万水计算机技术实用大全系列

C++Builder
精彩编程实例集锦

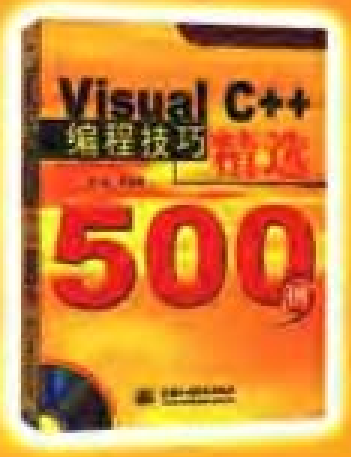
罗 斌 等编著

包含所有实例的源代码及可执行程序



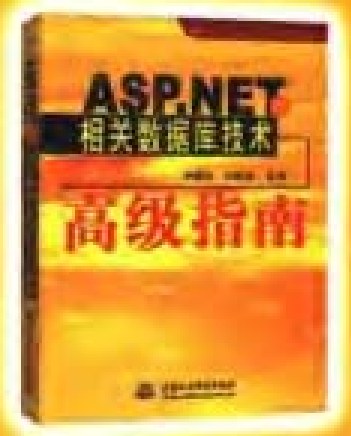
中国水利水电出版社
www.waterpub.com.cn

责任编辑：宋俊娥　　封面设计：俞兆君



- 精选500个Visual C++编程技巧
- 全方位直击Visual C++编程技术精要，涵盖MFC和NET精髓
- 以问答形式进行讲解
- 以编程实例形式加以明证

罗　斌　等编著
ISBN 7-5084-2585-5
定价：48.00元



- 网站建设一线的精英，最新的实践经验与知识积累
- 共同领略网站建设的玄机奥妙，轻松漫步互联网
- 精心收集的实用资料，网站实现过程分门别类的说明
- 初学者的借鉴技术手册，网站开发者的高级指南

申新旭　宋俊娥　主编
ISBN 7-5084-2616-5
定价：95.00元



- 精选250个经典实例，内容精彩
- 图形图像、网络通信、数据库系统开发等一网打尽
- 汇聚高手智慧，提升编程境界

罗　斌　等编著
ISBN 7-5084-2612-2
定价：48.00元

ISBN 7-5084-2812-9
9 787508 428123 >

北京万水电子信息有限公司
Beijing Multi-Channel Electronic Information Co.,Ltd.
地址：北京市海淀区长春桥路5号新起点国际4号楼1706室
电话：010-82562819(总机)
传真：010-82564371
邮编：100089
E-mail: mchannel@263.net

ISBN 7-5084-2812-9
定价：48.00元(含1CD)

万水计算机技术实用大全系列

C++Builder 精彩编程实例集锦

罗斌　等编著

中国水利水电出版社

内 容 提 要

本书通过 250 个编程实例比较全面地介绍了使用 C++Builder 6.0 开发应用程序的编程技巧、方法和经验，包括程序界面、图形图像、COM/OLL 应用、网络通信、系统设置、注册表操作和数据库使用等方面的内容。这些实例针对开发过程中的实际问题编写，范围较广、功能突出，具有较强的针对性和典型意义。使用本书的最好方法就是将书中提出的问题自己先做一遍，看看是否能够解决；如果不能解决，再查看配套光盘中的源代码；最后再看看书中所提示的关键内容。相信您做完这 250 个编程实例中提到的问题，您使用 C++Builder 解决实际问题的能力会大有长进。

本书特别适用于具有 C++Builder 编程基础知识，正在使用 C++Builder 开发管理项目的编程人员，也可作为 C++学习者提高编程水平的自学教材。

本书配套光盘中包括书中所有实例的源代码及可执行程序，可以参照学习及调用。

图书在版编目（CIP）数据

C++Builder 精彩编程实例集锦 / 罗斌等编著. —北京：中国水利水电出版社，2005

（万水计算机技术实用大全系列）

ISBN 7-5084-2812-9

Ⅰ. C… Ⅱ. 罗… Ⅲ. C 语言—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2005）第 028069 号

书　　名	C++Builder 精彩编程实例集锦
作　　者	罗斌　等编著
出版　发行	中国水利水电出版社（北京市三里河路 6 号　100044） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）63202266（总机）、68331835（营销中心）、82562819（万水）
经　　售	全国各地新华书店和相关出版物销售网点
排　　版	北京万水电子信息有限公司
印　　刷	北京市天竺颖华印刷厂
规　　格	787mm×1092mm　16 开本　27.75 印张　625 千字
版　　次	2005 年 5 月第 1 版　2005 年 5 月第 1 次印刷
印　　数	0001—5000 册
定　　价	48.00 元（含 1CD）

凡购买我社图书，如有缺页、倒页、脱页的，本社营销中心负责调换

版权所有 · 侵权必究

前　言

众所周知，Borland 公司是一家出色的语言平台开发商，其推出的 Turbo C 2.0、Borland C++ 3.1 等 C/C++语言开发工具曾经主导了一个时代，在广大程序员中享有崇高的声誉。然而，微软公司的 Visual C++开发平台的快速崛起，特别是 Visual C++ 4.2、Visual C++ 5.0、Visual C++ 6.0 的推出，使 Borland 公司在 C++开发工具领域节节败退，到了 1996 年，甚至终止了 Borland C++后续版本的开发，使广大的 Borland C++用户感到失望和茫然！

Delphi 在市场上的成功，再度为 Borland（Inprise）公司在语言开发平台领域树立了一个成功的丰碑。1997 年，Borland 公司借助 Delphi 的 VCL 组件推出了 C++Builder 3.0，重新为 Borland C++程序员打开了一片梦想的蓝天。此后，Borland 公司一鼓作气陆续推出了 C++Builder 的多个升级版本，使 C++Builder 迅速在快速开发工具领域拥有了一席之地。有人说，会使用 Delphi 开发工具就会使用 C++Builder，因为它们都是使用同一套 VCL 组件，VCL 组件能够造就 Delphi 的成功，当然也会推动 C++Builder 走向成功；也有人说，会使用 Visual C++就会使用 C++Builder，因为它们都是使用相同的 C/C++母语言，能够与 Windows 操作系统的 API 函数实现无缝对接，弥补了 Delphi 先天的不足。由此可见，C++Builder 所拥有的技术非同一般。

然而，有一位 IT 界的成功人士曾经这样说过：技术本身是毫无价值的，技术只有与社会的需求结合起来实现价值增值时，技术才具有价值。C++Builder 可以说是使 C++编程技术实现价值增值的开发平台典范。在使用 C++Builder 开发现代信息管理系统时，无须把 VCL 组件的来龙去脉了解得清清楚楚，也不必把 Windows 系统和程序的运行机制弄得明明白白（如果要了解这些，建议使用 Visual C++）；只要会熟练地使用这些 VCL 组件和方法搭建起自己的信息管理系统，实现自己的系统开发目标即可。本书就是本着这种指导思想，没有在原理性问题中纠缠，而是用简明扼要的语言和立竿见影的实例告诉读者怎样按图索骥使用这些 VCL 组件去实现自己的编程目标。

本书包括 250 个 C++Builder 精彩编程实例，一个实例就是一种方法，一个实例就能够帮助读者实实在在地去实现一个目标、解决一个问题。在这 250 个 C++Builder 编程实例中，可以找到程序界面、图形图像、COM/DLL 应用、网络通信、系统设置、注册表操作和数据库使用等方方面面的内容。特别是在数据库系统开发方面，可以发现许多在其他 C++开发工具中的老大难问题，使用 C++Builder 解决起来就会变成小菜一碟，真正体会到 VCL 组件的威力和 C++Builder 编程的精彩！“众里寻他千百度，蓦然回首，那人却在灯火阑珊处！”，如果读者能在本书中找到这种感觉，本书的创作目的也就达到了。

本书的所有内容和思想并非一人之力所能及，而是凝聚了众多 C++Builder 编程高手多年的心得并经过充分的提炼和总结而成，在此对他们的智慧表示崇高的敬意和衷心的感谢！参加本书编写的人员有罗斌、罗帅、雷治英、吴多和罗顺文，终稿由罗斌统筹完成。本书

所有实例程序均在 C++Builder 6.0 企业版/Windows XP/Windows 2000 企业版中编译成功，因此，请在以上环境中进行实例程序的学习（特别是数据库方面的实例程序），当然其中也可能存在一些疏漏之处，敬请读者批评指正。作者联系邮箱 binluobin@163.cm。

罗 斌

2005 年初于重庆长寿

目　　录

前言
第一部分　界面设计 1
实例 001　如何实现程序闪屏效果 1
实例 002　如何实现程序窗口闪烁 3
实例 003　如何制作吸附窗口程序 4
实例 004　如何制作透明程序窗口 5
实例 005　如何制作半透明程序窗口 7
实例 006　如何制作不规则程序窗口 8
实例 007　如何制作程序渐变背景 11
实例 008　如何设置程序背景图像 12
实例 009　如何设置 MDI 程序背景图像 13
实例 010　如何动态关闭 MDI 程序所有子窗口 17
实例 011　如何去掉 MDI 程序子窗口标题栏 18
实例 012　如何拖动非标题栏区域移动窗口 19
实例 013　如何控制程序最小化窗口 20
实例 014　如何限制程序最大最小化窗口 22
实例 015　如何控制程序窗口自动隐藏或显示 23
实例 016　如何隐藏应用程序 25
实例 017　如何控制程序在任务栏上是否出现 26
实例 018　如何使一个程序在任务栏同时出现多个标题 28
实例 019　如何使程序窗口总在最前面显示 29
实例 020　如何隐藏应用程序标题栏 31
实例 021　如何设置程序主标题 32
实例 022　如何设置程序图标 33
实例 023　如何在程序中改变菜单位置 34
实例 024　如何删除程序系统关闭菜单 35
实例 025　如何拷贝当前程序界面 36
实例 026　如何动态创建程序表单 37
实例 027　如何遍历窗体所有组件 39
实例 028　如何保证组件位置相对固定 40
实例 029　如何在窗体中动态增加组件 41

实例 030　如何使组件在程序运行时可以移动 42
实例 031　如何使用组件创建托盘程序 43
实例 032　如何使用动画组件播放动画 45
实例 033　如何在状态栏上加载进度条 46
实例 034　如何在编辑框内显示图像 48
实例 035　如何在列表框中增加水平滚动条 49
实例 036　如何在列表框中同时显示图像和文字 50
实例 037　如何设置属性表组件标签图像 51
实例 038　如何控制 Memo 组件的文字边界 52
实例 039　如何使用代码控制文字滚动 53
实例 040　如何纵向显示字符串 54
实例 041　如何旋转显示字符串 55
实例 042　如何设置显示文字分辨率 57
第二部分　程序设置 59
实例 043　如何使用消息框 59
实例 044　如何显示简单关于对话框 62
实例 045　如何在程序中增加热键 63
实例 046　如何在程序中自定义消息 64
实例 047　如何在程序中自定义系统菜单 67
实例 048　如何在执行程序间进行数据通信 70
实例 049　如何使用 OLE 技术启动画图 72
实例 050　如何在程序中添加 Word 文档表格 74
实例 051　如何在程序中设置 Word 文档的字体风格 76
实例 052　如何使用 OnIdle 事件 78
实例 053　如何使程序在循环时响应外界事件 79
实例 054　如何设置程序在系统启动时自动运行 81
实例 055　如何创建多线程程序 83
实例 056　如何创建可独立执行程序 86
实例 057　如何编写单文档应用程序 88
实例 058　如何编写多文档应用程序 89
实例 059　如何编写 COM 服务器程序 90
实例 060　如何编写 COM 客户端程序 94
实例 061　如何编写 DDE 服务器程序 95
实例 062　如何编写 DDE 客户端程序 96
实例 063　如何编写 DLL 服务器程序 98
实例 064　如何编写 DLL 客户端程序 101

