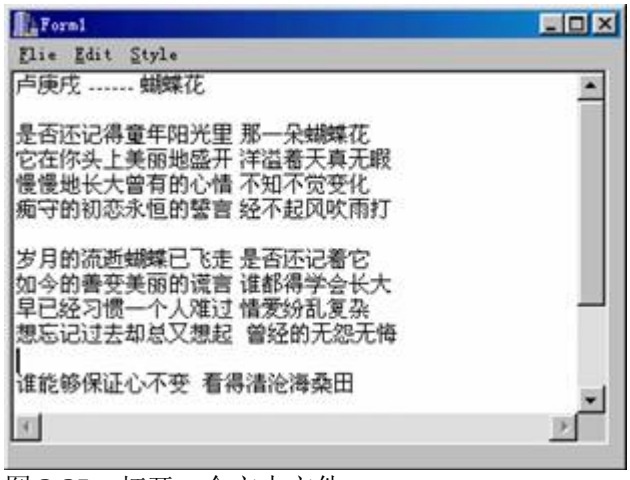


图 3-35　打开一个文本文件

然后，您就可以对此打开的文本进行简单的编辑，例如可以删改文本内容、设置文本的风格等。

如图 3-36 所示为作了某些修改后的结果。

图 3-36　设置风格后的文本

附程序完整的源代码如下所示，供读者参考学习：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Text.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Form1->Memo1->Clear();
 //清空文本框
 Form1->OpenDialog1->Title="请选择一个文本文件:";
 Form1->SaveDialog1->Title="请选择一个文本文件:";
 //设置对话框的标题
 Form1->OpenDialog1->InitialDir="c:\\pwin98";
 Form1->SaveDialog1->InitialDir="c:\\pwin98";
 //设置对话框的缺省路径
 Form1->OpenDialog1->Filter="Text Files(*.txt)|*.txt";
 Form1->SaveDialog1->Filter="Text Files(*.txt)|*.txt";
 //设置文件过滤器
 Form1->OpenDialog1->DefaultExt="txt";
 Form1->SaveDialog1->DefaultExt="txt";
 //设置缺省文件扩展名
}
//---------------------------------------------------------------------------

void __fastcall TForm1::New1Click(TObject *Sender)
{
 Form1->Memo1->Clear();
 //清空文本框
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Open1Click(TObject *Sender)
```

```
{
 if (Form1->OpenDialog1->Execute())
       //打开一个对话框
       {
       Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
       //打开文本文件
       }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Save1Click(TObject *Sender)
{
if (Form1->SaveDialog1->Execute())
    //打开一个对话框
    {
    Form1->Memo1->Lines->SaveToFile(Form1->SaveDialog1->FileName);
    //存储到文本文件中
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Exit2Click(TObject *Sender)
{
 Form1->Close();
 //结束程序的运行
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
 if (Form1->Memo1->SelText!="")
     //如果选中了文本内容
 {
 Form1->Memo1->CutToClipboard();
 //将选中的数据剪切到剪贴板上
 }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Copy1Click(TObject *Sender)
{
 if (Form1->Memo1->SelText!="")
     //如果选中了文本内容
 {
  Form1->Memo1->CopyToClipboard();
  //将选中的数据复制到剪贴板上
 }
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
 if (Form1->Memo1->SelText!="")
      //如果选中了文本内容
 {
   Form1->Memo1->PasteFromClipboard();
   //将当前剪贴板上的数据粘贴到当前位置上
 }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Delete1Click(TObject *Sender)
{
 Form1->Memo1->ClearSelection();
 //清除当前选中的数据
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SelectAll1Click(TObject *Sender)
{
 Form1->Memo1->SelectAll();
 //选中控件上的全部数据

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Left1Click(TObject *Sender)
{
 Form1->Memo1->Alignment=taLeftJustify;
 //设置文本左对齐方式
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Center1Click(TObject *Sender)
{
 Form1->Memo1->Alignment=taCenter;
 //设置文本居中对齐
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Right1Click(TObject *Sender)
{
 Form1->Memo1->Alignment=taRightJustify;
```

```
 //设置文本右对齐方式
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Bold1Click(TObject *Sender)
{
Form1->Bold1->Checked=! Form1->Bold1->Checked;
//设置复选状态
if (Form1->Bold1->Checked)
    //如果选中黑体
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style<<fsBold;
    //当前文本框中的文本以黑体的形式显示
    else
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style>>fsBold;
    //当前文本框中的文本不以黑体的形式显示
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Italic1Click(TObject *Sender)
{
Form1->Italic1->Checked=! Form1->Italic1->Checked;
//设置复选状态
if (Form1->Italic1->Checked)
    //如果选中斜体
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style<<fsItalic;
    //当前文本框中的文本以斜体的形式显示
    else
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style>>fsItalic;
    //当前文本框中的文本不以斜体的形式显示
}
//---------------------------------------------------------------------------

void __fastcall TForm1::UnderLine1Click(TObject *Sender)
{
Form1->UnderLine1->Checked=! Form1->UnderLine1->Checked;
//设置复选状态
if (Form1->UnderLine1->Checked)
    //如果选中下划线
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style<<fsUnderline;
    //当前文本框中的文本以下划线的形式显示
    else
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style>>fsUnderline;
    //当前文本框中的文本不以下划线的形式显示
}
//---------------------------------------------------------------------------

void __fastcall TForm1::StrikeOut1Click(TObject *Sender)
```

```
{
Form1->StrikeOut1->Checked=! Form1->StrikeOut1->Checked;
//设置复选状态
if (Form1->UnderLine1->Checked)
    //如果选中删除线
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style<<fsStrikeOut;
    //当前文本框中的文本以删除线的形式显示
    else
    Form1->Memo1->Font->Style=Form1->Memo1->Font->Style>>fsStrikeOut;
    //当前文本框中的文本不以删除线的形式显示
}
//---------------------------------------------------------------------------
```

3.3 小　结

在本章中，系统的介绍了程序运行过程中，人机交互的两种常用的手段——菜单和对话框的程序设计方法，在读者的程序开发实践中，可以利用 CBuilder 系统提供的菜单和对话框设置自己的应用程序，当然也可以自己定义菜单和对话框，这一点希望在不断的开发实践中自己加以体会。

第四章　绘制图形和图案

在 CBuilder 5 的应用程序中，图形的绘制和处理一直是一个比较活跃的领域，因为任何一个应用程序中都免不了要同图形打交道，而且处理图形又是进行多媒体程序设计的基础，所以在本书中将图形这块内容专门用一章来讲解。
下面就通过几个示例程序来说明在 CBuilder 5 中如何进行图形（以及图案）的绘制和处理。

4.1　基本图形的绘制

利用 CBuilder 5 中提供的方法可以绘制出简单的基本图案（如直线、圆和矩形等），同时也可以绘制出比较复杂的图案，如（水晶石图案、圆环等），下面我们就先对基本图形的绘制作一详细介绍。

4.1.1　直线的绘制

在 CBuilder 5 中绘制直线是通过 LineTo 方法来实现的，它的语法结构如下所示：
void __fastcall LineTo(int X, int Y);
下面通过一个示例来说明如何在程序中绘制直线，在程序运行的过程中，当用户在窗体上单击鼠标的左键时，程序就会调用相应的方法在窗体上绘制一条由上一个鼠标单击点到当前鼠标单击点的直线，步骤如下：

1. 开始工作

首先启动一个新项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，在 Button 组件的图标上双击鼠标的左键，空白窗体上就会出现一个按钮组件，添加组件后的窗体如图 4-1 所示。

图 4-1　添加组件后的窗体

2. 添加代码

在程序的设计过程中，用鼠标的左键双击窗体上的按钮组件，在屏幕上就会弹出一个代码窗口，把光标移动到组件和窗体的相应事件过程中，并且按照下面所列的程序清单添加程序代码：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Line.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
Boolean Start;
//定义标志变量
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Start=false;
//初始化标志变量
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Start=true;
//设置标志变量
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
if (Start)
   Form1->Canvas->LineTo(X,Y);
   //绘制直线到鼠标的单击点
}
//---------------------------------------------------------------------
```

程序说明：

在程序的变量声明段中，首先定义了一个布尔型变量 Start，它的作用是控制程序绘制直线的开始时间，在窗体的初始化过程中通过语句 Start=false;把标志变量 Start 设置为 false，如果在程序运行的过程中，用户单击按钮“绘制直线”，那么程序就自动的设置变量 Start 的值为 true，然后通过语句 Form1->Canvas->LineTo(X,Y);在窗体上绘制直线。

3. 运行程序

做完以上的工作后，按键盘上的功能键【F9】运行程序，在程序运行的初始画面中，首先单击按钮“绘制直线”，激活绘图程序后，用鼠标的左键在窗体上单击，程序的运行的结果如图 4-2 所示。

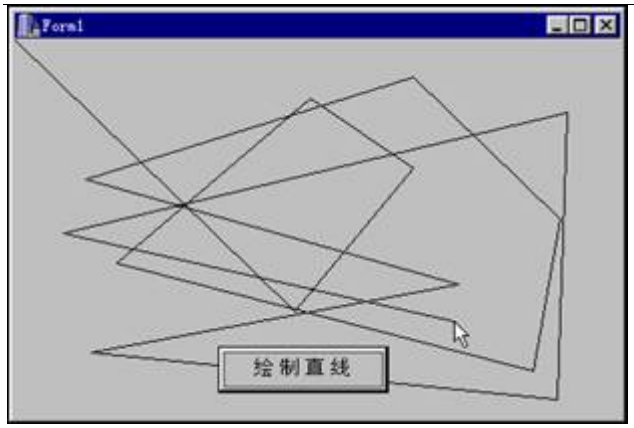


图 4-2　程序运行结果

4.1.2　椭圆的绘制

在 CBuilder 5 中绘制椭圆调用的是 Ellipse 方法，它的语法结构如下所示：
void __fastcall Ellipse(int X1, int Y1, int X2, int Y2);
下面通过一个示例来说明如何在程序中绘制椭圆，在程序运行的过程中，当用户在窗体上按下并且移动鼠标时，程序就会调用相应的方法在窗体上绘制一个椭圆。
具体的程序设计步骤如下：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体。

2. 添加代码

在程序的设计过程用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到代码段中的适当位置，并且按照下面的程序清单添加绘制椭圆的代码：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Ellipse.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
Boolean Start;
//定义标志变量
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Start=false;
//变量赋初值
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
Start=true;
//开始绘图
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
if (Start)
   Form1->Canvas->Ellipse(X,Y,20,20);
   //绘制椭圆
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
Start=false;
//结束绘图
}
//---------------------------------------------------------------------------
```

3. 运行程序

做完以上的工作后，按键盘上的功能键【F9】运行程序，在程序运行的窗体上，按下的同时移动鼠标，在窗体上就会绘制出一系列的椭圆，程序的运行结果如图 4-3 所示。

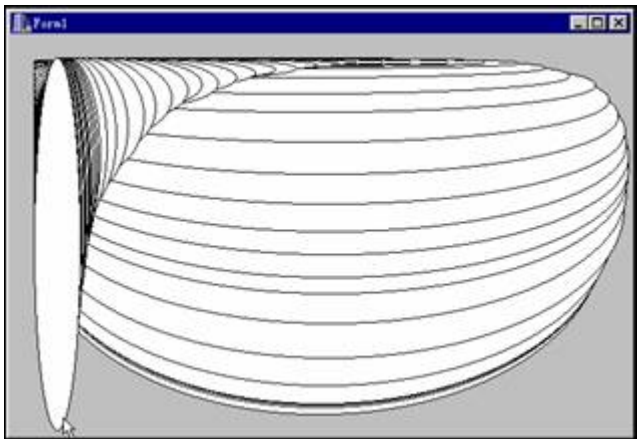


图 4-3　程序运行结果

4.1.3 绘制矩形

在 CBuilder 5 中绘制矩形调用的方法是 Rectangle 方法，它的语法结构如下所示：
void __fastcall Rectangle(int X1, int Y1, int X2, int Y2);
在调用 Rectangle 方法绘制矩形的时候，要为该方法提供四个顶点的坐标，并且顶点的坐标值为 Integer 类型。
下面通过一个示例来说明如何在程序中绘制一个矩形，在程序运行的过程中，每隔 1 秒钟，在窗体上就会以随机点绘制一个矩形。
具体的程序设计步骤如下：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 System 选项后，在 Timer 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个计时器组件.
其中计时器组件的属性设置如图 4-4 所示。

图 4-4　计时器组件的属性设置

2. 添加代码

为了保证在程序运行的过程中，能够每隔 1 秒钟就绘制一个矩形，在程序的初始化过程中应该设置计时器组件的有效状态为 True，具体的程序代码请参考下面所示的程序清单：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
```

```
Form1->Canvas->Pen->Color=RGB(random(255),random(255),random(255));
//设置画笔颜色
Form1->Canvas->Pen->Width=random(10)+1;
//设置画笔大小
Form1->Canvas->Pen->Mode=random(16);
//设置画笔模式
Form1->Canvas->Pen->Style=random(7);
//设置画笔风格
Form1->Canvas->Brush->Color=RGB(random(255),random(255),random(255));
//设置画刷颜色
Form1->Canvas->Brush->Style=random(8);
//设置画刷风格
Form1->Canvas->Rectangle(random(Form1->Width),random(Form1->Height),random(Form1->
Width),random(Form1->Height));
//以随机位置绘制矩形
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Timer1->Enabled=true;
//开始绘图
}
//---------------------------------------------------------------------------
```

程序说明：

由于是以随机点的形式绘制矩形，所以在程序运行的过程中要调用函数 random()来产生四个随机数，并且绘制矩形的代码要放在计时器组件的 Timer1Timer()过程事件中，这样才能够保证每隔 1 秒钟绘制一个随机点矩形。

3. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键【F9】运行程序，在程序运行的初始画面中，每隔 1 秒钟，程序就会自动的以随机点来绘制一个矩形，程序运行结果如图 4-5 所示。

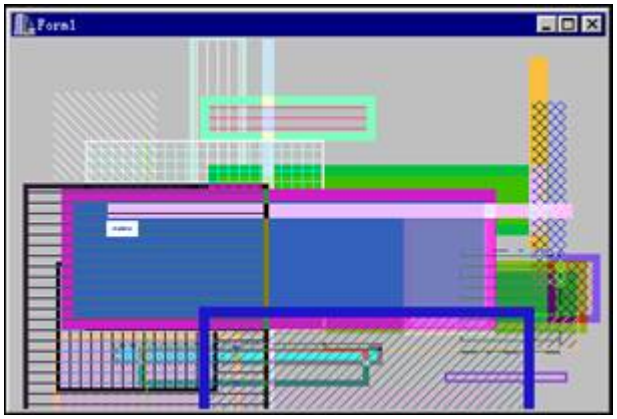


图 4-5　随机绘制的矩形

4.2 复杂图形的绘制

4.2.1 艺术图案的绘制

在日常的生活中，我们经常可以见到各种各样的复杂图案，如地毯上的艺术图案等，在CBuilder 5 中，利用一定的绘图方法，在程序中也可以绘制出丰富多彩的艺术图案，下面就以“水晶石”图案和“蜘蛛网”图案为例来说明艺术图案的绘制方法。在示例程序中，用户可以在程序运行的过程中自己选择绘制“水晶石”图案还是绘制“蜘蛛网”图案，具体的程序设计步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，向窗体上添加两个 Button 组件，然后选择组件工具栏上的 Additional 选项，把两个 Image 组件添加到窗体上，它们的属性设置如下所示：

```
object Form1: TForm1
  Left = 209
  Top = 93
  Width = 478
  Height = 332
  Caption = 'Form1'
  OnCreate = FormCreate
  object Image1: TImage
    Left = 8
    Top = 8
    Width = 353
    Height = 289
    AutoSize = True
    Center = True
    Stretch = True
    Transparent = True
  end
  object Image2: TImage
    Left = 8
    Top = 8
    Width = 353
    Height = 289
    Transparent = True
  end
  object Button1: TButton
    Left = 368
    Top = 8
    Width = 97
    Height = 33
    Caption = '蜘蛛网图案'
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 368
    Top = 48
    Width = 97
    Height = 33
```

```
    Caption = '水晶石图案'
    OnClick = Button2Click
  end
end
```

窗体中的各个组件的作用如下所示：

- Button1 组件：产生绘制“蜘蛛网”图案的动作和完成具体的绘制工作；
- Button2 组件：产生绘制“水晶石”图案的动作和完成具体的绘制工作；
- Image1 组件：作为程序运行过程过程中绘制“蜘蛛网”图案的容器；
- Image2 组件：作为程序运行过程过程中绘制“水晶石”图案的容器。

添加组件后的窗体如图 4-6 所示。

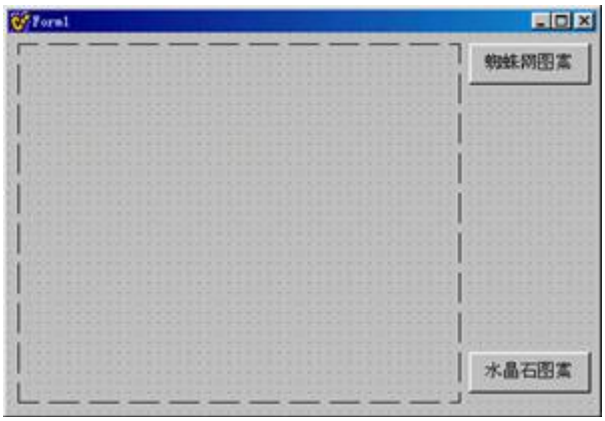


图 4-6　添加组件后的窗体

2. 添加代码

添加代码的过程包括两个过程——“蜘蛛网”图案的绘制和“水晶石”图案的绘制。由于篇幅的关系，在这里对按钮“蜘蛛网”图案的响应代码就不详细的加以介绍了，读者请参看附后的源程序代码。

在程序的设计过程中，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体 FormCreate()事件的处理过程中，并且添加如下所示的程序初始化代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Image1->Visible=false;
Form1->Image2->Visible=false;
//设置图像组件都处于不可见状态
Form1->Image1->Transparent=true;
Form1->Image2->Transparent=true;
//设置图像的背景色为透明
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的初期，系统会自动的激活窗体的 FormCreate()事件，然后程序通过语句 Form1->Image1->Visible=false;、Form1->Image2->Visible=false;设置两个图像组件都处于不可见状态，然后通过语句 Form1->Image1->Transparent=true;、Form1->Image2->Transparent=true;设置两个图像组件的背景色为透明，这样绘制的图案不会出现背景色。

添加完对程序的初始化代码后，用鼠标左键双击窗体上的 Button2 组件，并且在它的响应事件中添加如下代码：

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    float t,x0,y0;
    float r;
    int n,j,i;
    float x[50];
    float y[50];
```

```
Form1->Image2->Visible=true;
Form1->Image1->Visible=false;
n=15;
t=6.28318/n;
r=130;
x0= 180;
y0= 150;
for (i=0;i<=n;i++)
      {
      x[i]=r*cos(i*t)+x0;
      y[i]=r*sin(i*t)+y0;
      }
for (i=0;i<=n;i++)
      {
       for(j=i+1;j<=n-1;j++)
           {
               Form1->Image2->Canvas->MoveTo((x[i]),(y[i]));
               Form1->Image2->Canvas->LineTo((x[j]),(y[j]));
            }
       }
}
//---------------------------------------------------------------------------
```

3. 运行程序

按照附后的程序清单添加代码，做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键【F9】运行程序，在程序运行的初始画面中，单击“蜘蛛网图案”按钮，在窗体上就会绘制出一个如图 4-7 所示的“蜘蛛网”图案。

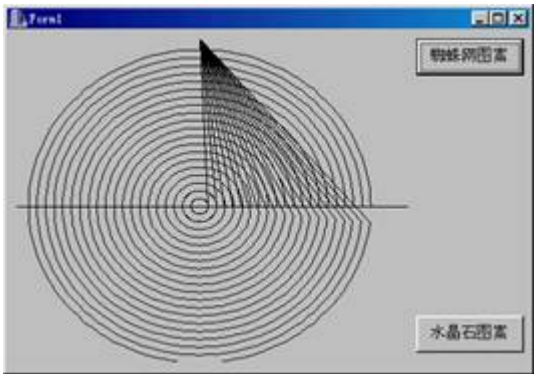


图 4-7　“蜘蛛网”渐变图案

如果用户单击“水晶石图案”按钮，在窗体的相应位置上就会绘制出如图 4-8 所示的“水晶石”图案。

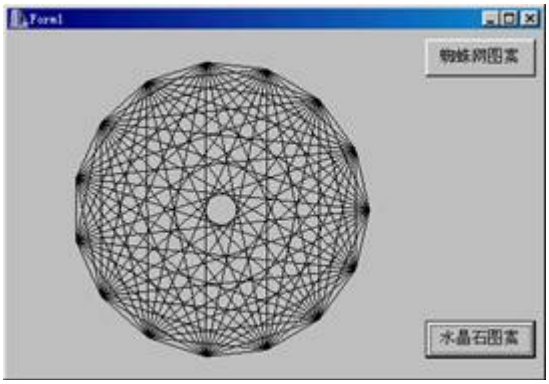


图 4-8　程序绘制的“水晶石”图案

附程序完整源代码如下所示。

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "math.h"
#include "Art.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Image1->Visible=false;
Form1->Image2->Visible=false;
//设置图像组件都处于不可见状态
Form1->Image1->Transparent=true;
Form1->Image2->Transparent=true;
//设置图像的背景色为透明
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    float x1[120];
    float y1[120];
    float x2[120];
    float y2[120];
    float pi;
    int n;
    int r;
    float a;
    int i;
    float t;
    float x;
    float y;
    int k;
    //变量定义
Form1->Image1->Visible=true;
Form1->Image2->Visible=false;
//设置组件可见性
pi=3.1415926;
n=20;
r=280;
i=0;
a=0;
//变量赋值
for (i=1;i<120;i++)
    {
    x1[i]=r*cos(a);
    y1[i]=(-1)*r*sin(a)/2;
    a=a+pi/60;
    }
```

```
r=100;
i=0;
a=0;
for (i=0;i<=120;i++)
      {
      t=(r*(1+1/2*sin(12*a)))*(1/2+1/2*sin(4*a));
      x2[i]=t*cos(a);
      y2[i]=(-1)*t*sin(a)/2;
      a=a+pi/60;
      }
for (k=0;k<=n;k++)
      {
      for(i=0;i<=120;i++)
            {
            x=(x2[i]-x1[i])/n*k+x1[i];
            y=(y2[i]-y1[i])/n*k+y1[i];
            x=x+300;
            y=y+150;
            if (i==0)
                {
                Form1->Image1->Canvas->MoveTo((x/2),(y));
                }
            Form1->Image1->Canvas->LineTo((x*1.1/2),(y));
            }
       }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
      float t,x0,y0;
      float r;
      int n,j,i;
      float x[50];
      float y[50];
      //变量定义
Form1->Image2->Visible=true;
Form1->Image1->Visible=false;
//设置组件可见性
n=15;
//设置分段数
t=6.28318/n;
r=130;
//设置半径
x0= 180;
y0= 150;
//设置图案的中心点
for (i=0;i<=n;i++)
      {
      x[i]=r*cos(i*t)+x0;
      y[i]=r*sin(i*t)+y0;
      //通过循环求出"水晶石"  图案的坐标点
      }
for (i=0;i<=n;i++)
      {
       for(j=i+1;j<=n-1;j++)
            {
                Form1->Image2->Canvas->MoveTo((x[i]),(y[i]));
                Form1->Image2->Canvas->LineTo((x[j]),(y[j]));
```

```
                //连线
            }
        }
}
//---------------------------------------------------------------------------
```

4.2.2 利用鼠标绘制图形

在图形图像处理的应用程序中，我们经常是利用鼠标来绘制图形，所以鼠标在图形处理中占有重要的地位，下面就通过一个示例应用程序来说明在图形处理应用程序中如何对鼠标的动作作出正确的响应，具体的程序设计步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Standard 选项后，选中 Button 组件，并且向窗体上添加三个 Button 组件和一个标签、四个 RadioButton 组件，然后在组件工具栏上选择 Additional 选项，并且把一个 Image 组件添加到窗体上，添加组件后的窗体如图 4-9 所示。

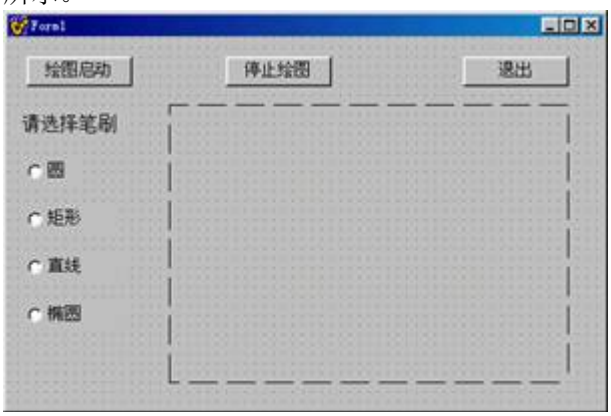


图 4-9　添加组件后的窗体

其中窗体上各个组件的属性设置如下所示：

```
object Form1: TForm1
   Left = 269
   Top = 111
   Width = 403
   Height = 308
   Caption = 'Form1'
   OnCreate = FormCreate
   object Image1: TImage
      Left = 8
      Top = 8
      Width = 265
      Height = 265
      Stretch = True
```

```
    OnMouseDown = Image1MouseDown
    OnMouseMove = Image1MouseMove
    OnMouseUp = Image1MouseUp
  end
  object Button1: TButton
    Left = 296
    Top = 8
    Width = 89
    Height = 25
    Caption = '绘图启动'
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 296
    Top = 56
    Width = 89
    Height = 25
    Caption = '停止绘图'
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 296
    Top = 104
    Width = 89
    Height = 25
    Caption = '退出'
    OnClick = Button3Click
  end
  object RadioButton1: TRadioButton
    Left = 296
    Top = 176
    Width = 41
    Height = 17
    Caption = '圆'
    OnClick = RadioButton1Click
  end
  object RadioButton2: TRadioButton
    Left = 296
    Top = 200
    Width = 73
    Height = 17
    Caption = '矩形'
    OnClick = RadioButton2Click
  end
  object RadioButton3: TRadioButton
    Left = 296
    Top = 224
    Width = 81
    Height = 17
    Caption = '直线'
    OnClick = RadioButton3Click
  end
  object RadioButton4: TRadioButton
    Left = 296
    Top = 248
    Width = 81
    Height = 17
    Caption = '椭圆'
    OnClick = RadioButton4Click
```

```
  end
end
```

其中各个组件的作用如下所示：

- Button1 组件：如果用户在程序运行的过程中激活它，那么就开始准备绘图了；
- Button2 组件：结束绘图；
- Button1 组件：结束程序的运行；
- Image1 组件：在程序运行的过程中用于完成具体的绘图操作，并且充当绘图的容器，所有绘图的操作都是在该组件上完成的；
- RadioButton1 组件：决定在绘图的过程中跟踪鼠标轨迹的方式为“圆”；
- RadioButton2 组件：决定在绘图的过程中跟踪鼠标轨迹的方式为“矩形”；
- RadioButton3 组件：决定在绘图的过程中跟踪鼠标轨迹的方式为“直线”；
- RadioButton4 组件：决定在绘图的过程中跟踪鼠标轨迹的方式为“椭圆”。

2. 响应鼠标动作

在这里仅以鼠标的移动操作来说明在程序中如何响应鼠标的动作，其余的代码请参看附后的源程序。

首先用鼠标的左键激活组件 Image1，在窗体的事件列表中选择 OnMouseMove，并且用鼠标左键双击右侧的输入框，在屏幕上就会弹出一个代码窗口，用户在其中就可以添加如下所示的响应鼠标移动的代码了：

```
void __fastcall TForm1::Image1MouseMove(TObject *Sender, TShiftState Shift,int X, int Y)
{
if (start)
    {
    if (move)
        {
        if (num==1)
            Form1->Image1->Canvas->Ellipse(X-5,Y-5,X+5,Y+5);
            //以圆的形式跟踪鼠标轨迹
        else if (num==2)
            Form1->Image1->Canvas->Rectangle(X-5,Y-5,X+5,Y+5);
            //以矩形的形式跟踪鼠标轨迹
        else if (num==3)
            Form1->Image1->Canvas->LineTo(X,Y);
            //以直线的形式跟踪鼠标轨迹
        else if (num==4)
            Form1->Image1->Canvas->Ellipse(X-5,Y-2,X+5,Y+2);
            //以椭圆的形式跟踪鼠标轨迹
        }
     }
//按照鼠标的运动轨迹绘图
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，当用户在窗体上移动鼠标时，就会激活组件的 Image1MouseMove 事件，然后程序就会通过条件判断语句来判断用户在移动鼠标之前是否按下“开始绘图”按钮和鼠标的左键；

如果单击“开始绘图”后在窗体上按下并且移动鼠标，程序就会对跟踪鼠标的绘图形式作出判断，最后通过：

```
Form1->Image1->Canvas->Ellipse(X-5,Y-5,X+5,Y+5);
Form1->Image1->Canvas->Rectangle(X-5,Y-5,X+5,Y+5);
Form1->Image1->Canvas->LineTo(X,Y);
Form1->Image1->Canvas->Ellipse(X-5,Y-2,X+5,Y+2);
```

等语句来完成绘制鼠标轨迹的操作。

3. 设置鼠标跟踪方式

在本程序中，是通过设置变量 num 的值来改变对鼠标跟踪方式的，例如在程序中如果选中“圆”选项，那么就会激活组件的 RadioButton1Click 事件，然后执行以下的程序代码：

```
void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
num=1;
//设置绘制鼠标轨迹的方式为圆
}
//---------------------------------------------------------------------------
```

程序说明：

通过以上的程序代码，设置了变量 num 的值为 1，在绘图的过程中通过判断 num 的值就可以用“圆”的形式对鼠标进行跟踪，进而完成绘图的操作。

4. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，单击“开始绘图”按钮，在窗体上按下左键并且移动鼠标。

这时，在窗体上就会出现一个跟踪鼠标的轨迹，程序运行出现结果，如图 4-10 和图 4-11 所示分别为选择“圆”笔刷和直线笔刷的效果。

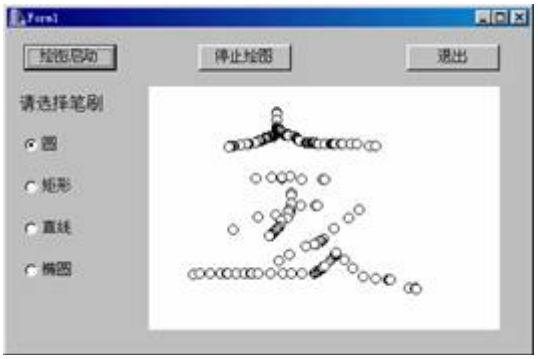


图 4-10　程序运行结果

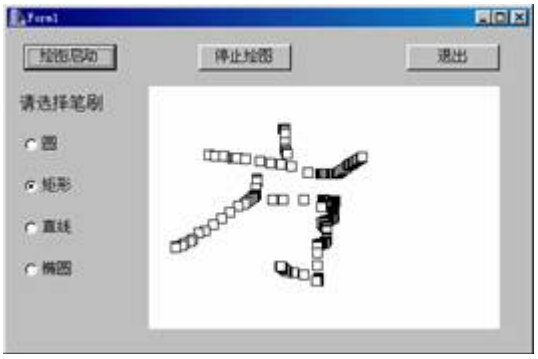


图 4-11　程序运行结果

附程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Mouse.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
```

```
TForm1 *Form1;
Boolean start;
Boolean move;
int num;
//声明变量
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
start=false;
move=false;
//变量初始化
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
start=true;
//准备绘图
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
start=false;
//结束绘图
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Close();
//结束程序运行
}
//---------------------------------------------------------------------

void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
num=1;
//设置绘制鼠标轨迹的方式为圆
}
//---------------------------------------------------------------------

void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
num=2;
//设置绘制鼠标轨迹的方式为矩形
}
//---------------------------------------------------------------------

void __fastcall TForm1::RadioButton3Click(TObject *Sender)
{
num=3;
//设置绘制鼠标轨迹的方式为直线
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton4Click(TObject *Sender)
{
num=4;
//设置绘制鼠标轨迹的方式为椭圆
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image1MouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
move=true;
//开始绘图
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Image1MouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
if (start)
    {
    if (move)
        {
        if (num==1)
            Form1->Image1->Canvas->Ellipse(X-5,Y-5,X+5,Y+5);
            //以圆的形式跟踪鼠标轨迹
        else if (num==2)
            Form1->Image1->Canvas->Rectangle(X-5,Y-5,X+5,Y+5);
            //以矩形的形式跟踪鼠标轨迹
        else if (num==3)
            Form1->Image1->Canvas->LineTo(X,Y);
            //以直线的形式跟踪鼠标轨迹
        else if (num==4)
            Form1->Image1->Canvas->Ellipse(X-5,Y-2,X+5,Y+2);
            //以椭圆的形式跟踪鼠标轨迹
        }
     }
//按照鼠标的运动轨迹绘图
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Image1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
move=false;
//结束绘图
}
//---------------------------------------------------------------------------
```

4.3 动画图形的实现

在对图形的处理中，不容忽视的一个方面就是图形动画的实现。其实动画技术当中两个非常重要的方面就是图形的产生和对时间的控制，在 CBuilder 5 中为我们解决这两个方面的问题提供了良好的方案，一方面可以利用现成的 Shape 组件产生常见的动画形体，另一个方面我们可以利用 Timer 组件还对动画的时间进行准确的控制。
下面就通过一个示例应用程序来说明在 CBuilder 5 中进行图形动画程序设计，具体的程序设计步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 Additional 选项后，选中 Shape 组件，并且向窗体上添加一个 Shape 组件，然后在组件工具栏上选择 System 选项，并且把一个 Timer 组件添加到窗体上，添加组件后的窗体如图 4-12 所示。

图 4-12　添加组件后的窗体

其中各个组件的作用如下：

- Image1 组件：在程序运行的过程中，利用该组件对动画运行时间等参数进行控制，并且在组件对应的 Timer1Timer 事件中产生具体的动画操作；
- Shape1 组件：在动画绘制的过程中用于产生动画形体。

2. 产生图形动画

在添加组件的过程中，把 Timer1 组件的 Interval 属性设置为 500，即每隔 0.5 秒的时间就会自动的激活一个 Timer1Timer 事件，所以我们可以把产生图形动画的代码放在这个事件中。
首先用鼠标的左键激活组件 Timer1，在窗体的事件列表中选择 OnTimer1Timer，并且用鼠标左键双击右侧的输入框，在屏幕上就会弹出一个代码窗口，用户在其中就可以添加如下所示的产生图形动画的代码了：