实例 065　如何注册 DLL 服务器程序 103
实例 066　如何制作聊天服务器程序 105
实例 067　如何制作聊天客户端程序 107
实例 068　如何获取程序名称和路径 109
实例 069　如何获取程序命令行参数 110
实例 070　如何析取动态链接库中的图标 111
实例 071　如何设置 Edit 组件的文本选择范围 113
实例 072　如何将小写金额转换成大写金额 114
实例 073　如何将全角字符串转换成半角字符串 115
实例 074　如何按照指定的有效位数转换数字 116
实例 075　如何获取汉字的拼音索引 118
实例 076　如何设置输入方法 120
实例 077　如何控制数据输入格式 121
实例 078　如何控制编辑框中的输入法 123
实例 079　如何设置回车键后的焦点 124
实例 080　如何将项目保存到对象库 125
实例 081　如何检测鼠标信息 127
实例 082　如何读取鼠标信息 128
实例 083　如何实现鼠标拖动操作 130
实例 084　如何限制鼠标活动范围 131
实例 085　如何读取键盘信息 133
第三部分　文件操作 135
实例 086　如何复制文件 135
实例 087　如何定制文件复制操作 136
实例 088　如何更名文件 141
实例 089　如何更改文件扩展名 142
实例 090　如何删除文件 143
实例 091　如何删除文件夹 145
实例 092　如何选择文件夹 146
实例 093　如何创建多层文件夹 147
实例 094　如何自动获取文件夹和文件 149
实例 095　如何搜索指定目录下的文件 150
实例 096　如何搜索指定目录下的隐藏文件 151
实例 097　如何搜索文本文件中的字符串 153
实例 098　如何替换文本文件中的字符串 154
实例 099　如何将长文件名转换为短文件名 157

实例 100　如何在文件打开对话框中选择多个文件 158
实例 101　如何定制文件打开和关闭对话框 159
实例 102　如何修改文件打开对话框的按钮标题 160
实例 103　如何使用查找与替换对话框 161
实例 104　如何获取和设置当前目录 166
实例 105　如何判断文件类型 167
实例 106　如何获取系统所有文件类型 168
实例 107　如何获取和设置文件属性 171
实例 108　如何获取和设置文件时间属性 173
实例 109　如何对文件名函数进行操作 176
实例 110　如何读写文本文件 178
实例 111　如何打印文本文件 181
实例 112　如何创建特大型文件 183
实例 113　如何制作临时文件 184
实例 114　如何启动帮助文件 185
实例 115　如何使用组件在主机间传送文件 187
实例 116　如何对文件进行编码和解码 189
实例 117　如何设置 RichEdit 组件的存盘格式 191
实例 118　如何在 WIN.INI 文件中保存程序信息 192
实例 119　如何将数据转换成 Word 文档格式 194
实例 120　如何直接从 Excel 文件导入数据 195
实例 121　如何直接将数据导入到 Excel 文件 198
实例 122　如何发送电子邮件（一） 200
实例 123　如何发送电子邮件（二） 203
第四部分　图像处理 206
实例 124　如何浏览图像 206
实例 125　如何滚动浏览大图像 207
实例 126　如何拖动图像 208
实例 127　如何动态显示图像 209
实例 128　如何维护多个图像 215
实例 129　如何绘制图形 217
实例 130　如何以不规则方式裁剪图像 219
实例 131　如何以任意角度旋转显示图像 220
实例 132　如何在桌面上显示图像 222
实例 133　如何拷贝当前屏幕 224
实例 134　如何打印图像文件 225

实例 135　如何制作马赛克图像 227
实例 136　如何实现图像中心扩散效果 228
实例 137　如何实现图像百叶窗效果 229
实例 138　如何实现图像底片化效果 231
实例 139　如何实现图像翻页效果 232
实例 140　如何实现图像镜像效果 233
实例 141　如何柔化处理图像 234
实例 142　如何锐化处理图像 236
实例 143　如何雾化处理图像 237
实例 144　如何将彩色图像转换成黑白图像 238
实例 145　如何将 BMP 图像文件转换为 JPG 图像文件 240
实例 146　如何全屏播放 AVI 文件 241
实例 147　如何制作媒体播放器 243
第五部分　系统编程 245
实例 148　如何获取键盘类型 245
实例 149　如何获取声卡配置 246
实例 150　如何获取处理器信息 248
实例 151　如何获取系统元素颜色 249
实例 152　如何获取显示元素信息 251
实例 153　如何获取系统支持的显示模式 252
实例 154　如何动态设置系统的显示模式 253
实例 155　如何获取内存空间信息 256
实例 156　如何获取磁盘类型 257
实例 157　如何获取磁盘卷信息 259
实例 158　如何获取磁盘空间信息 260
实例 159　如何在我的电脑中隐藏磁盘 262
实例 160　如何隐藏系统任务栏 266
实例 161　如何隐藏桌面图标 267
实例 162　如何在桌面上创建快捷方式 268
实例 163　如何获取和设置桌面墙纸 270
实例 164　如何设置桌面墙纸显示方式 271
实例 165　如何进行系统桌面切换 274
实例 166　如何禁止系统屏幕保护程序 277
实例 167　如何在程序中启动系统屏幕保护程序 278
实例 168　如何在开始右键菜单中添加菜单项 279
实例 169　如何清空系统回收站 280

实例 170　如何删除文件到回收站 281
实例 171　如何启动控制面板程序 284
实例 172　如何在系统控制面板中添加项目 287
实例 173　如何为计算机增加启动日志 291
实例 174　如何获取操作系统的启动模式 292
实例 175　如何变暗屏幕颜色 294
实例 176　如何关闭计算机 295
实例 177　如何禁止关闭 Windows 操作系统 297
实例 178　如何修改 IE 浏览器的背景 299
实例 179　如何在程序中控制 IE 窗口 300
实例 180　如何使用默认浏览器打开指定网页 301
实例 181　如何获取当前正在运行的程序 303
实例 182　如何监视系统剪贴板变化 304
实例 183　如何获取系统剪贴板的数据格式 305
实例 184　如何设置系统日期 307
实例 185　如何获取操作系统的产品序列号和产品名称 309
实例 186　如何获取和设置鼠标双击响应速度 310
实例 187　如何弹出和关闭光驱 311
实例 188　如何检测网络连接 312
实例 189　如何创建拨号网络 314
第六部分　数据库应用 317
实例 190　如何创建不写代码的数据库程序 317
实例 191　如何添加 TDBNavigator 组件的按钮功能 319
实例 192　如何对数据库记录求和 321
实例 193　如何对数据库记录求平均值 323
实例 194　如何使用 Locate 方法查询数据库记录 324
实例 195　如何查询固定范围的数据库记录 325
实例 196　如何提高数据库应用程序查询效果 327
实例 197　如何使用数据集的书签 328
实例 198　如何浏览数据库中的图像 330
实例 199　如何在删除数据库记录前给予提示 331
实例 200　如何取得数据集的当前状态 332
实例 201　如何创建主从关系的数据集 334
实例 202　如何在一个数据集中浏览另一个数据集的数据 336
实例 203　如何在数据库更新过程中增加事务操作 337
实例 204　如何在程序中实现查找字段 340

实例 205　如何在程序中实现自定义字段.....342
实例 206　如何使用字段拖放功能.....343
实例 207　如何为程序添加报表打印功能.....345
实例 208　如何使用向导创建打印报表.....348
实例 209　如何设置一对多数据库表.....349
实例 210　如何设置一对多报表打印功能.....351
实例 211　如何根据一对多数据库表绘制图形.....354
实例 212　如何根据数据库表显示饼图.....357
实例 213　如何根据数据库表绘制条形图.....358
实例 214　如何动态绘制数据库图表.....360
实例 215　如何设置多重群组的打印报表.....362
实例 216　如何编写商业决策分析程序.....364
实例 217　如何导出对象库中的商业决策分析程序.....367
实例 218　如何打印含有图像字段的数据库表.....368
实例 219　如何实现数据库表整表复制.....370
实例 220　如何实现数据库记录的批量删除.....371
实例 221　如何实现数据库记录的批量修改.....374
实例 222　如何使用 SQL Builder 工具创建 SQL 语句.....377
实例 223　如何在程序运行时指定 SQL 语句.....379
实例 224　如何在程序中创建并使用数据库.....381
实例 225　如何在程序中控制与数据库的连接.....385
实例 226　如何在程序中创建 Access 数据库 ODBC 数据源.....386
实例 227　如何在程序运行时创建数据库 BDE 别名.....388
实例 228　如何在安装程序中设置 BDE 引擎.....390
实例 229　如何消除 MS-SQL Server 数据库连接的登录框.....391
实例 230　如何连接 MS-SQL Server 数据库.....393
实例 231　如何连接 MS-Access 数据库.....395
实例 232　如何连接 dBase 数据库.....397
实例 233　如何使用 ODBC 连接 Excel 表.....399
实例 234　如何使用 ADO 操作 Access 数据库.....402
实例 235　如何定制 DBGrid 组件的列表头.....404
实例 236　如何统计 DBGrid 组件多选单元格.....406
实例 237　如何设置 DBGrid 组件指定单元格的颜色值.....408
实例 238　如何在 DBGrid 组件单元格中增加下拉菜单.....410
实例 239　如何定制 StringGrid 组件指定单元格的颜色.....411
实例 240　如何获取 StringGrid 组件鼠标指定单元格的值.....413

实例 241　如何使用 StringGrid 组件显示数据库查询结果.....414
实例 242　如何禁止数据控制组件与数据集组件同步刷新.....415
实例 243　如何把数据库表转换成文本文件.....417
实例 244　如何把数据库中的数据写入 Word 文档.....419
实例 245　如何设置数据库字段的输入显示格式.....421
实例 246　如何管理数据库程序的错误信息.....423
实例 247　如何获取数据库 BDE 别名.....425
实例 248　如何获取数据库 BDE 别名参数信息.....426
实例 249　如何获取数据库的工作目录.....427
实例 250　如何获取数据库驱动程序名称.....428

第一部分　界面设计

实例 001　如何实现程序闪屏效果

实例目的

此实例主要介绍了通过使用延时和动态创建窗体实现在程序主窗口出现前产生闪屏效果的编程技术。实例程序执行后，效果如图 1-1 所示。



图 1-1

实现方法

闪屏效果通常用于在程序启动后初始化时，提示用户等待。在现在发行的大多数软件中，都可以发现它们在程序主窗口出现前已经使用闪屏效果，特别是一些大型软件。要使程序产生闪屏效果，可以使用一些专用的闪屏组件来实现，也可以参考下面的方法，通过使用延时和动态创建窗体的技术实现程序闪屏效果。具体实现步骤如下：

（1）启动 C++Builder 创建一个新的工程项目，默认情况下，主窗体名称为 Form1。

（2）选择菜单 File\New\Form 产生一个新窗体，默认情况下，该窗体名称为 Form2。它将被作为闪屏效果图片的支撑窗体。

（3）在组件面板的 Additional 标签页上拖动图像控件 TImage 组件到 Form2 上，并设置一幅图像，如图 1-1 所示。

（4）选择菜单 Project\Oprtions...，弹出 Project Options for...对话框，在 Forms 标签页中，将 Form2 从 Auto_create forms 列表框移到 Available forms 列表框，如图 1-2 所示，单

击 OK 按钮关闭该对话框。



图 1-2

（5）按照如下所示的代码修改项目主文件。

```
#include <vcl.h>
#include "Showform.h" //闪屏窗体头文件
#pragma hdrstop
USEFORM("Mainform.cpp", Form1);
USEFORM("Showform.cpp", Form2);
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
     {
      Application->Initialize();
      Form2=new TForm2(Application);
      Form2->Position=poScreenCenter;
      Form2->Show();
      Form2->Update();
      Application->ProcessMessages();
      //闪屏延时时间
      Sleep(3000);
      Application->CreateForm(__classid(TForm1), &Form1);
      Form2->Hide();
      delete Form2;
      Application->Run();
     }
     catch(Exception & exception)
     {
       Application->ShowException(&exception);
     }
   return 0;
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分　界面设计\实例 001 如何实现程序闪屏效果”。

实例 002　如何实现程序窗口闪烁

实例目的

此实例主要介绍了如何制作闪烁程序窗口的编程技术。实例程序执行后，在“闪烁次数”编辑栏中输入闪烁次数后，再单击“开始闪烁”按钮，可以发现程序窗口就不停地闪烁（闪烁指定的次数），效果如图 2-1 所示。



图 2-1

实现方法

Windows 操作系统提供了一个 API 函数 FlashWindow()，该函数能够让指定程序窗口呈闪烁状态，该函数的声明如下：

```
BOOL FlashWindow(
    HWND hWnd,          //闪烁窗口句柄
    BOOL bInvert        //闪烁状态
  );
```

该函数的返回值为 BOOL 类型，它指明了指定窗口的闪烁状态。若窗口呈激活状态则返回非零值，否则返回值为零。但是这个函数仅能使窗口闪烁一次，若想使窗口不停闪烁，应该使用定时器组件配合实现。如果要使程序窗口实现有限次数的闪烁，可以使用另一个函数 FlashWindowEx()实现。使用函数 FlashWindowEx()实现窗口有限次数闪烁的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//闪烁程序窗口
  FLASHWINFO PFLASHWINFO;
  PFLASHWINFO.cbSize=sizeof(FLASHWINFO);
  PFLASHWINFO.hwnd=Handle;
```

```
    PFLASHWINFO.dwFlags=FLASHW_ALL;
    PFLASHWINFO.uCount=StrToInt(this->Edit1->Text);
    PFLASHWINFO.dwTimeout=500;
    FlashWindowEx(&PFLASHWINFO);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 002 如何实现程序窗口闪烁”。

实例 003 如何制作吸附窗口程序

实例目的

此实例介绍了制作具有类似吸附力类型的应用程序窗口的编程技术。实例程序运行后，拖动应用程序到屏幕左端或顶端，当程序窗口接近屏幕左端或顶端时，程序窗口像被磁铁吸引似地快速移动到屏幕左端或顶端，效果如图 3-1 所示。



图 3-1

实现方法

细心的用户会发现，Winamp 播放器窗口具有类似吸附的功能，当 Winamp 播放器窗体接近某一边界时，它会像被该边界吸引似地快速移动到该边界。实际上制作这种类型的窗口并不难，要使窗口在接近某边界时产生一种磁性效果，必须在窗口进行快速移动之前做出一个判断，判断窗口将要到达的新位置的坐标值是否小于该范围（即磁性距离，大小可由自己定义）。但是，在 C++Builder 中并没有窗体移动事件，因此必须自己在程序里面拦截窗体的移动消息。要在程序里拦截窗体移动消息，只需要重载 WndProc()函数即可。实例程序处理 WndProc()函数实现窗口吸附效果的示例代码如下（实例程序假定在离屏幕顶端 20 像素，左端 100 像素的范围为磁性距离）：

```
void __fastcall TForm1::WndProc(TMessage& Message)
{
  switch(Message.Msg)
```

```
{
 /*在窗口移动之前，判断窗口新位置的 Top 值是否在磁性范围内，如果在，让它归 0，
 表示不产生窗体移动，避免在磁性范围内移动窗口时窗口过于闪烁*/
 case WM_WINDOWPOSCHANGING:
    if ((LPWINDOWPOS(Message.LParam))->y<20)
      (LPWINDOWPOS(Message.LParam))->y=0;
    if ((LPWINDOWPOS(Message.LParam))->x<100)
      (LPWINDOWPOS(Message.LParam))->x=0;
    break;
 default:
    break;
 }
 TForm::WndProc(Message);
}
```

也许有人会问为什么不拦截 WM_MOVE 这个消息监视应用程序窗口的移动？其实，只要试一下就知道了，如果拦截这个消息，窗口在磁性范围内移动时会很闪烁。原因很简单，WM_WINDOWPOSCHANGING 是在窗口移动之前就做出了判断，如果窗口将要在磁性范围内移动（此时尚未移动），则不让窗口移动；而 WM_MOVE 在窗口移动以后才做判断，这样的结果就是不断让 Top 复位成 0，当然会造成窗口闪烁了。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分　界面设计\实例 003　如何制作吸附窗口程序”。

实例 004　如何制作透明程序窗口

实例目的

此实例介绍了如何制作透明程序窗口的编程技术。实例程序执行后，效果如图 4-1 所示。



图 4-1

实现方法

在 C++Builder 中，制作透明程序窗体通常分为两步：一是重载 CreateParams 函数，设置透明风格参数 WS_EX_TRANSPARENT；二是重载它的 WM_ERASEBKGND 消息更新背景。实例程序制作透明程序窗体（主窗口）的头文件和实现文件的内容如下：

```
//filename mainform.h
#ifndef MainformH
#define MainformH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
class TForm1 : public TForm
{
__published:    // IDE-managed Components
private:        // User declarations
       void __fastcall WMNC(TMessage &Message);
      BEGIN_MESSAGE_MAP
      VCL_MESSAGE_HANDLER(WM_ERASEBKGND,TMessage,WMNC);
       END_MESSAGE_MAP(TForm);
public:         // User declarations
       __fastcall TForm1(TComponent* Owner);
      virtual void __fastcall CreateParams(TCreateParams &Params);
};
extern PACKAGE TForm1 *Form1;
#endif

//filename: mainform.cpp
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
}
void __fastcall TForm1::WMNC(TMessage &Message)
{
  SetBkMode(GetWindowDC(this->Handle),OPAQUE);
  ReleaseDC(this->Handle,GetWindowDC(this->Handle)) ;
  Message.Result=1;
}
void __fastcall TForm1::CreateParams(TCreateParams &Params)
{
  TForm::CreateParams(Params);
  Params.ExStyle|=WS_EX_TRANSPARENT;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分　界面设计\实例 004　如何制作透明程序窗口”。

实例 005　如何制作半透明程序窗口

实例目的

此实例介绍了如何制作半透明程序窗体的编程技术。实例程序执行后，效果如图 5-1 所示。



图 5-1

实现方法

Windows 2000 以上版本的操作系统提供了一个 SetLayeredWindowAttributes()的 API 函数，它能够使窗体实现透明显示，并且可以调节透明级数。该函数的声明如下：

```
BOOL SetLayeredWindowAttributes(
HWND hwnd,          // 窗口句柄
COLORREF crKey,     // 颜色值
BYTE bAlpha,        // 透明度值，0~255 表示从全透明到不透明
DWORD dwFlags       // LWA_COLORKEY 或 LWA_ALPHA 标志值
);
```

以下是实例程序实现程序界面半透明显示的示例代码：

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{//处理主窗口 OnCreate 事件
  long lRt=GetWindowLong(Handle,GWL_EXSTYLE);
  lRt=lRt|WS_EX_LAYERED;
  SetWindowLong(Handle,GWL_EXSTYLE,lRt);
  SetLayeredWindowAttributes(Handle,clBlue,200,LWA_ALPHA);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 005 如何制作半透明程序窗口”。

实例 006 如何制作不规则程序窗口

实例目的

此实例主要介绍了如何创建不规则的应用程序窗口的编程技术。实例程序执行后，效果如图 6-1 所示。



图 6-1

实现方法

弹出一个不规则的应用程序界面，使应用程序更具有个性化是现代多媒体应用软件的特征。要创建一个不规则的应用程序窗口，需要熟悉 Windows API 函数：椭圆形（或圆形）区域创建函数 CreateEllipticRgn()、多边形区域创建函数 CreatePolygonRgn()、矩形区域创建函数 CreateRectRgn()、带圆角的矩形区域创建函数 CreateRoundRectRgn()。不过最简单的方法是在一些已有的图像中，通过滤色技术过滤掉一些程序窗口不需要的颜色来天然形成这些不规则的窗口。最后，通过 API 函数 SetWindowRgn()将创建的不规则图形用作应用程序的程序界面。SetWindowRgn()函数的声明如下：

```
int SetWindowRgn(
      HRGN hRgn,                    //不规则区域句柄
      BOOL bRedraw                  //是否重画不规则区域
);
```