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Randomize;
Form1->Shape1->Brush->Color=RGB(random(255),random(255),random(255));
//设置画刷颜色
Randomize;
……
Randomize;
Form1->Shape1->Pen->Width=random(10)+1;
……
}
```

在程序运行的过程中，每隔 0.5 秒就会激活一个 Timer1Timer 事件，程序首先通过语句 Randomize;来初始化随机数，然后通过语句 Form1->Shape1->Brush->Color=RGB(random(255)，random(255)random(255))来以随机颜色设置当前使用画刷的颜色，本示例程序动画产生的基本原理就是每隔一定的时间就对窗体上 Shape1 组件的形状、颜色等各种参数加以改变，并且改变的结果是由系统产生的随机数来决定的。
由于篇幅的关系，这里就不对动画产生的全过程加以叙述了，请读者参看附后的程序代码。

3. 运行程序

选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行结果如图 4-13、图 4-14 和图 4-15 所示。

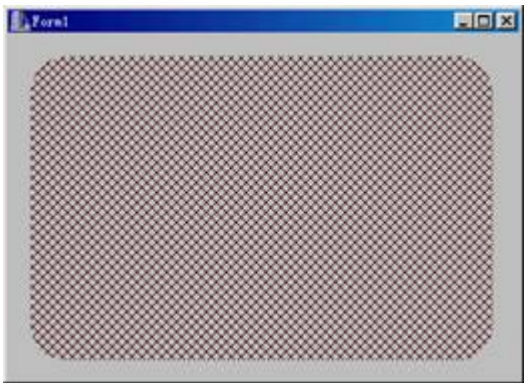


图 4-13　程序运行结果之 1

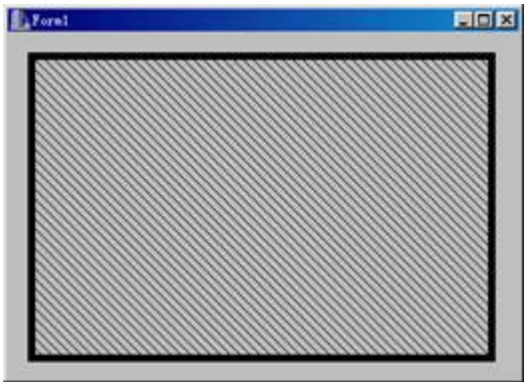


图 4-14　程序运行结果之 2

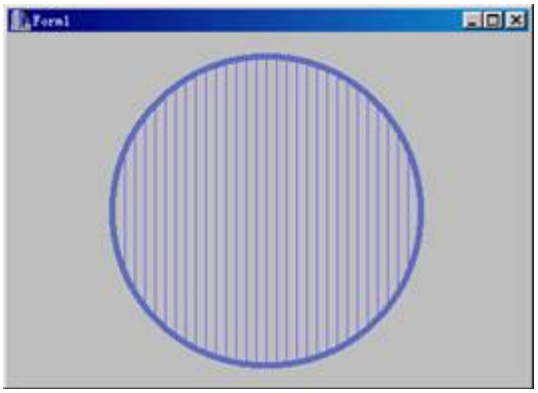


图 4-15　程序运行结果之 3

附程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include <stdlib.h>
```

```
#include "Animate.h"
//---------------------------------------------------------------------------

#pragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Randomize;
Form1->Shape1->Brush->Color=RGB(random(255),random(255),random(255));
//设置画刷颜色
Randomize;
switch (random(8))
        {
        case 0:Form1->Shape1->Brush->Style=bsSolid;
                break;
        case 1:Form1->Shape1->Brush->Style=bsClear;
                break;
        case 2:Form1->Shape1->Brush->Style=bsHorizontal;
                break;
        case 3:Form1->Shape1->Brush->Style=bsVertical;
                break;
        case 4:Form1->Shape1->Brush->Style=bsFDiagonal;
                break;
        case 5:Form1->Shape1->Brush->Style=bsBDiagonal;
                break;
        case 6:Form1->Shape1->Brush->Style=bsCross;
                break;
        case 7:Form1->Shape1->Brush->Style=bsDiagCross;
        }
        //设置画刷风格
Randomize;
Form1->Shape1->Pen->Color=RGB(random(255),random(255),random(255));
//设置画笔颜色
Randomize;
Form1->Shape1->Pen->Width=random(10)+1;
//设置画笔大小
Randomize;
switch (random(16))
        {
        case 0:Form1->Shape1->Pen->Mode=pmBlack;
                break;
        case 1:Form1->Shape1->Pen->Mode=pmWhite;
                break;
        case 2:Form1->Shape1->Pen->Mode=pmNop;
                break;
        case 3:Form1->Shape1->Pen->Mode=pmNot;
                break;
        case 4:Form1->Shape1->Pen->Mode=pmCopy;
                break;
        case 5:Form1->Shape1->Pen->Mode=pmNotCopy;
                break;
```

```
        case 6:Form1->Shape1->Pen->Mode=pmMergePenNot;
                break;
        case 7:Form1->Shape1->Pen->Mode=pmMaskPenNot;
                break;
        case 8:Form1->Shape1->Pen->Mode=pmMergeNotPen;
                break;
        case 9:Form1->Shape1->Pen->Mode=pmMaskNotPen;
                break;
        case 10:Form1->Shape1->Pen->Mode=pmMerge;
                break;
        case 11:Form1->Shape1->Pen->Mode=pmNotMerge;
                break;
        case 12:Form1->Shape1->Pen->Mode=pmMask;
                break;
        case 13:Form1->Shape1->Pen->Mode=pmNotMask;
                break;
        case 14:Form1->Shape1->Pen->Mode=pmXor;
                break;
        case 15:Form1->Shape1->Pen->Mode=pmNotXor;
        }
        //设置画笔绘图模式
Randomize;
switch (random(7))
        {
        case 0:Form1->Shape1->Pen->Style=psSolid;
                break;
        case 1:Form1->Shape1->Pen->Style=psDash;
                break;
        case 2:Form1->Shape1->Pen->Style=psDot;
                break;
        case 3:Form1->Shape1->Pen->Style=psDashDot;
                break;
        case 4:Form1->Shape1->Pen->Style=psDashDotDot;
                break;
        case 5:Form1->Shape1->Pen->Style=psClear;
                break;

        case 6:Form1->Shape1->Pen->Style=psInsideFrame;
        }
        //设置画笔绘图风格
Randomize;
switch (random(6))
        {
        case 0:Form1->Shape1->Shape=stRectangle;
                break;
        case 1:Form1->Shape1->Shape=stSquare;
                break;
        case 2:Form1->Shape1->Shape=stRoundRect;
                break;
        case 3:Form1->Shape1->Shape=stRoundSquare;
                break;
        case 4:Form1->Shape1->Shape=stEllipse;
                break;
        case 5:Form1->Shape1->Shape=stCircle;
        }
        //设置组件形状
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
Form1->Shape1->Brush->Color=RGB(random(255),random(255),random(255));
Form1->Shape1->Brush->Style=bsBDiagonal;
Form1->Shape1->Pen->Color=RGB(random(255),random(255),random(255));
Form1->Shape1->Pen->Width=random(10)+1;
Form1->Shape1->Pen->Mode=pmMergeNotPen;
Form1->Shape1->Pen->Style=psInsideFrame;
Form1->Shape1->Shape=stRoundSquare;
//设置绘图参数
}
//---------------------------------------------------------------------------
```

4.4 小　结

本章系统地介绍了在 CBuilder 5 中进行文本、图形和图像处理的各种方法，其中“文本编辑器”、“字符艺术”、“图形动画”、“图像动态观察器”、“图像动画”和“艺术图案”等示例程序都由很大的实用性，在应用程序的设计过程中。

本章的程序设计方法有很大的参考价值，希望读者在理解的基础上，设计出更好的文本、图形和图像处理程序。

第五章　管理文件、目录和驱动器

在 Windows 的操作环境中，文件是一个非常重要的概念，所谓的文件是指存放在外部存储介质上的数据和程序等。
为了方便用户存取和操作的方便，在 CBuilder 5 中带有强大的文件管理功能，在其中用户可以通过调用文件处理函数来对文件和目录进行各种各样的处理和操作，如创建目录，打开和删除文件等操作。
在本章中，将讨论以下几个问题：

- 利用文件类的函数来进行文件内部的操作
- 如何使用常用的文件类控件
- 如何利用文件类控件来设计文件系统的界面

下面首先来介绍一下文件类函数（目录函数、驱动器函数和文件函数）的使用方法。

5.1　目录操作函数

在 CBuilder 5 的文件程序设计中，由于要经常的访问和处理存储在文件中的数据，所以文件类函数在程序设计中就显得十分的重要，在 CBuilder 5 中，系统为用户提供了很多文件类的函数，其中有对文件的操作函数、对目录进行操作的函数，还有访问磁盘的函数等等，下面就首先对目录函数加以介绍和说明。
在 CBuilder 5 的目录类函数中，常用的函数有 chdir()、mkdir()和 rmdir()，它们的功能和使用方法如下。

5.1.1　改变目录

chdir()函数的功能是更改 DOS 逻辑驱动器和缺省工作目录，它的语法如下：

```
int chdir(const char *path);
```

如果执行下面的程序段：

```
#include "dir.h"

void __fastcall TForm1::FormCreate(TObject *Sender)
{
chdir("c:\\Windows\\");
}
//---------------------------------------------------------------------
```

程序运行的结果是系统的工作目录更改为 c:\Windows。

5.1.2 生成目录

mkdir()函数的功能是创建一个新的子目录，它的语法如下：
int mkdir(const char *path);
如果执行以下的程序段：

```
#include "dir.h"

void __fastcall TForm1::FormCreate(TObject *Sender)
{
chdir("c:\\Windows\\");
//更改工作目录
mkdir("test");
//在当前的工作目录下创建一个子目录
}
//---------------------------------------------------------------------------
```

程序运行结果如图 5-1 所示。

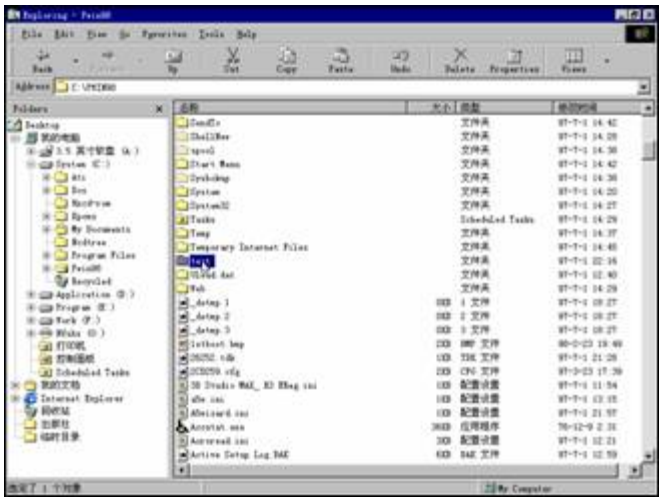


图 5-1　创建目录后的资源管理器

5.1.3 删除函数

rmdir()函数的功能是删除一个已经存在的目录，它的语法如下所示：

```
#include "dir.h"

int _rmdir(const char *path);
```

如果执行以下的程序段：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
chdir("c:\\Windows\\");
//更改工作目录
mkdir("test1");
mkdir("test2");
//在当前的工作目录下创建两个子目录
rmdir("test1");
//在当前的工作目录下删除一个子目录
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的初期，首先通过函数 ChDir()把 c:\Windows\设置为当前的工作目录，然后在当前的动作目录下创建了两个子目录 test1 和 test2，最后通过语句 rmdir("test1");，在当前的工作目录下删除一个子目录 test1。

做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序，程序的运行结果如图 5-2 所示。

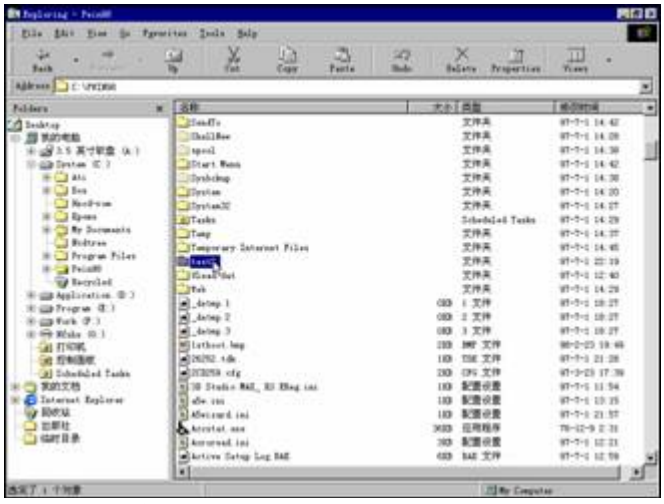


图 5-2　程序运行结果

5.2 驱动器操作函数

在 CBuilder 5 的驱动器操作函数中，常用的函数有 DiskSize()和 DiskFree()，它们的功能和使用方法如下。

5.2.1 显示磁盘容量

DiskSize()函数的功能是计算用户指定的驱动器的总容量，即所有的可用磁盘空间，它的语法如下：

```
extern PACKAGE __int64 __fastcall DiskSize(Byte Drive);
```

其中参数 Drive 的取值为 0 时表示当前驱动器，取 1 时表示 A 驱动器，取 2 时表示 B 驱动器，取 3 时表示 C 驱动器，其他依次类推。下面的示例就是利用 DiskSize()函数来达到获得计算机上驱动器信息的目的，编程的具体步骤如下所示。

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Memo 控件，并且在它的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 控件，属性设置如下所示：

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 291
   Height = 294
   Caption = 'Form1'
   OnCreate = FormCreate
   object Memo1: TMemo
      Left = 8
      Top = 8
      Width = 265
      Height = 249
   end
end
```

添加控件后的窗体如图 5-3 所示。

图 5-3　添加控件后的窗体

本示例程序的作用是获得硬盘上三个驱动器（C、D、E）的可用磁盘空间的信息，所以在程序变量的声明中要定义 3 个字符串变量 temp1、temp2、temp3，其中依次存储 C、D、E 驱动器的可用磁盘空间，如下所示：

```
String temp1;
String temp2;
String temp3;
```

在程序的设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，在其中添加如下所示的代码，用于获得和显示驱动器的信息。

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
temp1=IntToStr(DiskSize(3));
temp1="C 驱动器的可用空间："+temp1+"Byte"+"\n\n";
temp2=IntToStr(DiskSize(4));
temp2="D 驱动器的可用空间："+temp2+"Byte"+"\n\n";
temp3=IntToStr(DiskSize(5));
temp3="E 驱动器的可用空间："+temp3+"Byte"+"\n\n";
//获得驱动器信息
Form1->Memo1->Text=temp1+temp2+temp3;
}
//---------------------------------------------------------
```

最后，存储文件，按键盘上的功能键【F9】运行程序，程序的运行结果如图 5-4 所示。

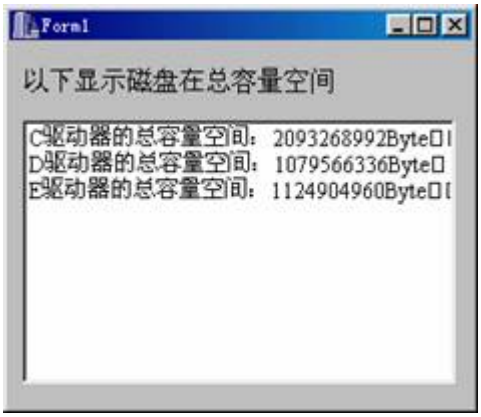


图 5-4　程序运行结果

5.2.2　显示剩余空间

DiskSize()函数的功能是计算用户指定的驱动器的剩余磁盘空间，它的语法结构如下所示：

```
extern PACKAGE __int64 __fastcall DiskFree(Byte Drive);
```

其中参数 Drive 的取值为 0 时表示当前驱动器，取 1 时表示 A 驱动器，取 2 时表示 B 驱动器，取 3 时表示 C 驱动器，其他依次类推。

下面的示例就是利用 DiskFree()函数来达到获得计算机上驱动器剩余磁盘空间信息的目的，本示例程序的基本框架是基于 DlskSize()函数的示例程序，但是在其中的代码略有变化，代码如下所示。

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
temp1=IntToStr(DiskFree(3));
temp1="C  盘的剩余磁盘空间："+temp1+"Byte"+"\n\n";
temp2=IntToStr(DiskFree(4));
```

```
temp2="D 盘的剩余磁盘空间："+temp2+"Byte"+"\n\n";
temp3=IntToStr(DiskFree(5));
temp3="E 盘的剩余磁盘空间：  "+temp3+"Byte"+"\n\n";
//获得驱动器信息
Form1->Memo1->Text=temp1+temp2+temp3;
}
//---------------------------------------------------------------------
```

存储文件，按键盘上的功能键 F9 运行程序，程序运行结果如图 5-5 所示。

图 5-5　显示驱动器剩余磁盘空间的信息

5.3　文件操作函数

在 CBuilder 5 的文件操作函数中，常用的函数有 DeleteFile()、RenameFile()和 ExpandFileName()、ExtractFileName()和 ExtractFileExt()，它们的功能和使用方法如下。

5.3.1　DeleteFile()函数

DeleteFile()函数的功能是删除有程序中指定的文件，它的语法结构如下所示：

```
extern PACKAGE bool __fastcall DeleteFile(const System::AnsiString FileName);
```

如执行下列代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
chdir("c:\\Windows\\temp");
//设置系统的工作路径
DeleteFile("temp.txt");
//删除指定的文件
}
//---------------------------------------------------------------------
```

将会删除目录 c:\Windows\temp 下的 temp.txt 文件。

5.3.2 RenameFile()函数

RenameFile()函数的作用是更改指定文件的名称，但是在程序中要指定文件的原名和目的名，它的语法结构如下所示：

extern PACKAGE bool __fastcall RenameFile(const AnsiString OldName, const AnsiString NewName);

如执行下列代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
chdir("c:\\Windows\\temp");
//设置系统的工作路径
RenameFile("temp.txt","test.txt");
//更改文件名
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的初期，窗体首先被加载，就会自动的激活窗体的 FormCreate()事件，设置系统的工作路径为 c:\Windows\temp，然后通过语句 RenameFile("temp.txt","test.txt");将当前系统工作目录下的 temp.txt 重新命名为 test.txt。

5.3.3 ExpandFileName()函数

ExpandFileName()函数的作用是返回文件的完整的路径名和文件名，它的语法结构如下所示：

extern PACKAGE AnsiString __fastcall ExpandFileName(const AnsiString FileName);

如果在程序的设计阶段，向窗体上添加一个 Edit 控件和一个 FileListBox 控件，添加控件的属性设置如下所示：

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 283
   Height = 314
   object FileListBox1: TFileListBox
      Left = 8
      Top = 8
      Width = 257
      Height = 241
      OnClick = FileListBox1Click
   end
   object Edit1: TEdit
      Left = 8
      Top = 256
      Width = 257
      Height = 24
   end
end
```

添加控件后的窗体如图 5-6 所示。

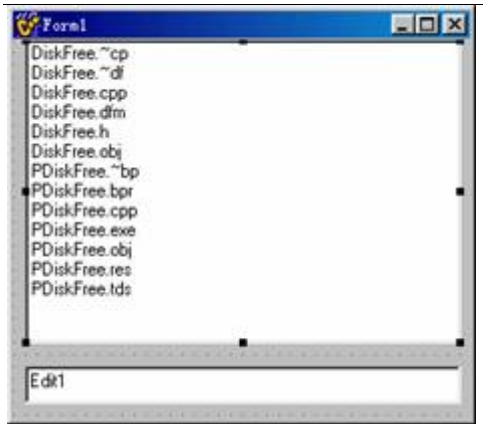


图 5-6　添加控件后的窗体

然后在控件 FileListBox1 的事件列表中用鼠标的左键双击 FileListBox1Click()事件，在弹出的代码窗口中添加如下所示的响应代码：

```
void __fastcall TForm1::FileListBox1Click(TObject *Sender)
{
Edit1->Text=ExpandFileName(Form1->FileListBox1->FileName);
//显示完整文件名
}
//---------------------------------------------------------------------------
```

选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，按键盘上的功能键 F9 运行程序，结果如图 5-7 所示。

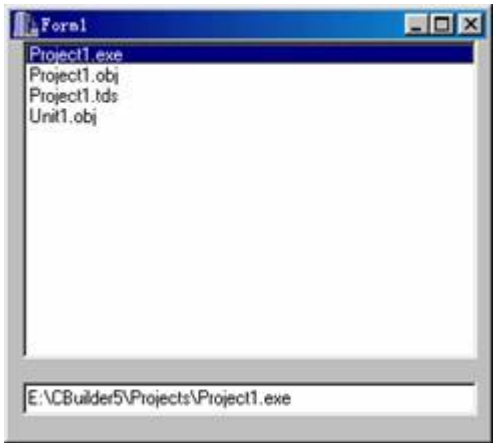


图 5-7　程序运行结果

在程序的运行过程中，用鼠标的左键单击文件列表框中的文件时，在窗体下方的文本框中就会显示出这个文件的完整路径和文件名。

5.3.4 ExtractFileName()函数

ExtractFileName()函数的作用是返回指定文件的文件名，与 ExpandFileName 函数有所不同，ExtractFileName()函数的返回值不包括文件的路径，如果在上例程序中的代码做如下的改动：

```
void __fastcall TForm1::FileListBox1Click(TObject *Sender)
{
Edit1->Text=ExtractFileName(Form1->FileListBox1->FileName);
//仅仅显示文件名
}
//---------------------------------------------------------------------
```

存储文件，运行程序，结果如图 5-8 所示。

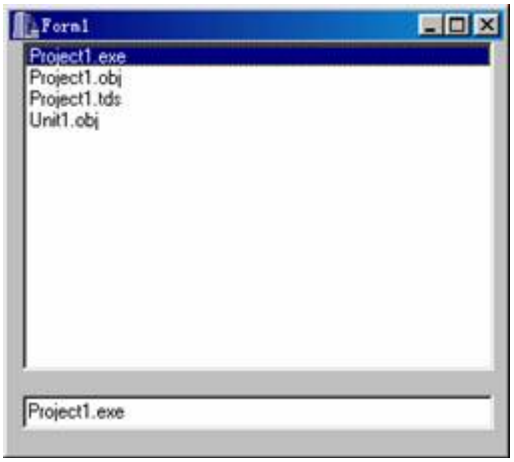


图 5-8　在程序中显示文件名

在程序的运行过程中，当用户在文件列表框中单击某个文件时，在窗体下方的文本输入框中就会显示出选中文件的文件名。

5.3.5 ExtractFileExt()函数

ExtractFileExt()函数的作用是返回指定文件的扩展名，与**ExtractFileName()**函数不同的是，利用 **ExtractFileExt()**函数，只能够得到文件的扩展名，如果在上例程序中对代码做如下改动：

```
void __fastcall TForm1::FileListBox1Click(TObject *Sender)
{
Edit1->Text=ExtractFileExt(Form1->FileListBox1->FileName);
//仅仅显示文件扩展名
}

//---------------------------------------------------------------------
```

那么在程序的运行过程中，文本输入框中就会显示选中文件的扩展名，运行结果如图 5-9 所示。

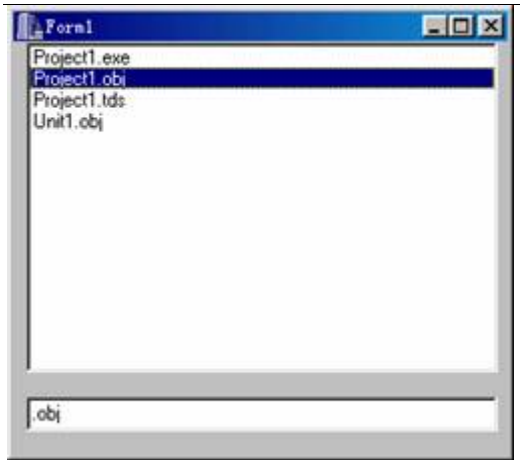


图 5-9　在程序中显示文件名

5.4　相关控件概述

CBuilder 5 中的文件类控件包括 DirectoryListBox 控件（目录列表框控件）、FileListBox 控件（文件列表框控件）、DriveComboBox 控件（驱动器组合列表框控件）和 FilterComboBox 控件（过滤组合列表框控件）四种。

5.4.1 驱动器类控件

DriveComboBox 控件（驱动器组合列表框控件）的功能是显示当前计算机中全部有效的驱动器，以供用户选择和操作，它的常用的属性如表 5-1 所示。

表 5-1 DriveComboBox 控件常用属性

Anchors	Color	Constraints
Ctl3D	Cursor	DirList
DragCursor	DragMode	Enabled
Font	Height	HelpContext
Hint	ImeMode	ImeName
Left	Name	ParentColor
ParentCtl3D	ParentFont	ParentShowHint
PopupMenu	ShowHint	TabOrder
TabStop	Tag	TextCase
Top	Visible	Width

DriveComboBox 控件的属性在这里就不一一的加以介绍了，下面以一个示例来说明 DriveComboBox 控件和它的属性的用法，具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Win 3.1 选项后，在 Drive ComboBox 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Drive ComboBox 控件，如图 5-10 所示。

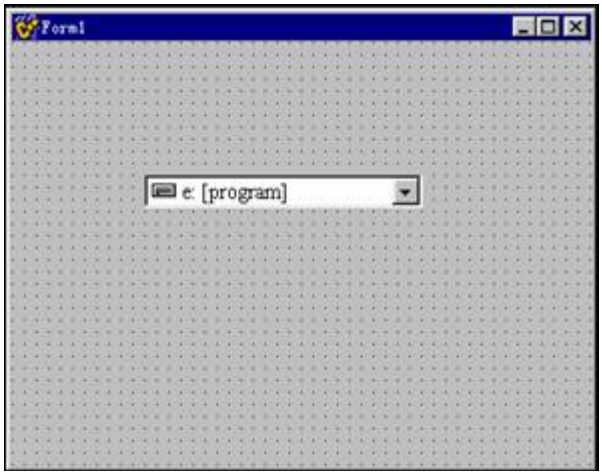


图 5-10 添加控件后的窗体

2. 添加代码

添加控件后，用鼠标的左键在窗体上的空白处双击，就会弹出一个代码窗口，用户可以在其中添加自己需要的代码。把光标移动到代码窗口中的适当位置，找到需要添加代码的事件，如窗体的 FormCreate()事件。读者可以参考下面的程序清单在其中添加程序代码：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
```

```
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------


void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->DriveComboBox1->Drive='d';
//设置驱动器为 D 盘
Form1->DriveComboBox1->TextCase=tcUpperCase;
//设置显示的字母为大写字母

}
//---------------------------------------------------------------------

void __fastcall TForm1::DriveComboBox1Click(TObject *Sender)
{
String str1;
//定义字符串变量
str1="您选择的是";
str1=str1+Form1->DriveComboBox1->Drive;
str1=str1+"盘";
//设置显示信息
ShowMessage(str1);
//显示用户选择
}
//---------------------------------------------------------------------
```

程序说明：

DriveComboBox 控件的 Drive 用于设置控件显示的驱动器的盘符，所以语句 Form1->DriveComboBox1->Drive='d'；的作用是设置控件显示的驱动器为 D 盘；而 TextCase 属性用于设置 DriveComboBox 控件中的字符显示形式，所以语句 Form1->DriveComboBox1->TextCase=tcUpperCase；的作用是设置控件显示的字符为大写字符。

在以上的程序段中，我们通过语句 Form1->DriveComboBox1->TextCase=tcUpperCase;来设置 DriveComboBox 控件中显示的字符为大写字符，那么可不可以通过程序来设置 DriveComboBox 控件中显示为小写字符呢？

与设置显示字符为大写字符一样，设置 DriveComboBox 控件中显示为小写字符的语句为：

```
Form1->DriveComboBox1->TextCase=tcLowerCase;
//设置控件显示字符为小写字符
```

3. 运行程序

做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序，程序的运行结果如图 5-11 所示。

图 5-11　程序运行结果

5.4.2　目录类控件

DirectoryListBox 控件（目录列表框控件）的功能是显示当前计算机有效驱动器中当前驱动器的树形目录结构，以供用户选择和操作，用户在选择的过程中可以在同一个驱动器的不同目录之间进行切换等操作，它的常用的属性如表 5-2 所示。

表 5-2　DirectoryListBox 控件的常用属性

Align	Anchors	Color
Columns	Constraints	Ctl3D
Cursor	DirLabel	DragCursor
DragMode	Enabled	FileList
Font	Height	HelpContext
Hint	ImeMode	ImeName
IntegralHeight	ItemHeight	Left
Name	ParentColor	ParentCtl3D
ParentFont	ParentShowHint	PopupMenu
ShowHint	TabOrder	TabStop
Tag	Top	Visible
Width		

DirectoryListBox 控件的属性在这里就不一一的加以介绍了，如果读者有什么不清楚的地方，请参考 DriveComboBox 控件即可。

下面以一个示例来说明 DirectoryListBox 控件控件和它的属性的用法，具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体。

在控件工具栏上选择 Win3.1 选项后，在 DirectoryListBox 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 DirectoryListBox 控件。

然后在控件工具栏上选择 Standard 选项，在 Label 控件的图标上双击鼠标的左键，在窗体上就会添加一个 Label 控件。

在窗体上添加控件后，下一步的工作就是为窗体和控件设置属性值，在本程序中，控件的属

性设置如下所示。

```
  object Label1: TLabel
    Left = 0
    Top = 240
    Width = 160
    Height = 16
    Caption = 'E:\Cbuilder5\Cbuilder5\Bin'
  end
  object DirectoryListBox1: TDirectoryListBox
    Left = 0
    Top = 0
    Width = 363
    Height = 225
    Align = alTop
    DirLabel = Label1
  end
end
```

以上属性设置后的控件具有如下的特性：

- DirectoryListBox 控件的始终位于窗体的顶部，不会因窗体的位置和大小改变；
- 两个 VCL 控件都处于有效的状态；
- Label 控件的大小会随着控件中的内容而改变；
- 与 DirectoryListBox 控件相关联的控件是 Label 控件。

设置属性后的窗体与控件如图 5-12 所示。

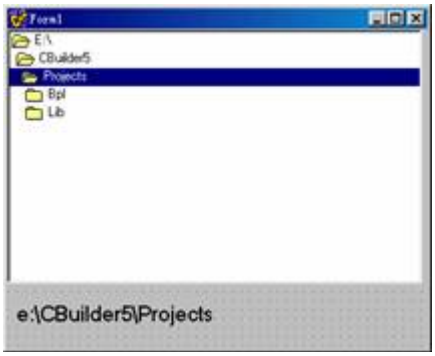


图 5-12　设置属性后的窗体

当然，控件和窗体的属性设置不但可以在程序的设计过程中由用户手动设置，还可以在程序的运行过程中有程序控制，如以上的两个控件的属性设置可以通过以下的程序段来实现。

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->DirectoryListBox1->Align=alTop;
//DirectoryListBox 控件始终位于窗体的顶部
Form1->DirectoryListBox1->DirLabel=Form1->Label1;
//与 DirectoryListBox1 相关联的控件为 label1
Form1->DirectoryListBox1->Drive='c';
//设置 DirectoryListBox1 驱动器
chdir("c:\\windows");
//设置系统的缺省路径为 c:\windows\
Form1->DirectoryListBox1->Update();
//显示更改结果
}
//---------------------------------------------------------------------------
```

2. 运行程序

做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序，程序的运行结果如图 5-13 所示。

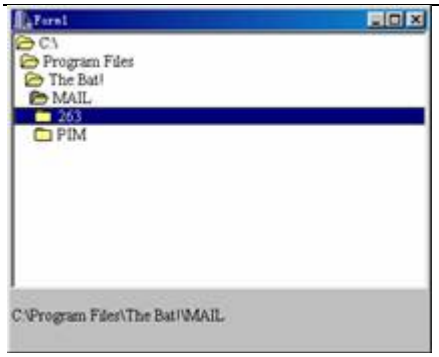


图 5-13　程序运行结果

在程序的运行过程中，可以选择不同的目录，同时在 Label 控件中会显示出用户在程序运行过程中所选择的路径。

5.4.3　文件列表和文件过滤控件

CBuilder 5 中常用的文件类控件中最后一种控件为 FilterComboBox 控件，作用是为用户提供从文件列表框中筛选文件的条件，它的作用决定了 FilterComboBox 控件要同其他几种文件类控件相互协调才能够完成文件的筛选功能，FilterComboBox 控件的常用属性如表 5-3 所示。

表 5-3　FilterComboBox 控件的常用属性

Action	ActiveControl	Align	Anchors
AutoScroll	AutoSize	BiDiMode	BorderIcons
BorderStyle	BorderWidth	Caption	ClientHeight
ClientWidth	Color	Constraints	Ctl3D
Cursor	DefaultMonitor	DockSite	DragKind
DragMode	Enabled	Font	FormStyle

（续表）

Height	HelpContext	HelpFile	Hint
HorzScrollBar	Icon	KeyPreview	Left
Menu	Name	ObjectMenuItem	OldCreateOrder
ParentBiDiMode	ParentFont	PixelsPerInch	PopupMenu
Position	PrintScale	Scaled	ShowHint
Tag	Top	UseDockManager	VertScrollBar
Visible	Width	WindowMenu	WindowState

FilterComboBox 控件的属性在这里就不一一的加以介绍了，如果读者有什么不清楚的地方，请参考有关的参考手册，下面以一个示例来说明 FilterComboBox 控件和它的属性的用法，另外还介绍了如何使 FilterComboBox 控件与 FileListBox 控件相互协调工作。制作示例程序的具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，选择 FileListBox 控件和 FilterComboBox 控件，并且把它们拖放到空白的窗体上，添加控件后的窗体如图 5-14 所示。

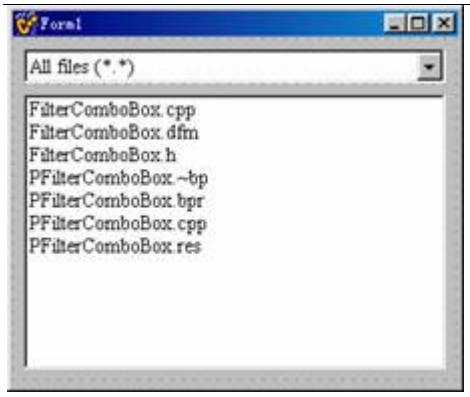


图 5-14　添加控件后的窗体

2. 设置控件属性

在程序的设计阶段，用鼠标的左键单击 FilterComboBox 控件属性列表中的 Filter 属性设置框中的按钮…，就会弹出一个如图 5-15 所示的过滤器设置对话框，用户可以在其中设置文件显示的过滤器。

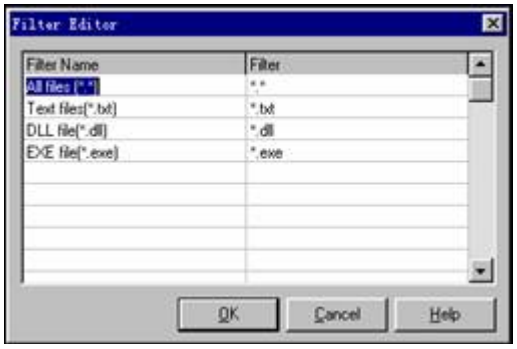


图 5-15　过滤器设置

在本程序中，过滤器的属性设置为 All files (*.*)|*.*|Text files(*.txt)|*.txt|DLL file(*.dll)|*.dll|EXE file(*.exe)|*.exe，也就是说，在程序的运行过程中，用户在显示文件类型上有 4 种选择：

- 显示全部文件；
- 显示以*.txt 结尾的文本文件；
- 显示以*.dll 结尾的动态链接库文件；
- 显示以*.exe 结尾的可执行文件。

3. 响应窗体事件

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，在缺省的情况下，代码窗口中显示的是窗体的 FormCreate()事件，将光标移动代码窗口中，并且添加下列代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
FilterComboBox1->FileList=FileListBox1;
//设置关联控件
FileListBox1->Drive='c';
//设置控件的驱动器
FileListBox1->Directory="c:\\Windows\\system\\";
//设置控件的文件路径
}
//---------------------------------------------------------------------
```

窗体的 FormCreate()事件在窗体的装入的同时就会被激活，进入到窗体的 FormCreate()事件

的过程中，首先通过语句 FilterComboBox1->FileList=FileListBox1;设置 FileListBox1 控件为 FilterComboBox 控件的关联控件，然后设置文件列表框控件的文件路径为 c:\Windows\system。

4. 运行程序

做完以上的工作后，存储文件，按键盘上的功能键【F9】运行程序，程序的运行结果如图 5-16 所示。

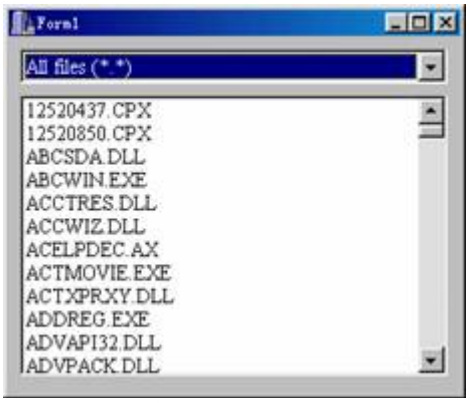


图 5-16　程序运行结果

在程序的运行过程中，用户可以选择显示文件的类型，如图 5-17 所示即为显示为以*.dll 结尾的可执行文件，当然用户也可以选择其他的文件筛选方式，如显示所有的以*.exe 结尾的动态链接库文件等。

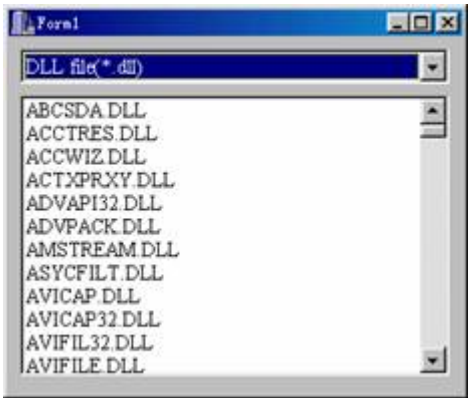


图 5-17　显示“*.dll”文件

5.5 利用文件类控件

前面分别的介绍了文件类控件中常用的四种 VCL 控件，但是到现在为止，各个控件之间还是独立运行，还没有使它们协调的工作，下面将通过一个示例程序来说明如何使得四个文件类控件协调工作。
综合利用文件类控件制作文件处理程序的具体步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 FileListBox 控件、Edit 控件、DriveComboBox 控件、DirectoryListBox 控件和 FilterComboBox 控件，并且把它们拖放到空白的窗体上，添加控件后的窗体如图 5-18 所示。