该函数如果操作成功，返回一个非 0 数字，否则为 0。实例程序创建一个不规则的应

用程序窗口的示例步骤和代码如下：

（1）在应用程序窗体上加载一个 TImage 控件，并设置好图像。

（2）设置应用程序窗口的 BorderStyle 属性为 bsNone。

（3）为应用程序窗口的 OnCreate 事件加上如下代码：

```
void __fastcall TForm1::OnCreate(TObject *Sender)
{
  register int x,y;
  int l,r;
  POINT *a;
  bool bCount,bHeight;
  HRGN WndRgn,TRgn1,TRgn2;
  if((a=(POINT *)malloc(800*4*(sizeof(POINT))))==NULL)
  {
   MessageBox(Handle,"分配内存失败！","信息提示",MB_OK);
   exit(0);
  }
  Width=Image1->Width;
  Height=Image1->Height;
  //重绘窗口，使窗口大小等于图像大小
  Repaint();
  l=0;
  r=Image1->Height*2-1;
  WndRgn=CreateRectRgn(0,0,Image1->Width,Image1->Height);
   //产生轮廓坐标点数组
  for(y=0;y<Image1->Height;y++)
  {
   bCount=true;
   for(x=0;x<Image1->Width;x++)
    if(Image1->Canvas->Pixels[x][y]!=clWhite)
    {
     a[l].x=x+1;
     a[l].y=y;
     bCount=false;
     break;
    }
  if(bCount)
    a[l]=a[l-1];
  l++;
  bHeight=true;
  for(x=Image1->Width-1;x>=0;x--)
  //根据去掉部分的具体颜色而定，此例为白色 clWhite
  if(Image1->Canvas->Pixels[x][y]!=clWhite)
  {
   a[r].x=x;
   a[r].y=y;
   bHeight=false;
   break;
  }
  if(bHeight)
```

```
        a[r]=a[r+1];
      r--;
    }
    // 抠去图片内凹部分
    r=Image1->Height*2-1;
    for(y=0;y<Image1->Height;y++)
    {
      for(x=a[y].x;x<a[r].x;x++)
      if(Image1->Canvas->Pixels[x][y]==clWhite)
      {
        TRgn2=CreateRectRgn(x,y,x+1,y+1);
        CombineRgn(WndRgn,WndRgn,TRgn2,RGN_XOR);
        DeleteObject(TRgn2);
      }
      r--;
    }
    //将图片外围部分抠去
    TRgn1=CreatePolygonRgn(a,Image1->Height*2,ALTERNATE);
    CombineRgn(WndRgn,WndRgn,TRgn1,RGN_AND);
    DeleteObject(TRgn1);
    free(a);
    //显示不规则窗体
    SetWindowRgn(Handle,WndRgn,true);
    SetWindowPos(Handle,HWND_TOP,0,0,0,0,SWP_NOMOVE|SWP_NOSIZE);
}
```

最后，编译执行程序即可。如果要使程序支持鼠标拖动功能，请参考本书中关于拖动非标题栏区域移动窗口的技术。函数 CombineRgn()用于根据指定的模式将两个不规则图像进行组合而产生第三个图像。该函数的声明如下：

```
int CombineRgn(
    HRGN hrgnDest,              //目标区域句柄
    HRGN hrgnSrc1,              //源区域一句柄
    HRGN hrgnSrc2,              //源区域二句柄
    int fnCombineMode           //区域组合模式
  );
```

参数 int fnCombineMode 的可能值包括 RGN_AND、RGN_COPY、RGN_DIFF、RGN_OR、RGN_XOR。

常用的创建椭圆的函数 CreateEllipticRgn()的声明如下：

```
HRGN CreateEllipticRgn(
int nLeftRect,      //椭圆边界矩形左上角 X 坐标
int nTopRect,       //椭圆边界矩形左上角 Y 坐标
    int nRightRect, //椭圆边界矩形右下角 X 坐标
    int nBottomRect //椭圆边界矩形右下角 Y 坐标
  );
```

创建圆角矩形的函数 CreateRoundRectRgn()的声明如下：

```
HRGN CreateRoundRectRgn(
int nLeftRect,      //左上角区域的 X 坐标
int nTopRect,       //左上角区域的 Y 坐标
 int nRightRect,    //右下角区域的 X 坐标
```

```
    int nBottomRect,    //右下角区域的 Y 坐标
    int nWidthEllipse, //圆角椭圆的高度
    int nHeightEllipse //圆角椭圆的宽度
);
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 006 如何制作不规则程序窗口”。

实例 007　如何制作程序渐变背景

实例目的

此实例介绍了如何使程序窗口产生渐变色背景的编程技术。实例程序执行后，效果如图 7-1 所示。



图 7-1

实现方法

渐变色技术以前曾经作为一种画面的特殊显示效果出现在多个场合，例如许多安装程序的主界面，许多大型系统的欢迎画面等。C++Builder 的画布提供了一个 Color 属性，根据颜色渐变的基本原理（从 0~255 逐渐改变其值）填充显示渐变色显示区域的 Color 属性即可。制作程序渐变背景的示例代码如下：

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{//处理 OnPaint 事件，显示渐变背景
  double y,Step=Height/256.00;
  for(int i=255;i>0;i--)
  {
    Form1->Canvas->Brush->Color=TColor(0x00000000+i*0x10000);
    Form1->Canvas->FillRect(Rect(0,int(y),Width,int(y+Step)));
    y=y+Step;
  }
}
void __fastcall TForm1::FormResize(TObject *Sender)
{//处理 OnResize 事件，更新背景
```

```
    Form1->Refresh();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 007 如何制作程序渐变背景”。

实例 008 如何设置程序背景图像

实例目的

此实例介绍了如何设置应用程序窗口背景图像的编程技术。实例程序执行后，效果如图 8-1 所示。

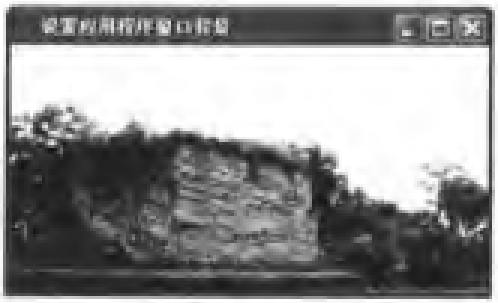


图 8-1

实现方法

设置应用程序窗口背景图像的最简单方法是使用 Image 组件，但是这种方法的明显缺点是会造成程序尺寸过大和组件排列等其他问题。如果将程序窗口背景图像以文件的形式存放则灵活得多，它不仅减少了程序尺寸，而且可以根据实际需要变换图像文件。通常采用以上方式设置程序窗口背景需要处理程序窗口的 OnPaint 事件和 OnResize 事件。如果只处理 OnPaint 事件而不处理 OnResize 事件，则在窗口进行缩放时可能造成背景画面不能更新。实例程序实现设置程序背景图像的示例代码如下：

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{//处理窗口的 OnPaint 事件，显示背景图像
  Graphics::TBitmap* pBmp=new Graphics::TBitmap();
  pBmp->LoadFromFile("Show.bmp");
  TRect MyRect=TRect(0,0,Width,Height);
  Form1->Canvas->StretchDraw(MyRect,pBmp);
  delete pBmp;
}
void __fastcall TForm1::FormResize(TObject *Sender)
{//处理窗口的 OnResize 事件，以便在窗口缩放时更新
  this->Refresh();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 008 如何设置程序背景图像”。

实例 009　如何设置 MDI 程序背景图像

实例目的

此实例介绍了在多文档应用程序中设置主窗口背景图像的编程技术。实例程序执行后，效果如图 9-1 所示。



图 9-1

实现方法

在 C++Builder 中，SDI 程序和对话框程序比较容易实现漂亮的软件界面，但在 MDI 程序中，由于向导默认生成的 MDI 程序隐藏了窗口管理的许多技术细节，用 SDI 程序的方法就不能轻易实现漂亮的软件界面。一般情况下，MDI 程序的窗口主要由两类组成：一类是 MDI 主窗口，另一类是 MDI 的客户窗口。客户窗口覆盖了主窗口的客户区，并提供大量的 MDI 支持。在 MDI 程序中，可以通过窗口的 Canvas->Draw()函数设置 MDI 主窗口的背景图像。在设置主窗口的背景图像时，为使背景图像能够不断刷新，必须处理 WM_PAINT 和 WM_ERASEBKGND 消息。在此可以利用钩子函数截住 Windows 操作系统发送给 MDI 客户窗口的 WM_PAINT 和 WM_ERASEBKGND 消息，从而实现客户窗口的刷新和重绘。通常编写 Windows 钩子函数分为 3 步：定义钩子、安装钩子和卸载钩子。

（1）定义钩子函数。钩子函数是一种特殊的回调函数，不同事件的钩子其函数头是不一样的，实例程序用到的钩子函数头声明如下：

```
LRESULT CALLBACK GetMsgProc(int nCode, WPARAM wParam, LPARAM lParam);
```

参数 int nCode 表示本钩子函数是否必须处理该消息；参数 WPARAM wParam 表明这次传递的消息是否已从 Windows 消息队列中删除；参数 LPARAM lParam 用来传送消息。

在钩子函数中，必须将系统发送的消息继续回送给系统，以使其他程序可以继续使用该消息，处理该问题的函数声明为：

```
LRESULT CallNextHookEx(HHOOK hhk, int nCode, WPARAM wParam, LPARAM lParam );
```

参数 HHOOK hhk 是安装钩子函数时安装函数返回的句柄，nCode、wParam 和 lParam 参数是系统传给钩子函数的值。

（2）安装钩子函数。定义完钩子函数后，必须将该钩子安装到 Windows 操作系统中才能生效，安装钩子的函数声明为：

```
HHOOK SetWindowsHookEx(int idHook,
HOOKPROC lpfn, HINSTANCE hMod, DWORD dwThreadId);
```

参数 int idHook 表示待安装的钩子函数类型，可以是键盘、鼠标或外壳等钩子，参数 HOOKPROC lpfn 表示钩子函数的地址，参数 HINSTANCE hMod 表示是全局钩子还是局部钩子，如果是全局钩子则钩子函数必须在 DLL 文件中，参数 DWORD dwThreadId 表示钩子将要起作用的程序 ID。

（3）卸载钩子函数。钩子函数使用完后必须卸载。卸载钩子的函数声明为：

```
BOOL UnhookWindowsHookEx(HHOOK hhk);
```

参数 HHOOK hhk 表示待卸载的钩子句柄。

下面是实例程序设置 MDI 类型程序主窗口背景图像的具体步骤：

（1）选择 File\New\Other...菜单，弹出 New Items 对话框。在该对话框中单击 Projects 标签页，选择其中的 MDI Application 对象，如图 9-2 所示，然后单击 OK 按钮，C++Builder 将默认产生一个 MDI 程序框架。

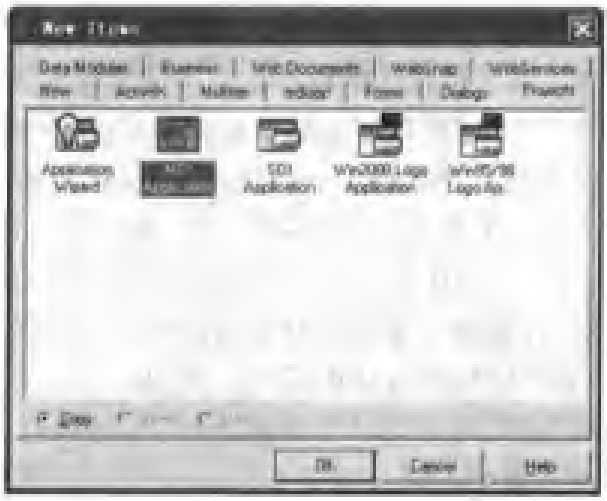


图 9-2

（2）按照如下所示的代码修改 C++Builder 默认产生的 main.cpp 文件。

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

```
#include "Main.h"
#include "About.h"
#pragma resource "*.dfm"
HHOOK hMsgHook;                                    //钩子句柄
int iClientHeight, iClientWidth;                   //待画的客户区高和宽
Graphics::TBitmap *BackBmp;
HBITMAP hFaceBitmap;                               //位图句柄
HWND hClientHandle, hMdiHandle;                    //MDI 主窗口和 MDI 客户窗口句柄
TMainForm *MainForm;
//在指定的窗口中画位图，填充整个客户窗口,true 为绘制成功，false 为绘制失败
BOOL DrawBitmap(HWND Handle, HBITMAP hBitmap, int iClientHeight, int
iClientWidth)
{
 if(hBitmap==NULL)
   return false;
 BITMAP Bmp;
 int iBitmapH, iBitmapW;
 GetObject( hBitmap, sizeof( BITMAP), &Bmp);
 iBitmapH = Bmp.bmHeight;
 iBitmapW = Bmp.bmWidth;
 HDC hClientDC, hMemDC;
 hClientDC = GetDC(Handle);
 if(hClientDC==NULL)
  return false;
 hMemDC = CreateCompatibleDC( hClientDC );
 if(hMemDC==NULL)
 {
  DeleteDC(hClientDC);
  return false;
 }
 SelectObject(hMemDC,hBitmap);
 int x=0, y=0;
 while(x<iClientWidth )
 {
  y = 0;
  while(y<iClientHeight)
  {
    MainForm->Canvas->Draw(x, y, BackBmp);
    BitBlt(hClientDC, x, y,iBitmapW, iBitmapH, hMemDC, 0, 0,SRCCOPY );
    y = y + iBitmapH;
  }
  x = x + iBitmapW;
 }
DeleteDC( hMemDC );
DeleteDC( hClientDC );
return true;
```

```
}
//钩子函数，处理系统 WM_PAINT 和 WM_ERASEBKGND 消息
LRESULT CALLBACK GetMsgProc(int nCode, WPARAM wParam, LPARAM lParam )
{
  LRESULT lReturn=0;
  MSG *cwMessage;
  cwMessage=(MSG*)lParam;
  if(cwMessage->hwnd==hClientHandle||cwMessage->hwnd==hMdiHandle)
  {  //是发送给子窗口的消息则处理
    if(cwMessage->message==WM_PAINT||cwMessage->message==WM_ERASEBKGND)
    {//重画客户窗口
      DrawBitmap(hClientHandle, hFaceBitmap, iClientHeight, iClientWidth);
    }
  }
  if(hMsgHook != NULL)
  {//将消息继续下传
    lReturn = CallNextHookEx(hMsgHook, nCode, wParam, lParam );
}
  return lReturn;
}
__fastcall TMainForm::TMainForm(TComponent *Owner)
    : TForm(Owner)
{
}
void __fastcall TMainForm::CreateMDIChild(String Name)
{
    TMDIChild *Child;
    //--- create a new MDI child window ----
    Child = new TMDIChild(Application);
    Child->Caption = Name;
    if (FileExists (Name))
        Child->Memo1->Lines->LoadFromFile(Name);
}
void __fastcall TMainForm::FileNew1Execute(TObject *Sender)
{
    CreateMDIChild("NONAME" + IntToStr(MDIChildCount + 1));
}
void __fastcall TMainForm::FileOpen1Execute(TObject *Sender)
{
    if (OpenDialog->Execute())
        CreateMDIChild(OpenDialog->FileName);
}
void __fastcall TMainForm::HelpAbout1Execute(TObject *Sender)
{
    AboutBox->ShowModal();
}
```

```
void __fastcall TMainForm::FileExit1Execute(TObject *Sender)
{
     Close();
}
void __fastcall TMainForm::FormPaint(TObject *Sender)
{
iClientHeight = ClientHeight;
iClientWidth = ClientWidth;
}
void __fastcall TMainForm::FormShow(TObject *Sender)
{//从文件中调入位图
  BackBmp = new Graphics::TBitmap();
  BackBmp->LoadFromFile("Show.bmp");
  hFaceBitmap = BackBmp->Handle;        //保存位图句柄
  hClientHandle = ClientHandle;         //保存窗口句柄
  hMdiHandle = Handle;                  //保存 MDI 主窗口句柄
  //安装截取程序消息的钩子函数
   hMsgHook = SetWindowsHookEx(WH_GETMESSAGE,
(HOOKPROC)GetMsgProc, NULL, GetCurrentThreadId());
}
void __fastcall TMainForm::FormClose(TObject *Sender, TCloseAction
&Action)
{//卸载钩子函数
 if ( hMsgHook != NULL)
   UnhookWindowsHookEx( hMsgHook );
 if ( BackBmp != NULL )
   delete BackBmp;
}
```

最后，编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 009 如何设置 MDI 程序背景图像”。

实例 010 如何动态关闭 MDI 程序所有子窗口

实例目的

此实例主要介绍了在 MDI 应用程序中，如何一次性地关闭用户打开的多个子窗口。实例程序启动后，多次单击工具栏上的 New 按钮创建多个子窗口，再单击“关闭所有子窗口”按钮，则刚才创建的所有子窗口将被关闭，效果如图 10-1 所示。



图 10-1

实现方法

在 MDI 应用程序中，可以同时打开多个子窗口。C++Builder 组件提供了许多优秀的属性来维护这些子窗口。属性 MDIChildCount 统计子窗口的数量，属性 WindowState 可以设置最大化、最小化等状态。下面是实例程序应用这些属性关闭 MDI 应用程序所有子窗口的实现代码。

```
void __fastcall TMainForm::BitBtn1Click(TObject *Sender)
{//动态关闭 MDI 程序所有子窗口
  for(int i=0;i<this->MDIChildCount;i++)
    this->MDIChildren[i]->Close();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 010 如何动态关闭 MDI 程序所有子窗口”。

实例 011 如何去掉 MDI 程序子窗口标题栏

实例目的

此实例介绍了如何去掉 MDI 程序子窗口标题栏的编程技术。实例程序执行后，效果如图 11-1 所示。



图 11-1

实现方法

Windows 在创建窗体时使用了一个 CreateWindowEx()函数，通过传给这个函数的参数来确定窗口的样式。而 CreateParams()函数是 VCL 在创建窗口之前调用的，用来取得要创建的窗口样式及其他一些相关属性（Params 参数）。通过设置 Params.Style^=WS_CAPTION;可以把窗口样式里的标题栏属性去掉。以下是实例程序实现去掉 MDI 程序子窗口标题栏的示例代码（注意：必须在子窗口文件中实现）。

```
void __fastcall TMDIChild::CreateParams(Controls::TCreateParams & Params)
{ //取消子窗口标题栏
  TCustomForm::CreateParams(Params);
  Params.Style^=WS_CAPTION;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 011 如何去掉 MDI 程序子窗口标题栏”。

实例 012　如何拖动非标题栏区域移动窗口

实例目的

此实例主要介绍了如何实现拖动应用程序非标题栏区域移动程序窗口的编程技术。当实例程序执行后，用鼠标拖动图像区域，程序窗口将跟随移动，效果如图 12-1 所示。



图 12.1

实现方法

在 Windows 操作系统中，通常移动程序窗口是通过拖动应用程序标题栏实现的，但是也可以通过拖动应用程序非标题栏区域实现程序窗口的移动，如一些多媒体播放器就具有此功能。移动应用程序窗口是通过处理 WM_NCHITTEST 消息来实现的，WM_NCHITTEST

消息称为“非客户区命中测试”，它优先于所有其他的客户区和非客户区鼠标消息，Windows 应用程序通常把这个消息传送给 DefWindowProc，然后 Windows 用这个消息产生基于鼠标位置的所有其他鼠标消息。实际编程时，可以使用函数 GetAsyncKeyState()来实时检查鼠标按钮的状态，通过检测虚拟键码 VK_LBUTTON 的高位是否被置位来探测鼠标动作，如果按下鼠标左键，此键码的高位被置为 1，此时 GetAsyncKeyState()函数将返回一个负数，同时通知 Windows 鼠标所在区域是标题栏区，Windows 就会按要求完成拖动操作。实例程序实现此技术的具体步骤如下：

（1）在窗体类头文件中声明 WM_NCHITTEST 消息映射和 WM_NCHITTEST 消息处理函数。代码如下所示：

```
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TImage *Image1;
private:        // User declarations
public:         // User declarations
        __fastcall TForm1(TComponent* Owner);
    void __fastcall WMNchitTest(TMessage &Message);
    BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(WM_NCHITTEST,TMessage,WMNchitTest)
    END_MESSAGE_MAP(TForm)
};
```

（2）在窗体类的实现中添加 WM_NCHITTEST 消息处理函数的实现代码。代码如下所示：

```
void __fastcall TForm1::WMNchitTest(TMessage &Message)
{
  if(GetAsyncKeyState(VK_LBUTTON)< 0)
    Message.Result=HTCAPTION;
  else
    Message.Result=HTCLIENT;
}
```

最后编译执行程序即可。注意，程序没有对标题栏按钮进行处理，在具体应用时，可以通过对鼠标所在位置的感知来判断有关按钮。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 012 如何拖动非标题栏区域移动窗口”。

实例 013 如何控制程序最小化窗口

实例目的

此实例介绍了采用 3 种不同的方式在程序中最小化应用程序窗口的编程技术。实例程

序执行后，效果如图 13-1 所示。



图 13-1

实现方法

在 C++Builder 中，可以使用下面 3 种方法之一来实现最小化应用程序窗口。

方法一：调用 SendMessage() API 函数发送一条 Windows 消息到主窗口的 Handle 属性或 Application->Handle。这条消息就是 WM_SYSCOMMAND，将 wParam 设为 SC_MINIMIZE 即可。例如：

```
SendMessage(Application->Handle, WM_SYSCOMMAND, SC_MINIMIZE, 0);
```

方法二：调用 ShowWindow()函数。调用时必须传送 Application 对象句柄到 ShowWindow()函数。如果传送给 ShowWindow()函数的句柄是主窗口，那么主窗口将最小化到桌面而不是任务条。例如：

```
ShowWindow(Application->Handle, SW_MINIMIZE);
```

方法三：调用 Application 对象的 Minimize()函数。例如：

```
Application->Minimize();
```

调用 Application 的方法较易用，但发送 WM_SYSCOMMAND 消息功能更强。另外，WM_SYSCOMMAND 消息允许最大化程序，改变光标为帮助光标，滚动程序，移动一个窗口，改变窗口大小，甚至模拟 Alt+Tab 切换到另一个窗口。以下是在实例程序中最小化应用程序窗口的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //最小化应用程序窗口
  //SendMessage(Application->Handle, WM_SYSCOMMAND, SC_MINIMIZE, 0);
  //ShowWindow(Application->Handle, SW_MINIMIZE);
  Application->Minimize();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 013 如何控制程序最小化窗口”。

实例 014 如何限制程序最大最小化窗口

实例目的

此实例介绍了如何限制程序的最大最小化窗口的编程技术。实例程序执行后，当单击右上角的放大按钮时，程序界面在放大到一定程度（因为已经限制）时不再放大，而不是全屏幕显示，效果如图 14-1 所示。



图 14-1

实现方法

每当 Windows 需要知道窗口最大化的位置和大小，或者是最大、最小跟踪尺寸时，就会向窗口发送 WM_GETMINMAXINFO 消息，通过重写这一消息处理函数，可以实现对应用程序窗口大小的限制。其中所需要的结构 MINMAXINFO 的定义如下：