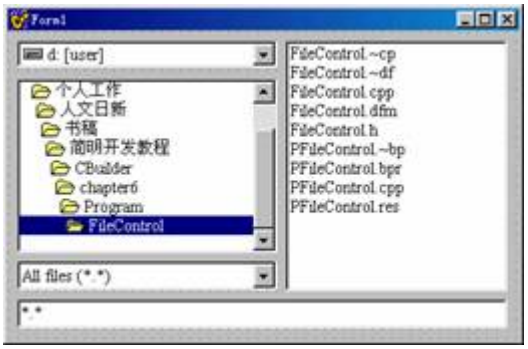


图 5-18　添加控件后的窗体
其中各个控件的作用如下所示：

- DriveComboBox 控件的作用是显示当前计算机中全部有效的驱动器，以供用户选择和操作；
- DirectoryListBox 控件的作用是显示当前计算机有效驱动器中当前驱动器的树形目录结构，以供用户选择和操作；
- FileListBox 控件的作用是显示当前路径中符合条件的所有文件列表，以供用户选择和操作；
- FilterComboBox 控件的作用是为用户提供从文件列表框中筛选文件的条件；
- Edit 控件的作用是显示选中文件的路径和文件名。

2. 控件属性设置

添加控件后的工作就是设置控件的属性，在设置属性的过程中，我们要力争发挥各个控件的作用，如果能够在控件的属性设置过程中充分的发挥控件的独特作用，那么就可以减少很多编程的工作量。
在本示例程序中各个控件的属性设置如下所示：

```
object DriveComboBox1:
        TDriveComboBox
    Left = 8
    Top = 8
    Width = 217
    Height = 22
    OnChange = DriveComboBox1Change
end
object DirectoryListBox1:
        TDirectoryListBox
    Left = 8
```

```
    Top = 40
    Width = 217
    Height = 145
    OnChange = DirectoryListBox1Change
  end
  object FileListBox1: TFileListBox
    Left = 232
    Top = 8
    Width = 185
    Height = 209
    OnChange = FileListBox1Change
  end
  object FilterComboBox1:
    TFilterComboBox
    Left = 8
    Top = 192
    Width = 217
    Height = 24
    Filter = 'All files (*.*)|*.*|DLL
    files(*.dll)|*.dll|Text files(*.txt)|*.t' +
    'xt|BMP flies(*.bmp)|*.bmp|ICO
    files(*.ico)|*.ico'
  end
  object Edit1: TEdit
    Left = 8
    Top = 224
    Width = 409
    Height = 24
    Text = '*.*'
  end
```

3. 添加响应驱动器改变代码

如果想要在程序的运行过程中，动态的跟踪驱动器的改变，就要在 DriveComboBox 控件的 DriveComboBox1Change()事件中添加响应驱动器改变的代码。在程序的设计过程中，用鼠标的左键双击控件 DriveComboBox1，在屏幕上就会弹出一个代码窗口。在缺省的情况下，代码窗口中显示的是 DriveComboBox 控件的 DriveComboBox1Change()事件代码段，把光标移动到 DriveComboBox1Change()事件的代码段中，添加如下代码：

```
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
Form1->DirectoryListBox1->Drive=Form1->DriveComboBox1->Drive;
//设置控件 DirectoryListBox1 的驱动器
}
//---------------------------------------------------------------------
```

程序说明：

当程序运行过程中，当用户改变控件 DriveComboBox1 中的驱动器时，就会激活 DriveComboBox1 控件的 DriveComboBox1Change()，然后通过语句 Form1->Directory ListBox1->Drive=Form1->DriveComboBox1->Drive;来动态的跟踪系统驱动器的变化。

4. 添加响应目录改变代码

在程序的设计过程中，用鼠标的左键双击控件 DirectoryListBox1，在屏幕上就会弹出一个代码窗口，代码窗口中显示的是 DirectoryListBox 控件的 DirectoryListBox1Change()事件代码段，把光标移动到 DirectoryListBox1Change()事件的代码段中，添加如下代码：

```
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
```

```
{
Form1->FileListBox1->Directory=Form1->DirectoryListBox1->Directory;
//设置 FileListBox1 控件的目录
}
//---------------------------------------------------------------------------
```

5. 添加其余代码

当然，仅仅依靠前面所添加的代码是不够的，还需要添加响应建立窗体等事件的代码，如下所示：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
chdir("c:\\Windows\\");
//初始化 DriveComboBox1 控件
Form1->DriveComboBox1->Left=56;
Form1->DriveComboBox1->Top=56;
Form1->DriveComboBox1->Width=217;
Form1->DriveComboBox1->Height=19;
//设置控件 DriveComboBox1 的位置与大小
……
//初始化控件 Edit1
Form1->Edit1->Left=56;
Form1->Edit1->Top=272;
Form1->Edit1->Width=409;
Form1->Edit1->Height=21;
//设置控件 Edit1 的位置与大小
}
//---------------------------------------------------------------------------
```

注意：

✍ 由于篇幅的限制，在这里就不对代码一一的加以说明，详细的代码可以参看附后的源程序代码。

6. 运行程序

做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序，程序的运行结果如图 5-19 所示。

图 5-19　程序运行结果

以上程序的源程序代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
```

```
#include <dir.h>
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
chdir("c:\\Windows\\");
//初始化 DriveComboBox1 控件
Form1->DriveComboBox1->Left=56;
Form1->DriveComboBox1->Top=56;
Form1->DriveComboBox1->Width=217;
Form1->DriveComboBox1->Height=19;
//设置控件 DriveComboBox1 的位置与大小
Form1->DriveComboBox1->Drive='c';
//设置控件 DriveComboBox1 的初始驱动器
//Form1->DriveComboBox1->DirList=Form1->DirectoryListBox1;
//设置 DriveComboBox1 控件的关联控件为 DirectoryListBox1
//初始化 DirectoryListBox1 控件
Form1->DirectoryListBox1->Left=56;
Form1->DirectoryListBox1->Top=88;
Form1->DirectoryListBox1->Width=217;
Form1->DirectoryListBox1->Height=145;
//设置控件 DirectoryListBox1 的位置与大小
Form1->DirectoryListBox1->Directory="c:\\Windows\\";
//设置控件 DirectoryListBox1 的初始工作目录为 c:\Windows
//Form1->DirectoryListBox1->FileList=Form1->FileListBox1;
//设置 DirectoryListBox1 控件的关联控件为 FileListBox1
//初始化 FileListBox1 控件
Form1->FileListBox1->Left=280;
Form1->FileListBox1->Top=56;
Form1->FileListBox1->Width=185;
Form1->FileListBox1->Height=209;
//设置控件 FileListBox1 的位置与大小
Form1->FileListBox1->FileEdit=Form1->Edit1;
//设置 FileListBox1 控件的关联控件为 Edit1
//初始化 FilterComboBox 控件
Form1->FilterComboBox1->Left=56;
Form1->FilterComboBox1->Top=240;
Form1->FilterComboBox1->Width=217;
Form1->FilterComboBox1->Height=21;
//设置控件 FilterComboBox1 的位置与大小
Form1->FilterComboBox1->FileList=Form1->FileListBox1;
//设置 FileListBox1 控件为 FilterComboBox1 控件的关联控件
//初始化控件 Edit1
Form1->Edit1->Left=56;
Form1->Edit1->Top=272;
Form1->Edit1->Width=409;
Form1->Edit1->Height=21;
//设置控件 Edit1 的位置与大小
```

```
}
//---------------------------------------------------------------------

void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
Form1->DirectoryListBox1->Drive=Form1->DriveComboBox1->Drive;
//设置控件 DirectoryListBox1 的驱动器
}
//---------------------------------------------------------------------

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
Form1->FileListBox1->Directory=Form1->DirectoryListBox1->Directory;
//设置 FileListBox1 控件的目录
}
//---------------------------------------------------------------------

void __fastcall TForm1::FileListBox1Change(TObject *Sender)
{
Edit1->Text=FileListBox1->Directory+"\\"+Edit1->Text;
//显示文件路径和文件名
}
//---------------------------------------------------------------------
```

5.6 小 结

文件处理在 Windows 的程序设计中一直是一个比较活跃的领域，同样，在 CBuilder 5 的程序设计中，也把文件处理放在一个比较重要的位置，可以说，文件处理是进行大的应用程序的基础。

在本章中，介绍了在 CBuilder 5 中进行程序设计的几种常用的方法，如通过文件类控件设计文件处理应用程序、调用系统函数来进行系统的目录和文件的维护等，特别的，编者为读者准备了两个文件格式转换的示例程序，希望读者在理解的基础上，设计出有个人特色的文件处理应用程序。

第六章　用户窗体的设计和使用

在 CBuilder 中，常见的用户窗体开发包括三个方面的内容：普通文档界面窗体（SDI）、多页窗体设计和复合窗体设计。
普通文档窗体相对来说比较容易让人理解，就是应用程序运行的过程中在同一个时间内只能打开一个文档来进行处理；多页窗体指的是在一个窗体上同时包含多个页面，通过单击页面的标签可以让其中任意一页在窗体的最前面显示，每一页可以象设计普通窗体那样设计；复合窗体程序指的是在一个应用程序中有多个窗口界面，按照程序的要求分别的显示在屏幕上，但是要对每一个窗体进行单独的设计。
下面就对以上提到的三种窗体编程技术分别的加以介绍。

6.1　普通文档窗体设计

SDI 界面的一个典型例子就是“记事本”应用程序如图 6-1 所示。在“记事本”应用程序中，在同一个时间内只能打开一个文本文件，想要打开另一个文本文件时，就必须先关闭以前打开的文本文件。

图 6-1　一个单文档界面（SDI）应用程序
在这里我们就通过一个简单的示例程序向读者说明，在 CBuilder 5 中如何设计一个 SDI 应用程序，这个示例程序可以对文本进行简单的处理，具体的程序设计步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，向窗体上添加一个 ActionList 控件、一个 Memo 控件和一个 MainMenu 控件，然后在控件工具栏上选择 Win32 选项，并且向窗体上添加一个 ToolBar 控件、一个 StatusBar 控件和一个 ImageList 控件，在控件工具栏上选择 Additional 选项，向窗体上添加七个 SpeedButton 控件，最后选择控件工具栏上的 Dialogs 选项，向窗体上添加一个 OpenDialog 控件和一个 SaveDialog 控件，各个控件的功能将在后面分别的加以介绍。添加控件后的窗体如图 6-2 所示。

图 6-2　添加控件后的窗体

添加到窗体上各个控件的作用如下所示：

- StatusBar 控件：在程序运行过程中显示应用程序当前的状态和各控件的提示文本；
- MainMenu 控件：为程序运行提供菜单设计的容器；
- ActionList 控件：为程序运行提供一些标准和非标准操作；
- ImageList 控件：在其中装有一些图像文件，可以显示显示在菜单和工具栏中；
- OpenDialog 控件：显示一个对话框，在其中可以选择一个待打开的文本文件；
- SaveDialog 控件：显示一个对话框，在其中可以选择一个待存储的文本文件；
- 工具栏控件：在其中可以设计菜单的快捷操作方式。

2. 菜单设计

在程序设计的过程中用鼠标双击窗体上的 MainMenu 控件，在窗体上就会弹出一个菜单编辑窗口，读者可以参看图 6-3 来进行菜单设计。

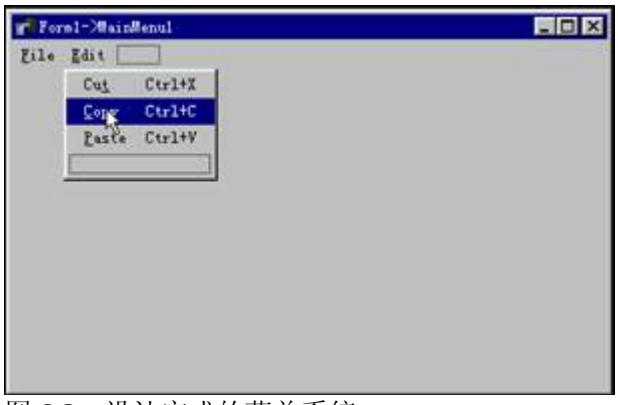


图 6-3　设计完成的菜单系统

其中添加到窗体上的 File 菜单中各个子菜单项的作用如下所示：

- New 菜单：创建一个新的文件；
- Open 菜单：显示一个打开文件的对话框，并且可以根据用户的选择打开一个文本文件；
- Save 菜单：显示一个存储文件的对话框，并且可以根据用户的选择保存一个文本文件。

在 Edit 主菜单下包含有三个子菜单，它们是 Cut、Copy 和 Paste，它们的作用如下所示：

- Cut 菜单：将当前选中的内容剪切到剪贴板上；
- Copy 菜单：将当前选中的内容复制到剪贴板上；
- Paste 菜单：将当前剪贴板上的内容放置到当前位置上。

3. ActionList 控件

添加到窗体上的 ActionList 控件的作用是在程序中为设计人员提供一些标准的操作，如 TDataSetFirst、TDataSetPrior、TDataSetNext、TDataSetLast、TDataSetInsert、TDataSetDelete、TDataSetEdit、TDataSetPost、TDataSetCancel、TDataSetRefresh、TEditCut、TEditCopy、TEditPaste、TWindowClose、TWindowCascade、TWindowTileHorizontal、TWindowTileVertical、TwindowMinimizeAll 和 TWindowArrange 等。
在程序设计的过程中，用户可以随时的向当前的应用程序中添加一些标准操作。
在本示例程序的设计过程中，用鼠标双击窗体上的 ActionList 控件，在窗体上就会弹出一个如图 6-4 所示的对话框。

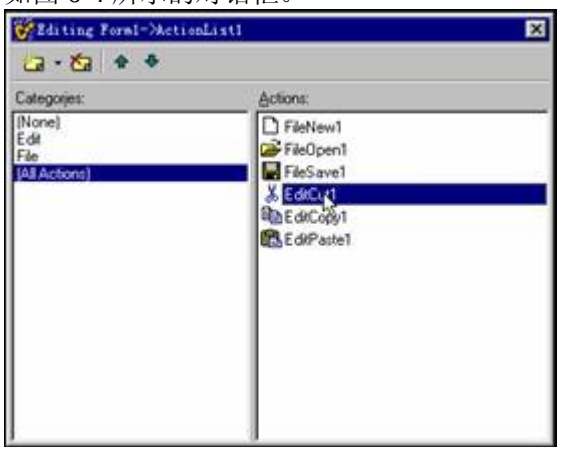


图 6-4　添加标准操作

单击工具栏上 New Action 选项右侧的下三角，就会弹出一个 New Standard Action 选项，单击这个选项，在窗体上就会弹出一个显示所有标准操作的对话框，在其中添加 TEditCut、TEditCopy 和 TEditPaste 三个标准操作，并且添加 FileNew、FileOpen 和 FileSave 三个用户自定义的操作。
TEditCut、TEditCopy 和 TEditPaste 三个标准操作的响应代码已经封装在 ActionList 控件中，不需要用户自己编写，但是 FileNew、FileOpen 和 FileSave 三个操作的响应代码却需要用户自己添加，如下所示：

```
void __fastcall TForm1::FileNew1Execute(TObject *Sender)
{
Form1->Memo1->Clear();
//清除文本框中的内容
}
//---------------------------------------------------------------------

void __fastcall TForm1::FileOpen1Execute(TObject *Sender)
{
Form1->OpenDialog->Title="请选择一个文本文件:";
//设置对话框标题
Form1->OpenDialog->InitialDir="c:\\pwin98";
//设置对话框缺省标题
Form1->OpenDialog->Filter="All files(*.*)|*.*|Text files(*.txt)|*.txt";
//设置文件过滤器
   if (OpenDialog->Execute())
        //显示一个对话框
        {
        Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog->FileName);
        //显示指定文件的内容
        }
}
//---------------------------------------------------------------------
```

```
void __fastcall TForm1::FileSave1Execute(TObject *Sender)
{
Form1->SaveDialog->Title="请选择一个文本文件:";
//设置对话框标题
Form1->SaveDialog->InitialDir="c:\\pwin98";
//设置对话框缺省标题
Form1->SaveDialog->Filter="All files(*.*)|*.*|Text files(*.txt)|*.txt";
//设置文件过滤器
   if (Form1->SaveDialog->Execute())
        {
        Form1->Memo1->Lines->SaveToFile(Form1->SaveDialog->FileName);
        //存储文件
        }
}
//---------------------------------------------------------------------
```

在这里就不对以上程序代码做过多的解释了，如果读者有什么不清除的地方，请参看前面几章的有关内容。

4. 响应窗体事件

在程序设计的过程中，用鼠标左键双击窗体上的空白处，在弹出的代码窗口中选择窗体的 FormCreate 事件，并且添加如下所示的响应代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->FileNewItem->OnClick=Form1->FileNew1->OnExecute;
Form1->FileOpenItem->OnClick=Form1->FileOpen1->OnExecute;
Form1->FileSaveItem->OnClick=Form1->FileSave1->OnExecute;
//关联 File 菜单事件
Form1->CutItem->OnClick=Form1->EditCut1->OnExecute;
Form1->CopyItem->OnClick=Form1->EditCopy1->OnExecute;
Form1->PasteItem->OnClick=Form1->EditPaste1->OnExecute;
//关联 Edit 菜单事件
Form1->ToolButton0->OnClick=Form1->FileNewItem->OnClick;
Form1->ToolButton1->OnClick=Form1->FileOpenItem->OnClick;
Form1->ToolButton2->OnClick=Form1->FileSaveItem->OnClick;
Form1->ToolButton4->OnClick=Form1->CutItem->OnClick;
Form1->ToolButton5->OnClick=Form1->CopyItem->OnClick;
Form1->ToolButton6->OnClick=Form1->PasteItem->OnClick;
//关联工具栏事件
}
//---------------------------------------------------------------------
```

5. 运行程序

按照附后的程序清单添加剩余的程序代码。做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序。在程序运行的过程中，用户即可以通过菜单来进行各种文件操作，如新建一个文本文件、打开文本文件和存储文件等，也可以进行剪切、复制和粘贴等编辑操作，当然还可以通过工具栏上快捷按钮直接进行操作。程序运行结果如图 6-5 所示。

图 6-5　程序运行结果

SDI 程序设计的完整代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "SDI.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FileNew1Execute(TObject *Sender)
{
Form1->Memo1->Clear();
//清除文本框中的内容
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FileOpen1Execute(TObject *Sender)
{
Form1->OpenDialog->Title="请选择一个文本文件:";
//设置对话框标题
Form1->OpenDialog->InitialDir="c:\\pwin98";
//设置对话框缺省标题
Form1->OpenDialog->Filter="All files(*.*)|*.*|Text files(*.txt)|*.txt";
//设置文件过滤器
   if (OpenDialog->Execute())
        //显示一个对话框
        {
        Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog->FileName);
        //显示指定文件的内容
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FileSave1Execute(TObject *Sender)
{
```

```
Form1->SaveDialog->Title="请选择一个文本文件:";
//设置对话框标题
Form1->SaveDialog->InitialDir="c:\\pwin98";
//设置对话框缺省标题
Form1->SaveDialog->Filter="All files(*.*)|*.*|Text files(*.txt)|*.txt";
//设置文件过滤器
   if (Form1->SaveDialog->Execute())
       {
       Form1->Memo1->Lines->SaveToFile(Form1->SaveDialog->FileName);
       //存储文件
       }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->FileNewItem->OnClick=Form1->FileNew1->OnExecute;
Form1->FileOpenItem->OnClick=Form1->FileOpen1->OnExecute;
Form1->FileSaveItem->OnClick=Form1->FileSave1->OnExecute;
//关联 File 菜单事件
Form1->CutItem->OnClick=Form1->EditCut1->OnExecute;
Form1->CopyItem->OnClick=Form1->EditCopy1->OnExecute;
Form1->PasteItem->OnClick=Form1->EditPaste1->OnExecute;
//关联 Edit 菜单事件
Form1->ToolButton0->OnClick=Form1->FileNewItem->OnClick;
Form1->ToolButton1->OnClick=Form1->FileOpenItem->OnClick;
Form1->ToolButton2->OnClick=Form1->FileSaveItem->OnClick;
Form1->ToolButton4->OnClick=Form1->CutItem->OnClick;
Form1->ToolButton5->OnClick=Form1->CopyItem->OnClick;
Form1->ToolButton6->OnClick=Form1->PasteItem->OnClick;
//关联工具栏事件
}
//---------------------------------------------------------------------------
```

6.2 多页窗体设计

设计多页窗体通常要用到 TabbedNotebook 控件，它可以建立一个多页标签，经常在其上放置不同的组选项按钮，也可以为相似的控件分组。在它的设计阶段，用户可以通过它的 Pages 属性来定制 TabbedNotebook 控件的标签页，在其中可以编辑、添加、删除各个标签页，还可以改变各个标签页之间的相对位置关系。
下面就通过一个绘制正弦曲线、余弦曲线、正切曲线和余切曲线的示例程序向读者说明用 TabbedNote 控件设计多页窗体的方法，具体的程序设计步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Win3.1 选项后，在 TabbedNotebook 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 TabbedNotebook 控件。
在 TabbedNotebook 控件上添加四个标签页——“正弦曲线”、“余弦曲线”、“正切曲线”和“余切曲线”，并且在每个标签页上都放置一个 Image 控件。
添加控件后的窗体如图 6-6 所示。

图 6-6　添加控件后的窗体
添加到窗体上控件的作用如下所示：

- 标签页“正弦曲线”：显示绘制的正弦曲线，并且隐藏其余三个标签页上的曲线；
- 标签页“余弦曲线”：显示绘制的余弦曲线，并且隐藏其余三个标签页上的曲线；
- 标签页“正切曲线”：显示绘制的正切曲线，并且隐藏其余三个标签页上的曲线；
- 标签页“余切曲线”：显示绘制的余切曲线，并且隐藏其余三个标签页上的曲线；
- Image1 控件：作为绘制正弦曲线的容器，并且完成绘制正弦曲线的动作；
- Image2 控件：作为绘制余弦曲线的容器，并且完成绘制余弦曲线的动作；
- Image3 控件：作为绘制正切曲线的容器，并且完成绘制正切曲线的动作；
- Image4 控件：作为绘制余切曲线的容器，并且完成绘制余切曲线的动作。

窗体和控件的属性设置如下所示：

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 354
   Height = 265
   Caption = 'Form1'
   object TabbedNotebook1:
            TTabbedNotebook
      Left = 16
      Top = 16
      Width = 313
```

```
    Height = 209
    object TTabPage
      Left = 4
      Top = 27
      Caption = '正弦曲线'
      object Image1: TImage
        Left = 8
        Top = 16
        Width = 289
        Height = 153
        OnClick = Image1Click
      end
    end
    object TTabPage
      Left = 4
      Top = 27
      Caption = '余弦曲线'
      object Image2: TImage
        Left = 8
        Top = 8
        Width = 289
        Height = 161
        OnClick = Image2Click
      end
    end
    object TTabPage
      Left = 4
      Top = 27
      Caption = '正切曲线'
      object Image3: TImage
        Left = 8
        Top = 8
        Width = 289
        Height = 161
        OnClick = Image3Click
      end
    end
    object TTabPage
      Left = 4
      Top = 27
      Caption = '余切曲线'
      object Image4: TImage
        Left = 8
        Top = 8
        Width = 289
        Height = 161
        OnClick = Image4Click
      end
    end
  end
end
```

2. 响应窗体事件

本示例程序的作用是在不同的标签页上能够绘制出正弦、余弦等四种曲线，为此要在程序的初始化阶段定义画笔的属性和作为绘图容器——Image 控件——的显示属性。
为此，在程序的设计阶段，用鼠标左键双击窗体上的空白处，在弹出的代码窗口中选择窗体

的 FormCreate 事件，并且添加如下所示的响应代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Image1->Canvas->Pen->Width=5;
Form1->Image2->Canvas->Pen->Width=5;
Form1->Image3->Canvas->Pen->Width=5;
Form1->Image4->Canvas->Pen->Width=5;
//设置画笔大小
Form1->Image1->Transparent=true;
Form1->Image2->Transparent=true;
Form1->Image3->Transparent=true;
Form1->Image4->Transparent=true;
//设置四个控件具有透明色
}
//---------------------------------------------------------------------------
```

程序说明：

窗体的 FormCreate 事件在程序运行初期就会被执行，程序通过：

```
Form1->Image1->Canvas->Pen->Width=5;
```

等四条语句设置了画笔的大小，然后通过 Form1->Image1->Trans parent=true;等四条语句来设置四个 Image 控件在程序运行的过程中背景色都是透明的。

3. 响应绘图事件

在程序运行的过程中，切换到不同的标签页就会隐藏其余三个标签页上的控件和图形，但是并不能绘制图形，图形绘制是通过用鼠标单击 Image 控件来实现的，为此要在程序中添加如下代码（由于篇幅的关系，在这里仅列举了 Image1 控件的响应代码，其余控件的代码请参看附后的程序清单）：

```
void __fastcall TForm1::Image1Click(TObject *Sender)
{
float x1;
float y1;
float x2;
float y2;
float n;
//定义变量
x1=0;
y1=(Form1->Image1->Left+Form1->Image1->Width)/4;
x2=x1;
y2=y1;
Form1->Image1->Canvas->MoveTo((x2),(y2));
//定义绘图起始点
n=0;
while (n<3.1415926*6)
        {
        x2=x2+3.1415926/12;
        y2=y1-40*sin(n);
        Form1->Image1->Canvas->LineTo((x2),(y2));
        //绘制正弦曲线
        n=n+3.1415926/180;
        }
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，切换到标签页“正弦曲线”，那么 Image1 控件就已经准备好绘制正弦曲线了，如果用户用鼠标左键单击控件 Image1，那么就会激活控件的 Image1Click 事件，开始绘图。

在绘图开始阶段，首先定义了五个变量 x1、x2、y1、y2、n，其中 x1、y1 用于存储曲线绘制的起始点，而 x2、y2 则用于存储正弦曲线绘制的当前点，变量 n 用于控件曲线绘制的长度；然后程序通过语句 Form1->Image1->Canvas->MoveTo((x2),(y2));把当前绘图坐标移动到绘图的起始点；最后通过一个循环中的 Form1->Image1->Canvas->LineTo((x2) ,(y2));语句完成绘制正弦曲线的操作。

4. 运行程序

按照附后的程序清单添加代码。

做完以上各步的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后在键盘上按功能键【F9】运行程序，在程序运行的初始画面中可以选择四个标签页，并且单击上面的 Image 控件，就可以绘制出相应的曲线，程序运行结果如图 6-7 所示。

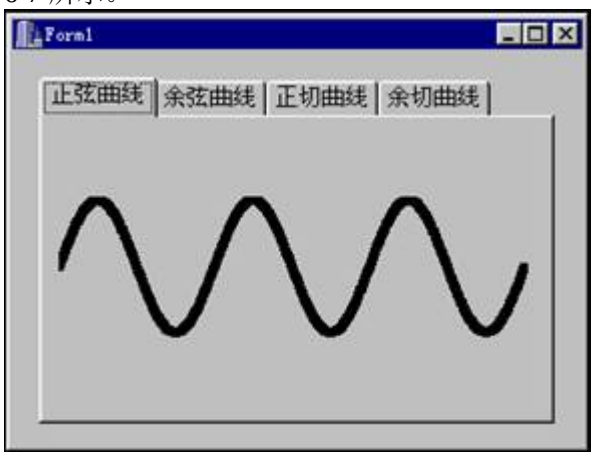


图 6-7　程序运行结果

在这个程序运行的过程中，用户可以选择四种不同的曲线——“正弦曲线”、“余弦曲线”、“正切曲线”和“余切曲线”。

程序的完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "math.h"
#include "Page.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Image1Click(TObject *Sender)
{
float x1;
float y1;
float x2;
float y2;
float n;
//定义变量
```

```
x1=0;
y1=(Form1->Image1->Left+Form1->Image1->Width)/4;
x2=x1;
y2=y1;
Form1->Image1->Canvas->MoveTo((x2),(y2));
//定义绘图起始点
n=0;
while (n<3.1415926*6)
        {
        x2=x2+3.1415926/12;
        y2=y1-40*sin(n);
        Form1->Image1->Canvas->LineTo((x2),(y2));
        //绘制正弦曲线
        n=n+3.1415926/180;
        }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Image2Click(TObject *Sender)
{
float x1;
float y1;
float x2;
float y2;
float n;
//定义变量
x1=0;
y1=(Form1->Image2->Left+Form1->Image2->Width)/4;
x2=x1;
y2=y1;
Form1->Image2->Canvas->MoveTo((x2),(y2));
//定义绘图起始点
n=0;
while (n<3.1415926*6)
        {
        x2=x2+3.1415926/12;
        y2=y1-40*cos(n);
        Form1->Image2->Canvas->LineTo((x2),(y2));
        //绘制正弦曲线
        n=n+3.1415926/180;
        }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Image3Click(TObject *Sender)
{
float x1;
float y1;
float x2;
float y2;
float n;
//定义变量
x1=0;
y1=(Form1->Image3->Left+Form1->Image3->Width)/4;
x2=x1;
y2=y1;
Form1->Image3->Canvas->MoveTo((x2),(y2));
//定义绘图起始点
n=0;
```

```
while (n<3.1415926*2)
        {
        x2=x2+3.1415926/3;
        if (abs(cos(n))<0.000001)
            {
             y2=y1-30*sin(n)/cos(n);
             Form1->Image3->Canvas->LineTo((x2),(y2));
             //绘制正切曲线
            }
         n=n+3.1415926/180;
         }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Image4Click(TObject *Sender)
{
float x1;
float y1;
float x2;
float y2;
float n;
//定义变量
x1=0;
y1=(Form1->Image4->Left+Form1->Image4->Width)/4;
x2=x1;
y2=y1;
Form1->Image4->Canvas->MoveTo((x2),(y2));
//绘图起始点
n=3.1415926;
while (n<3.1415926*3)
        {
        x2=x2+3.1415926/3;
        if (abs(sin(n))<0.00001)
            {
            y2=y1-30*cos(n)/sin(n);
            Form1->Image4->Canvas->LineTo((x2),(y2));
            //绘制余切曲线
            }
        n=n+3.1415926/180;
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Image1->Canvas->Pen->Width=5;
Form1->Image2->Canvas->Pen->Width=5;
Form1->Image3->Canvas->Pen->Width=5;
Form1->Image4->Canvas->Pen->Width=5;
//设置画笔大小
Form1->Image1->Transparent=true;
Form1->Image2->Transparent=true;
Form1->Image3->Transparent=true;
Form1->Image4->Transparent=true;
//设置四个控件具有透明色
}
//---------------------------------------------------------------------------
```

6.3 复合窗体设计

如果读者想要设计一个比较复杂的应用程序，那么仅仅利用一个窗体是不够的，这时通常要涉及到复合窗体的程序设计。在复合窗体的程序设计过程中，用户可以对程序中的每一个窗体按照自己的编程意图来设计和编码，并且可以随时的控制每个窗体的显示、隐藏、加载和卸载等操作。

我们已经知道，利用 CBuilder 中提供的方法可以绘制出简单的基本图案（如直线、圆和矩形等），同时也可以绘制出比较复杂的图案。下面，我们一个可以绘制各种图形的复合窗体应用为例来说明复合窗体程序的开发和设计技术。

6.3.1 绘制直线的子窗体设计

下面，我们继续添加绘制直线的子窗体，这个窗体的作用是：当用户在窗体上单击鼠标的左键时，程序就会调用相应的方法在窗体上绘制一条由上一个鼠标单击点到当前鼠标单击点的直线，步骤如下：

1. 新建窗体

首先启动一个新的项目，选择菜单“File”中的“New Application”项。在本项目的基础上，选择菜单“File”中的“New Form”项，在 CBuilder 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“Standart”选项后，在“Button”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，添加控件后的窗体如图 6-8 所示。

图 6-8　绘制直线的子窗体

2. 添加代码

在程序的设计过程中，用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口，把光标移动到控件和窗体的相应事件过程中，并且按照下面的程序清单添加程序代码：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Line.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
```

```
#pragma resource "*.dfm"
TForm_Line *Form_Line;
Boolean Start;
//定义标志变量
//---------------------------------------------------------------------------
__fastcall TForm_Line::TForm_Line(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm_Line::FormCreate(TObject *Sender)
{
Start=false;
//初始化标志变量
}
//---------------------------------------------------------------------------

void __fastcall TForm_Line::Button1Click(TObject *Sender)
{
Start=true;
//设置标志变量
}
//---------------------------------------------------------------------------

void __fastcall TForm_Line::FormMouseDown(TObject *Sender, TMouseButton
Button,TShiftState Shift, int X, int Y)
{
if (Start)
    Form_Line->Canvas->LineTo(X,Y);
    //绘制直线到鼠标的单击点
}
//---------------------------------------------------------------------------
```

程序说明：

在程序的变量声明段中，首先定义了一个布尔型变量 Start，它的作用是控制程序绘制直线的开始时间，在窗体的初始化过程中通过语句 start:=false;设置为 False，如果在程序运行的过程中，用户单击按钮“绘制直线”，那么程序就自动的设置变量 Start 的值为 True，然后通过语句 form1.Canvas.LineTo(x,y);在窗体上绘制直线。

6.3.2 绘制椭圆的子窗体设计

下面，我们再创建用于绘制椭圆的子窗体，它的作用是：当用户在子窗体上按下并且移动鼠标时，程序就会调用相应的方法在窗体上绘制一个椭具体的程序设计步骤如下：

1. 建立窗体

首先启动一个新的项目，选择菜单“File”中的“New Application”项，在 CBuilder 的集成开发环境中就会弹出一个新建的窗体。

图 6-9　绘制椭圆子窗体

2. 添加代码

在程序的设计过程用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到到代码段中的适当位置，并且添加如下绘制绘制椭圆的代码：

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Ellipse.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm_Ellipse *Form_Ellipse;
Boolean Start;
//定义标志变量
//---------------------------------------------------------------------
__fastcall TForm_Ellipse::TForm_Ellipse(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm_Ellipse::FormCreate(TObject *Sender)
{
Start=false;
//变量赋初值
}
//---------------------------------------------------------------------

void __fastcall TForm_Ellipse::FormMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
Start=true;
```

```
//开始绘图
}
//---------------------------------------------------------------------------
void __fastcall TForm_Ellipse::FormMouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
if (Start)
   Form_Ellipse->Canvas->Ellipse(X,Y,20,20);
   //绘制椭圆
}
//---------------------------------------------------------------------------

void __fastcall TForm_Ellipse::FormMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
Start=false;
//结束绘图
}
//---------------------------------------------------------------------------
```

以上程序能够实现在程序运行的初始画面中，按下的同时移动鼠标，在窗体上就会绘制出一系列的椭圆。

6.3.3 绘制矩形子窗体

在 CBuilder 5 中绘制矩形调用的方法是 Rectangle 方法，它的语法结构如下所示：

```
void __fastcall Rectangle(int X1, int Y1, int X2, int Y2);
```

在调用 Rectangle 方法绘制矩形的时候，要为该方法提供四个顶点的坐标，并且顶点的坐标值为 Integer 类型。下面通过一个示例来说明如何在程序中绘制一个矩形，在程序运行的过程中，每隔 1 秒钟，在窗体上就会以随机点绘制一个矩形，具体的程序设计步骤如下：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 System 选项后，在 Timer 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个计时器组件，其中计时器组件的属性设置如图 6-10 所示。