```
typedef struct tagMINMAXINFO {
    POINT ptReserved;          //保留为内部使用
    POINT ptMaxSize;           //指定窗口的最大化宽度(X)和最大化高度(Y)
    POINT ptMaxPosition;       //指定最大化窗口的左边位置(X)和顶部位置(Y)
    POINT ptMinTrackSize;      //指定窗口的最小跟踪宽度(X)和最小跟踪高度(Y)
    POINT ptMaxTrackSize;      //指定窗口的最大跟踪宽度(X)和最大跟踪高度(Y)
} MINMAXINFO;
```

以下是实例程序通过处理 WM_GETMINMAXINFO 消息实现限制程序窗口大小的头文件和实现文件的具体内容。

```
//FileName:Mainform.h
#ifndef MainformH
#define MainformH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
```

```
    class TForm1 : public TForm
    {
    __published:    // IDE-managed Components
          TImage *Image1;
    private:        // User declarations
    public:         // User declarations
          __fastcall TForm1(TComponent* Owner);
          void __fastcall WMGetMinMaxInfo(TWMGetMinMaxInfo& Msg);
          BEGIN_MESSAGE_MAP

MESSAGE_HANDLER(WM_GETMINMAXINFO,TWMGetMinMaxInfo,WMGetMinMaxInfo)
          END_MESSAGE_MAP(TForm)
    };
    extern PACKAGE TForm1 *Form1;
    #endif

    //FileName:Mainform.cpp
    #include <vcl.h>
    #pragma hdrstop
    #include "Mainform.h"
    #pragma package(smart_init)
    #pragma resource "*.dfm"
    TForm1 *Form1;
    __fastcall TForm1::TForm1(TComponent* Owner)
          : TForm(Owner)
    {
    }
    void __fastcall TForm1::WMGetMinMaxInfo(TWMGetMinMaxInfo& Msg)
    { //限制程序的最大最小窗口
      Msg.MinMaxInfo->ptMaxSize.x=500;
      Msg.MinMaxInfo->ptMaxSize.y=500;
      Msg.MinMaxInfo->ptMaxPosition.x=250;
      Msg.MinMaxInfo->ptMaxPosition.y=100;
      Msg.MinMaxInfo->ptMaxTrackSize.x=500;
      Msg.MinMaxInfo->ptMaxTrackSize.y=500;
      Msg.MinMaxInfo->ptMinTrackSize.x=200;
      Msg.MinMaxInfo->ptMinTrackSize.y=200;
    }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 014 如何限制程序最大最小化窗口”。

实例 015　如何控制程序窗口自动隐藏或显示

实例目的

此实例介绍了如何实现程序窗口的自动隐藏或显示的编程技术。实例程序执行后，当

鼠标在程序窗口区域内时，程序自动显示；当鼠标离开程序窗口区域内时，程序自动隐藏。效果如图 15-1 所示。



图 15-1

实现方法

用过“金山快译.Net2001”的用户可能都知道，金山快译的窗口界面能够实现自动隐藏或自动显示。当用户刚启动金山快译时，它的程序窗口显示在屏幕的最上面，当鼠标离开金山快译窗口界面时，金山快译窗口会逐渐隐藏；当将鼠标重新放置在屏幕最上面时，金山快译窗口又会显示出来。在 C++Builder 中，通过使用定时器不断检测当前鼠标的位置，也能够实现类似的效果。实例程序实现这种显示效果的示例代码如下：

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{//通过不断读取光标位置，决定是否隐藏程序窗口
  POINT* pMyPoint=new POINT;
  GetCursorPos(pMyPoint);
  if((pMyPoint->x>Form1->Left)&&(pMyPoint->x<(Form1->Left+Form1->Width))&&
        (pMyPoint->y>Form1->Top)&&(pMyPoint->y<(Form1->Top+Form1->Height)))
  {
    Form1->Show();
  }
  else
  {
    Form1->Hide();
  }
  delete pMyPoint;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 015 如何控制程序窗口自动隐藏或显示”。

实例 016　如何隐藏应用程序

实例目的

此实例主要介绍如何在操作系统中隐藏正在运行的应用程序。当实例程序启动后，操作系统的任务栏不出现该实例程序标题，“Windows 任务管理器”的“应用程序”标签页中也不出现该实例程序标题，但是在“Windows 任务管理器”的“进程”标签页中可以发现该实例程序已经启动，效果如图 16-1 所示。



图 16-1

实现方法

在 C++Builder 中，设置程序主窗口的显示属性为 false，可以实现类似实例程序的效果。具体实现方法是按照如下所示的代码修改程序的 WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)函数。

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    Application->CreateForm(__classid(TForm1), &Form1);
    Application->ShowMainForm=false;
    ShowWindow(Application->Handle,SW_HIDE);
    Application->Run();
  }
  catch (Exception &exception)
```

```
    {
      Application->ShowException(&exception);
    }
    catch (...)
    {
      try
      {
        throw Exception("");
      }
      catch (Exception &exception)
      {
        Application->ShowException(&exception);
      }
    }
    return 0;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 016 如何隐藏应用程序”。

实例 017 如何控制程序在任务栏上是否出现

实例目的

此实例主要介绍了如何在操作系统的任务栏上控制应用程序的图标及标题栏文字是否显示。当实例程序执行后，单击“隐藏任务栏”按钮，应用程序在操作系统任务栏上隐藏；单击“显示任务栏”按钮，应用程序在操作系统任务栏上出现。效果如图 17-1 所示。



图 17-1

实现方法

操作系统任务栏是可以在屏幕上伸展的工具栏，它是应用程序图标所在的位置，标准的 Windows 应用程序运行时一般都会在任务栏上显示应用程序图标，用户可直接单击任务

栏图标进行程序切换，但有些应用程序不使用任务栏图标，如典型的 Office 工具条，也有些程序可由用户定制显示方式显示或隐藏任务栏图标。要隐藏应用程序的任务栏图标，可以应用 ShowWindow()函数并传给它 Application->Handle 窗口句柄实现。ShowWindow()函数用于设定指定窗口的显示状态，该函数的声明如下：

```
BOOL ShowWindow(
    HWND hWnd,          // 窗口句柄
    int nCmdShow        // 窗口显示状态
   );
```

其中，参数 int nCmdShow 的值包括 SW_HIDE、SW_MAXIMIZE、SW_MINIMIZE、SW_RESTORE、SW_SHOW、SW_SHOWDEFAULT、SW_SHOWMAXIMIZED、SW_SHOWMINIMIZED、SW_SHOWMINNOACTIVE、SW_SHOWNA、SW_SHOWNOACTIVATE、SW_SHOWNORMAL。应用该函数控制应用程序是否在操作系统任务栏上显示的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//在任务栏上隐藏应用程序
  ShowWindow(Application->Handle, SW_HIDE);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//在任务栏上显示应用程序
   ShowWindow(Application->Handle, SW_SHOW);
}
```

说明：通过 ShowWindow()函数来隐藏窗口的任务栏图标是不持久的。某些动作会使应用程序在任务栏上的图标重新出现。但可以将隐藏的应用程序窗口设为 Tool Window 来移走程序的任务栏图标而避免它再次出现。Tool Window 永远不会有任务栏图标。使应用程序窗口成为一个 Tool Window 有一个副作用：当按下 Alt+Tab 组合键时它将不在程序列表中出现。调用 API 函数 GetWindowLong()和 SetWindowLong()可以使应用程序窗口成为一个 Tool Window。示例代码如下：

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  DWORD dwExStyle = GetWindowLong(Application->Handle, GWL_EXSTYLE);
  dwExStyle |= WS_EX_TOOLWINDOW;
  SetWindowLong(Application->Handle, GWL_EXSTYLE, dwExStyle);
  try
  {
    Application->Initialize();
    Application->CreateForm(__classid(TForm1), &Form1);
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

SetWindowLong()函数的声明如下：

```
LONG SetWindowLong(
HWND hWnd,          //窗口句柄
int nIndex,         //索引
LONG dwNewLong      //定义要替换的值
);
```

其中，参数 Int nIndex 的取值如下：GWL_EXSTYLE（设置新窗口的扩展类型），GWL_STYLE（设置新窗口类型），GWL_WNDPROC（为窗口过程设置新地址），GWL_HINSTANCE（设置一个新的应用程序实例句柄），GWL_ID（为窗口设置一个新的识别号），GWL_USERDATA（为窗口设置一个相联系的 32 位值，每一个窗口都有一个 32 位值，应用程序产生窗口时，要利用这个值）。如果 HWND hWnd 参数为对话框类型，参数 int nIndex 还可以取如下值：DWL_DLGPROC（为对话框过程设置新地址），DWL_MSGRESULT（为对话框返回值），DWL_USER（为应用程序设置新的额外值）。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 017 如何控制程序在任务栏上是否出现”。

实例 018 如何使一个程序在任务栏同时出现多个标题

实例目的

此实例介绍了如何使一个程序在任务栏上同时出现多个标题的编程技术。实例程序执行后，效果如图 18-1 所示。



图 18-1

实现方法

在 Office 软件的最近版本中，它提供了一个特殊的功能，就是每打开或新建一个文档，在操作系统的任务栏上均显示该文档的标题栏内容，好像这些文档就是一个独立的应用程

序。此实例程序就是实现类似的功能。当实例程序启动时，操作系统任务栏上同时出现 3 个不同（可以为多个）的程序标题，单击不同的程序标题，实例程序就切换到该程序标题对应的操作界面。单击子窗口关闭按钮，子窗口关闭，操作系统任务栏上对应的程序标题消失；单击主窗口关闭按钮，则程序所有窗口关闭。使用操作系统的 API 函数 GetWindowLong()、SetWindowLong()设置程序窗口属性，即可实现这种效果。实例程序在创建完 3 个显示表单以后，在主界面表单的 OnCreate 事件中加入如下代码：

```
void __fastcall TForm1::TaskbarSwitch(TObject *Sender)
{
int i=GetWindowLong(this->Handle,GWL_STYLE);
TForm2 *Pic2Form;
Pic2Form=new TForm2(NULL);
Pic2Form->ParentWindow=GetDesktopWindow();
Pic2Form->Caption="任务栏标题切换:图像二";
SetWindowLong(Pic2Form->Handle,GWL_STYLE,i);
Pic2Form->Show();
TForm3 *Pic3Form;
Pic3Form=new TForm3(NULL);
Pic3Form->ParentWindow=GetDesktopWindow();
Pic3Form->Caption="任务栏标题切换:图像三";
SetWindowLong(Pic3Form->Handle,GWL_STYLE,i);
Pic3Form->Show();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 018 如何使一个程序在任务栏同时出现多个标题”。

实例 019　如何使程序窗口总在最前面显示

实例目的

此实例主要介绍了如何使应用程序窗口总显示在系统其他应用程序前面的编程技巧。实例程序运行后，程序可以在最前面显示和正常显示两种模式间进行切换，效果如图 19-1 所示。