图 6-10　计时器组件的属性设置

2. 添加代码

为了保证在程序运行的过程中，能够每隔 1 秒钟就绘制一个矩形，在程序的初始化过程中应该设置计时器组件的有效状态为 True，具体的程序代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->Canvas->Pen->Color=RGB(random(255),random(255),random(255));
//设置画笔颜色
Form1->Canvas->Pen->Width=random(10)+1;
//设置画笔大小
Form1->Canvas->Pen->Mode=random(16);
//设置画笔模式
Form1->Canvas->Pen->Style=random(7);
//设置画笔风格
Form1->Canvas->Brush->Color=RGB(random(255),random(255),random(255));
//设置画刷颜色
Form1->Canvas->Brush->Style=random(8);
//设置画刷风格
Form1->Canvas->Rectangle(random(Form1->Width),random(Form1->Height),random(Form1->
Width),random(Form1->Height));
//以随机位置绘制矩形
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Timer1->Enabled=true;
//开始绘图
}
//---------------------------------------------------------------------------
```

程序说明：

由于是以随机点的形式绘制矩形，所以在程序运行的过程中要调用函数 random()来产生四个随机数，并且绘制矩形的代码要放在计时器组件的 Timer1Timer()过程事件中，这样才能够保证每隔 1 秒钟绘制一个随机点矩形。

3. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名

保存文件。

然后按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，每隔 1 秒钟，程序就会自动的以随机点来绘制一个矩形，程序运行结果如图 6-11 所示。

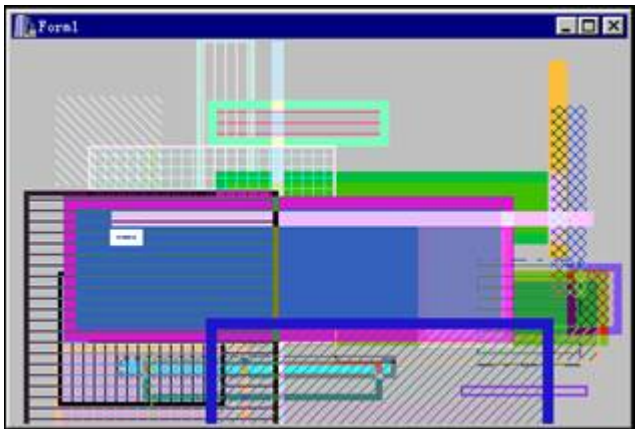


图 6-11　随机绘制的矩形

6.3.4　主窗体的完整设计

最后，我们再来建立主窗体，并添加必要的程序代码，以便能由主窗体直接调用上面建立的子窗体程序。

1. 新建窗体

首先新建一个窗体，在窗体添加一个 Panel 控件，然后在这个 Panel 控件上再添加三个 Button 控件；之后，继续在窗体上添加第四个按钮控件和一个图片控件 Image，添加完控件后的窗体如图 6-12 所示。

图 6-12　主窗体设计界面

下面是主窗体的属性设置，请读者参考设计自己喜好的主窗体界面：

```
object Form_Main: TForm_Main
   Left = 177
   Top = 139
   Width = 454
   Height = 313
   Caption = 'Form_Main'
   Color = clBtnFace
```

```
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Image1: TImage
    Left = 272
    Top = 16
    Width = 169
    Height = 145
    Picture.Data = { …….       }          //图像编码
  end
  object Panel1: TPanel
    Left = 8
    Top = 8
    Width = 249
    Height = 273
    Caption = 'Panel1'
    TabOrder = 0
  end
  object Button1: TButton
    Left = 32
    Top = 40
    Width = 209
    Height = 41
    Caption = '单击进入绘制直线子窗体'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 1
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 32
    Top = 120
    Width = 201
    Height = 41
    Caption = '单击进入绘制椭圆子窗体'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clRed
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 2
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 32
    Top = 208
    Width = 201
    Height = 41
    Caption = '单击进入绘制矩形子窗体'
    Font.Charset = DEFAULT_CHARSET
```

```
      Font.Color = clRed
      Font.Height = -16
      Font.Name = 'MS Sans Serif'
      Font.Style = []
      ParentFont = False
      TabOrder = 3
      OnClick = Button3Click
   end
   object Button4: TButton
      Left = 296
      Top = 208
      Width = 113
      Height = 33
      Caption = '退出总程序'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clBlack
      Font.Height = -16
      Font.Name = 'MS Sans Serif'
      Font.Style = []
      ParentFont = False
      TabOrder = 4
      OnClick = Button4Click
   end
end
```

2. 添加代码

在程序的设计阶段，用鼠标的左键双击窗体，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中的适当位置。
分别双击“进入直线绘制子窗体”按钮、“进入圆形绘制子窗体”按钮和“进入矩形绘制子窗体”按钮和“退出总程序”按钮，添加如下所示的程序代码：

```
void __fastcall TForm_Main::Button1Click(TObject *Sender)
{
      Form_Line=new TForm_Line(Application);
      Form_Line->ShowModal();
      delete Form_Line;
}
//---------------------------------------------------------------------------

void __fastcall TForm_Main::Button2Click(TObject *Sender)
{
      Form_Ellipse=new TForm_Ellipse(Application);
      Form_Ellipse->ShowModal();
      delete Form_Ellipse;
}
//---------------------------------------------------------------------------

void __fastcall TForm_Main::Button3Click(TObject *Sender)
{
      Form_Rectangle=new TForm_Rectangle(Application);
      Form_Rectangle->ShowModal();
      delete Form_Rectangle;
}
//---------------------------------------------------------------------------

void __fastcall TForm_Main::Button4Click(TObject *Sender)
{
Form_Main->Close();
```

```
}
//---------------------------------------------------------------------------
```

3. 运行程序

代码添加完毕，我们将本次创建的应用程序保存下来（这里，项目保存为 P_Muti，主窗体的单元文件保存 Main，绘制直线子窗体的单元文件保存为 unit1，绘制椭圆子窗体的单元文件保存为 unit2，绘制矩形子窗体的单元文件保存为 unit3）。
按 F9 键运行程序，如图 6-13 上所示为初始主窗体画面。

图 6-13　程序运行主窗体
单击“进入绘制椭圆子窗体”按钮，就会打开绘制椭圆子窗体，如图 6-14 所示。在本子窗体上，我们可以用鼠标绘制一系列的椭圆。

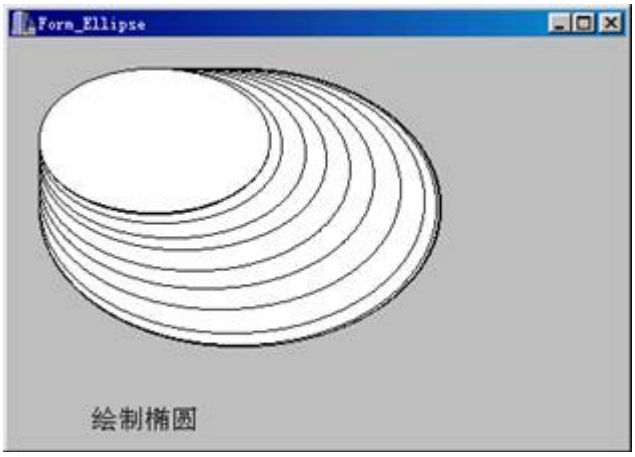


图 6-14　子窗体运行结果

6.4　小　结

通过本章的学习，用户可以掌握窗体程序设计中的一般方法和常用的技巧，如多页面窗体设计、复合窗体程序设计和简单的 SDI 程序设计等，在介绍窗体设计技术的同时，编者还向读者介绍了 ActionList 控件的使用，利用这个控件可以大大减轻程序设计的工作量和强度。最后，希望读者能够熟练的利用本章介绍的知识，自己独立设计出丰富多彩的应用程序。下一章，我们将介绍 CBuilder 5 中丰富多彩的数据库开发应用。

第七章　数据库开发和应用

CBuilder 5 最吸引人之处就是它强大的数据库访问和网络功能，利用大量的系统数据库控件（DataSource 控件、Table 控件和 Query 控件等）和网络控件(HTML 控件、NMURL 控件和 NMHTTP 控件等)可以很方便的设计出功能强大的数据库和网络应用程序，不但可以编制访问本地数据库的应用程序，也可以设计能够访问远程大型数据的应用程序。
本章的重点不是向读者介绍 CBuilder 5 的内部数据库访问和网络机制，而是通过几个具体的应用程序示例来向读者说明在 CBuilder 5 中的数据库和网络应用程序的设计方法，所以有很大的实用性。

7.1　使用数据库向导

设计数据库应用程序有两种基本的方法，一是从一个空白的窗体开始开始设计，在窗体上放置各种控件，在代码窗口中添加代码，另一种方法就是利用 CBuilder 5 提供的数据库应用程序向导来完成简单数据库应用程序的设计，下面来介绍一下如何利用向导程序制作数据库应用程序，具体的步骤如下所示。

1. 开始工作

首先选择菜单 File 中的 New 选项，在屏幕上就会弹出一个如图 7-1 所示的 New Items 对话框。

图 7-1　New Items 对话框

在 New Items 对话框中选择 Business 选项中的 DatabaseFormWizard，即打开了一个数据库向导应用程序，单击 OK 按钮进入下一步。

2. 选择数据库类型

在如图 7-2 所示的选择数据库类型的对话框中，用户可以对窗体的类型和数据控件类型进行选择，用户可以选择如下类型的窗体：创建一个简单的窗体或创建一个主/细节窗体；同时用户还可以选择如下的数据控件：创建的窗体中包含 Tabel 控件或创建的窗体中包括 Query 控件。

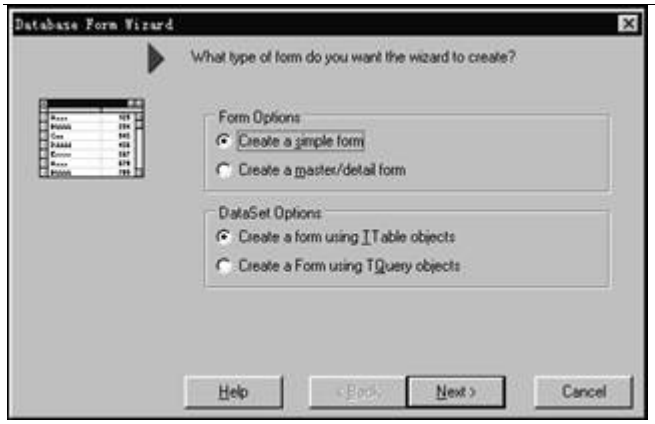


图 7-2　选择数据库类型对话框

选择 Create a simple form 和 Create a form using TTable objects 选项，即创建一个包含 Table 控件的简单窗体。单击 Next 按钮进入下一步。

3. 选择数据库文件

在如图 7-3 所示的选择数据库文件的对话框中，用户可以选择本地机上的所有有效的数据库文件。

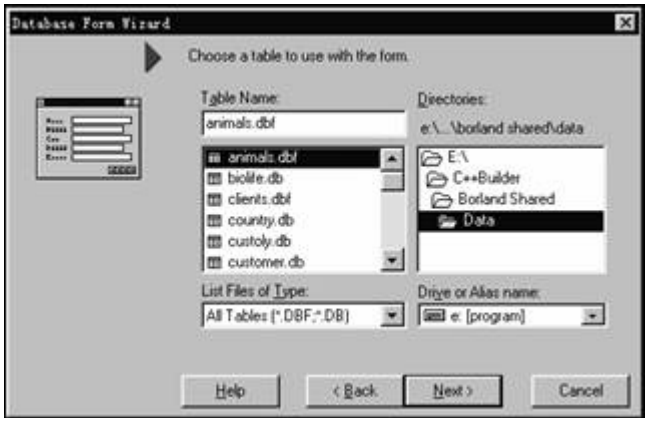


图 7-3　选择数据库文件

本示例程序中我们选择的数据库文件为 E:\C++Builder\Borland Shared\Data\animals.db，单击 Next 按钮进入下一步。

4. 选择显示域

在如图 7-4 所示的对话框中，用户可以选择在程序运行过程中想要显示的域，单击　　按钮可以向窗体中添加选中的域，而单击　　则可以添加所有的域，同样单击　　按钮和  可以删除已经添加的域。

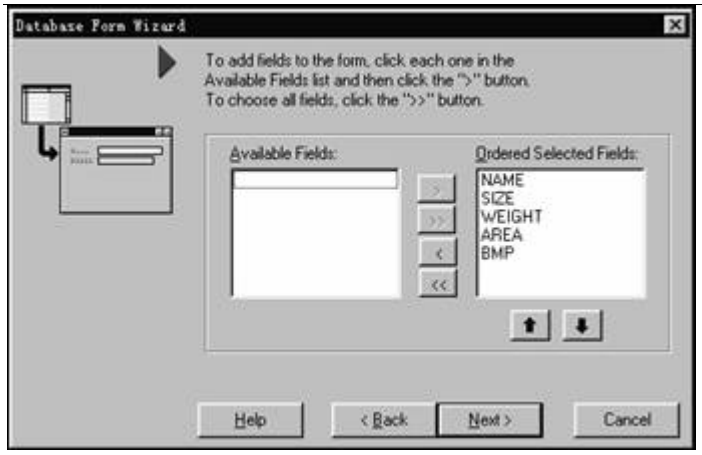


图 7-4　选择显示域

在选择显示域的对话框中，用户还可以选择各个域在程序运行过程中的索引位置。在本示例程序中，把所有的域都设置为可见，并且，各个域在程序中的索引位置采用默认值，单击 Next 按钮进入下一步。

5. 选择布局样式

在向导应用程序中，可以选择窗体中的控件的排列方式，如水平排列方式和垂直排列的方式，对话框如图 7-5 所示。

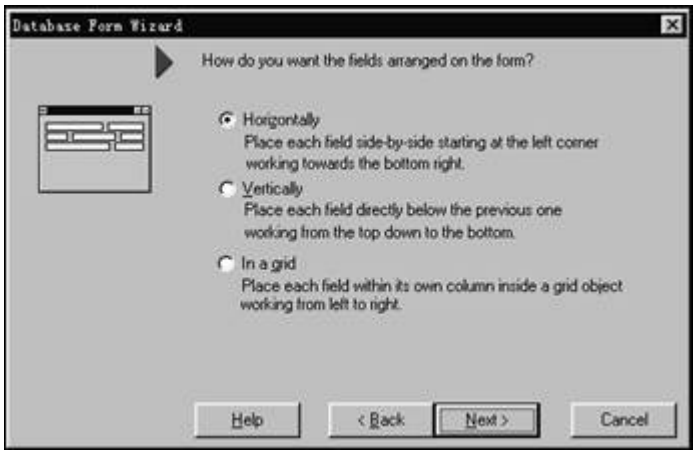


图 7-5　选择布局样式对话框

如果在选择窗体布局样式对话框中选择的是 Horizontally 方式，那么窗体上的控件将会以水平方式对齐，如果选择的是 Vertically 方式，那么窗体上的控件将会以垂直方式排列。

如果选择的是 In a grid，那么在中将会使用 DBGird 控件。

在本示例程序中选择 Horizontally 方式，单击 Next 按钮进入下一步。

6. 选择窗体形式

在前一步中选择窗体布局后，下一步的工作就是选择窗体的形式，用户可以在如图 7-6 所示的对话框中设置窗体形式的参数。

用户可以通过复选 Generate a main form 来决定在项目中是否加入一个主窗体。同时用户还可以选择 Form Only 或者 Form and DataModule，如果选择 Form Only 选项，那么在项目中将只包括一个窗体，所有的控件都会放置在一个窗体中，如果选择 Form and DataModule 选项，那么控件将会分别的放置在 2 个窗体中。完成以上的工作后，单击 Finish 按钮，就创建了一个简单的数据库应用程序，完成后的窗体如图 7-7 所示。

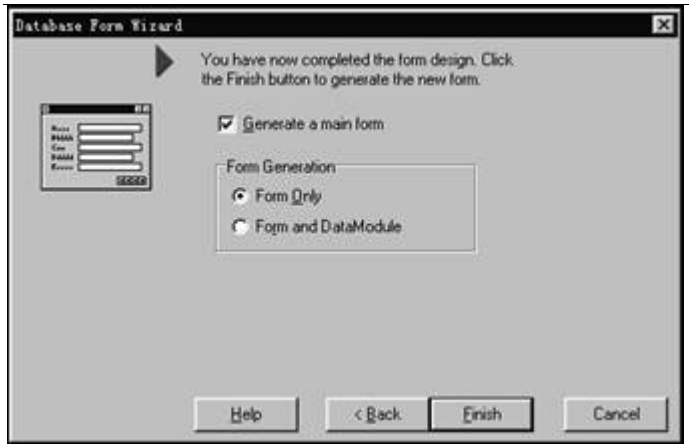


图 7-6　选择窗体形式

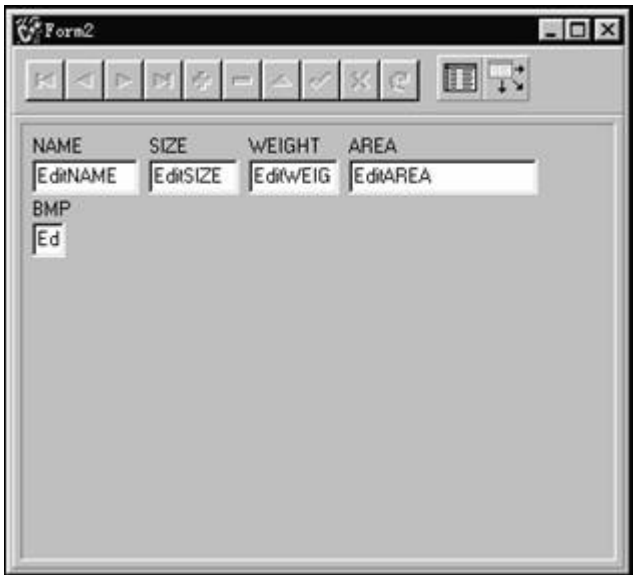


图 7-7　完成后的窗体结构

在窗体上放置有 5 个 Label 控件、2 两个 Panel 控件、1 个 Table 控件、2 个 StringField、2 个 SmallintField、1 个 BlobField 控件、1 个 ScrollBox、5 个 DBEdit、1 个 DBNavigator 和 1 个 DataSource 控件。

完成以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序。

程序运行结果如图 7-8 所示。

图 7-8　程序运行结果

7.2 设计自己的数据库

在前面通过向导生成的数据库应用程序，所能够实现的功能还是比较简单的，而且数据库的各种操作都是通过 DBNavigator 控件来完成的。
下面我们就从一个空白的窗体开始设计一个数据库应用程序，在本示例程序中，所有的数据库编辑操作都是通过用户自定义的按钮控件来完成的。
在程序运行的过程中，用户不但可以浏览数据库文件中的各种数据，还可以进行插入、删除等编辑操作。
具体的程序设计步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 选项项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，在 Button 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，接着向窗体上添加 8 个 Button 控件、1 个 Timer 控件、1 个 DBGrid 控件、1 个 Table 控件和 1 个 DataSource 控件。各个控件的功能将在后面分别的加以介绍。
添加控件后的窗体如图 7-9 所示。

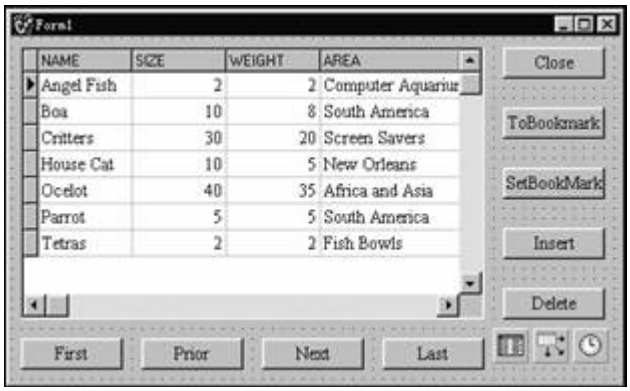


图 7-9　添加控件后的窗体
其中窗体和控件的属性设置如下所示：

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 493
   Height = 302
   Caption = 'Form1'
   OnCreate = FormCreate
   object DBGrid1: TDBGrid
      Left = 8
      Top = 8
      Width = 369
      Height = 217
      DataSource = DataSource1
   end
   object Button1: TButton
      Left = 8
      Top = 240
      Width = 81
      Height = 25
      Caption = 'First'
   end
   object Button2: TButton
      Left = 104
```

```
    Top = 240
    Width = 81
    Height = 25
    Caption = 'Prior'
  end
  object Button3: TButton
    Left = 200
    Top = 240
    Width = 81
    Height = 25
    Caption = 'Next'
  end
  object Button4: TButton
    Left = 296
    Top = 240
    Width = 81
    Height = 25
    Caption = 'Last'
  end
  object Button6: TButton
    Left = 392
    Top = 200
    Width = 81
    Height = 25
    Caption = 'Delete'
  end
  object Button5: TButton
    Left = 392
    Top = 152
    Width = 81
    Height = 25
    Caption = 'Insert'
  end
  object Button7: TButton
    Left = 392
    Top = 104
    Width = 81
    Height = 25
    Caption = 'SetBookMark'
  end
  object Button8: TButton
    Left = 392
    Top = 56
    Width = 81
    Height = 25
    Caption = 'ToBookmark'
  end
  object Button9: TButton
    Left = 392
    Top = 8
    Width = 81
    Height = 25
    Caption = 'Close'
  end
  object DataSource1: TDataSource
    DataSet = Table1
    Left = 416
    Top = 232
  end
  object Timer1: TTimer
```

```
    Interval = 100
    Left = 448
    Top = 232
  end
  object Table1: TTable
    Active = True
    DatabaseName = 'BCDEMOS'
    TableName = 'animals.dbf'
    Left = 384
    Top = 232
  end
end
```

在控件的属性设置过程中，一定要注意以下几个控件的属性设置：

- Table 控件的 DatabaseName 属性要设置为'DBDEMOS'，这时单击 TableName 属性右侧的下拉式列表框，就可以选择数据库文件'animals.dbf'了，最后要把它的 Active 属性设置为 True；
- DataSource 控件的 DataSet 属性设置为 Table1；
- DBGrid 控件的 DataSource 属性设置为 DataSource1，即指明了数据显示的来源。

2. 响应按钮事件

在窗体上放置有九个 Button 控件，它们可以实现以下的功能：

- 控件 Button1：将数据库的指针移动到第一条记录处；
- 控件 Button2：将数据库的指针移动移动到前一条记录处；
- 控件 Button3：将数据库的指针移动移动到下一条记录处；
- 控件 Button4：将数据库的指针移动到最后一条记录处；
- 控件 Button5：在当前位置插入一条数据；
- 控件 Button6：将当前位置处的数据删除；
- 控件 Button7：在当前位置设置一个书签；
- 控件 Button8：跳转到上一次设置的书签处；
- 控件 Button9：关闭当前的数据库。

由于篇幅的关系，在这里仅以 Button1 控件和 Button5 控件为例来说明如何添加响应按钮事件的代码。在程序设计的过程中，用鼠标的左键双击窗体中的 Button1 和 Button5 控件，在屏幕上就会弹出一个代码窗口，把光标分别移动到控件 Button1 和控件 Button5 的事件响应过程中。添加如下所示的代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->Table1->First();
//移动到第一条记录
}
//---------------------------------------------------------------------
……
void __fastcall TForm1::Button5Click(TObject *Sender)
{
Form1->Table1->Insert();
//在当前位置插入一条记录
}
//---------------------------------------------------------------------
```

程序说明：

程序中各个按钮功能都是通过调用 Table 控件的相应方法来实现的，如 First 按钮就是通过调用 Table 控件的 First 方法来实现把当前数据库指针移动到第一条记录的功能的，而 Insert 按钮是通过调用 Table 控件的 Insert 方法来实现在当前位置插入一条记录的功能的。在程序运行的过程中，当用户用鼠标的左键单击窗体上的各个按钮时，程序就会自动的调用 Table 控件的相应方法来实现对应的功能。如用户单击 Insert 按钮，程序就会调用 Insert 方法在数据库的当前位置插入一条记录，结果如图 7-10 所示。

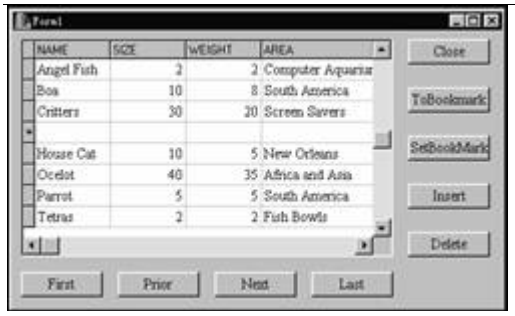


图 7-10　在当前位置插入一条记录

其它几个按钮的代码添加过程就不一一的加以介绍了，读者可以参看附后的源程序代码，在这里仅仅列举出各个按钮所对应的方法：

- First 按钮：Table 控件的 First 方法；
- Prior 按钮：Table 控件的 Prior 方法；
- Next 按钮：Table 控件的 Next 方法；
- Last 按钮：Table 控件的 Last 方法；
- Insert 按钮：Table 控件的 Insert 方法；
- Delete 按钮：Table 控件的 Delete 方法；
- ToBookMark 按钮：Table 控件的 GotoBookmark 方法；
- SetBookMark 按钮：Table 控件的 GetBookmark 方法；
- Close 按钮：Table 控件的 Close 方法。

3. 响应计时器事件

窗体上添加的 Timer 控件的作用是在程序运行的过程中动态的设置各个按钮控件的有效状态，例如在浏览或者编辑数据库中的数据时，如果当前的数据库指针已经移动到了数据库的开头或结尾，再执行 Tabel 控件的 Last 方法、Next 方法、First 方法和 Prior 方法，程序就会出现运行错误。

为了在程序运行的过程中避免以上错误的出现，在程序中应该有设置各个按钮有效状态的代码，为此，在程序设计的过程中，用鼠标的左键双击窗体上的 Timer 控件，在屏幕上就会弹出一个代码窗口，在其中可以添加动态设置按钮有效状态的代码。

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
if (Form1->Table1->Eof)
    {
    Form1->Button4->Enabled=false;
    Form1->Button3->Enabled=false;
    }
    else
    {
    Form1->Button4->Enabled=true;
    Form1->Button3->Enabled=true;
    }
    //动态设置“Next”和“Last”按钮的有效状态
if (Form1->Table1->Bof)
    {
    Form1->Button1->Enabled=false;
    Form1->Button2->Enabled=false;
    }
    else
    {
    Form1->Button1->Enabled=true;
    Form1->Button2->Enabled=true;
    }
    //动态设置“First”和“Prior”按钮的有效状态
```

```
}
//---------------------------------------------------------------------------
```

程序说明：

Timer 控件的 Interval 属性设置为 100，即在程序的运行过程中，每隔 100 微秒就会自动的激活一个 Timer1Timer()事件，随时的检测数据库文件指针的位置，同时设置按钮的有效状态，结果如图 7-11 所示。

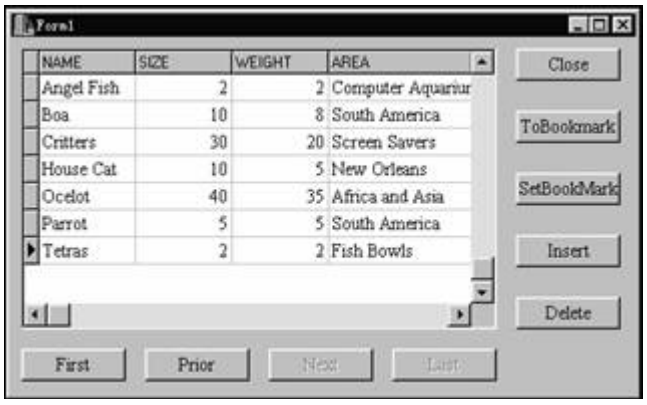


图 7-11　动态设置按钮有效状态

4. 运行程序

按照附后的源程序，添加剩余的程序代码后。做完以上的工作后，用鼠标选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的结果如图 7-12 所示。

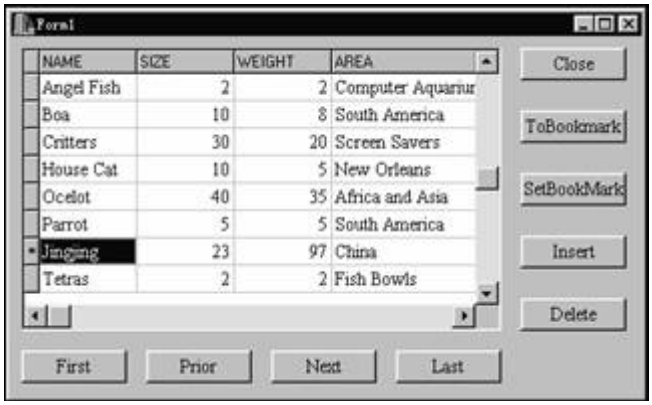


图 7-12　程序运行结果

程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "DataEdit.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
TBookmark SavePlace;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
```

```
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Timer1->Enabled=true;
//设置控件的有效状态
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->Table1->First();
//移动到第一条记录
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->Table1->Prior();
//向前移动一条记录
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Form1->Table1->Next();
//移动到下一条记录
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
Form1->Table1->Last();
//移动到最后一条记录
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
Form1->Table1->Insert();
//在当前位置插入一条记录
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
Form1->Table1->Delete();
//删除当前记录
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button7Click(TObject *Sender)
{
    SavePlace=Form1->Table1->GetBookmark();
    //在当前位置设置一个书签
}
//---------------------------------------------------------------------
```

```
void __fastcall TForm1::Button8Click(TObject *Sender)
{
if (SavePlace!="")
     {
     Form1->Table1->GotoBookmark(SavePlace);
     //跳转到指定书签
     }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button9Click(TObject *Sender)
{
Form1->Table1->Close();
//关闭数据库
}
//---------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
if (Form1->Table1->Eof)
     {
     Form1->Button4->Enabled=false;
     Form1->Button3->Enabled=false;
     }
     else
     {
     Form1->Button4->Enabled=true;
     Form1->Button3->Enabled=true;
     }
     //动态设置“Next”和“Last”按钮的有效状态
if (Form1->Table1->Bof)
     {
     Form1->Button1->Enabled=false;
     Form1->Button2->Enabled=false;
     }
     else
     {
     Form1->Button1->Enabled=true;
     Form1->Button2->Enabled=true;
     }
     //动态设置“First”和“Prior”按钮的有效状态
}
//---------------------------------------------------------------------
```

7.3 设计数据库报表

在数据库应用程序设计中要实现的很重要的任务之一就是报表的设计，CBuilder 5 中为用户进行报表设计提供了一个 QuickReport 报表制作工具，利用 QuickReport 可以很简单地制作了打印出报表，用户可以根据实际需要来建立各种各样的报表，而且它的最大的一个优点就是几乎不用程序代码，所有的工作都是在设计的过程中通过设置控件属性来完成的。下面就通过一个具体报表的设计来向读者说明 CBuilder 5 中报表设计的具体步骤和方法。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New 选项项，在 CBuilder 5 的集成开发环境中就会弹出一个如图 7-13 所示的 New Items 对话框。

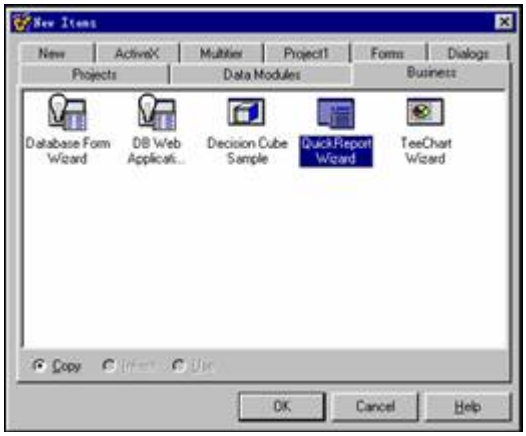


图 7-13　New Items 对话框

在 New Items 对话框中选择 Business 页下面的 QuickReport Wizard 选项，单击 OK 按钮，就开始了报表设计向导，接下来就会显示一个 New Report Wizard 对话框，在其中用鼠标单击 Start Wizard 按钮进入下一步，在弹出的对话框中做如图 7-14 所示的选择。

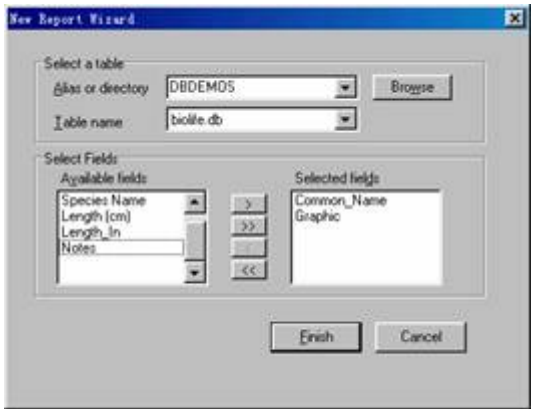


图 7-14　New Report Wizard 对话框

上面一步中，我们在 New Report Wizard 对话框的 Alias or directory 选项框中选择 DBDEMOS，在 Table name 选项框中选择 biolife.dbf，在 Selected fields 框中选择 Common_Name 和 Graphic，单击 Finish 按钮就完成了报表的设计。

注意：

- 上面向导设计完成的报表还不能浏览图片，需要进行修改。

2. 修改报表界面

在设计完成的窗体中，在 ColumnHeaderBand 控件上添加一个 QRDBImage 控件

QRDBImage1，QRDBImage 控件的作用是在程序运行的过程中显示数据库中的图像，并且调整 ColumnHeaderBand 和 QRDBImage1 的大小添加控件后的窗体如图 7-15 所示。

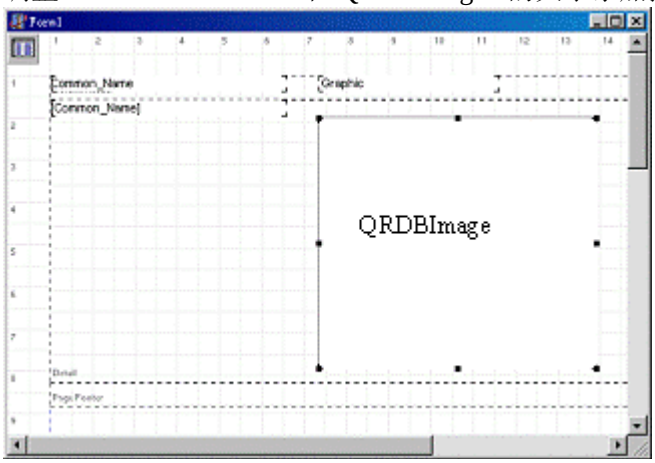


图 7-15　设计完成的窗体

为了实现 QRDBImage1 控件和数据库的接口，请按照如图 7-16 所示对 QRDBImage1 进行属性设置。

图 7-16　设置 QRDBImage1 控件的属性

本报表窗体用文字描述如下：

object QRDBImage1: TQRDBImage
Left = 392
Top = 0
Width = 129
Height = 121
Frame.Color = clBlack
Frame.DrawTop = False
Frame.DrawBottom = False
Frame.DrawLeft = False
Frame.DrawRight = False
Size.Values = (
320.145833333333
1037.16666666667
0
341.3125)
DataField = 'graphic'
DataSet = Table1
end

3. 查看报表

做完以上的工作后，用鼠标选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件。

在 QuickRep 控件上单击鼠标的右键，在窗体上就会弹出一个对话框，在其中选择 Preview 选项，就可以预览设计完成的报表了，结果如图 7-17 所示。

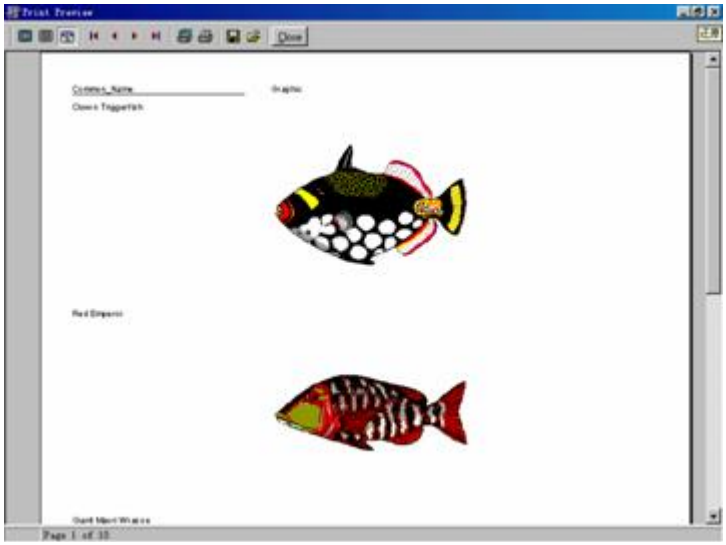


图 7-17　设计完成的报表预览效果

7.4 SQL 在数据库中的使用

如果设计一个比较简单的数据库应用程序，利用 Table 控件就足够了，但是如果要求数据库的功能比较完善，比如在程序运行的过程中动态的显示数据库中各个域，那么就需要在数据库程序的设计过程中嵌入 SQL 语句了。
在 CBuilder 5 中，利用 Query 控件可以很方便的实现 SQL 语句与数据控件之间的连接操作，下面就以一个示例程序来说明在 CBuilder 5 的数据库程序设计的过程中如何嵌入 SQL 语句，示例程序不但可以实现本章第二节示例中的全部功能，还可以在程序运行的过程中随时的编辑 SQL 语句，然后在程序中调用，程序设计的具体步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 System 选项后，在 Timer 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个计时器控件，接着向窗体上添加 10 个 Button 控件、1 个 DataSource 控件、1 个 Query 控件、1 个 DBGrid 控件和 1 个 Memo 控件，各个控件的功能将在后面分别的加以介绍，
其中窗体以及窗体上各个控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 491
  Height = 350
  Caption = 'Form1'
  OnClose = FormClose
  OnCreate = FormCreate
  object Memo1: TMemo
    Left = 312
    Top = 8
    Width = 161
    Height = 145
  end
  object Button1: TButton
    Left = 312
    Top = 160
    Width = 73
    Height = 25
    Caption = '执行 SQL'
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 400
    Top = 160
    Width = 73
    Height = 25
    Caption = '清除 SQL'
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 312
    Top = 192
    Width = 73
    Height = 25
    Caption = 'Insert'
```

```
    OnClick = Button3Click
  end
  object Button4: TButton
    Left = 400
    Top = 192
    Width = 73
    Height = 25
    Caption = 'Delete'
    OnClick = Button4Click
  end
  object Button5: TButton
    Left = 312
    Top = 224
    Width = 73
    Height = 25
    Caption = 'First'
    OnClick = Button5Click
  end
  object Button6: TButton
    Left = 400
    Top = 224
    Width = 73
    Height = 25
    Caption = 'Last'
    OnClick = Button6Click
  end
  object Button7: TButton
    Left = 312
    Top = 256
    Width = 73
    Height = 25
    Caption = 'Prior'
    OnClick = Button7Click
  end
  object Button8: TButton
    Left = 400
    Top = 256
    Width = 73
    Height = 25
    Caption = 'Next'
    OnClick = Button8Click
  end
  object DBGrid1: TDBGrid
    Left = 8
    Top = 8
    Width = 297
    Height = 305
    DataSource = DataSource1
  end
  object Button9: TButton
    Left = 312
    Top = 288
    Width = 73
    Height = 25
    Caption = 'Set Mark'
    OnClick = Button9Click
  end
  object Button10: TButton
    Left = 400
    Top = 288
```

```
    Width = 73
    Height = 25
    Caption = 'To Mark'
    OnClick = Button10Click
  end
  object Timer1: TTimer
    OnTimer = Timer1Timer
    Left = 192
    Top = 144
  end
  object DataSource1: TDataSource
    AutoEdit = False
    DataSet = Query1
    Left = 136
    Top = 144
  end
  object Query1: TQuery
    Active = True
    BeforeClose = Query1BeforeClose
    DatabaseName = 'BCDEMOS'
    RequestLive = True
    SQL.Strings = ('select custno,company
    from customer.db')
    Left = 80
    Top = 144
  end
end
```