图 19-1

实现方法

一般情况下，通过使用 API 函数 SetWindowPos()可以实现应用程序窗口总在最前面显示。该函数的声明如下：

```
BOOL SetWindowPos(
    HWND hWnd,                          //窗体句柄
    HWND hWndInsertAfter,               //放置窗体位置的句柄
    int X,                              //水平位置坐标
    int Y,                              //垂直位置坐标
    int cx,                             //窗体宽度
    int cy,                             //窗体高度
    UINT uFlags                         //窗体位置标志
   );
```

其中，参数 HWND hWndInsertAfter 的取值包括 HWND_BOTTOM、HWND_NOTOPMOST、HWND_TOP、HWND_TOPMOST；参数 UINT uFlags 的取值包括 SWP_DRAWFRAME、SWP_FRAMECHANGED、SWP_HIDEWINDOW、SWP_NOACTIVATE、SWP_NOCOPYBITS、SWP_NOMOVE、SWP_NOOWNERZORDER、SWP_NOREDRAW、SWP_NOREPOSITION、SWP_NOSENDCHANGING、SWP_NOSIZE、SWP_NOZORDER、SWP_SHOWWINDOW。该函数一般在窗体创建时使用。

在 C++Builder 中，用户也可以在表单属性中设置表单窗口在最前面显示。在表单的窗口属性中，有两个属性：Position 和 FormStyle，这两个属性都有关于 Top 的值，正确的设置是取 FormStyle 的 fsStayOnTop 值，而不是 Position 的 poDesktopCenter 值。实例程序通过设置窗体的FormStyle属性值为fsStayOnTop实现应用程序窗口总在最前面显示的示例代码如下：

```
void __fastcall TForm1::Form1OnCreate(TObject *Sender)
{  //设置系统菜单
  AppendMenu(GetSystemMenu(Handle,false),MF_SEPARATOR,0,"");
  AppendMenu(GetSystemMenu(Handle,false),MF_STRING,200,"最顶层显示(&A)");
}
void __fastcall TForm1::SystemMenuCommand(TWMMenuSelect &Msg)
{
   TForm::Dispatch(&Msg);
   if(Msg.IDItem==200)
   {
     if(Form1->FormStyle==fsNormal)
     {//将窗体设置为总在最前面显示
      Form1->FormStyle=fsStayOnTop;
      //重新设置系统菜单
      AppendMenu(GetSystemMenu(Handle,false),MF_SEPARATOR,0,"");
      AppendMenu(GetSystemMenu(Handle,false),MF_STRING,200,"正常显示(&A)");
     }
     else
```

```
        (//将窗体设置为正常显示模式
         Form1->FormStyle=fsNormal;
         //重新设置系统菜单
         AppendMenu(GetSystemMenu(Handle,false),MF_SEPARATOR,0,'');
         AppendMenu(GetSystemMenu(Handle,false),MF_STRING,200,'最顶层显示(&A)');
        }
      }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 019 如何使程序窗口总在最前面显示”。

实例 020　如何隐藏应用程序标题栏

实例目的

此实例主要介绍了如何隐藏应用程序标题栏。当实例程序执行后，程序标题栏将被隐藏，效果如图 20-1 所示。



图 20-1

实现方法

在 C++Builder 中，隐藏应用程序标题栏与其他编程工具隐藏标题栏的方式不同，程序必须通过设置窗体创建时的参数控制标题栏是否显示，它通过在 TForm 的派生类中重载 CreateParams 函数，去除相应的窗口标识位 WS_CAPTION，才能在运行时删去窗体的标题栏。实例程序实现此效果的具体实现步骤如下：

（1）在窗体头文件中声明 CreateParams(TCreateParams& Params)函数，代码如下所示：

```
class TForm1 : public TForm
{
```

```
...
public:         // User declarations
 __fastcall TForm1(TComponent* Owner);
 void __fastcall CreateParams(TCreateParams& Params);
};
```

（2）在窗体实现文件中定义 CreateParams(TCreateParams& Params)函数。代码如下所示：

```
void __fastcall TForm1::CreateParams(TCreateParams& Params)
{
TForm::CreateParams(Params);
Params.Style &=~WS_CAPTION;
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 020 如何隐藏应用程序标题栏”。

实例 021 如何设置程序主标题

实例目的

此实例主要解决应用程序在操作系统任务栏上不显示应用程序文件名，而显示定制的应用程序特殊文字串的问题。当实例程序执行后，将在操作系统任务栏上显示自定义的程序标题，效果如图 21-1 所示。



图 21-1

实现方法

使用 C++Builder 开发程序时，默认情况下，它会自动生成一个名为 Project1.exe 的执行程序，并且在操作系统的任务栏上也显示同样的标题。其实，可以通过对项目的设置，改变程序在任务栏上的显示标题。步骤如下：选择菜单 Project/Options...，弹出 Project Options for Project1.exe 对话框，单击 Application 标签页，在 Title 编辑框中输入程序主标

题“这是我的应用程序”即可，如图 21-2 所示。



图 21-2

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 021 如何设置程序主标题”。

实例 022　如何设置程序图标

实例目的

此实例主要解决如何在应用程序中用特定的图标替换掉 C++Builder 默认情况下给程序定义的图标。当实例程序启动后，在操作系统任务栏和应用程序标题栏显示定制的图标，效果如图 22-1 所示。



图 22-1

实现方法

在使用 C++Builder 开发程序时，默认情况下，它会使用 C++Builder 自己的图标作为

程序图标。其实，可以通过对项目的设置，改变程序的图标。步骤如下：选择菜单 Project/Options...，弹出 Project Options for Project1.exe 对话框，单击 Application 标签页，单击 Load Icon...按钮，弹出选择图标对话框，选择一个图标文件即可，如图 22-2 所示。



图 22-2

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 022 如何设置程序图标”。

实例 023 如何在程序中改变菜单位置

实例目的

此实例介绍了如何在程序中将菜单项从左边移到右边的编程技术。实例程序执行后，效果如图 23-1 所示。



图 23-1

实现方法

在 C++Builder 中，如果需要改变程序菜单项的布局，只需设置菜单风格即可。如将菜单的类型设置为 MFT_RIGHTJUSTIFY，则选择的子菜单将在右边显示。以下是在单文档应用程序中调整菜单显示位置的示例代码。

```
__fastcall TSDIAppForm::TSDIAppForm(TComponent *AOwner)
    : TForm(AOwner)
{
  char StrTemp[255];
  MENUITEMINFO MenuInfo;
  MenuInfo.cbSize=sizeof(MENUITEMINFO);
  MenuInfo.fMask=MIIM_TYPE;
  MenuInfo.dwTypeData=StrTemp;
  GetMenuItemInfo(MainMenu1->Handle,2,true,&MenuInfo);
  MenuInfo.fType|=MFT_RIGHTJUSTIFY;
  SetMenuItemInfo(MainMenu1->Handle,2,true,&MenuInfo);
  DrawMenuBar(Handle);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 023 如何在程序中改变菜单位置”。

实例 024　如何删除程序系统关闭菜单

实例目的

此实例介绍了如何在程序中删除默认的系统关闭菜单的编程技术。实例程序执行后，效果如图 24-1 所示。



图 24-1

实现方法

在 C++Builder 中，删除应用程序的系统菜单通常需要两个步骤：一是使用 GetSystemMenu()

函数取得系统菜单句柄，二是通过 RemoveMenu()函数删除指定的菜单。其中删除菜单函数 RemoveMenu()的声明如下：

```
BOOL RemoveMenu(
    HMENU hMenu,      //菜单句柄
    UINT uPosition,   //菜单标识符或位置
    UINT uFlags       //菜单标志，包括 MF_BYCOMMAND 和 MF_BYPOSITION 取值
   );
```

实例程序使用上述函数实现删除系统关闭菜单的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//删除系统关闭菜单
  HWND hSysMenu;
  hSysMenu=GetSystemMenu(Form1->WindowHandle,0);
  RemoveMenu(hSysMenu, 6, MF_BYPOSITION);
  MessageBox(Handle,'删除系统关闭菜单操作成功！','信息提示',MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 024 如何删除程序系统关闭菜单”。

实例 025 如何拷贝当前程序界面

实例目的

此实例主要介绍了如何以位图的形式复制当前程序界面上所显示内容的编程技巧。实例程序运行后，效果如图 25-1 所示。



图 25-1

实现方法

当应用程序出现错误时，可以通过 try...catch 块进行捕获，并进行处理。对于已知的预料中的错误，程序可以给出相应的提示，对于一些未知的错误，则可以采用下面的方法，

将错误的内容以位图的形式整个屏幕复制下来进行分析处理。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  TCanvas *dtCanvas = new TCanvas;
  dtCanvas->Handle=GetDC(0);
  Graphics::TBitmap *bitmap=new Graphics::TBitmap;
  bitmap->Width=Width;
  bitmap->Height=Height;
  int nColors=GetDeviceCaps(Canvas->Handle,SIZEPALETTE);
  LOGPALETTE* logPal=(LOGPALETTE*)new Byte[sizeof(LOGPALETTE)+(nColors-1)*sizeof
(PALETTEENTRY)];
  logPal->palVersion=0x300;
  logPal->palNumEntries=(Word)nColors;
  GetSystemPaletteEntries(Canvas->Handle,0,nColors,logPal->palPalEntry);
  bitmap->Palette=CreatePalette(logPal);
   delete[] logPal;
  TRect src=BoundsRect;
  TRect dest=Rect(0,0,Width,Height);
  bitmap->Canvas->CopyRect(dest,dtCanvas,src);
  this->Image1->Picture->Bitmap=bitmap;
  delete bitmap;
  delete dtCanvas;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 025 如何拷贝当前程序界面”。

实例 026　如何动态创建程序表单

实例目的

此实例主要解决如何在程序中包含大量表单的情况下动态生成表单，以减少程序所需要的系统资源。当实例程序启动后，单击“动态创建表单”按钮，则程序将自动动态创建一个表单，效果如图 26-1 所示。



图 26-1

实现方法

在 C++Builder 中，通常由 Form 组件创建的表单，它会在存储器内实际地配置内存空间给这些 Form 使用，即在程序一开始运行时，Form 组件创建了多少个表单，程序就立即分配多少个表单的内存空间；在表单太多的情况下，这通常会导致程序启动速度降低甚至根本无法运行。动态创建表单，则在程序需要使用表单时才分配表单所需要的存储空间，使用完毕之后立即释放所占用的存储空间，这样应用程序就不会占用系统太多的资源。

实例程序动态创建表单的步骤如下：首先在项目中新建两个表单 Form1、Form2，Form1 为程序主窗口界面，Form2 为动态创建的表单；然后选择菜单 Project\Options...，弹出 Project Options For DynamicForm.exe 对话框，单击 Forms 标签页，在 Auto-create forms 列表框中选择 Form2，单击“>”按钮，则 Form2 将从 Auto-create forms 列表框转移到 Available forms 列表框，如图 26-2 所示，单击 OK 按钮完成操作。



图 26-2

在程序中动态创建表单时，则使用以下代码实现。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  TForm2* DynaForm;
  try
  {
    DynaForm=new TForm2(Application);
    DynaForm->ShowModal();
    delete DynaForm;
  }
  catch(Exception &e)
  {
     delete DynaForm;
     Application->ShowException(&e);
```

```
    }
}
```