在控件的属性设置过程中，一定要注意以下几个控件的属性设置：

- Timer 控件的 Enable 属性设置为 True，同时 Interval 属性设置为 100，即每隔 100 微秒程序就会激活一个计时器事件；
- DataSource 控件的 DataSet 属性设置为 Query1，即将 DataSource 控件与 Query 相关联；
- DBGrid 控件的 DataSource 属性设置为 DataSource1，即指明了数据显示的来源；
- Query 控件的 DatabaseName 属性设置为 'DBDEMOS'，SQL 属性输入框中输入字符串"select custno,company from customer.db"，同时 Active 属性设置为 True，即控件处于活动的状态。

设置属性后的控件如图 7-18 所示。

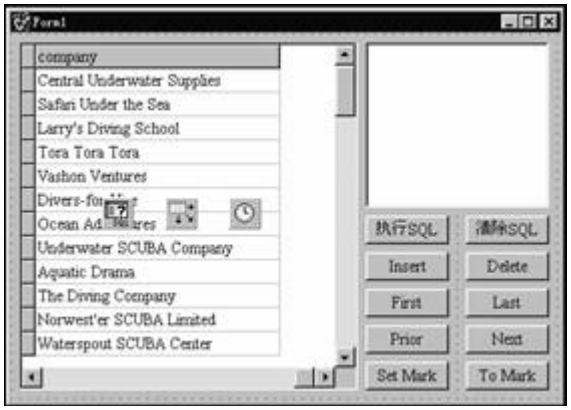


图 7-18　设置属性后的窗体和控件

2. 响应 SQL 事件

本程序（也是本节）的重点在于向读者说明在 CBuilder 5 中 SQL 语句的用法，所以在窗体上放置了一个 Memo 控件，用于接收用户的输入，而具体的执行动作是由按钮“执行 SQL”来完成的。

在程序的设计阶段，用鼠标的左键双击窗体上的“执行 SQL”按钮，在屏幕上就会弹出一个代码窗口，把光标移动到相应的事件处理过程中，并且添加如下代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->Query1->Close();
//关闭数据库
Form1->Query1->SQL->Clear();
//清除 SQL 语句
Form1->Query1->SQL->Add(Form1->Memo1->Text);
//读入新的 SQL 语句
Form1->Query1->Open();
//打开数据库
}
//---------------------------------------------------------------------------
```

程序说明：

这样，在程序的运行过程中，当用户在文本框中输入 SQL 语句并且单击按钮“执行 SQL”时，程序就会激活控件的 Button1Click()事件，首先通过控件 Query 的 Close 方法把当前打开的数据库关闭，然后清除执行完的 SQL 语句，再通过语句 Form1->Query1->SQL->Add(Form1->Memo1->Text);从文本框中读取新的 SQL 语句，最后通过控件 Query 的 Open 方法来重新打开数据库，即执行新的 SQL 语句。

3. 响应数据浏览操作

在窗体上有 8 个用于进行数据浏览和编辑操作的按钮——Insert、Delete、Last、Next、Prior、First、Set Mark 和 To Mark 按钮，它们的功能如下所示：

- 控件 First：将数据库的指针移动到第一条记录处；
- 控件 Prior：将数据库的指针移动到前一条记录处；
- 控件 Next：将数据库的指针移动到下一条记录处；
- 控件 Last：将数据库的指针移动到最后一条记录处；
- 控件 Insert：在当前位置插入一条数据；
- 控件 Delete：将当前位置处的数据删除；
- 控件 Set Mark：在当前位置插入一个书签；
- 控件 To Mark：跳转到书签处。

下面仅以 Next 按钮为例来说明在程序设计中添加数据浏览操作代码的过程，其余几个按钮的代码请参看附后的源程序。在程序的设计阶段，用鼠标的左键双击窗体上的 Next 按钮，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口的 Button8Click()事件处理过程中，并且添加如下所示的响应代码：

```
void __fastcall TForm1::Button8Click(TObject *Sender)
{
Form1->Query1->Next();
//移动到下一条记录
}
//---------------------------------------------------------------------------
```

程序说明：

程序中各个按钮功能都是通过调用 Query 控件的相应方法来实现的，如 First 按钮就是通过调用 Query 控件的 First 方法来实现把当前数据库指针移动到第一条记录的功能的，而 Insert 按钮是通过调用 Query 控件的 Insert 方法来实现在当前位置插入一条记录的功能的。

在程序运行的过程中，当用户用鼠标的左键单击窗体上的各个按钮时，程序就会自动的调用 Query 控件的相应方法来实现对应的功能。如用户单击 Insert 按钮，程序就会调用 Insert 方法在数据库的当前位置插入一条记录，结果如图 7-19 所示。

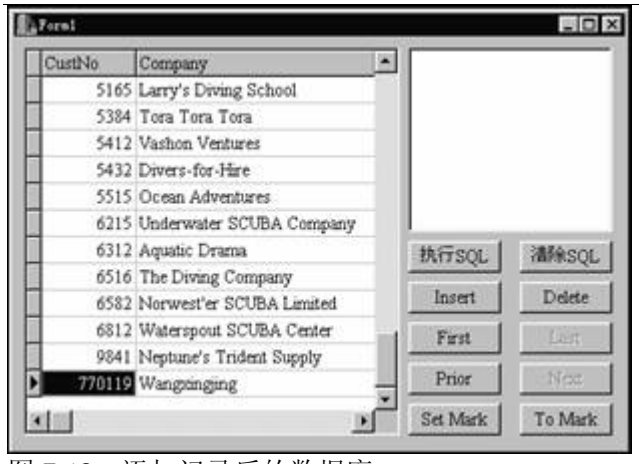


图 7-19　添加记录后的数据库

其它几个按钮的代码添加过程就不一一的加以介绍了，读者可以参看附后的源程序代码，在这里仅仅列举出各个按钮所对应的方法：

- First 按钮：Query 控件的 First 方法；
- Prior 按钮：Query 控件的 Prior 方法；
- Next 按钮：Query 控件的 Next 方法；
- Last 按钮：Query 控件的 Last 方法；
- Insert 按钮：Query 控件的 Insert 方法；
- Delete 按钮：Query 控件的 Delete 方法；
- Set Mark 按钮：Query 控件的 GetBookmark 方法；
- To Mark 按钮：Query 控件的 GotoBookmark 方法。

4. 响应计时器事件

本程序中的计时器控件的功能与 9.2 程序中的程序功能基本相同，添加代码的原理也是一致的，在这里就不多加叙述了，仅仅列出一段代码供读者参考。

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
if (Form1->Query1->Eof)
    {
    Form1->Button6->Enabled=false;
    Form1->Button8->Enabled=false;
    }
    else
    {
    Form1->Button6->Enabled=true;
    Form1->Button8->Enabled=true;
    }
if (Form1->Query1->Bof)
    {
    Form1->Button5->Enabled=false;
    Form1->Button7->Enabled=false;
    }
    else
    {
    Form1->Button5->Enabled=true;
    Form1->Button7->Enabled=true;
    }
//动态设置按钮的有效状态
}
//---------------------------------------------------------------------------
```

5. 运行程序

按照附后的源程序，添加剩余的程序代码后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的初始画面如图 7-20 所示。

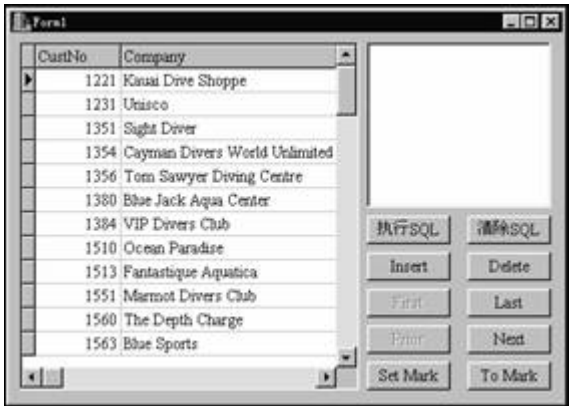


图 7-20　程序运行的初始画面

在窗体右侧的文本框中输入 SQL 语句“select * from items.db”（注：即显示数据库 items.db 中的所有数据）后，单击“执行 SQL”按钮，程序运行结果如图 7-21 所示，在数据控件中显示了所有字段的数据。

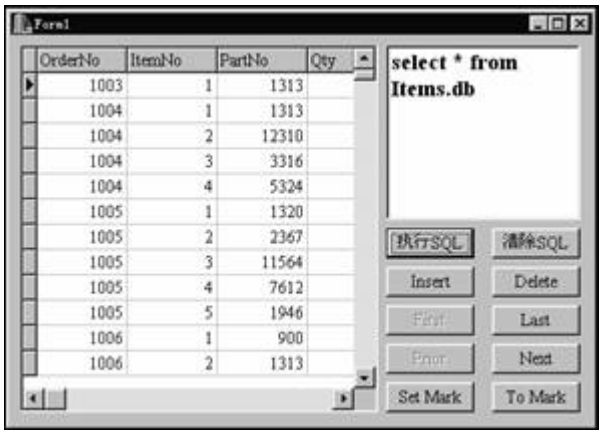


图 7-21　程序运行结果

程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "SQL.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)

#pragma resource "*.dfm"
TForm1 *Form1;
TBookmark SavePlace;
//---------------------------------------------------------------------------

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button9Click(TObject *Sender)
{
    SavePlace=Form1->Query1->GetBookmark();
    //在当前位置设置一个书签
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button10Click(TObject *Sender)
{
if (SavePlace!="")
    {
    Form1->Query1->GotoBookmark(SavePlace);
    //跳转到指定书签
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Query1BeforeClose(TDataSet *DataSet)
{
if (Form1->Query1->Modified)
    {
    if (MessageDlg("数据库已经被修改过，是否保存结果?", mtWarning, TMsgDlgButtons() <<
       mbYes << mbNo, 0) == mrYes)
    Query1->Post();
```

```
    else
    Query1->Cancel();
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Form1->Query1->Insert();
//插入一条新的记录
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
if (!Form1->Query1->Modified)
    {
    if (MessageDlg("数据库已经被修改过，是否保存结果?", mtWarning, TMsgDlgButtons() <<
       mbYes << mbNo, 0) == mrYes)
    Query1->Post();
    else
    Query1->Cancel();
    }
Query1->Close();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
Form1->Query1->Delete();
//删除一条记录
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
Form1->Query1->First();
//移动到第一条记录
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
Form1->Query1->Last();
//移动到最后一条记录
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button7Click(TObject *Sender)
{
Form1->Query1->Prior();
//移动到前一条记录
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button8Click(TObject *Sender)
{
Form1->Query1->Next();
//移动到下一条记录
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->Query1->Close();
//关闭数据库
Form1->Query1->SQL->Clear();
//清除 SQL 语句
Form1->Query1->SQL->Add(Form1->Memo1->Text);
//读入新的 SQL 语句
Form1->Query1->Open();
//打开数据库
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->Memo1->Clear();
//清除文本框中的内容
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
if (Form1->Query1->Eof)
    {
    Form1->Button6->Enabled=false;
    Form1->Button8->Enabled=false;
    }
    else
    {
    Form1->Button6->Enabled=true;
    Form1->Button8->Enabled=true;
    }
```

```
if (Form1->Query1->Bof)
    {
    Form1->Button5->Enabled=false;
    Form1->Button7->Enabled=false;
    }
    else
    {
    Form1->Button5->Enabled=true;
    Form1->Button7->Enabled=true;
    }
//动态设置按钮的有效状态
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Memo1->Clear();
//清空文本框
Form1->Timer1->Enabled=true;
//设置控件的有效状态
}
//---------------------------------------------------------------------
```

7.5 小 结

在本章中，通过了 4 个具体的程序示例向读者展示了 CBuilder 5 最吸引人之处——强大的数据库开发功能，在程序的设计过程中，我们利用大量的系统数据库控件（DataSource 控件、Table 控件和 Query 控件等），让读者对数据库设计的全过程和概念都有了系统的理解。

另外读者还可注意到，在 CBuilder 5 中设计数据库程序不用添加很多的代码，很多工作都是在控件属性设置的过程中完成的，设计是如此之简便，这也正是 CBuilder 开发数据库的优点之一。

第八章　多媒体开发和使用

个人电脑发展到今天，多媒体已经成为 PC 领域不可缺少的一部分，是计算机领域发展最为迅速的市场，而且随着技术的进步，多媒体正在慢慢的成为计算机接口的基础，在市场上也有很多的多媒体设计和制作软件，同 CBuilder 5 相比，或者是功能不够强大，或者是程序设计过于繁琐。

CBuilder 5 的出现打破了这种局面，利用 CBuilder 5 设计多媒体应用程序是非常简单的，为了方便用户，在 CBuilder 5 中提供了一个了多媒体控制接口（MCI）。通过 MCI（多媒体控制接口）这个公用的接口，用户无须介入实际的设备就可以操纵所有的多媒体设备。例如，对 MediaPlayer 组件的简单编程就可以实现视频文件、音频文件的播放和复制等功能。

好了，下面就首先介绍一下 CBuilder 5 中最常用的多媒体组件——MediaPlayer。

8.1 多媒体组件概述

在 CBuilder 5 中，MediaPlayer 组件通常位于组件工具栏上的 System 组件页上，在程序的设计过程中，用户可以通过以下几种方法添加 MediaPlayer 组件。

- 在组件工具栏 System 组件页上用鼠标的左键双击 MediaPlayer 组件的图标，在窗体上就会自动的添加一个 MediaPlayer 组件；
- 在组件工具栏 System 组件页上用鼠标的左键单击击 MediaPlayer 组件的图标，然后把鼠标移动到窗体上，按下的同时移动鼠标，在窗体上就会添加一个 MediaPlayer 组件。

如图 8-1 所示即为添加组件后的窗体。

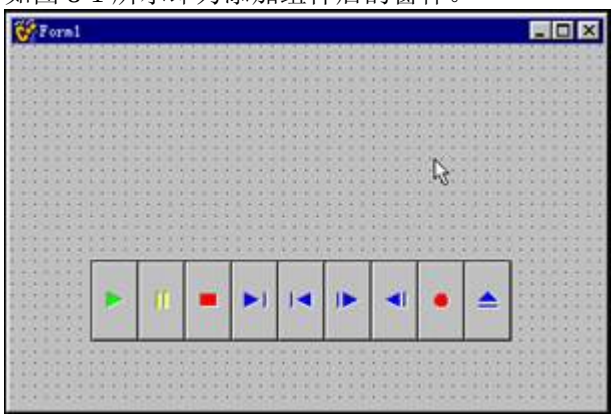


图 8-1　窗体上的 MediaPlayer 组件

MediaPlayer 组件有很多的属性和事件，能够熟练的运用这些属性和事件是进行多媒体程序设计的基础，在本节中，将会对 MediaPlayer 组件中常用的属性和事件作一个简单的介绍，详细的说明请读者参见有关的技术手册。

MediaPlayer 组件中常用的属性如表 8-1 所示。

表 8-1　MediaPlayer 组件常用的属性

Anchors	AutoEnable	AutoOpen
AutoRewind	ColoredButtons	Constraints
Cursor	DeviceType	Display
Enabled	EnabledButtons	FileName
Height	HelpContext	Hint
Left	Name	ParentShowHint
PopupMenu	Shareable	ShowHint
TabOrder	TabStop	Tag
Top	Visible	VisibleButtons
Width		

下面就对 MediaPlayer 组件中常用的属性作一个简单的说明。

8.1.1 AutoEnable 属性

AutoEnable 属性的功能是设置系统是否具有自动检测 MediaPlayer 组件中各个按钮有效状态能力，它的取值有 True 和 False 两种状态。
如果把 MediaPlayer 组件的 AutoEnable 属性值设置为 True，那么 MediaPlayer 组件就会根据组件和系统当前的状态自动的设置组件中哪个按钮应该处于有效的状态，哪个按钮应该处于无效的状态。
如图 8-2 所示即为组件的 AutoEnable 属性值设置为 True 时程序的运行结果。

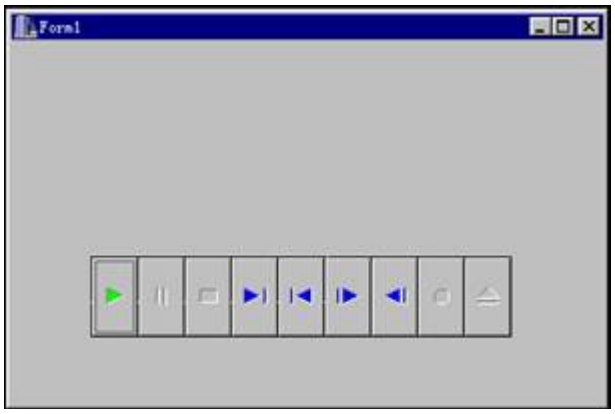


图 8-2　组件的 AutoEnable 属性值设置为 True
在程序的执行过程中，由于已经打开了多媒体播放设备和待播放的声音文件，所以在程序运行的开始阶段，Play 按钮处于有效的状态，而 Pause、Stop 等按钮则处于无效的状态。实现以上功能的程序代码如下所示，代码的具体功能在后面将详细的加以介绍，在这里就不加赘述了。

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->MediaPlayer1->VisibleButtons<<btPause<<btStop<<btStep<<btBack
<<btRecord<<btEject;
Form1->MediaPlayer1->EnabledButtons<<btPause<<btStop<<btStep<<btBack
<<btRecord<<btEject;
//设置处于有效状态的按钮
Form1->MediaPlayer1->FileName="c:\\pwin98\\media\\tada.wav";
//设置播放文件名
Form1->MediaPlayer1->Open();
//打开多媒体播放设备
Form1->MediaPlayer1->AutoEnable=true;
//设置组件具有自动检测有效状态的功能
}
//---------------------------------------------------------------------------
```

程序说明：
在程序的运行过程中，单击 Play 按钮，在音箱中就会播放声音文件 c:\windows\media\tada.wav，在播放的过程中，程序会自动的检测各个按钮的有效状态，如把 Play 按钮设置为无效的状态。如图 8-3 所示。

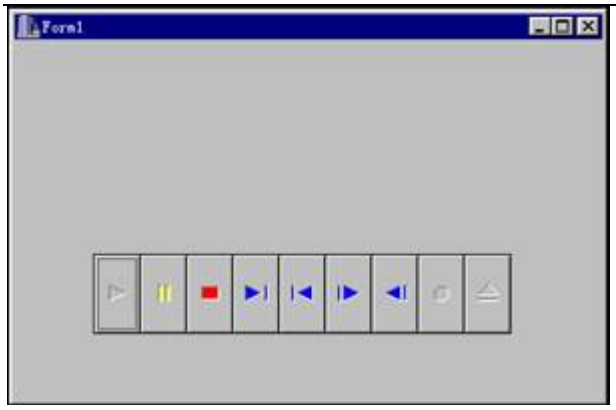


图 8-3　播放文件过程中的按钮有效状态

上面是将组件的 AutoEnable 属性值设置为 True，如果将组件的 AutoEnable 属性值设置为 False 时，MediaPlayer 组件就不会自动的检测和设置组件中各个按钮的有效状态，按钮的状态就需要用户自己去设置。

我们把上面的程序代码做一下小改动，如下所示。

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->MediaPlayer1->VisibleButtons<<btPause<<btStop<<btStep<<btBack;
Form1->MediaPlayer1->EnabledButtons<<btPause<<btStop<<btStep<<btBack;
//设置处于有效状态的按钮
Form1->MediaPlayer1->FileName="c:\\pwin98\\media\\tada.wav";
//设置播放文件名
Form1->MediaPlayer1->Open();
//打开多媒体播放设备
Form1->MediaPlayer1->AutoEnable=false;
//设置组件具有自动检测有效状态的功能
}
//---------------------------------------------------------------------------
```

做完以上的工作后，按键盘上的功能键 F9 运行程序，结果如图 8-4 所示。

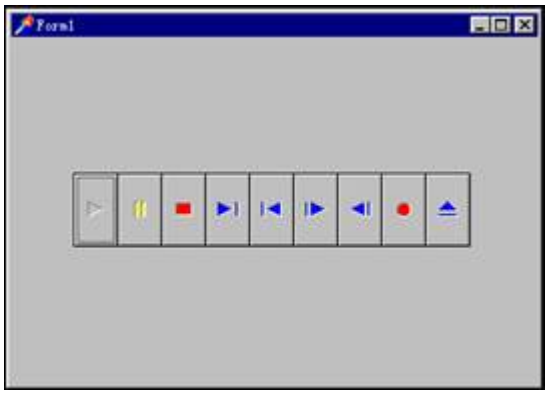


图 8-4　组件的 AutoEnable 属性值设置为 False

从以上的示例可以看出，当把 MediaPlayer 组件的 AutoEnable 属性值设置为 True 时，系统就会屏蔽程序中有关设置组件按钮状态的代码，反之当把 MediaPlayer 组件的 AutoEnable 属性值设置为 False 时，系统不会根据组件的状态来设置按钮的有效状态，这时就需要用户对 MediaPlayer 组件的按钮进行控制。

8.1.2 Display 属性

MediaPlayer 组件的 Display 属性用于设置在多媒体设备播放动画等多媒体文件时的容器，当然，在多媒体程序设计的过程中，也可以不给多媒体播放设备指定播放容器，这样用户就很难控制播放动画的位置，如图 8-5 所示。

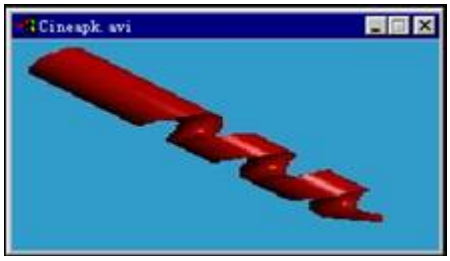


图 8-5　不指定播放容器

在大多数的情况下，在程序中都会给多媒体播放设备指定一个放映的容器，这个容器可以是按钮、文本框，甚至还可以是多媒体播放组件本身，下面就是一个指定按钮为动画播放容器的示例，具体步骤如下所示：

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，在组件工具栏上选择 System 选项后，在 MediaPlayer 组件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 MediaPlayer 组件，然后在向窗体上添加一个 Panel 组件，其中 Panel 组件用于在程序播放动画的过程中充当容器。添加组件后的窗体如图 8-6 所示。

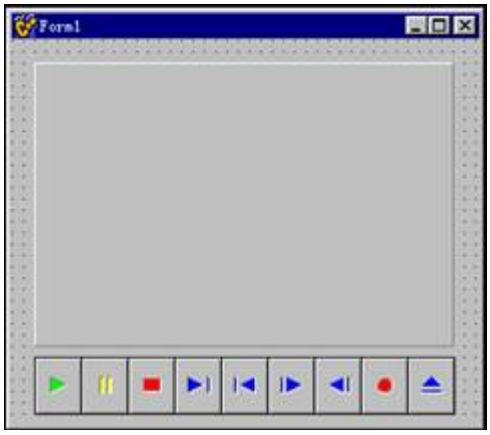


图 8-6　添加组件后的窗体

添加组件后，根据如图 8-6 所示的组件及窗体布局，设置相应的组件和窗体的属性，然后进入下一步。

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，缺省的情况下，光标应该位于在窗体的 FormCreate()事件的过程处理代码段中，在程序的适当位置添加下列代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->MediaPlayer1->AutoEnable=true;
//自动检测有效状态
Form1->MediaPlayer1->DeviceType=dtAVIVideo;
//设置播放设备类型
Form1->MediaPlayer1->FileName="d:\\ASP\聊天室\\Clock.avi";
//设置打开的文件名
Form1->MediaPlayer1->Open();
//打开多媒体播放设备
Form1->MediaPlayer1->Display=Form1->Panel1;
//设置播放容器
}
//---------------------------------------------------------------------------
```

程序说明：

在添加的程序中，需要注意的是语句 Form1->MediaPlayer1->Display=Form1->Panel1;它的作用是为程序中播放动画指定一个容器，在本程序中指定的动画播放容器为一个 Panel1 组件。做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择一个合适的文件名来保存文件。在键盘上按下功能键 F9 运行程序，程序运行的初始画面如图 8-7 所示。

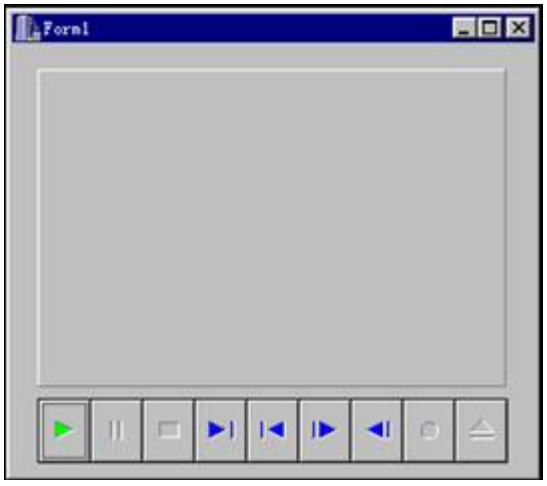
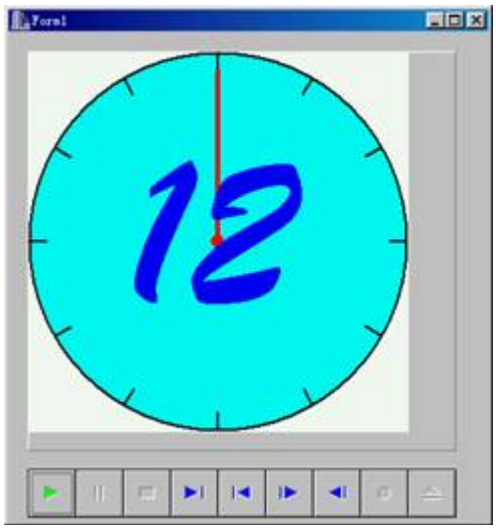


图 8-7　程序运行初始画面

用鼠标的左键单击 Play 按钮，在按钮组件上就会播放指定的动画文件，程序运行结果如图 8-8 所示。



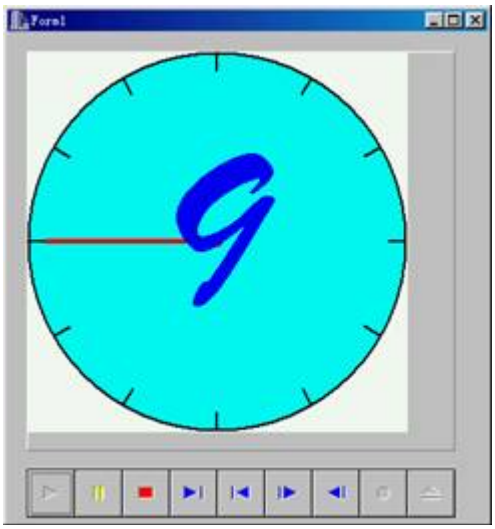


图 8-8　程序运行结果（Clock.avi）

8.1.3 Filename 属性

MediaPlayer 组件中的 FileName 属性用于为多媒体播放设备指定一个待播放的多媒体文件名，FileName 属性在组件的属性列表框中的位置如图 8-9 所示。

图 8-9　FileName 属性在组件的属性列表框中的位置

在 FileName 属性的设置输入框中可以直接输入多媒体文件的路径和文件名，当然也可以用鼠标左键单击属性输入框右侧的…按钮，就会弹出一个如图 8-10 所示的对话框，在该对话框中用户可以输入或者选择一个有效的多媒体文件，可以打开的多媒体文件类型有 All files (*.*)、 Wave files (*.wav)、Midi files (*.mid)和 Video for Windows (*.avi)等。

图 8-10　“打开”对话框

8.1.4 DeviceType 属性

MediaPlayer 组件中的 DeviceType 属性用于为打开多媒体设备指定一个设备类型，它的有效取值范围如下：

dtAutoSelect, dtAVIVideo, dtCDAudio, dtDAT, dtDigitalVideo, dtMMMovie, dtOther, dtOverlay, dtScanner, dtSequencer, dtVCR, dtVideodisc, dtWaveAudio

MediaPlayer 组件中的 DeviceType 属性的缺省取值为 dtAutoSelect，即程序在运行的过程中会根据所打开的文件类型自己设置设备类型。

用户可以在程序的设计过程中为多媒体设备指定一个设备类型，同样由于 DeviceType 属性在程序的运行过程中是可读可写的，所以可以在程序运行的过程中由程序代码指定，典型的 DeviceType 属性设置语句如下：

```
MediaPlayer1->DeviceType = dtAVIVideo;
//设置播放设备类型
```

8.1.5 EnabledButtons 属性、VisibleButtons 属性

MediaPlayer 组件中的 EnabledButtons 属性用于指定 MediaPlayer 组件中各个按钮的有效状态，而 VisibleButtons 属性则用来设置组件中的各个按钮的可见状态。
在缺省的状态下，所有的按钮都处于有效和可见的状态。
如图 8-11 为 EnabledButtons 属性和 VisibleButtons 属性都设置为缺省值时的显示情况。

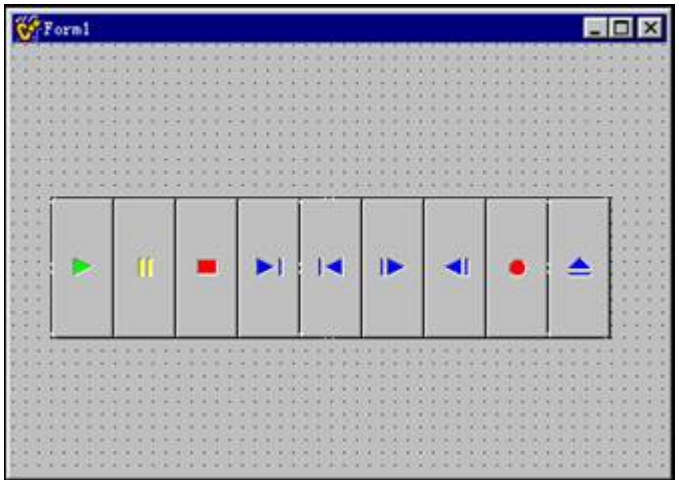


图 8-11　添加组件后的窗体
如果在程序的设计过程中，用鼠标的左键双击窗体的空白处，在弹出的代码窗口中添加下列代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
MediaPlayer1->FileName= "d:\\FileCopy.avi";
//指定播放文件
MediaPlayer1->VisibleButtons.Clear();
Form1->MediaPlayer1->EnabledButtons<<btPlay<<btPause<<btStop<<btStep<<btStep;
//设置按钮有效状态
Form1->MediaPlayer1->VisibleButtons<<btPlay<<btPause<<btStop<<btStep<<btStep<<btBack;
//设置按钮可见状态
MediaPlayer1->Open();
//打开播放设备
}
//---------------------------------------------------------------------------
```

完成以上工作后，按键盘上的功能键 F9 运行程序。
结果如图 8-12 所示。

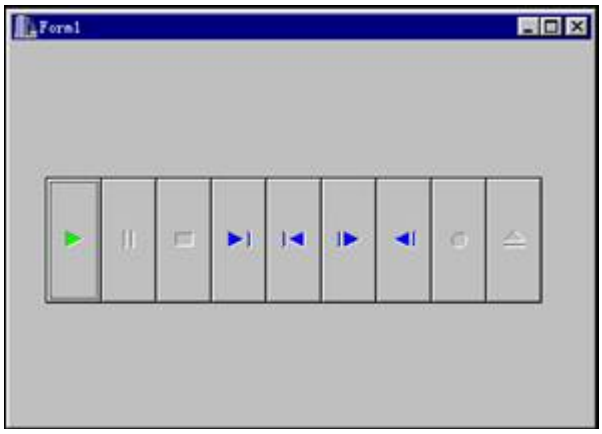


图 8-12　程序运行结果

当执行程序时，首先装入窗体，然后执行窗体 FormCreate()事件中的代码，即通过语句
Form1->MediaPlayer1->EnabledButtons<<btPlay<<btPause<<btStop<<btStep<<btStep;
来设置组件中各个按钮的有效状态；通过语句
Form1->MediaPlayer1->VisibleButtons<<btPlay<<btPause<<btStop<<btStep<<btStep<<btBack;
来设置组件中各个按钮的可见状态。

8.1.6 组件中常用的方法

MediaPlayer 组件中常用的方法有 Open()、Close()、Play()、Stop()、Pause()、Step()、Back()、Previous()、Next()和 Eject()方法。利用以上的方法可以达到打开多媒体设备、播放文件、弹出光碟等操作，调用这些方法的典型代码如下所示：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->MediaPlayer1->DeviceType=dtAVIVideo;
//设置设备类型
MediaPlayer1->FileName= "d:\\FileCopy.avi";
//指定播放文件
Form1->MediaPlayer1->EnabledButtons<<btPlay<<btPause<<btStop<<btStep<<btStep;
//设置按钮有效状态
Form1->MediaPlayer1->VisibleButtons<<btPlay<<btPause<<btStop<<btStep<<btStep<<btBack;
//设置按钮可见状态
MediaPlayer1->Open();
//打开多媒体播放设备
MediaPlayer1->Play();
//播放文件
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
Form1->MediaPlayer1->Close();
//关闭多媒体播放设备
}
//---------------------------------------------------------------------------
```

以上我们分别的讲述了 MediaPlayer 组件常用的属性和方法，下面将利用这些属性和方法编制一些多媒体处理应用程序。

8.2 声音文件播放器

声音属于多媒体的一个范畴，其实 Windows 操作系统自己就带有一个简单好用的声音播放程序——“录音机”，如图 8-13 所示。
当然，在 CBuilder 5 的多媒体应用中，制作一个简单的声音播放器是一件很容易的事情。

图 8-13　Windows 的声音播放器——“录音机”
下面就利用前面讲到的 MediaPlayer 组件常用的属性和方法来制作一个简单的声音播放程序，它的功能是能够任意的选择一个有效的以*.wav 为后缀的声音文件。
然后在程序中播放，在播放声音文件的同时，在窗体中还将有一个滑动条来随时的显示声音文件的播放进度。
制作能够实现以上功能的声音播放程序的具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，然后向空白的窗体添加三个 Button 组件、一个 TrackBar 组件、一个 MediaPlayer 组件、一个 OpenDialog 组件和一个 Timer 组件。
添加组件后的窗体如图 8-14 所示。

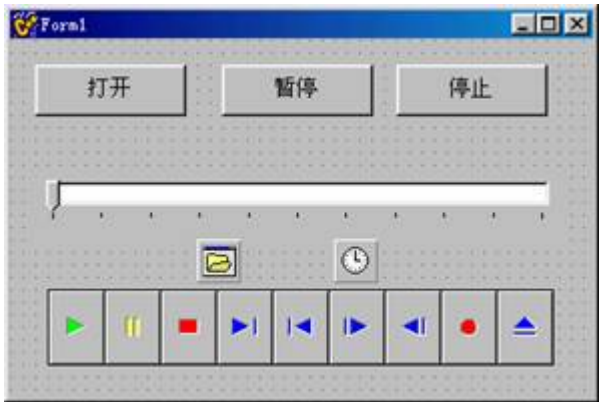


图 8-14　添加组件后的窗体
所添加的组件的功能如下所示：

- MediaPlayer 组件：为播放声音文件和系统多媒体设备之间建立连接的关系，同时负责打开多媒体播放设备；
- Button 组件：实现打开文件、暂停播放和结束程序运行等功能；
- TrackBar 组件：显示程序中播放声音文件的进度，能够给用户留下一个比较直观的印象；
- OpenDialog 组件：它的作用是显示一个的“打开”的对话框，同时还能够实现打开声音文件的功能，在其中还可以对多媒体文件进行筛选操作；
- Timer 组件：为程序运行提供时间控制，如随时显示声音文件播放进度。

2. 属性设置

添加组件后的下一步工作就是为所添加的组件设置各自的属性，窗体与组件的属性设置可以在程序的设计阶段通过改变属性列表框中的内容来达到，同样，也可以在程序中通过代码来达到，这两种方法的效果是相同的。
在本示例程序中，窗体及组件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 382
  Height = 256
  Caption = 'Form1'
  OnClose = FormClose
  OnCreate = FormCreate
  object MediaPlayer1: TMediaPlayer
    Left = 24
    Top = 160
    Width = 325
    Height = 49
    Display = MediaPlayer1
    FileName = 'C:\WINDOWS\MEDIA\The Microsoft Sound.wav'
  end
  object Button1: TButton
    Left = 24
    Top = 112
    Width = 97
    Height = 33
    Caption = '打开'
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 136
    Top = 112
    Width = 97
    Height = 33
    Caption = '暂停'
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 248
    Top = 112
    Width = 97
    Height = 33
    Caption = '停止'
    OnClick = Button3Click
  end
  object TrackBar1: TTrackBar
    Left = 24
    Top = 48
    Width = 337
    Height = 41
    Orientation = trHorizontal
    Frequency = 1
    Position = 0
    SelEnd = 0
    SelStart = 0
  end
  object OpenDialog1: TOpenDialog
```

```
    Left = 120
    Top = 80
  end
  object Timer1: TTimer
    Enabled = False
    Interval = 100
    OnTimer = Timer1Timer
    Left = 224
    Top = 80
  end
end
```

3. 设置文件过滤器

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体的 FormCreate()事件处理代码中，并且添加如下所示的设置文件过滤器的代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->OpenDialog1->Filter="All Files(*.*)|*.*|WAVE Files(*.wav)|*.wav";
//设置文件过滤器
Form1->Timer1->Enabled=false;
//设置组件有效状态
}
//---------------------------------------------------------------------
```

4. 响应“打开”按钮

在“打开文件”的按钮上双击鼠标的左键，就会弹出一个如图 8-14 所示的代码窗口，在其中就可以添加对按钮“打开文件”的响应代码如下所示。

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //打开一个对话框
    {
    Form1->MediaPlayer1->FileName=Form1->OpenDialog1->FileName;
    //设置播放文件
    Form1->MediaPlayer1->Open();
    //打开多媒体播放设备
    Form1->Timer1->Enabled=true;
    //设置组件有效状态
    }
}
//---------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，如果用户用鼠标的左键单击按钮“打开文件”，就会激活组件的 Button1Click()事件，然后程序通过语句：

```
Form1->OpenDialog1->Execute();
```

来打开一个对话框，在其中用户可以选择一个有效的声音文件，最后通过：

```
Form1->MediaPlayer1->FileName    =Form1->OpenDialog1->FileName;
Form1->MediaPlayer1->Open();
```

来打开一个多媒体播放设备。其中代码窗口如图 8-15 所示。

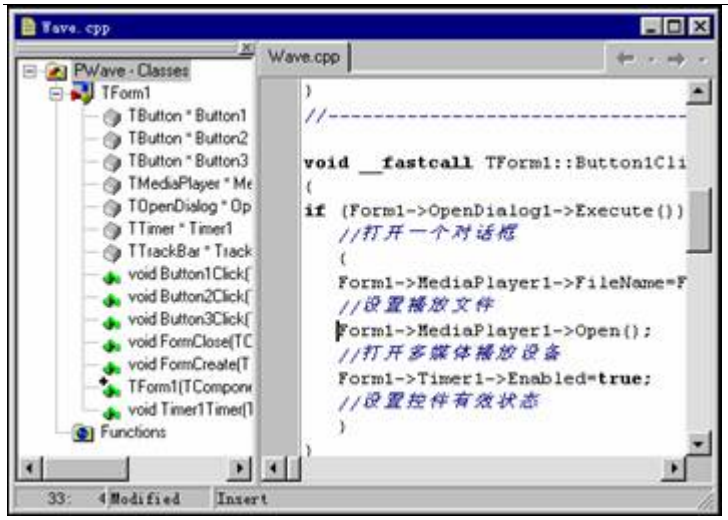


图 8-15　程序中的代码窗口

5. 响应计时器事件

窗体上 Timer 组件的作用是为程序运行提供时间控制，如随时显示声音文件播放进度等，为此就需要在程序的设计阶段将 Timer 组件设置为有效的状态，并且为组件指定一个响应的时间长度。

在本示例程序中，Timer 组件的时间控制设置为 100，它的意思是在程序运行的过程中，每隔 100 毫秒就会自动的激活一个 Timer1Timer()事件，为了在程序中动态的显示声音文件播放的进度，可以在 Timer1Timer()事件中添加如下代码：

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->TrackBar1->Min=0;
//设置滑动条最小值
Form1->TrackBar1->Max=Form1->MediaPlayer1->Length;
//设置滑动条最大值
Form1->TrackBar1->Position=Form1->MediaPlayer1->Position;
//设置滑动条的位置
}
//---------------------------------------------------------------------------
```

6. 运行程序

提示：

✍　在这里对程序其它代码的添加过程就不一一加以介绍了，如果读者有什么不清楚之处，请参看附后的源程序代码。

做完以上的工作后，存储文件，按键盘上的功能键【F9】运行程序，程序运行的初始画面如图 8-16 所示。

图 8-16　程序运行的初始画面

在程序运行的过程中，由于没有给多媒体播放设备指定程序播放文件，所以 MediaPlayer 组件的 Play、Pause 和 Stop 等按钮都处于无效的状态。

用鼠标的左键单击“打开文件”按钮，在屏幕上就会弹出一个如图 8-17 所示的对话框。

图 8-17　打开文件对话框

在打开文件的对话框中用键盘输入一个路径和文件名，或者用鼠标来选择一个有效的声音文件，单击“打开”按钮返回到程序运行的画面中，这时 MediaPlayer 组件的 Play 按钮变为有效的状态。

单击 Play 按钮，在音箱中就会播放选中的声音文件，同时滑动条会动态的显示处声音播放的进度。

程序运行结果如图 8-18 所示。

图 8-18　程序运行结果

示例程序中的完整代码如下所示。

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Wave.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
```

```
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->OpenDialog1->Filter="All Files(*.*)|*.*|WAVE Files(*.wav)|*.wav";
//设置文件过滤器
Form1->Timer1->Enabled=false;
//设置组件有效状态
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //打开一个对话框
    {
    Form1->MediaPlayer1->FileName=Form1->OpenDialog1->FileName;
    //设置播放文件
    Form1->MediaPlayer1->Open();
    //打开多媒体播放设备
    Form1->Timer1->Enabled=true;
    //设置组件有效状态
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->MediaPlayer1->Pause();
//暂停播放
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Close();
//结束程序的运行
}
//---------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->TrackBar1->Min=0;
Form1->TrackBar1->Max=Form1->MediaPlayer1->Length;
Form1->TrackBar1->Position=Form1->MediaPlayer1->Position;
//设置滑动条的位置
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
Form1->MediaPlayer1->Close();
//关闭多媒体播放设备
}
```

//----------

8.3 MIDI 音乐播放器

MIDI 音乐也是声频的一部分。只是由于 MIDI 音乐本身的特殊性，所以本章特地将 MIDI 音乐与普通的声音区别开来。
下面就利用前面讲到的 MediaPlayer 组件常用的属性和方法来制作一个 MIDI 音乐播放器程序，它的功能是能够任意的选择一个有效的以*.mid 为后缀的音乐文件，然后在程序中播放，在播放音乐的同时，在窗体中还将有一个滚动条来随时的显示声音文件的播放进度。
制作能够实现以上功能的音乐播放程序的具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，然后向空白的窗体添加四个 Button 组件、一个 ScrollBar 组件、一个 MediaPlayer 组件、一个 OpenDialog 组件和一个 Timer 组件，添加组件后的窗体如图 8-19 所示。

图 8-19　添加组件后的窗体
所添加的组件的功能如下所示：

- MediaPlayer 组件：为播放音乐文件和系统多媒体设备之间建立连接的关系，同时负责打开多媒体播放设备；
- Button 组件：实现打开文件、播放文件、暂停播放和停止播放等功能；
- ScrollBar 组件：显示程序中播放声音文件的进度，并且能够通过拖动滚动条来手动控制播放的进度；
- OpenDialog 组件：它的作用是显示一个的“打开”的对话框，同时还能够实现打开音乐文件的功能，在其中还可以对多媒体文件进行筛选操作；
- Timer 组件：为程序运行提供时间控制，如随时显示声音文件播放进度。

2. 属性设置

添加组件后的下一步工作就是为所添加的组件设置各自的属性，窗体与组件的属性设置可以在程序的设计阶段通过改变属性列表框中的内容来达到，同样，也可以在程序中通过代码来达到，这两种方法的效果是相同的。
在本示例程序中，窗体及组件的属性设置如下所示：

```
object Form1: TForm1
   Left = 192
   Top = 107
   Width = 329
   Height = 232
```

```
  Caption = 'Form1'
  OnCreate = FormCreate
  object MediaPlayer1: TMediaPlayer
    Left = 32
    Top = 24
    Width = 253
    Height = 41
    FileName =
    'c:\Windows\system\sound\Fmtest.mid'
  end
  object ScrollBar1: TScrollBar
    Left = 32
    Top = 72
    Width = 249
    Height = 25
    LargeChange = 5
    OnScroll = ScrollBar1Scroll
  end
  object Button1: TButton
    Left = 32
    Top = 112
    Width = 81
    Height = 25
    Caption = '打开'
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 32
    Top = 152
    Width = 81
    Height = 25
    Caption = '播放'
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 200
    Top = 112
    Width = 81
    Height = 25
    Caption = '暂停'
    OnClick = Button3Click
  end
  object Button4: TButton
    Left = 200
    Top = 152
    Width = 81
    Height = 25
    Caption = '停止'
    OnClick = Button4Click
  end
  object Timer1: TTimer
    Interval = 100
    OnTimer = Timer1Timer
    Left = 144
    Top = 112
  end
  object OpenDialog1: TOpenDialog
    Filter = 'MIDI 音乐文件(*.mid)|*.mid'
    Left = 144
    Top = 152
```

```
   end
end
```

3. 程序的初始化

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体的 FormCreate()事件处理代码中，并且添加如下所示的程序初始化代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Timer1->Enabled=false;
//设置计时器工作状态
Form1->Button2->Enabled=false;
Form1->Button3->Enabled=false;
Form1->Button4->Enabled=false;
//设置按钮有效状态
}
//---------------------------------------------------------------------------
```

程序说明：

窗体 FormCreate 事件中的代码在程序运行初期就会被执行，由于在程序刚刚开始运行时还没有选择有效的音乐文件，所以 Timer1 组件的有效状态设置为无效，然后通过三条语句：

```
Form1->Button2->Enabled=false;
Form1->Button3->Enabled=false;
Form1->Button 4->Enabled=false;
```

设置按钮组件的有效状态。

经过以上初始化处理后的窗体如图 8-20 所示。

图 8-20　程序初始化后的窗体

4. 打开一个音乐文件

在“打开文件”的按钮上双击鼠标的左键，就会弹出一个代码窗口，在其中就可以添加对按钮“打开文件”的响应代码如下所示。

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->OpenDialog1->InitialDir="e:\\midi";
//设置对话框缺省路径
Form1->OpenDialog1->Title="请选择一个音乐文件名:";
//设置对话框标题
Form1->OpenDialog1->DefaultExt="mid";
//设置对话框缺省文件名
if (Form1->OpenDialog1->Execute())
    {
    Form1->MediaPlayer1->FileName=Form1->OpenDialog1->FileName;
```

```
    //返回音乐文件名
    Form1->MediaPlayer1->Open();
    //打开多媒体播放设备
    Form1->Timer1->Enabled=true;
    Form1->Button2->Enabled=true;
    //设置按钮有效状态
    }
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，如果用户用鼠标的左键单击按钮“打开文件”，就会激活组件的Button1Click()事件，程序首先设置对话框缺省路径为 e:\midi，对话框的标题是“请选择一个音乐文件名:”，在对话框中显示的文件缺省扩展名是 mid，然后调用 OpenDialog1 组件的Execute()方法来显示一个对话框，在其中用户可以选择一个有效的音乐文件，最后通过Form1->MediaPlayer1->Open()；来打开一个多媒体播放设备，并且设置了按钮的有效状态。

5. 播放音乐文件

在窗体上的按钮“播放文件”的作用是播放一个用户选中的音乐文件，在 MediaPlayer 组件中有一个 Play()方法可以在程序运行的过程中播放一个打开的音乐文件，为此，在按钮“播放文件”的响应过程中添加如下代码：

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->MediaPlayer1->Play();
//播放音乐
Form1->Button3->Enabled=true;
Form1->Button4->Enabled=true;
}
//---------------------------------------------------------------------------
```

值得注意的是，在程序的运行过程中，只有用户在对话框中选择一个有效的音乐文件，单击“打开”按钮，才能成功的向系统多媒体设备传送文件的路径和将要打开的文件名，这时单击“播放文件”按钮才能调用 MediaPlayer 组件中的 Play()方法来播放一个选中的音乐文件。

6. 响应计时器事件

窗体上 Timer 组件的作用是为程序运行提供时间控制，如随时显示音乐文件播放进度等。在本示例程序中，Timer 组件的时间控制设置为 100，它的意思是在程序运行的过程中，每隔 100 毫秒就会自动的激活一个 Timer1Timer()事件，为了在程序中动态的显示音乐文件播放的进度，可以在 Timer1Timer()事件中添加如下代码：

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->ScrollBar1->Min=0;
Form1->ScrollBar1->Max=Form1->MediaPlayer1->Length;
Form1->ScrollBar1->Position=Form1->MediaPlayer1->Position;
//随时跟踪滚动条位置
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，如果组件 Timer1 处于有效的状态，那么每隔 100 毫秒就会自动的激活一个 Timer1Timer() 事件，通过语句 Form1->ScrollBar1->Position=Form1->MediaPlayer1->Position;在程序中动态的显示音乐文件播放的进度。

7. 调节音乐播放进度

在本示例程序中，对音乐播放进度的调节是通过滚动条来实现，响应滚动条动态拖动的事件是 ScrollBar1Scroll，为此，添加如下所示的代码：

```
void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,TScrollCode ScrollCode, int &ScrollPos)
{
Form1->MediaPlayer1->Pause();
//暂停播放
Form1->MediaPlayer1->Position=Form1->ScrollBar1->Position;
//拖放播放进度
Form1->MediaPlayer1->Play();
//开始播放
}
//---------------------------------------------------------------------------
```

程序说明：

如果在程序运行的过程中用户用鼠标左键拖动滚动条上的滑动块，就会激活组件的 ScrollBar1Scroll 事件，程序首先调用 MediaPlayer1 组件的 Pause()方法来暂停音乐的播放，然后把当前音乐播放的进度通过语句 Form1->MediaPlayer1->Position=Form1->ScrollBar1->Position；调节到当前滑动块的位置，最后调用 MediaPlayer1 组件的 Play()方法重新播放音乐文件。

8. 运行程序

在这里对程序其它代码的添加过程就不一一加以介绍了，如果读者有什么不清楚之处，请参看附后的源程序代码。做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序。

在程序运行的过程中，由于没有给多媒体播放设备指定程序播放文件，所以窗体上的“播放文件”、“暂停播放”和“停止播放”等按钮都处于无效的状态。用鼠标的左键单击“打开文件”按钮，在屏幕上就会弹出一个提示用户选择音乐文件的对话框。

在对话框中用键盘输入一个路径和文件名，或者用鼠标来选择一个有效的音乐文件，单击“打开”按钮返回到程序运行的画面中，这时窗体上的“播放文件”按钮变为有效的状态，单击“播放文件”按钮，在音箱中就会播放选中的音乐文件，同时滚动条会动态的显示出音乐播放的进度。

程序运行结果如图 8-21 所示。

图 8-21　程序运行结果

示例程序中的完整代码如下所示。

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
```

```
#include "Midi.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Form1->Timer1->Enabled=false;
//设置计时器工作状态
Form1->Button2->Enabled=false;
Form1->Button3->Enabled=false;
Form1->Button4->Enabled=false;
//设置按钮有效状态
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->OpenDialog1->InitialDir="e:\\midi";
//设置对话框缺省路径
Form1->OpenDialog1->Title="请选择一个音乐文件名:";
//设置对话框标题
Form1->OpenDialog1->DefaultExt="mid";
//设置对话框缺省文件名
if (Form1->OpenDialog1->Execute())
    {
    Form1->MediaPlayer1->FileName=Form1->OpenDialog1->FileName;
    //返回音乐文件名
    Form1->MediaPlayer1->Open();
    //打开多媒体播放设备
    Form1->Timer1->Enabled=true;
    Form1->Button2->Enabled=true;
    //设置按钮有效状态
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Form1->MediaPlayer1->Pause();
//暂停播放或继续播放
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->MediaPlayer1->Play();
//播放音乐
Form1->Button3->Enabled=true;
Form1->Button4->Enabled=true;
//设置按钮有效状态
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
Form1->MediaPlayer1->Stop();
//停止播放音乐
Form1->Button3->Enabled=false;
Form1->Button4->Enabled=false;
//设置按钮有效状态
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
Form1->MediaPlayer1->Pause();
//暂停播放
Form1->MediaPlayer1->Position=Form1->ScrollBar1->Position;
//拖放播放进度
Form1->MediaPlayer1->Play();
//开始播放
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->ScrollBar1->Min=0;
Form1->ScrollBar1->Max=Form1->MediaPlayer1->Length;
//设置滚动条参数
Form1->ScrollBar1->Position=Form1->MediaPlayer1->Position;
//随时跟踪滚动条位置
}
//---------------------------------------------------------------------------
```

8.4 视频播放程序

前面详细的讲述了如何利用 MediaPlayer 组件制作声音播放程序的示例，但是作为多媒体本身来讲，不仅仅包括声音，还应该有动画、音乐等，所以接下来制作一个能够播放视频文件的动画播放程序。

在 Windows 98/Windows 2000 的操作系统中，有一个叫做“ActiveMoive 控制”的应用程序，它不但可以播放以*.avi 为结尾的视频文件，同时也可以播放其它类型的动画文件，甚至可以播放声音文件。

下面我们就自己动手来制作一个动画播放器，预计实现的功能如下：

- 能够从文件列表中选择想要播放的文件；
- 在程序中 MediaPlayer 组件处于可见状态；
- 动画的播放等操作是通过用户自定义的按钮完成的；
- 在程序中能够通过一个滚动条来显示动画的播放进度；

能够实现以上功能的动画播放器程序制作步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，然后向空白的窗体添加 8 个 Button 组件、1 个 ScrollBar 组件、1 个 MediaPlayer 组件、1 个 OpenDialog 组件和 1 个 Timer 组件，添加组件后的窗体如图 8-22 所示。

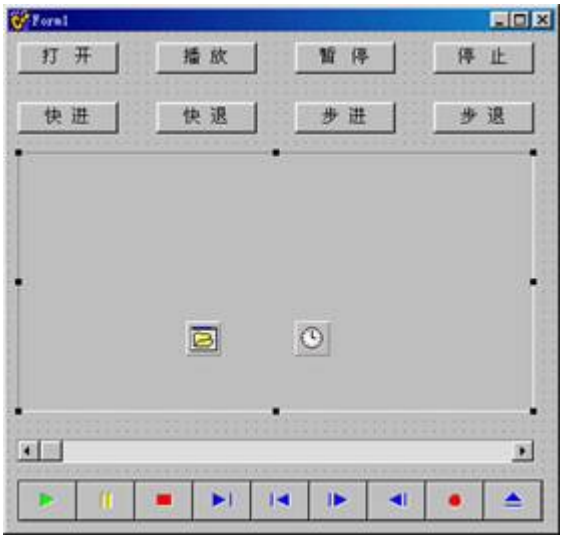


图 8-22　添加组件后的窗体

所添加的组件的功能如下所示：

- MediaPlayer 组件：为播放动画文件和系统多媒体设备之间建立连接的关系，同时负责打开多媒体播放设备；
- Button 组件：实现打开文件、播放文件和步进播放动画等功能；
- ScrollBar 组件：显示程序中播放动画的进度，同时还可以手动调节动画播放进度；
- OpenDialog 组件：它的作用是显示一个提示用户选择动画文件的对话框，同时还

能够实现打开动画文件的功能，在其中还可以对多媒体文件进行筛选操作；

- Timer 组件：为程序运行提供时间控制，如随时显示动画文件播放进度。

2. 属性设置

添加组件后的下一步工作就是为所添加的组件设置各自的属性，在本示例程序中，窗体及组件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 454
  Height = 288
  Caption = 'Form1'
  OnCreate = FormCreate
  object MediaPlayer1: TMediaPlayer
    Left = 16
    Top = 208
    Width = 406
    Height = 41
    AutoRewind = False
    DeviceType = dtAVIVideo
    Display = Panel1
  end
  object Button1: TButton
    Left = 16
    Top = 16
    Width = 81
    Height = 25
    Caption = '播　放'
    OnClick = Button1Click
  end
  object Panel1: TPanel
    Left = 112
    Top = 16
    Width = 217
    Height = 153
  end
  object Button3: TButton
    Left = 16
    Top = 112
    Width = 81
    Height = 25
    Caption = '停　　止'
    OnClick = Button3Click
  end
  object Button4: TButton
    Left = 16
    Top = 160
    Width = 81
    Height = 25
    Caption = '快　进'
    OnClick = Button4Click
  end
  object Button5: TButton
    Left = 344
    Top = 16
    Width = 81
    Height = 25
```

```
    Caption = '快　退'
    OnClick = Button5Click
  end
  object Button6: TButton
    Left = 344
    Top = 64
    Width = 81
    Height = 25
    Caption = '步　进'
    OnClick = Button6Click
  end
  object Button7: TButton
    Left = 344
    Top = 112
    Width = 81
    Height = 25
    Caption = '步　退'
    OnClick = Button7Click
  end
  object Button8: TButton
    Left = 344
    Top = 160
    Width = 81
    Height = 25
    Caption = '打　　开'
    OnClick = Button8Click
  end
  object Button2: TButton
    Left = 16
    Top = 64
    Width = 81
    Height = 25
    Caption = '暂　　停'
    OnClick = Button2Click
  end
  object ScrollBar1: TScrollBar
    Left = 112
    Top = 168
    Width = 217
    Height = 17
    OnScroll = ScrollBar1Scroll
  end
  object OpenDialog1: TOpenDialog
    Left = 160
    Top = 72
  end
  object Timer1: TTimer
    Interval = 100
    Left = 248
    Top = 72
  end
end
```

3. 程序的初始化

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体的 FormCreate()事件处理代码中，并且添加如下所示的程序初始化代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
Form1->MediaPlayer1->Visible=true;
//设置组件的可见状态
Form1->MediaPlayer1->Display=Form1->Panel1;
//设置播放容器
Form1->Button1->Enabled=false;
Form1->Button2->Enabled=false;
Form1->Button3->Enabled=false;
Form1->Button4->Enabled=false;
Form1->Button5->Enabled=false;
Form1->Button6->Enabled=false;
Form1->Button7->Enabled=false;
Form1->Button8->Enabled=true;
//设置按钮有效状态
Form1->ScrollBar1->Enabled=false;
Form1->Timer1->Enabled=false;
//设置组件有效状态
}
//---------------------------------------------------------------------
```

程序说明：

在开始执行程序时，系统首先激活窗体的 FormCreate()事件，然后通过代码 Form1->MediaPlayer1->Display=Form1->Panel1；设置动画播放的容器是窗体上的 Panel1 组件，最后通过 Form1->ScrollBar1->Enabled=false；Form1->Timer1->Enabled=false；Form1->Button 1->Enabled=false；Form1->Button2->Enabled=false；Form1->Button3->Enabled=false；Form 1->Button4->Enabled=false；Form1->Button5->Enabled=false；Form1->Button6->Enabled=fal se；Form1->Button7->Enabled=false;和 Form1->Button8->Enabled=true;设置了窗体上各个组件的有效状态和可见状态。

经过以上初始化处理后的窗体如图 8-23 所示。

图 8-23　初始化后的窗体

4. 响应计时器事件

窗体上 Timer 组件的作用是为程序运行提供时间控制，如随时显示动画文件播放进度等，为此就需要在程序的运行阶段将 Timer 组件设置为有效的状态，并且为组件指定一个响应的时间长度。在本示例程序中，Timer 组件的时间控制设置为 100，它的意思是在程序运行的过程中，每隔 100 毫秒就会自动的激活一个 Timer1Timer()事件，为了在程序中动态的显示动画文件播放的进度，可以在程序设计的过程中用鼠标双击窗体中的 Timer 组件，在屏幕上就会弹出一个如图 8-24 所示的代码窗口。

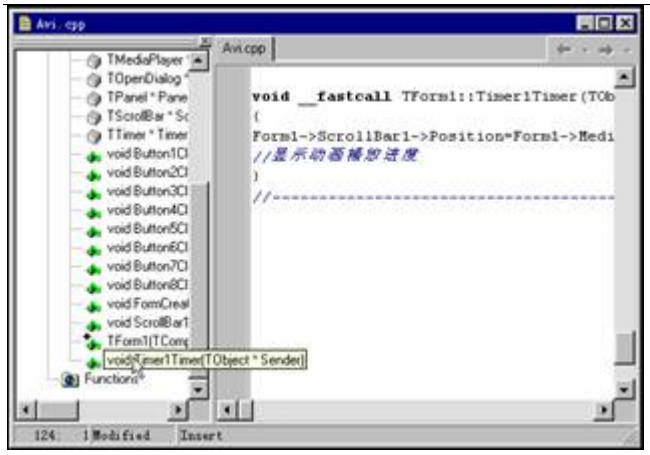


图 8-24　Timer 组件的代码窗口

将光标移动到代码窗口中的 Timer1Timer()事件中，并且在它的事件处理中添加下列代码：

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->ScrollBar1->Position=Form1->MediaPlayer1->Position;
//显示动画播放进度
}
//---------------------------------------------------------------------
```

程序说明：

由于组件属性设置的原因，在程序的执行过程中，每隔 100 毫秒的时间就会自动的激活一个 Timer1Timer()事件，通过语句

Form1->ScrollBar1->Position=Form1->MediaPlayer1->Position;

来随时的显示动画播放的进度。

5. 打开动画文件

在窗体上放置有一个“打开”按钮，它实现诸如打开对话框以及为设置播放文件等功能，所以在它的 Button8Click()事件中添加如下所示的代码：

```
void __fastcall TForm1::Button8Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //显示对话框
    {
    Form1->MediaPlayer1->FileName=Form1->OpenDialog1->FileName;
    //设置播放文件
    Form1->MediaPlayer1->Open();
    //打开多媒体播放设备
    Form1->ScrollBar1->Min=0;
    Form1->ScrollBar1->Max=Form1->MediaPlayer1->Length;
    //初始化滚动条的状态
    Form1->ScrollBar1->Enabled=true;
    Form1->Button1->Enabled=true;
    }
}
//---------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，用鼠标的左键单击“打开”按钮，就会自动的激活组件的 Button8Click()事件，程序首先调用 OpenDialog1 组件的 Execute()方法来显示一个提示用户选择动画文件的对话框，用户在该对话框中可以选择待播放的动画文件，之后程序就会通过语句 Form1->MediaPlayer1->FileName=Form1->OpenDialog1->FileName；把用户的选择传递给多媒体播放设备，并且调用组件 MediaPlayer1 的 Open()方法来打开多媒体播放设备，然后通过一系列语句设置滚动条和其它按钮组件的参数和有效状态。

6. 播放动画文件

在窗体上的按钮“播放”的作用是播放一个用户选中的动画文件，在 MediaPlayer 组件中有一个 Play()方法可以在程序运行的过程中播放一个打开的动画文件，为此，在按钮“播放”的响应过程中添加如下代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->MediaPlayer1->Play();
//播放动画
Form1->Button2->Enabled=true;
Form1->Button3->Enabled=true;
Form1->Button4->Enabled=true;
Form1->Button5->Enabled=true;
Form1->Button6->Enabled=true;
Form1->Button7->Enabled=true;
//设置按钮的有效状态
}
//---------------------------------------------------------------------
```

程序说明：

值得注意的是，在程序的运行过程中，只有用户在对话框中选择一个有效的动画文件，单击“打开”按钮，才能成功的向系统多媒体设备传送文件的路径和将要打开的文件名，这时单击“播放”按钮才能调用 MediaPlayer 组件中的 Play()方法来播放一个选中的音乐文件，并且设置窗体上其它组件的有效状态。

7. 手动调节动画播放进度

在本示例程序中，对动画播放进度的调节是通过滚动条来实现，响应滚动条动态拖动的事件是 ScrollBar1Scroll，为此，添加如下所示的代码：

```
void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
Form1->MediaPlayer1->Pause();
//暂停播放
Form1->MediaPlayer1->Position=Form1->ScrollBar1->Position;
//拖动到指定位置
Form1->MediaPlayer1->Play();
//重新播放
}
//---------------------------------------------------------------------
```

程序说明：

如果在程序运行的过程中用户用鼠标左键拖动滚动条上的滑动块，就会激活组件的 ScrollBar1Scroll 事件，程序首先调用 MediaPlayer1 组件的 Pause()方法来暂停动画的播放。然后把当前动画播放的进度通过语句 Form1->MediaPlayer1->Position=Form1->ScrollBar1->Position；调节到当前滑动块的位置，最后调用 MediaPlayer1 组件的 Play()方法重新播放动画。

8. 运行程序

在这里对程序其它代码的添加过程就不一一加以介绍了，如果读者有什么不清楚之处，请参看附后的源程序代码。做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序。

在程序运行的过程中，由于没有给多媒体播放设备指定程序播放文件，所以窗体上的“播放”、“暂停”和“停止”等按钮都处于无效的状态。

用鼠标的左键单击“打开”按钮，在屏幕上就会弹出一个提示用户选择动画文件的对话框。在对话框中用键盘输入一个路径和文件名，或者用鼠标来选择一个有效的动画文件，单击“打开”按钮返回到程序运行的画面中，这时窗体上的“播放”按钮变为有效的状态。

单击“播放”按钮，在窗体的 Panel1 组件上就会播放选中的动画文件，同时滚动条会动态的显示出动画播放的进度。

程序运行结果如图 8-25 所示。

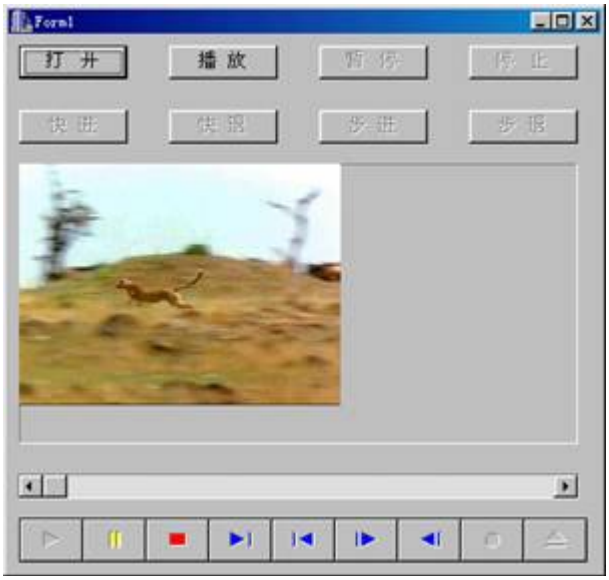


图 8-25　程序运行结果

在窗体上的各个按钮的功能如下：

- “打开”按钮：打开一个对话框，从中可以选择一个有效的动画文件；
- “播放”按钮：如果用户打开了多媒体播放设备，那么单击这个按钮就可以播放指定的动画文件；
- “暂停”按钮：能够暂停动画文件的播放动作；
- “停止”按钮：停止播放动画文件；
- “快进”按钮：把当前的播放为止定位在动画文件的末尾；
- “快退”按钮：把把当前的播放为止定位在动画文件的开头；
- “步进”按钮：根据组件的属性设置来控制单步前进的桢数；
- “步退”按钮：根据组件的属性设置来控制单步步退的桢数。

在上面的程序运行过程中，用户可以通过“暂停”、“快进”和“停止”等按钮对动画文件的播放过程加以控制，也可以用鼠标来拖动滚动条上的滑动块来改变动画播放进度。

完整的程序源代码如下所示。

程序清单

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Avi.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
Form1->MediaPlayer1->Visible=true;
//设置组件的可见状态
Form1->MediaPlayer1->Display=Form1->Panel1;
//设置播放容器
Form1->Button1->Enabled=false;
Form1->Button2->Enabled=false;
Form1->Button3->Enabled=false;
Form1->Button4->Enabled=false;
Form1->Button5->Enabled=false;
Form1->Button6->Enabled=false;
Form1->Button7->Enabled=false;
Form1->Button8->Enabled=true;
//设置按钮有效状态
Form1->ScrollBar1->Enabled=false;
Form1->Timer1->Enabled=false;
//设置组件有效状态
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form1->MediaPlayer1->Play();
//播放动画
Form1->Button2->Enabled=true;
Form1->Button3->Enabled=true;
Form1->Button4->Enabled=true;
Form1->Button5->Enabled=true;
Form1->Button6->Enabled=true;
Form1->Button7->Enabled=true;
//设置按钮的有效状态
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form1->MediaPlayer1->Pause();
//暂停播放操作
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Form1->MediaPlayer1->Stop();
//停止播放
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
Form1->MediaPlayer1->Next();
//快进
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
Form1->MediaPlayer1->Previous();
//返回到文件开头
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
Form1->MediaPlayer1->Step();
//步进式播放动画
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button7Click(TObject *Sender)
{
Form1->MediaPlayer1->Back();
//步退一格
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button8Click(TObject *Sender)
{
if (Form1->OpenDialog1->Execute())
    //显示对话框
    {
    Form1->MediaPlayer1->FileName=Form1->OpenDialog1->FileName;
    //设置播放文件
    Form1->MediaPlayer1->Open();
    //打开多媒体播放设备
    Form1->ScrollBar1->Min=0;
    Form1->ScrollBar1->Max=Form1->MediaPlayer1->Length;
    //初始化滚动条的状态
    Form1->ScrollBar1->Enabled=true;
    Form1->Button1->Enabled=true;
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
Form1->MediaPlayer1->Pause();
//暂停播放
Form1->MediaPlayer1->Position=Form1->ScrollBar1->Position;
//拖动到指定位置
Form1->MediaPlayer1->Play();
//重新播放
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->ScrollBar1->Position=Form1->MediaPlayer1->Position;
//显示动画播放进度
}
//---------------------------------------------------------------------------
```

8.5 CD 光碟播放器

前面详细的讲述了如何利用 MediaPlayer 组件制作声音播放程序和动画播放程序，接下来制作一个能够播放光碟音乐的 CD 播放器。
在 Windows 98/2000 等操作系统中，有一个叫做“CD 播放器”的应用程序，在 Windows 的开始菜单中选择“程序”/“附件”/“多媒体”/“CD 播放器”项，在屏幕上就会弹出一个如图 8-26 所示的播放 CD 音乐的程序执行画面。

图 8-26 CD 播放器应用程序
下面我们就自己动手来制作一个 CD 播放器，预计能够实现播放 CD 音乐、自动弹碟等功能，实现以上功能的 CD 播放器程序制作步骤如下所示。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，然后向空白的窗体添加五个 Button 组件、五个 Edit 组件、五个 Label 组件、一个 MediaPlayer 组件和一个 Timer 组件，添加组件后的窗体如图 8-27 所示。

图 8-27 添加组件后的窗体
所添加的组件的功能如下所示：

- MediaPlayer 组件：为播放 CD 和系统多媒体设备之间建立连接的关系，同时负责打开多媒体播放设备；
- Button 组件：实现播放 CD、暂停播放动作和停止播放等功能；
- Timer 组件：为程序运行提供时间控制，如随时显示 CD 播放位置；
- Edit 组件：作为 CD 播放过程中播放信息显示的容器；
- Label 组件：显示固定的文本信息。

2. 设置组件属性

添加组件后的下一步工作就是为所添加的组件设置各自的属性，在本示例程序中，窗体中各个按钮组件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 352
  Height = 292
  Caption = 'Form1'
  OnCreate = FormCreate
  object Label1: TLabel
    Left = 136
    Top = 32
    Width = 70
    Height = 16
    Caption = '曲目总数：'
  end
  object Label2: TLabel
    Left = 136
    Top = 72
    Width = 70
    Height = 16
    Caption = '播放位置：'
  end
  object Label3: TLabel
    Left = 136
    Top = 112
    Width = 70
    Height = 16
    Caption = '唱片长度：'
  end
  object Label4: TLabel
    Left = 136
    Top = 152
    Width = 70
    Height = 16
    Caption = '起始位置：'
  end
  object Label5: TLabel
    Left = 136
    Top = 192
    Width = 70
    Height = 16
    Caption = '播放模式：'
  end
  object Button1: TButton
    Left = 16
    Top = 16
    Width = 97
    Height = 33
    Caption = '播放  CD'
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 16
    Top = 56
    Width = 97
```

```
    Height = 33
    Caption = '暂停播放'
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 16
    Top = 96
    Width = 97
    Height = 33
    Caption = '下一首'
    OnClick = Button3Click
  end
  object Button4: TButton
    Left = 16
    Top = 136
    Width = 97
    Height = 33
    Caption = '前一首'
    OnClick = Button4Click
  end
  object Button5: TButton
    Left = 16
    Top = 176
    Width = 97
    Height = 33
    Caption = '停止播放'
    OnClick = Button5Click
  end
  object MediaPlayer1: TMediaPlayer
    Left = 16
    Top = 216
    Width = 307
    Height = 33
  end
  object Edit1: TEdit
    Left = 216
    Top = 24
    Width = 105
    Height = 24
    Text = 'Edit1'
  end
  object Edit2: TEdit
    Left = 216
    Top = 64
    Width = 105
    Height = 24
    Text = 'Edit2'
  end
  object Edit3: TEdit
    Left = 216
    Top = 104
    Width = 105
    Height = 24
    Text = 'Edit3'
  end
  object Edit4: TEdit
    Left = 216
    Top = 144
    Width = 105
    Height = 24
```

```
      Text = '0'
    end
    object Edit5: TEdit
      Left = 216
      Top = 184
      Width = 105
      Height = 24
      Text = 'Edit5'
    end
    object Timer1: TTimer
      Interval = 100
      OnTimer = Timer1Timer
      Left = 120
    end
end
```

3. 程序的初始化

在程序的设计过程中，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出如图 8-28 所示的代码窗口，在缺省的情况下，显示的应该是窗体的 FormCreate()事件。

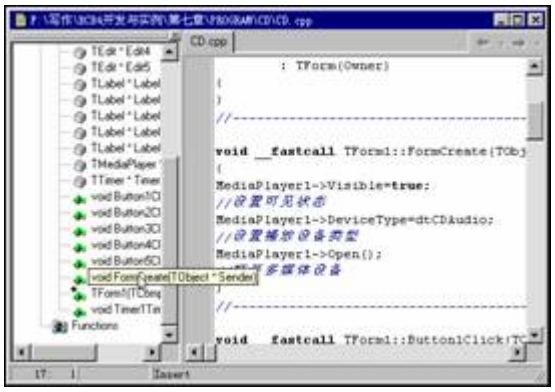


图 8-28　弹出的代码窗口

把光标移动到代码窗口中的窗体 FormCreate()事件处理过程中，并且添加如下所示的程序初始化代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
MediaPlayer1->Visible=true;
//设置可见状态
MediaPlayer1->DeviceType=dtCDAudio;
//设置播放设备类型
MediaPlayer1->Open();
//打开多媒体设备
}
//---------------------------------------------------------------------------
```

开始执行程序时，系统首先调用窗体的 FormCreate()事件，程序首先通过语句设置了 MediaPlayer 组件的可见状态，然后通过语句 MediaPlayer1->DeviceType=dtCDAudio;设置多媒体设备类型为 dtCDAudio，然后通过调用 MediaPlayer1 组件的 Open()方法来打开 CD 播放设备，为用户播放 CD 音乐作好了准备工作。

4. 响应计时器事件

窗体上 Timer 组件的作用是为程序运行提供时间控制，如随时显示 CD 播放位置等，为此就需要在程序的运行阶段将 Timer 组件设置为有效的状态，并且为组件指定一个响应的时间长度。在本示例程序中，Timer 组件的时间控制设置为 100，它的意思是在程序运行的过程中，

每隔 100 毫秒就会自动的激活一个 Timer1Timer()事件，在程序设计的过程中用鼠标双击窗体中的 Timer 组件，在屏幕上就会弹出的代码窗口中添加下列代码：

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->Edit1->Text=IntToStr(Form1->MediaPlayer1->Tracks);
//显示曲目总数
Form1->Edit2->Text=IntToStr(Form1->MediaPlayer1->Position);
//显示播放位置
Form1->MediaPlayer1->TimeFormat=tfMilliseconds;
//设置时间格式
Form1->Edit3->Text=FloatToStr((Form1->MediaPlayer1->Length)/60000)+"分";
//显示唱片长度
Form1->Edit4->Text=IntToStr(Form1->MediaPlayer1->Start);
//显示播放起始位置
switch (Form1->MediaPlayer1->Mode)
        {
        case mpNotReady: Form1->Edit5->Text="Not ready";
                break;
        case mpStopped:    Form1->Edit5->Text="Stopped";
                break;
        case mpPlaying:    Form1->Edit5->Text="Playing";
                break;
        case mpRecording:Form1->Edit5->Text="Recording";
                break;
        case mpSeeking:    Form1->Edit5->Text="Seeking";
                break;
        case mpPaused:     Form1->Edit5->Text="Paused";
                break;
        case mpOpen:         Form1->Edit5->Text="Open";
        }
//显示播放模式
}
//---------------------------------------------------------------------------
```

由于组件属性设置的原因，在程序的执行过程中，每隔 100 毫秒的时间就会自动的激活一个 Timer1Timer()事件，在它的响应事件中通过一系列语句在窗体的文本框中分别显示 CD 曲目总数、当前播放位置、时间格式、唱片长度、播放起始位置和播放模式。

5. 响应按钮事件

在窗体上我们放置了五个 Button 组件，它们的作用如下所示：

- “播放 CD”按钮：开始播放光盘中的 CD 音乐；
- “暂停播放”按钮：能够暂停 CD 的播放动作；
- “停止播放”按钮：停止播放 CD；
- “前一首”按钮：跳转到当前曲目的前一首；
- “后一首”按钮：跳转到当前曲目的后一首；

在程序的设计过程中，通过双击相应按钮组件就可以直接进入该组件的事件处理过程中，在这里对各个按钮的代码添加过程就不加赘述了，如果读者有什么不清楚的地方，请参看附后的源程序代码。

6. 运行程序

完成以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择一个合适的文件名存储文件。按键盘上的功能键 F9 运行程序，程序运行结果如图 8-29 所示。

图 8-29　程序运行结果

附程序完整源代码如下所示：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "CD.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
MediaPlayer1->Visible=true;
//设置可见状态
MediaPlayer1->DeviceType=dtCDAudio;
//设置播放设备类型
MediaPlayer1->Open();
//打开多媒体设备
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Form1->MediaPlayer1->Play();
    //播放 CD
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Form1->MediaPlayer1->Pause();
    //暂停播放 CD
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
```

```
    Form1->MediaPlayer1->Next();
    //播放下一首曲目
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
    Form1->MediaPlayer1->Previous();
    //播放前一首曲目
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
    Form1->MediaPlayer1->Stop();
    //停止播放 CD
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
Form1->Edit1->Text=IntToStr(Form1->MediaPlayer1->Tracks);
//显示曲目总数
Form1->Edit2->Text=IntToStr(Form1->MediaPlayer1->Position);
//显示播放位置
Form1->MediaPlayer1->TimeFormat=tfMilliseconds;
//设置时间格式
Form1->Edit3->Text=FloatToStr((Form1->MediaPlayer1->Length)/60000)+"分";
//显示唱片长度
Form1->Edit4->Text=IntToStr(Form1->MediaPlayer1->Start);
//显示播放起始位置
switch (Form1->MediaPlayer1->Mode)
            {
            case mpNotReady: Form1->Edit5->Text="Not ready";
                    break;
            case mpStopped:    Form1->Edit5->Text="Stopped";
                    break;
            case mpPlaying:    Form1->Edit5->Text="Playing";
                    break;
            case mpRecording:Form1->Edit5->Text="Recording";
                    break;
            case mpSeeking:    Form1->Edit5->Text="Seeking";
                    break;
            case mpPaused:     Form1->Edit5->Text="Paused";
                    break;
            case mpOpen:         Form1->Edit5->Text="Open";
            }
//显示播放模式
}
//---------------------------------------------------------------------------
```

8.6 小　结

本章集中的介绍了利用 CBuilder 5 进行多媒体程序设计的一般思想，通过本章的学习，读者可以有重点的掌握多媒体组件 MediaPlayer 的应用，利用它们可以设计自己的声音播放器、MIDI 播放器和视频播放器等多媒体应用程序。

学习完本章，我们希望读者能够在本章的基础上，自己独立设计一个综合的“媒体播放机”应用程序，在其中不但可以播放 Wave 声音、MIDI 音乐和 CD 音乐，还可以播放电影视频以及动画等，相信读者一定能运用本章所学是知识顺利完成。

第九章　Windows 信息共享技术的应用

应用程序间的数据交换和信息共享是像 Windows 这样的多任务环境的重要特性，有时候在程序中需要调用程序内部另一文档或另外一个应用程序的数据或信息，如何实现这种程序之间或者在程序内部的信息共享呢？在应用程序中，通常要执行一些比较费时的任务，例如读取磁盘、科学计算等，通过什么技术可以避免无谓的等待呢？

为了解决以上的问题，CBuilder 5 提供了如下的解决方案：剪贴板（ClipBoard）、对象链接与嵌入（OLE）以及进程（Thread）技术。

所以，在本书的最后一章，我们将专门讨论剪贴板、对象链接与嵌入（OLE）和进程等 Windows 信息共享技术的应用。

9.1　剪贴板及其应用

本质上，剪贴板只是一个全局内存块。当一个应用程序将数据传送给剪贴板后，通过修改内存块分配标志，把相关内存块的所有权从应用程序移交给 Windows 自身。其他应用程序可以通过一个句柄找到这个内存块，从而能够从内存块中读取数据。这样就实现了数据在不同应用程序间的传输。

剪贴板虽然功能较为简单，且不能实现实时传输，但却是更为复杂的 DDE 和 OLE 的基础。对于一些只是偶尔需要使用其他应用程序数据的程序来说，使用剪贴板不失为一种方便、快捷的方式。

CBuilder 5 把剪贴板的大部分功能封装到一个 TClipboard 类中，同时把使用频度最高的文本传输功能（包括 DBImage 的图像传输功能）置入相应部件作为部件的方法，从而使用户可以十分方便地使用剪贴板进行编程。

9.1.1 使用剪贴板传输文本

剪贴板传输文本主要是应用如下的 3 个方法：CopyToClipboard、CutToClipboard 和 PasteFromClipboard。包含这些方法的部件如表 9-1 所示。

表 9-1 包含剪贴板方法的部件

方 法	部 件
CopyToClipboard	TDBEdit TDBMemo TDBImage
	Tedit Tmemo TmaskEdit
	TOLEContainer TDDEServerItem
CutToClipboard	TDBEdit TDBMemo TDBImage Tedit
	Tmemo TMaskEdit
PasteFromClipboard	TDBEdit TDBMemo TDBImage
	Tedit Tmemo TMaskEdit

说明：

✍ 除了 TDBImage 外，其余全是有关文本的控件。

在把文本传输到剪贴板之前，文本必须被选中。

若选 TMaskEdit 的 AutoSelect 属性为 True，则当 MaskEdit 获得输入焦点时文本自动被选中；若选 TEdit、TMemo 的 HideSelection 属性为 True，则失去焦点时，文本选中状态自动隐藏，重新获得焦点时再显示。

下面的语句把 MaskEdit 中选中的文本剪切到剪贴板：

```
MaskEdit ->CutToClipboard;
```

下面的语句把剪贴板中的文本粘贴到 Memo 的当前光标处：

```
Memo->PasteFromClipboard;
```

利用剪贴板类也可以实现文本的传输，见下节内容中的介绍。

9.1.2 剪贴板类

为方便剪贴板的操作，CBuilder 5 在 Clipbrd 库单元中定义了一个 TClipboard 类，并且预定义了一个变量 Clipboard 作为类 TClipboard 的实例，从而使用户在绝大多数场合不必自己去定义一个 TClipboard 的实例。

利用剪贴板类可以进行文本、图像和部件的传输，剪贴板类为实现这些方法提供了相应的属性和方法。表 9-2、表 9-3 列出了 TClipboard 属性和方法的意义。

表 9-2　TClipboard 的属性

属　性	意　义
AsText	保存剪贴板的文本，只有运行时才可设置
FormatCount	可用剪贴板格式的数目
Formats	可用剪贴板格式链

表 9-3　TClipboard 的方法

方　法	参　数	意　义
Clear	无	清除剪贴板的内容
Assign	Source:Tpersistent	把 Source 参数指定的对象拷贝到剪贴板，常用于图形、图像对象
Open	无	打开剪贴板，阻止其他应用程序改变它的内容
Close	无	关闭打开的剪贴板
SetComponent	Source:Tpersistent	把部件拷贝到剪贴板
GetComponent	OwnerParent:Tpersistent	从剪贴板取回一个部件并放置
SetAsHandle	Format:Word 返回类型：Thandle	把指定格式数据的句柄交给剪贴板
GetAsHandle	Format:Word 返回类型：THandle	返回剪贴板指定格式数据的句柄
HasFormat	Format:Word 返回类型：Boolean	判断剪贴板是否拥有给定的格式
SetTextBuf	Buffer:Pchar	设置剪贴板的文本内容

剪贴板中可能的数据格式如表 9-4 所示。

表 9-4　剪贴板数据格式及其意义

数据格式	意　义
CF_TEXT	文本。每行以 CF_LF 结束，nil 标志文本结束
CF_BITMAP	Windows 位图
CF_METAFILE	Windows 元文件
CF_PICTURE	TPicture 类型的对象
CF_OBJECT	任何 TPersistent 类型的对象

利用 TClipboard 实现文本的传输可以使用 AsText 属性和 SetTextBuf 方法。

AsText 属性为非控件部件的剪贴板操作提供了方便。如：

Clipboard-> AsText = Form1->Caption ;

作用是把 Form1 的标题拷贝到剪贴板。

Label1->Caption = Clipboard->AsText;

作用把剪贴板中的文本写入 Label1。

SetTextBuf 用于把超过 255 个字符的字符串拷入剪贴板。

9.1.3 利用剪贴板传输图像

1. 拷贝操作

Image 部件上的内容和窗体上的图形可以直接拷贝到剪贴板。图像拷贝利用 Clipboard 的 Assign 方法。
例如：
Clipboard->Assign(Image1->Picture);
把 Image1 上的图像拷贝到剪贴板。

2. 剪切操作

图像的剪切是首先把图像拷贝到剪贴板，而后在原位置用空白图像进行覆盖。
下面一段程序表示了图像的剪切：

```
Trect   *Arect;

void _fastcall   TForm1::Cut1Click(Tobject   *Sender)
      {
  Clipboard->Assign(Image1->Picture);
       Do while( Image->Canvas)
          {
     CopyMode = cmWhiteness;
           ARect = Rect(0, 0, Image.Width, Image.Height);
           CopyRect(ARect, Image.Canvas, ARect);
           CopyMode = cmSrcCopy;
          }
}
```

3. 粘贴操作

从剪贴板上粘贴图像，首先检测剪贴板上的数据格式。如果格式为 CF_BITMAP，则调用目标位图的 Assign 方法粘贴图像。
程序清单如下：

```
Tbitmap * Bitmap;

     Void _fastcall   TForm1::PasteButtonClick( Tobject   *Sender)
     {
  if Clipboard.HasFormat(CF_BITMAP)
       {      Bitmap = Tbitmap->Create;
            try
              Bitmap->Assign(Clipboard);
              Image->Canvas->Draw(0, 0, Bitmap);
            finally
```

```
    }
  Bitmap->Free;
```

9.1.4 建立自己的剪贴板观察程序

前面已经说过，剪贴板是我们在不同应用程序之间进行数据交换的容器，而我们可以建立自己的剪贴板观察程序。
其实，在 Windows 操作系统的“附件”中就带有一个“剪贴板查看器”应用程序，只要用户在 Windows 的“开始”菜单中。
选择“程序”/“附件”/“系统工具”中的“剪贴板查看程序”选项，在屏幕上就会弹出一个“剪贴板查看器”。
本节，我们使用 CBuilder 5 建立一个类似但是功能相对简单的多的剪贴板查看器。
下面就利用 Bitbtn 控件来制作一个“剪贴板查看器”应用程序，它可以实时的显示当前剪贴板上的文本和图像数据。
具体的程序设计步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 CBuilder 5 的集成开发环境中就会弹出一个新建的窗体，接着向窗体上添加 6 个 Button 控件、1 个 OpenDialog 控件、1 个 OpenPictureDilog 控件和 1 个 RichEdit 控件、1 个 Bitbtn 控件。
注意：

- RichEdit 控件和 Bitbtn 控件放置的位置是重合的，且大小一样，用来分别显示文本和图像数据。为了让读者看清楚，我们特意将 Bitbtn 控件调节的比 Richedit 控件稍大，如图 9-1 所示。

各个控件的功能将在后面分别加以介绍。
添加控件后的窗体如图 9-1 所示。

图 9-1 添加控件后的窗体
在上面所添加的各种控件中，在功能上可以分为两类：显示类和操作类。

- 所谓显示类控件，即是指在窗体上 RichEdit 控件，它的作用是在程序运行的过程中显示剪贴板上的数据；
- 所谓的操作类控件指的是窗体上的按钮控件和对话框控件，它们将要完成对文件的各种操作。

添加到窗体上各个控件的属性设置如下所示：

```
object Button1: TButton
  Left = 312
  Top = 120
```

```
    Width = 97
    Height = 33
    Caption = '查看剪贴板'
    TabOrder = 0
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 312
    Top = 48
    Width = 97
    Height = 33
    Caption = '拷贝数据'
    TabOrder = 1
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 48
    Top = 264
    Width = 97
    Height = 33
    Caption = '打开文本'
    TabOrder = 2
    OnClick = Button3Click
  end
  object Button4: TButton
    Left = 176
    Top = 264
    Width = 97
    Height = 33
    Caption = '打开图像'
    TabOrder = 3
    OnClick = Button4Click
  end
  object Button5: TButton
    Left = 312
    Top = 192
    Width = 97
    Height = 33
    Caption = '清除剪贴板'
    TabOrder = 4
    OnClick = Button5Click
  end
  object BitBtn1: TBitBtn
    Left = 16
    Top = 16
    Width = 281
    Height = 225
    TabOrder = 5
  end
  object RichEdit1: TRichEdit
    Left = 16
    Top = 16
    Width = 281
    Height = 225
    Font.Charset = GB2312_CHARSET
    Font.Color = clWindowText
    Font.Height = -14
    Font.Name = 'Times New Roman'
    Font.Style = []
    Lines.Strings = ( 'RichEdit1')
```

```
    ParentFont = False
    TabOrder = 6
  end
  object Button6: TButton
    Left = 312
    Top = 264
    Width = 89
    Height = 33
    Caption = '退　出'
    TabOrder = 7
    OnClick = Button6Click
  end
  object OpenDialog1: TOpenDialog
    Left = 32
    Top = 160
  end
  object OpenPictureDialog1: TOpenPictureDialog
    Left = 88
    Top = 160
  end
```

2. 程序初始化

程序的初始化过程，主要是指对窗体 FormCreate()事件的初始化，在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Clipboard()->Clear();
 //清除剪贴板上的数据
 Form1->Button1->Enabled=true;
 Form1->Button2->Enabled=true;
 Form1->Button3->Enabled=true;
 Form1->Button4->Enabled=true;
 Form1->Button5->Enabled=true;
 Form1->Button6->Enabled=true;
 //设置按钮有效状态
 Form1->RichEdit1->Text="";
 //清空文本框中的内容
 Form1->OpenDialog1->Title="请选择一个文本文件：";
 //设置对话框标题
 Form1->OpenDialog1->InitialDir="c:\\pwin98\\";
 //设置缺省路径
 Form1->OpenDialog1->Filter="文本文件(*.txt)|*.txt";
 //设置文件过滤条件
 Form1->OpenPictureDialog1->Title="请选择一个图像文件：";
 //设置对话框标题
 Form1->OpenPictureDialog1->InitialDir="c:\\pwin98\\";
 //设置缺省路径
 Form1->OpenPictureDialog1->Filter="文本文件(*.bmp)|*.bmp";
 //设置文件过滤条件
}
//---------------------------------------------------------------------
```

程序说明：

窗体 FormCreate()事件中的语句 Clipboard->Clear; 用来清除剪贴板上的数据，然后通过五条语句：

```
 Form1->Button1->Enabled=true;
 Form1->Button2->Enabled=true;
 Form1->Button3->Enabled=true;
 Form1->Button4->Enabled=true;
 Form1->Button5->Enabled=true;
 Form1->Button6->Enabled=true;
```

来设置各个按钮的有效状态。

3. 查看剪贴板

在程序运行的过程中，用户在“查看剪贴板”控件上按下鼠标的左键时，在查看类控件上就会显示出当前剪贴板上的文本和图像数据，为了实现这一功能，需要在“查看剪贴板”控件的事件响应中添加如下代码：

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   if (Clipboard()->HasFormat(CF_TEXT))
        //如果剪贴板上的数据为文本格式
       {
       Form1->RichEdit1->Visible=true;
       Form1->BitBtn1->Visible=false;
       //设置控件可见状态
       RichEdit1->Text = Clipboard()->AsText;
       //显示剪贴板上的文本
       }
   else if (Clipboard()->HasFormat(CF_BITMAP))
     //如果剪贴板上的数据为图像格式
     {
     Form1->BitBtn1->Visible=true;
     Form1->RichEdit1->Visible=false;
     //设置控件可见状态
     Form1->BitBtn1->Glyph->Assign(Clipboard());
     //显示剪贴板上的图像
     }
     else
     {
     Form1->BitBtn1->Visible=false;
     Form1->RichEdit1->Visible=false;
     //设置控件可见状态
     }
}
//---------------------------------------------------------------------------
```

程序说明：

如果用户在程序运行的过程中，用鼠标的左键单击“查看剪贴板”按钮，程序首先会判断当前剪贴板上的数据格式；

如果当前剪贴板上放置的是图像数据，那么通过下面的语句在显示类控件上显示当前剪贴板上的图像：

```
Form1->BitBtn1->Glyph->Assign(Clipboard)
```

反之如果当前剪贴板上放置的是文本数据，那么就通过下面的语句在显示类控件上显示当前剪贴板上的文本，最后设置按钮控件的有效状态：

```
Form1->BitBtn1->Caption=Clipboard ->AsText
```

4. 拷贝数据

对剪贴板操作中相当重要的一个环节就是如何把当前选中的数据拷贝到剪贴板上，其中的一

个难点就就在于如何判断用户所选中的数据类型，在本程序中是通过显示类控件上的文本状态来判断数据类型的，具体的代码如下所示：

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 if (Form1->RichEdit1->Visible)
       //如果文本框控件可见
       Clipboard()->AsText=RichEdit1->Text;
       //把当前显示文本复制到剪贴板上
 if (Form1->BitBtn1->Visible)
       //如果 BitBtn1 控件可见
       Clipboard()->Assign(Form1->BitBtn1->Glyph);
       //把当前显示图像复制到剪贴板上
}
//---------------------------------------------------------------------------
```

程序说明：

在程序运行的过程中，用户用鼠标左键单击按钮“拷贝数据”时。如果显示类控件上的文本为空，那么通过语句 clipboard()->Assign(form1->bitbtn1->glyph)来把当前按钮上的图像复制到剪贴板上。否则通过语句 clipboard()->AsText=Richedit1->Text；把当前按钮上的文本复制到剪贴板上。

5. 运行程序

按照附后的源程序，添加剩余的程序代码后，选择菜单“File”中的“Save All”选项，在弹出的对话框中选择合适的文件名保存文件。然后按 F9 键运行程序，程序运行的初始画面如图 9-2 所示。

图 9-2　程序运行的初始画面

在程序运行的过程中，用户可以通过单击各个按钮来完成对剪贴板的查看和编辑操作，比如我们在 Windows 的“画笔”程序中，将一个打开的图像文件某一区域复制下来。如图 9-3 所示。

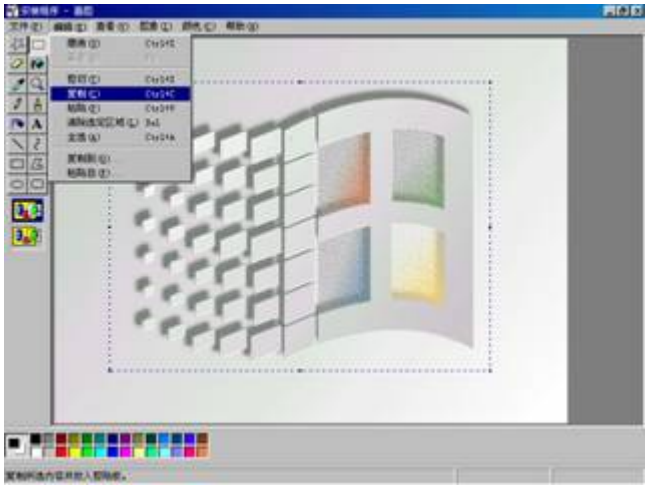


图 9-3　程序运行的初始画面

然后，回到上面设计的剪贴板程序中，单击“查看剪贴板”按钮，程序运行结果如图 9-4 所示，刚才被复制的一块区域显示在了我们自己的剪贴板查看区域中。

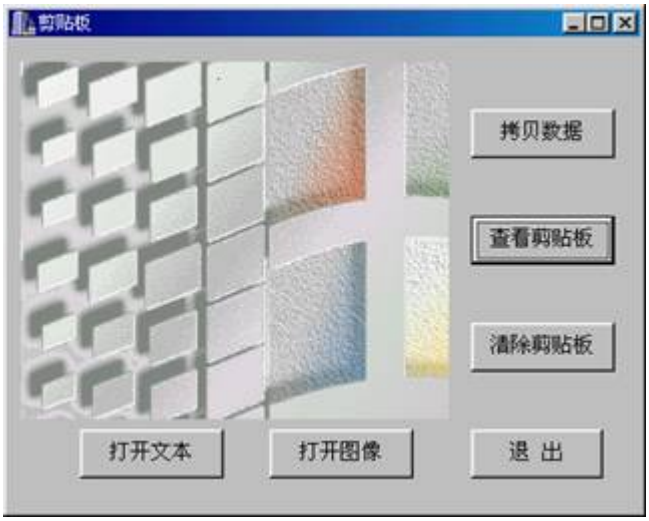


图 9-4　程序运算的结果

在这个示例应用程序中，不但可以观察当前剪贴板上的图像数据，同时还可以显示文本数据，这里我们就不再演示了，请读者自己制作并练习。

通过对这个程序的介绍，以下几点是应该注意的：

（1）提供了一种自己截获和处理剪贴板上内容的方法。读者可以根据需要进一步扩充；

（2）提供了响应 Windows 消息的方法。

（3）最后的一点启示是：在 CBuilder 5 程序开发中巧妙应用传统的 Windows 方法(如消息处理、API 函数等)仍是很有必要的。而在应用这些方法中所体现的方便之处，正是 CBuilder 5 胜过其他可视化开发工具的一个重要方面。

程序完整的源代码如下所示，供读者参考学习：

程序清单

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Clipbrd.hpp"
#include "Clip.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Clipboard()->HasFormat(CF_TEXT))
     //如果剪贴板上的数据为文本格式
     {
     Form1->RichEdit1->Visible=true;
     Form1->BitBtn1->Visible=false;
     //设置控件可见状态
     RichEdit1->Text = Clipboard()->AsText;
     //显示剪贴板上的文本
     }
  else if (Clipboard()->HasFormat(CF_BITMAP))
```

```
    //如果剪贴板上的数据为图像格式
    {
    Form1->BitBtn1->Visible=true;
    Form1->RichEdit1->Visible=false;
    //设置控件可见状态
    Form1->BitBtn1->Glyph->Assign(Clipboard());
    //显示剪贴板上的图像
    }
    else
    {
    Form1->BitBtn1->Visible=false;
    Form1->RichEdit1->Visible=false;
    //设置控件可见状态
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
 if (Form1->RichEdit1->Visible)
       //如果文本框控件可见
       Clipboard()->AsText=RichEdit1->Text;
       //把当前显示文本复制到剪贴板上
 if (Form1->BitBtn1->Visible)
       //如果 BitBtn1 控件可见
       Clipboard()->Assign(Form1->BitBtn1->Glyph);
       //把当前显示图像复制到剪贴板上
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
 if (Form1->OpenDialog1->Execute())
      //显示一个对话框
      {
      Form1->RichEdit1->Visible=true;
      Form1->BitBtn1->Visible=false;
      //设置控件可见状态
      Form1->RichEdit1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
      //显示指定文本文件的内容
      }
}
//---------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
 if (Form1->OpenPictureDialog1->Execute())
      //显示一个对话框
      {
      Form1->BitBtn1->Visible=true;
      Form1->RichEdit1->Visible=false;
      //设置控件可见状态
      Form1->BitBtn1->Glyph->LoadFromFile(Form1->OpenPictureDialog1->FileName);
      //显示指定图像文件的内容
      }
}
//---------------------------------------------------------------------
```

```
void __fastcall TForm1::Button5Click(TObject *Sender)
{
  Clipboard()->Clear();
  //清除剪贴板上的数据
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Form1->Button1->Enabled=true;
  Form1->Button2->Enabled=true;
  Form1->Button3->Enabled=true;
  Form1->Button4->Enabled=true;
  Form1->Button5->Enabled=true;
  Form1->Button6->Enabled=true;
  //设置按钮有效状态
  Form1->RichEdit1->Text="";
  //清空文本框中的内容
  Form1->OpenDialog1->Title="请选择一个文本文件：";
  //设置对话框标题
  Form1->OpenDialog1->InitialDir="c:\\pwin98\\";
  //设置缺省路径
  Form1->OpenDialog1->Filter="文本文件(*.txt)|*.txt";
  //设置文件过滤条件
  Form1->OpenPictureDialog1->Title="请选择一个图像文件：";
  //设置对话框标题
  Form1->OpenPictureDialog1->InitialDir="c:\\pwin98\\";
  //设置缺省路径
  Form1->OpenPictureDialog1->Filter="文本文件(*.bmp)|*.bmp";
  //设置文件过滤条件
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
Application->Terminate();
}
//---------------------------------------------------------------------------
```

9.2 对象的链接与嵌入

对象链接和嵌入（Object Linking and Embeding）是一组服务功能，它提供了一种用源于不同应用程序的信息创建复合文档的强有力方法。 对象可以是几乎所有的信息类型，如文字、位图、矢量图形，甚至于声音注解和录像剪辑等。

CBuilder 5 完美的支持 OLE 技术，使用 CBuilder 5，用户可以创建 OLE 应用程序，还可创

建 OLE 自动化服务器和控制器程序。因为 OLE 是一种技术，所以本节先介绍 OLE 的原理，然后通过例子介绍对象链接与嵌入的基本概念，CBuilder 5 创建 OLE 对象的方法，OLE 自动化的概念以及如何开发 OLE 自动化服务器和控制器。

9.2.1 OLE 简介

OLE 是 Windows 环境下应用程序之间信息传递和共享的一种技术，和 DDE 有类似的作用，但不同的是，它允许应用程序使用其他应用程序的数据创建复合文件。

使用 OLE 技术，一个 Windows 应用程序可以启动其他的 Windows 应用程序，也可以显示和控制其他的应用程序中的数据，并在创建该数据的时中对它进行编辑，这里所说的数据可以是一幅图像、一个电子表格、一段文本或者视频剪辑等，他们通常称为 OLE 对象。

在 CBuilder 5 中，用于交换信息的应用程序分为两种：一种称之为服务程序，由叫做源程序，它是数据的提供者；另一种是数据的接收者，称之为客户程序，又叫做目的程序。

OLE 实际上包括两种技术：嵌入和链接，因此 OLE 对象也分为两种，分别是嵌入对象和链接对象，分别介绍如下。

（1）嵌入

嵌入指的是在应用程序的 OLE 控件中插入的对象是真正的数据。嵌入对象保存在 OLE 客户应用程序中，其他应用程序不能访问该对象。只有在 OLE 应用程序中激活 OLE 对象才能对其进行编辑。嵌入的 OLE 对象不需要保存在文件中，所有数据都在应用程序中，这就确保了 OLE 数据不会被偶然地删除或修改。不足之处是应用程序的规模因为保存了 OLE 数据而增大了。如果用户想保存对嵌入对象的修改，可以把 OLE 数据存入文件中。

（2）链接

链接指的是在应用程序中的 OLE 控件中插入的对象是数据的地址，而不是真正的原始数据。链接对象的数据保存在 OLE 服务器程序创建的文件中，而嵌入对象的数据保存在 OLE 客户应用程序中。虽然用户似乎在客户程序中看到链接的数据，但是实际上它仅仅只是存放指向服务器程序的指针，其实用户看到的仅仅是原始数据的映象。

链接对象必须以文件形式保存，只有对 OLE 服务器已经创建好的 OLE 对象，才能进行 OLE 链接，链接的 OLE 对象文件可被 OLE 应用程序或其他程序进行修改，OLE 服务器和其他 OLE 应用程序也可以访问和修改 OLE 对象。对象数据保存在某一处，但可以被多个应用程序访问。

CBuilder 5 应用程序可以得到 OLE 链接对象文件中的最新数据。当 OLE 对象数据被应用程序修改时，这些变化将在所有包含该对象的其他应用程序中体现。

最后说明几点：

（1）迄今为止，已经出现过两种版本的 OLE：OLE 1.0 和 OLE 2.0。当用户在 OLE 1.0 服务器中激活 OLE 对象，服务器程序在前台打开自己的窗体，并获得焦点。OLE 窗体失去焦点，存在于单独的窗体之中。

（2）OLE 2.0 是 OLE 1.0 的发展，当 OLE 2.0 服务器采用“本地”（in place）激活方式。本地激活意味着服务器菜单与应用程序菜单要进行融合，服务器的状态条更换应用程序状态条，服务器的工具条更换应用程序工具条。OLE 对象在应用程序窗体中进行编辑，但所有过程均由服务器处理。

（3）创建 OLE 对象的服务器决定了 OLE 的激活方式。如果一个 OLE 1.0 的对象在 OLE 2.0 编译的应用程序中打开，它将采用 OLE 1.0 的方式。

表 9-4　使用链接或嵌入的原则。

何时使用链接	何时使用嵌入
想要对源对象进行修改及将这些修改反映到其他与源对象链接的应用程序或文本中	对源对象进行修改，并将这些修改反映在一个特定的应用程序或文本中
源对象可能被多个 OLE 应用程序应用程序频繁修改	源对象不可能被一个 OLE 应用程序频繁修改

源对象的文件不会被频繁移动，且不会被删除	源对象的文件可能被频繁移动，且不会被删除
对象很大，一般通过网络或电子邮件进行分配	对象很小，或对象很大却无法通过网络或电子邮件进行分配

Windows 附件组中的“写字板”是应用 OLE 的实例，使用单击“对象 | 插入”菜单项， 写字板弹出插入对话框，对话框中列出了多个 OLE 服务器程序，如公式编辑工具，绘图工具，报表生成工具，如图 9-5 所示。
用户双击鼠标左键，可激活一个 OLE 服务器。在 OLE 服务器中可编辑 OLE 对象，当用户返回到书写器中时，在书写器文档中将出现 OLE 对象。

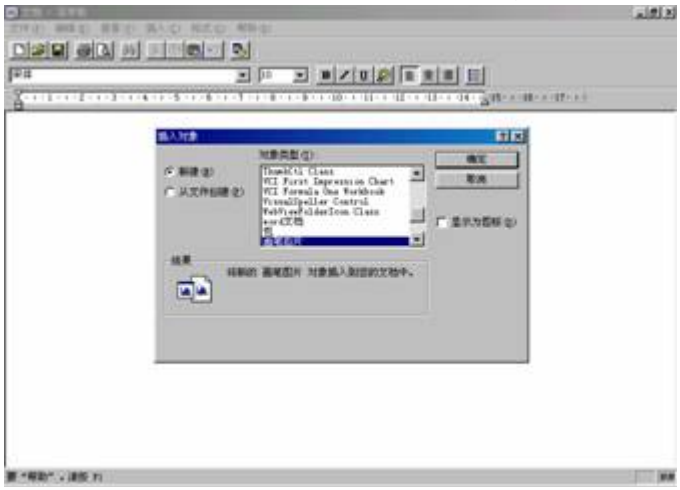


图 9-5　使用 OLE 的写字板

9.2.2　OLE 控件

应用程序部件包含链接或嵌入的对象，要创建 OLE 对象，需在窗体中加入 OLE 包容器部件。CBuilder 5 提供了一个 OLEContainer 控件，用于连接和嵌入 OLE 对象。它位于组件板上的 System 页标签上，如图 9-6 所示。

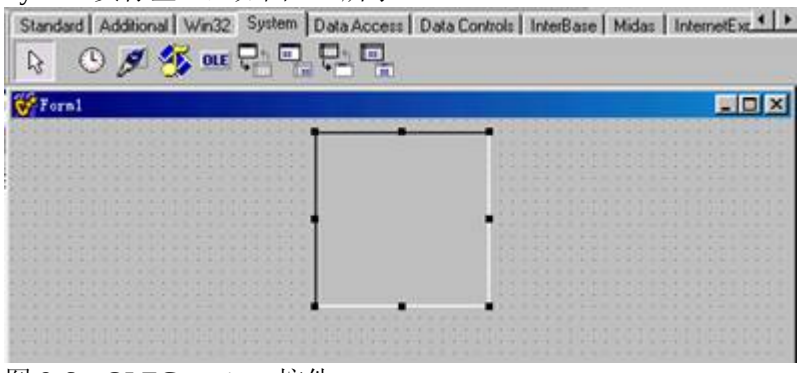


图 9-6　OLEContainer 控件

用 OLEContainer 控件可显示在 OLE 服务器编辑的数据。部件的 ObjClass、ObjDoc、ObjItem 属性分别定义 OLE 类、文件、项目。要定义 OLE 对象是否本地激活，使用 InPlaceActive 属性。如果 OLE 对象可以本地激活，OLE 服务器菜单将与 OLE 应用程序的菜单进行融合，GroupIndex 属性的值将决定菜单融合情况。
要使用 OLEContainer 控件，首先需要熟练的了解它的属性和使用方法，现具体介绍如下：
（1）AutoActivate 属性
该属性决定了 OLE 容器内的对象的激活方式，它有下面 3 种可以选择的值：

- aaManual

OLE 对象必须在程序中通过调用 DoVerd 方法来激活。

- aaGetFocus

当 OLE 容器获得焦点的时候激活 OLE 对象。

- aaDoubelClick

双击 OLE 容器或者当焦点在其上的时候按 Enter 键激活 OLE 对象，这是缺省值。

（2）CanPaste

该属性指示 Windows 剪贴板上是否可以粘贴到 OLE 容器上的 OLE 对象，为 True 时表示有，为 False 时表示没有。

（3）Linked

该属性指示 OLE 对象是链接的还是嵌入的，如果它的值是 True，表示是链接的，反之是嵌入的。

注意：

✍ 在访问这个属性前，OLE 对象必须已经装载到了 OLE 容器中。

（4）Modified

该属性指示 OLE 对象是否已经被修改过，包括被删除或者被另一个 OLE 对象替换。如果容器中没有 OLE 对象，或者对象没有被修改，则该属性的值为 True。

（5）SizeMode

该属性确定 OLE 容器内的 OLE 对象的大小的改变方式，也就是如何在容器内显示 OLE 对象。可以选择的值如下：

- smClip

以正常的大小显示 OLE 对象，超出范围的部分将被剪切，这是缺省值。

- smCenter

以正常的大小显示 OLE 对象，并使它在容器内居中。

- smScale

按照比例拉伸或者收缩显示 OLE 对象，使它适应容器。

- smStretch

拉伸或者收缩显示 OLE 对象，使它适应容器，但是不考虑比例。

- smAutoSize

以正常的大小显示 OLE 对象，并自动的调整容器的大小，使得容器适应对象的大小。

（6）State

该属性指示 OLE 对象的状态，可能的值包括如下：

- osEmpty

容器没有 OLE 对象

- osLoaded

容器中有一个 OLE 对象，但是它的服务器应用程序当前没有激活

- osRuning

OLE 对象的服务器应用程序当前正在激活

- osOpen

OLE 对象在一个独立的窗口中打开

- osInplaceActive

OLE 对象已经被激活，但是还没有合并其菜单系统和工具栏。

注意：

✍ 这是一种中间状态，只有菜单系统和工具栏被合并，才会变为 osulActive。

- osulActive

OLE 对象已经被激活，并且菜单系统和工具栏也已经被合并。

9.2.3 OLE 应用程序的开发

下面，通过具体的示例向用户介绍 OLE 的使用。
在应用程序设计的时候，可以向 OLEContainer 控件中插入 OLE 对象，具体的示例和操作步骤如下：

1. 开始工作

选择 File | New Application 新建一个项目，为了从不同的侧面说明 OLEContainer 控件的作用，我们在窗体上放置 4 个标签和 4 个 OLEContainer 控件，如图 9-7 所示。

图 9-7　为窗体添加 OLE 控件
调整窗体和控件窗体和控件的属性设置如下：

```
object Form1: TForm1
   Left = 190
   Top = 111
   Width = 354
   Height = 362
   Caption = 'OLE 应用'
   Color = clBtnFace
   Font.Charset = DEFAULT_CHARSET
   Font.Color = clWindowText
   Font.Height = -11
   Font.Name = 'MS Sans Serif'
   Font.Style = []
   OldCreateOrder = False
   PixelsPerInch = 96
   TextHeight = 13
   object Label1: TLabel
      Left = 24
      Top = 8
      Width = 73
```

```
    Height = 20
    Caption = 'OLE 控件 1'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object Label2: TLabel
    Left = 216
    Top = 8
    Width = 73
    Height = 20
    Caption = 'OLE 控件 2'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object Label3: TLabel
    Left = 24
    Top = 176
    Width = 73
    Height = 20
    Caption = 'OLE 控件 3'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object Label4: TLabel
    Left = 224
    Top = 176
    Width = 73
    Height = 20
    Caption = 'OLE 控件 4'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object OleContainer1: TOleContainer
    Left = 16
    Top = 32
    Width = 121
    Height = 121
    Caption = 'OleContainer1'
    TabOrder = 0
  end
  object OleContainer2: TOleContainer
    Left = 200
    Top = 32
    Width = 121
```

```
      Height = 121
      Caption = 'OleContainer2'
      TabOrder = 1
    end
    object OleContainer3: TOleContainer
      Left = 16
      Top = 200
      Width = 121
      Height = 121
      Caption = 'OleContainer3'
      TabOrder = 2
    end
    object OleContainer4: TOleContainer
      Left = 200
      Top = 200
      Width = 121
      Height = 121
      Caption = 'OleContainer4'
      TabOrder = 3
    end
end
```

2. OLE 创建

右击 OLEContainer1 控件，在弹出菜单上选择“Insert Object”命令，出现如图 9-8 所示的插入对象对话框。

图 9-8　插入对象对话框

选择新建复选框，并从对象类型中选择 BMP 图像，然后单击“确定”按钮，这时就会启动 Windows 画笔程序，如图 9-9 所示。

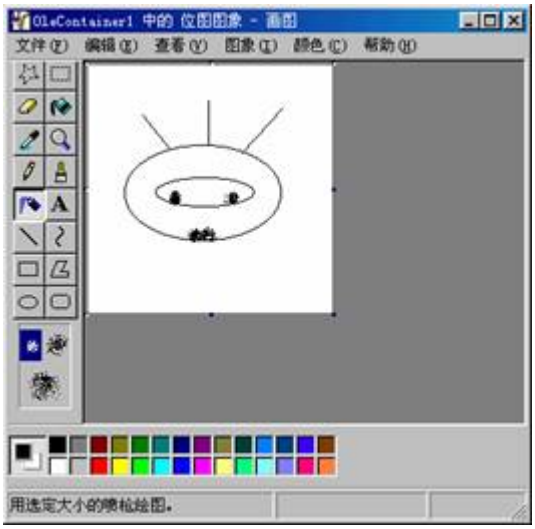


图 9-9　画笔程序

在画笔程序中绘制一个简单的图形，然后关闭该程序，用户就会看到所绘制的图形出现在

OLEContainer 控件中，如图 9-10 所示。

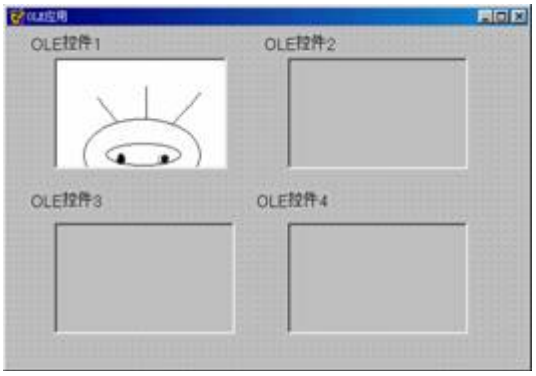


图 9-10　在 OLEContainer 控件中添加的图像

然后右击第二个 OLEContainer 控件，并从弹出菜单中选择 Insert Object，在出现的插入对象对话框中选择从文件创建复选框，然后选择一个可执行程序，比如 Windows 的记事本应用程序，如图 9-11 所示。请选中链接和显示为图标复选框。

图 9-11　选择应用程序作为 OLE 对象

然后单击“确定”按钮，这时 OLEContainer 控件中就回出现记事本应用程序的图标，如图 9-12 所示。

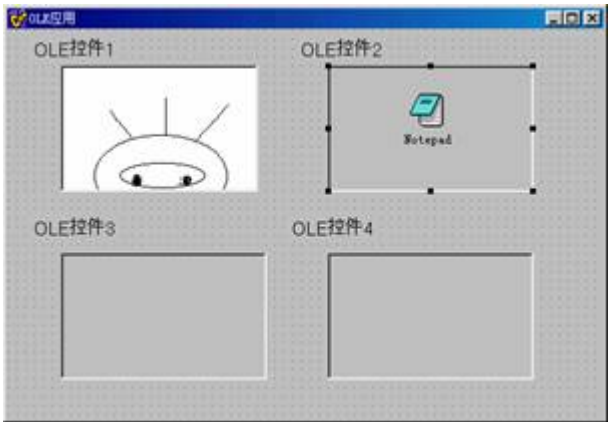


图 9-12　为 OLEContainer 控件添加应用程序 OLE 对象

右击 OLEContainer 控件，并从弹出菜单中选择 Insert object 命令，在出现的对话框中选择新建单选按钮，在对象类型列表中选择视频剪辑应用程序，并选择显示为图标复选框，如图 9-13 所示。

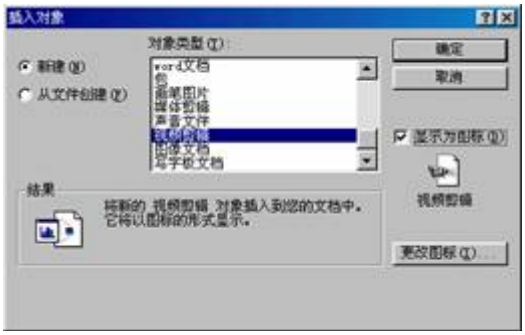


图 9-13　选择新建应用程序 OLE 对象

单击确定按钮后，用户就可以看到窗体上的 OLE 控件 OLEContainer3 中出现视频剪辑图标，

并且视频剪辑播放程序也会激活，如图 9-14 所示。

图 9-14　添加视频剪辑程序作为 OLE 对象

最后，我们选择添加一个现有的文件(HTML)作为 OLE 控件中的 OLE 对象，在添加对象对话框中作出如图 9-15 所示的选择。

图 9-15　添加 HTML 文件作为 OLE 对象

最后，添加完 OLE 对象的窗体如图 9-16 所示。

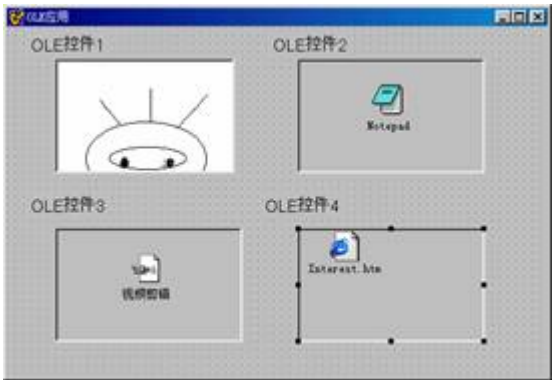


图 9-16　设置完毕的窗体

3. 运行程序

保存本次创建的 CBuilder 5 项目和文件，然后编译并运行之。如图 9-17 所示为程序运行的初始界面，为了激活某个控件中的 OLE 对象，用户只要右击该 OLEContainer 控件，然后在弹出菜单中选择激活内容命令即可。

图 9-17　准备激活 OLE 对象

现在，我们就可以从自己创建的 CBuilder 5 程序中直接激活 IE 浏览器来查看所添加的 HTML 文档了，如图 9-18 所示。

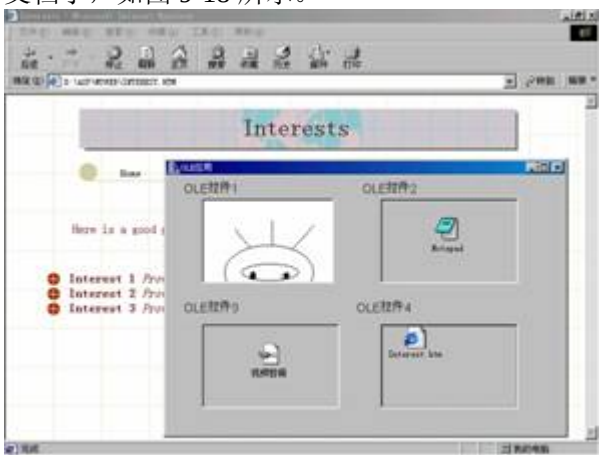


图 9-18　OLE 的使用示例

注意：

- 1.链接或者嵌入的 OLE 对象的类型不同，弹出菜单的内容也有所不同。
- 2.上面的应用程序从不同的侧面介绍了向 OLEContainer 控件中添加各种不同的 OLE 对象，用户可以自己去练习使用其他的类型的 OLE 对象。
- 3.为了插入并使用某种类型的 OLE 对象，用户的计算机上必须装载该类型 OLE 对象的服务器应用程序。

9.3 进程概述

在 Windows 应用程序中，当用户执行一些比较费时的任务时，如果系统一直处于等待状态，将会是一件非常头痛的事情。但是，通过多进程技术就可以避免这种问题的发生，当一个耗时很长的任务在执行时，用户可以去执行其他的应用，甚至在一个程序中同时运行所有并行的处理器。

本节，我们主要介绍的是有关进程的概念、原理和简单使用，并且和下一节联合起来，通过两个典型的实例来讲述如何在 CBuilder 5 中实现多进程应用。

9.3.1 进程的概念

进程是一个应用程序中的一条基本的执行路径，也就是 Windows 进程中的最小单元。每一个进程都可以访问进程中的所有资源。一个进程是由一个或者多个进程、代码、数据和应用程序在内存中的其他资源组成。

典型的应用程序资源是打开的文件和动态分配的内存空间。应用程序在系统调度时将控制权交给它的一个进程，系统调度决定了哪些进程运行以及什么时候运行。在多处理器的计算机上，系统调度可以将单独的进程分配到不同的处理器中来平衡 CPU 负载。

进程中的每一个进程都是独立的，除非用户似的进程之间可以互相可见，否则进程将独立于进程中的其他进程。

多进程的优点就是可以使得应用程序中的许多工作可以同时运行，例如在一个长时间的计算过程中，可以允许用户与应用程序交互或者终止计算过程。为此，可以建立一个与用户交互的进程，并为它赋予较高的优先级，而用其他优先级较低的进程去执行一些后台的工作。

注意：

> 与用户交互的进程优先级高可以使得应用程序及时的响应用户的输入。

多进程虽然在控制上比较复杂，但是它有下面明显的优点：

- 避免瓶颈

使用单进程，应用程序在等待一个缓慢操作完成的过程中，必须停止其他所有的工作，这样导致 CPU 直到操作完成才真正进入空闲状态。而使用多进程，应用程序可以在一个现成进入等待的过程中，转而去执行其他的进程，避免了瓶颈。

- 并行操作

应用程序的行为一般可以分为一些可以独立的、并行的操作，使用多进程可以让这些操作并行执行。此外，还可为不同的任务指定不同的优先级，以便使 CPU 有更多的时间去执行更为紧迫的任务。

- 提高效率

如果运行应用程序的系统有多个处理器，可以将应用程序的任务分配到不同的处理器中，并让它们在不同的处理器上同步运行以提高系统的性能。

9.3.2 进程应用方法

CBuilder 5 的 VCL（Visual Component Library）和 RTL（Run-Time Library）库把几乎全部的 Windows API 函数封装起来，而且在其基础上增加了一些安全措施。在实际运用时，程序员几乎没有必要直接调用 Windows API 函数，利用各单元所提供的函数库或例程库就可直接对系统底层进行操作。

那么如何在 CBuilder 5 中利用进程呢？其基本思想就是直接调用 WindowsAPI 的 CreateThread 函数来创建一个进程。那么，利用 CBuilder 5 自身所定义的丰富的标准例程库和数量更多、内容更广泛的（非）可视类库进行处理，以取代直接调用 Windows API 函数可否实现呢？回答是肯定的，而且更有效、更保险：

1. 使用进程函数

利用 RTL 库的 System 单元中定义的一个标准例程 BeginThread。此例程完整封装了 Win32 的 CreateThread 函数，是一个带有异常处理的标准 Pascal 函数，几乎可以处理所有自身的异常，相对于使用 Win32 的 CreateThread 函数，其安全系数大大增强。

BeginThread 函数在创建时，不是如 CreateThread 函数仅仅完成两项任务：创建一个进程；创建一个能作为进程入口的函数。还增加了几项保护措施：

把 System 单元中声明的全局变量 IsMultiThread 设为 TRUE， 这样 CBuilder 5 的堆栈管理器就知道当前有多个进程在运行，从而防止多个进程同时修改它的内部结构；

另外，在调用 BeginThread 函数时，可创建一个异常框架，允许系统缺省的异常处理句柄捕获任何未有被处理的异常进程。如果在进程函数中有任何未被处理的异常，会自动产生一个退出代码，或者进程返回的句柄为 0，表示进程没有创建成功，则应用程序将会调用 EndThread 过程（Procedure EndThread(ExitCiode:Integer)），自动终止进程的运行。

其完整声明如下：

```
function   BeginThread( SecurityAttributes: Pointer;
                        StackSize: Integer;
                        ThreadFunc: TThreadFunc;
                        Parameter: Pointer;
                        CreationFlags: Integer;
                        var ThreadId: Integer): Integer;
```

各参数的使用特点类似 CreateThread 函数：

- SecurityAttributes 参数是一个指向 SECURITY_ATTRIBUTES 结构的指针，其目的用于设置进程的访问权限，nil 表示为默认的安全属性；
- StackSize 参数用于设置分配给进程的栈空间大小 0 表示用默认值；
- ThreadFunc 用于指定一个函数，该函数在进程创建后开始执行代码时调用；
- Parameter 参数传递给 ThreadFunc 参数所指定的函数，常为 nil，或者设为一个 32 位的指针，指向一个数据结构；
- CreationFlags 参数用于指定进程创建后是不是立即执行，0 表示立即执行，CREATE_SUSPENDED 表示处于挂起状态；
- ThreadId 参数表示为每个进程唯一的识别号，当 BeginThread 函数返回后，此参数就是进程的识别号。

返回值为该进程的句柄，如果为 0，表示表示进程没有创建成功，可以调用 Windows 的 GetLastError 函数分析错误的原因。

2. 使用进程对象

利用 CBuilder 5 的 VCL 库中 TThread 对象。
CBuilder 5 的一个缺陷是不支持多个进程同时访问它的 VCL 库， 但 CBuilder 5 的设计者们并没有刻意掩饰这个缺陷，而是专门创建了一个 TThread 对象以解决这个问题。
这个 TThread 对象封装了 Windows API 和 System 单元中有关进程运用的多个函数和例程，利用操作系统分时段给各进程方式控制各个进程的“休眠”与“唤醒”以达到进程工作的同步，当被“唤醒”后就调用 TThread 对象的 Synchronize 过程通知主进程，让主进程去真正地访问 VCL，使得在一个应用程序中同时访问多个 VCL 库成为了可能。
当然，对于进行一般的多进程编程，就更加简单了。
在使用上它与 CBuilder 5 中大多数对象不同的是 TThread 类是一个带有虚拟抽象对象方法的类，我们不能直接创建 TThread 对象的实例。
必须先声明一个以 TThread 对象为基类的类，再用这个派生类创建实例和操纵进程具体的类属性和方法。

9.4 多进程技术应用

上面的示例中，我们讲解了进程的概念和应用。下面，我们再列举一个多进程的应用示例。本程序包含 3 个进程，分别用数学算法中的起泡排序、选择排序和快速排序对随即随机线条进行排序。

1. 设计窗体

在 Dlephi 5 中创建一个 Form 窗体，Caption 设为“排序算法进程示例”。
在窗体中放入 3 个 Label 控件，分别设置它们的 Caption 为“冒泡排序”、“选择排序”、“快速排序”。
然后再添加 3 个控件 PaintBox（在位于 System 标签页），Name 分别为“Bubblesortbox”、“Selectionsortbox”和“Quicksortbox”。
最后，再添加 1 个按钮控件，Caption 设置为“开始排序”，Name 属性为 Startbtn。
程序窗体的设计界面如图 9-19 所示。

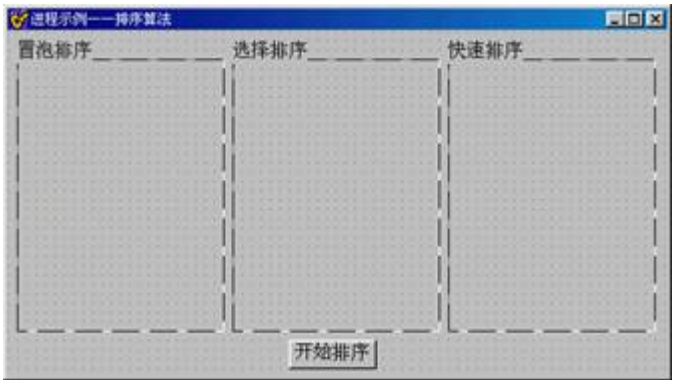


图 9-19 排序程序示例的窗体界面
有关窗体和其上的控件属性设置可以参考如下：

```
object ThreadSortForm: TThreadSortForm
  Left = 130
  Top = 95
```

```
  BorderStyle = bsDialog
  Caption = '进程示例--排序算法'
  ClientHeight = 295
  ClientWidth = 562
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = True
  Position = poScreenCenter
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object Bevel1: TBevel
    Left = 8
    Top = 24
    Width = 177
    Height = 233
  end
  object Bevel3: TBevel
    Left = 376
    Top = 24
    Width = 177
    Height = 233
  end
  object Bevel2: TBevel
    Left = 192
    Top = 24
    Width = 177
    Height = 233
  end
  object BubbleSortBox: TPaintBox
    Left = 8
    Top = 24
    Width = 177
    Height = 233
    OnPaint = BubbleSortBoxPaint
  end
  object SelectionSortBox: TPaintBox
    Left = 192
    Top = 24
    Width = 177
    Height = 233
    OnPaint = SelectionSortBoxPaint
  end
  object QuickSortBox: TPaintBox
    Left = 376
    Top = 24
    Width = 177
    Height = 233
    OnPaint = QuickSortBoxPaint
  end
  object Label1: TLabel
    Left = 8
    Top = 8
    Width = 44
    Height = 13
    Caption = '冒泡排序'
```

```
  end
  object Label2: TLabel
    Left = 192
    Top = 8
    Width = 44
    Height = 13
    Caption = '选择排序'
  end
  object Label3: TLabel
    Left = 376
    Top = 8
    Width = 44
    Height = 9
    Caption = '快速排序'
  end
  object StartBtn: TButton
    Left = 480
    Top = 264
    Width = 75
    Height = 25
    Caption = '开始排序'
    TabOrder = 0
    OnClick = StartBtnClick
  end

end
```

2. 程序设计

本示例应用程序项目我们设计了两个单元文件，分别是 Thsort.cpp 和 SortThd.cpp。
单元文件 SortThd.cpp 主要作用是定义程序中需要使用的三个进程类：TbubbleSort、TselectionSort 和 TquickSort，它们都是派生类 TsortThread 类的派生类，而 TsortThread 类是由 CBuilder 5 提供的进程基类 Tthread 派生而来。
由于篇幅所限，我们不再详细讲解程序的设计思路，下面给出该单元文件的详细代码，请读者下去研究。

程序清单（SortThd 单元文件代码）

```
#include <vcl.h>
#pragma hdrstop

#include "sortthd.h"
//---------------------------------------------------------------------------
void __fastcall PaintLine(TCanvas *Canvas, int I, int Len)
{
  TPoint points[2];

  points[0] = Point(0, I*2+1);
  points[1] = Point(Len, I*2+1);

  Canvas->Polyline(EXISTINGARRAY(points));
}
//---------------------------------------------------------------------------
__fastcall TSortThread::TSortThread(TPaintBox *Box, int *SortArray,
      const int SortArray_Size)
      : TThread(False)
{
  FBox = Box;
```

```
   FSortArray = SortArray;
   FSize = SortArray_Size + 1;
   FreeOnTerminate = True;
}
//---------------------------------------------------------------------

void __fastcall TSortThread::DoVisualSwap()
{
   TCanvas *canvas;
   canvas = FBox->Canvas;
   canvas->Pen->Color = TColor(clBtnFace);
   PaintLine(canvas, FI, FA);
   PaintLine(canvas, FJ, FB);
   canvas->Pen->Color = clRed;
   PaintLine(canvas, FI, FB);
   PaintLine(canvas, FJ, FA);
}
//---------------------------------------------------------------------

void __fastcall TSortThread::VisualSwap(int A, int B, int I, int J)
{
   FA = A;
   FB = B;
   FI = I;
   FJ = J;

   Synchronize(DoVisualSwap);
}
//---------------------------------------------------------------------
/* The Execute method is called when the thread starts */

void __fastcall TSortThread::Execute()
{
   Sort(FSortArray, FSize-1);
}
//---------------------------------------------------------------------

__fastcall TBubbleSort::TBubbleSort(TPaintBox *Box, int *SortArray,
   const int SortArray_Size)
   : TSortThread(Box, SortArray, SortArray_Size)
{
}
//---------------------------------------------------------------------

void __fastcall TBubbleSort::Sort(int *A, int const AHigh)
{
   int I, J, T;

   for (I=AHigh; I >= 0; I--)
      for (J=0; J<=AHigh-1; J++)
         if (A[J] > A[J + 1])
         {
            VisualSwap(A[J], A[J + 1], J, J + 1);
            T = A[J];
            A[J] = A[J + 1];
            A[J + 1] = T;
            if (Terminated)
               return;
         }
}
```

```
//---------------------------------------------------------------------

__fastcall TSelectionSort::TSelectionSort(TPaintBox *Box, int *SortArray,
   const int SortArray_Size)
   : TSortThread(Box, SortArray, SortArray_Size)
{
}
//---------------------------------------------------------------------

void __fastcall TSelectionSort::Sort(int *A, int const AHigh)
{
   int I, J, T;

   for (I=0; I <= AHigh-1; I++)
      for (J=AHigh; J >= I+1; J--)
         if (A[I] > A[J])
         {
            VisualSwap(A[I], A[J], I, J);
            T = A[I];
            A[I] = A[J];
            A[J] = T;
            if (Terminated)
               return;
         }
}
//---------------------------------------------------------------------

__fastcall TQuickSort::TQuickSort(TPaintBox *Box, int *SortArray,
   const int SortArray_Size)
   : TSortThread(Box, SortArray, SortArray_Size)
{
}
//---------------------------------------------------------------------

void __fastcall TQuickSort::QuickSort(int *A, int const AHigh, int iLo, int iHi)
{
   int Lo, Hi, Mid, T;

   Lo = iLo;
   Hi = iHi;
   Mid = A[(Lo+Hi)/2];

   do
   {
      if (Terminated)
         return;
      while (A[Lo] < Mid)
            Lo++;
      while (A[Hi] > Mid)
            Hi--;
      if (Lo <= Hi)
      {
         VisualSwap(A[Lo], A[Hi], Lo, Hi);
         T = A[Lo];
         A[Lo] = A[Hi];
         A[Hi] = T;
         Lo++;
         Hi--;
      }
   }
```

```
  while (Lo <= Hi);

  if (Hi > iLo)
    QuickSort(A, AHigh, iLo, Hi);
  if (Lo < iHi)
    QuickSort(A, AHigh, Lo, iHi);
}
//---------------------------------------------------------------------

void __fastcall TQuickSort::Sort(int *A, int const AHigh)
{
  QuickSort(A, AHigh, 0, AHigh);
}
//---------------------------------------------------------------------
```

示例程序项目中另外一个单元文件是 Thsort.cpp，它是程序中所设计的窗体的单元文件，其中包含响应按钮事件的代码，以及用于启动三个进程的代码。完整的程序代码如下，也请读者自己学习。

程序清单（ThSort 单元文件代码）

```
#include <vcl.h>
#pragma hdrstop

#include <stdlib.h>
#include "thsort.h"
#include "sortthd.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TThreadSortForm *ThreadSortForm;
//---------------------------------------------------------------------
Boolean ArraysRandom;
TSortArray BubbleSortArray, SelectionSortArray, QuickSortArray;

//---------------------------------------------------------------------
__fastcall TThreadSortForm::TThreadSortForm(TComponent *Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TThreadSortForm::PaintArray(TPaintBox *Box, int const *A,
  int const ASize)
{
  int i;
  TCanvas *canvas;

  canvas = Box->Canvas;
  canvas->Pen->Color = clRed;

  for (i=0; i <= ASize; i++)
    PaintLine(canvas, i, A[i]);
}
//---------------------------------------------------------------------

void __fastcall TThreadSortForm::BubbleSortBoxPaint(TObject * /*Sender*/)
{
  PaintArray(BubbleSortBox, EXISTINGARRAY(BubbleSortArray));
}
//---------------------------------------------------------------------
void __fastcall TThreadSortForm::SelectionSortBoxPaint(TObject * /*Sender*/)
```

```
{
   PaintArray(SelectionSortBox, EXISTINGARRAY(SelectionSortArray));
}
//---------------------------------------------------------------------

void __fastcall TThreadSortForm::QuickSortBoxPaint(TObject * /*Sender*/)
{
   PaintArray(QuickSortBox, EXISTINGARRAY(QuickSortArray));
}
//---------------------------------------------------------------------

void __fastcall TThreadSortForm::FormCreate(TObject * /*Sender*/)
{
   RandomizeArrays();
}
//---------------------------------------------------------------------
void __fastcall TThreadSortForm::StartBtnClick(TObject * /*Sender*/)
{
   TBubbleSort *bubble;
   TSelectionSort *selsort;
   TQuickSort *qsort;

   RandomizeArrays();
   ThreadsRunning = 3;

   bubble = new TBubbleSort(BubbleSortBox, EXISTINGARRAY(BubbleSortArray));
   bubble->OnTerminate = ThreadDone;

   selsort = new TSelectionSort(SelectionSortBox,
      EXISTINGARRAY(SelectionSortArray));
   selsort->OnTerminate = ThreadDone;

   qsort = new TQuickSort(QuickSortBox, EXISTINGARRAY(QuickSortArray));
   qsort->OnTerminate = ThreadDone;

   StartBtn->Enabled = False;
}
//---------------------------------------------------------------------

void __fastcall TThreadSortForm::RandomizeArrays()
{
   int i;

   if (! ArraysRandom)
   {
      Randomize();
      for (i=0; i < ARRAYSIZE(BubbleSortArray); i++)
         BubbleSortArray[i] = random(170);

      memcpy(SelectionSortArray, BubbleSortArray, sizeof(SelectionSortArray));
      memcpy(QuickSortArray, BubbleSortArray, sizeof(QuickSortArray));

      ArraysRandom = True;
      Repaint();
   }
}
//---------------------------------------------------------------------
void __fastcall TThreadSortForm::ThreadDone(TObject * /*Sender*/)
{
   ThreadsRunning--;
```

```
    if (! ThreadsRunning)
    {
        StartBtn->Enabled = True;
        StartBtn->SetFocus();
        ArraysRandom = False;
    }
}
//---------------------------------------------------------------------------
```

3. 运行程序

做完以上的工作后，选择菜单“File”中的“Save All”选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序。
程序运行时的初始画面如图 9-20 所示，窗体上 3 个 PainBox 控件中充满相同的随机线条。

图 9-20　程序运行初始画面

单击按钮“开始排序”，显示如下的结果：程序开始启动 3 个进程（排序线条），从中可以看到三种排序方法速度的不同——快速排序最快，选择排序其次，冒泡排序最慢，排序过程如图 9-21 所示。

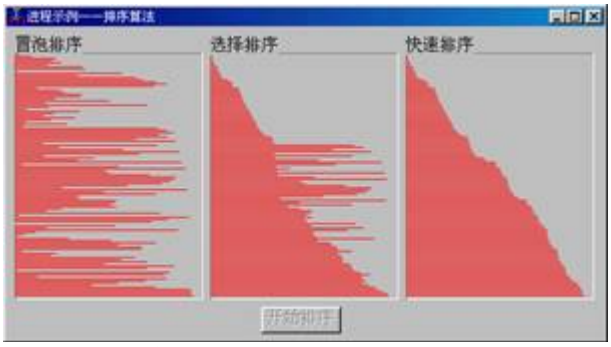


图 9-21　排序过程中

最后排序完毕，结果如图 9-22 所示。

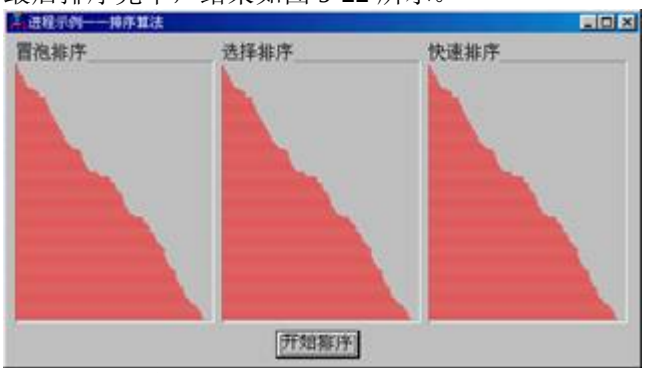


图 9-22　排序完毕

注意：

- 排序过程中，按钮“开始排序”无效；结束后有效，用户可以开始新一轮排序计算。

9.5 小　结

本章，我们讲解了剪贴板、OLE 和进程技术，它们是 Windows 下信息共享的三种主要的方式。CBuilder 5 以简便、友好的方式实现了相应的功能，为用户编程提供了方便。一般说来，剪贴板多用于静态数据传输，而 OLE 可以针对不同的数据类型进行交换和共享，进程则实现了应用程序的多任务和资源共用。按照全书的风格，我们还是通过一些比较实用的例子向用户介绍它们的用法。
本章所讲解的内容在 CBuilder 5 编程中属于比较高级的应用，如果使用合适，它可以大大的丰富程序的功能，并可以提高程序的开发效率。这些内容对于多用户环境下的 Windows 程序开发是很重要的，希望读者能够熟练掌握。

至此，本书的所有内容讲述完毕。古诗云：“宝剑锋从磨砺出，梅花香自苦寒来”，希望读者以后继续深入钻研 CBuilder 的编程技术，勤学苦练，坚持不懈，成为一名出色的 CBuilder 程序员。