注意：除了可以使用 delete 释放分配的系统资源外，也可以在表单的 OnClose()消息处理函数中通过 TCloseAction &Action 参数进行处理。TCloseAction &Action 参数有 4 种状态，包括 caNone、caHide、caFree、caMinimize，其中的 caFree 就会把动态创建的表单资源释放掉。代码如下所示：

```
void __fastcall TForm2::CloseDynaForm(TObject *Sender, TCloseAction &Action)
{
  Action=caFree;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 026 如何动态创建程序表单”。

实例 027　如何遍历窗体所有组件

实例目的

此实例主要介绍了在 C++Builder 中，如何使用 TForm 属性遍历窗体所有控件的技术。当实例程序执行后，效果如图 27-1 所示。



图 27-1

实现方法

使用 C++Builder 编程与使用其他 C++编程语言工具进行编程的优势在于：C++Builder 提供了大量的成体系的组件，通过对这些组件的简单拖放即可实现程序的功能。因此，在程序中如何对窗体上的组件进行控制就显得非常重要，特别是群集同类型组件的快速处理。例如：如何一次性禁止窗体所有组件？如何一次性给窗体所有编辑栏赋值或清空？一旦掌握了怎样搜索和判断窗体所有组件技术，这些问题就会轻松解决，而不用一个组件一个组件地个别处理。因为 TForm 组件在维护窗体上的所有组件时，是通过属性 ControlCount

统计窗体组件的数量，通过属性 Controls 存放窗体上的所有组件，在编程时通过对这两个属性的简单操作即可实现相当丰富的功能。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  AnsiString StrControl="组件类型及名称:";
  for(int i=0;i<this->ControlCount;i++)
  {
    StrControl+=this->Controls[i]->ClassName();
    StrControl+="[";
    StrControl+=this->Controls[i]->Name;
    StrControl+="]";
    StrControl+="\t";
    if(this->Controls[i]->Name.Pos("TEdit1")==0)
    {
      TEdit* pEdit=(TEdit*)this->Controls[i];
      pEdit->Color=clYellow;
    }
  }
  this->Edit1->Text=StrControl;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 027 如何遍历窗体所有组件”。

实例 028 如何保证组件位置相对固定

实例目的

此实例主要介绍如何在程序窗口缩放时，保持窗口上控件的位置相对整个窗口的位置比例保持不变。实例程序执行后，拖动程序边框缩放窗口，可以发现图像和按钮始终居中，效果如图 28-1 所示。



图 28-1

实现方法

在使用 C++Builder 编程时，一旦将组件拖放到 Form 后，组件的位置就相对固定死了。这往往造成：随着 Form 的大小或屏幕分辨率的改变，组件和 Form 本身的位置往往变得非常混乱。不过，C++Builder 的 Form 组件有一个 OnResize 事件，Form 的所有与大小变化有关的变化都可以触发这个事件，包括 Form 的创建、最大化/最小化/还原、用鼠标拖动改变大小等。因此，在该事件中动态地改变各个组件的位置可以确保其在 Form 中的相对位置不变。示例代码如下：

```
void __fastcall TForm1::OnResize(TObject *Sender)
{
    int ixPos=Width/2;
    int iyPos=Height/2;
    TControl *pControl;
    for(int i=0; i<ControlCount; i++)
    {
      pControl=Controls[i];
      pControl->Left=ixPos-pControl->Width/2;
      pControl->Top=iyPos-pControl->Height/2-15;
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 028 如何保证组件位置相对固定”。

实例 029　如何在窗体中动态增加组件

实例目的

此实例介绍了如何在窗体中动态增加组件的编程技术。实例程序执行后，单击“动态增加组件”按钮，将新增两个标签文本组件和编辑组件，效果如图 29-1 所示。

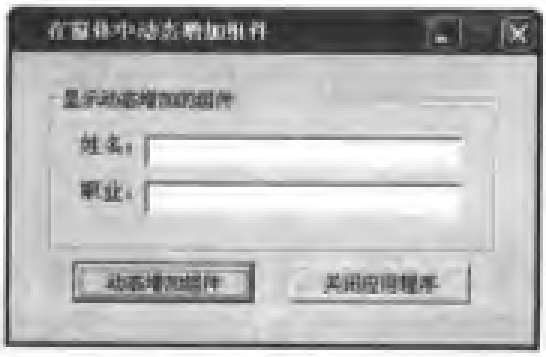


图 29-1

实现方法

在使用 C++Builder 开发应用程序时，通常是从组件面板上将组件拖到表单上直接使用。但是在某种情况下，需要在程序中动态增加组件。这样做的最大好处就是能够减少系统资源占用，优化程序运行效率，缺点是不直观，不易把握。以下是实例程序在运行时直接增加组件的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//动态增加组件
    TLabel* pLab1=new TLabel(this->GroupBox1);
    TLabel* pLab2=new TLabel(this->GroupBox1);
    pLab1->Parent=this->GroupBox1;
    pLab2->Parent=this->GroupBox1;
    pLab1->Caption="姓名：";
    pLab2->Caption="职业：";
    pLab1->Left=20;
    pLab1->Top=30;
    pLab2->Left=20;
    pLab2->Top=60;
    TEdit* pE1=new TEdit(this->GroupBox1);
    TEdit* pE2=new TEdit(this->GroupBox1);
    pE1->Parent=this->GroupBox1;
    pE2->Parent=this->GroupBox1;
    pE1->Left=60;
    pE1->Top=30;
    pE1->Width=200;
    pE2->Left=60;
    pE2->Top=60;
    pE2->Width=200;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 029 如何在窗体中动态增加组件”。

实例 030 如何使组件在程序运行时可以移动

实例目的

此实例介绍了如何在程序运行时用户能够移动组件的编程技术。实例程序执行后，用鼠标拖动编辑栏，可以发现编辑栏组件可以被拖动到程序界面的任意地方，效果如图 30-1 所示。



图 30-1

实现方法

通常情况下，在程序界面上的组件是不可移动的，也就是说，组件在程序界面上的位置是相对固定的。不过，通过使用 Perform()函数传递系统消息，也可以实现在程序运行时移动组件。示例代码如下：

```
void __fastcall TForm1::Edit1MouseDown(TObject *Sender,
                    TMouseButton Button, TShiftState Shift, int X, int Y)
{//编辑栏组件的 OnMouseDown 事件
   WORD SC_DragMove=0xF012;
   ReleaseCapture();
   ((TWinControl*)Sender)->Perform(WM_SYSCOMMAND,SC_DragMove,0);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 030 如何使组件在程序运行时可以移动”。

实例 031　如何使用组件创建托盘程序

实例目的

此实例介绍了如何在 C++Builder 中直接使用 TrayIcon 组件而不是使用 API 函数和消息编写代码创建托盘程序的编程技巧。实例程序运行后，效果如图 31-1 所示。



图 31-1

实现方法

C++Builder 在组件模板的 Samples 标签页中提供了一个 TrayIcon 组件，它封装了创建托盘程序的大多数 API 功能，直接使用 TrayIcom 组件即可创建一个具有丰富功能的托盘程序。TrayIcon 组件的常用属性意义如表 31-1 所示。

表 31-1

TrayIcon 组件属性	属性说明
Animate	设为 true 时，顺序更改动画图标以产生动画效果
AnimateInterval	设置更改动画图标的时间（单位：毫秒）
Hide	设为 true 时，最小化时隐藏任务栏上的标题栏
Hint	设置鼠标移动到 Tary 图标上时出现的应用程序提示信息
IconIndex	设置动画图标的索引值
Icons	设置图标源（通常是 ImageList 组件，动画图标就在该组件中）
PopupMenu	设置 Tray 要弹出的鼠标右键菜单（通常是 PopupMenu 组件）
PopupMenuOn	设置触发弹出菜单的鼠标操作
RestoreOn	设置恢复最小化到正常状态的鼠标操作
Visible	设为 true 时，在任务栏最右边显示图标

属性 PopupMenuOn 所表示的鼠标动作意义如下：imClick（鼠标单击）；imDoubleClick（鼠标双击）；imLeftClickUp（单击鼠标左键，放开）；imLeftDoubleClick（双击鼠标左键）；imMouseDown（按下鼠标键）；imMouseUp（按下鼠标键，放开）；imNone（不设置鼠标操作）；imRightClickUp（单击鼠标右键，放开）；imRightDoubleClick（双击鼠标右键）。下面是实例程序使用组件 TrayIcon 创建托盘程序的示例步骤：

（1）在组件模板的 Win32 标签页中选择 ImageList 组件到 Form1 上，双击它，按照图 31-2 所示，增加托盘区域中显示的动画图标。

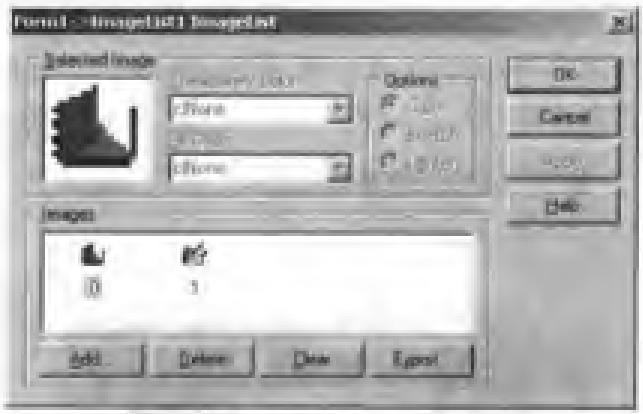


图 31-2

（2）在组件模板的 Standard 标签页中选择 Popupmenu 组件到 Form1 上，双击它，按照图 31-3 所示，增加托盘右键菜单。



图 31-3

（3）在组件模板的 Samples 标签页中选择 TrayIcon 组件到 Form1 上，按照图 31-4 所示设置 TrayIcon 组件的属性。

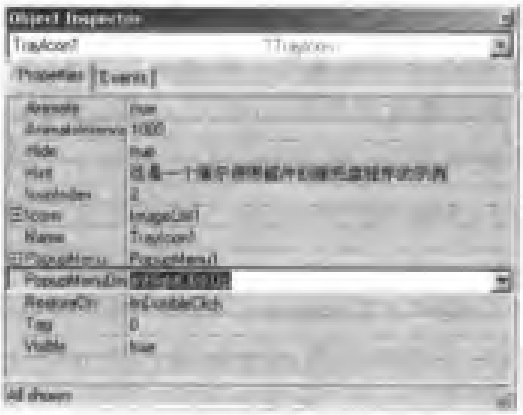


图 31-4

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分　界面设计\实例 031 如何使用组件创建托盘程序”。

实例 032　如何使用动画组件播放动画

实例目的

此实例主要介绍了如何使用动画组件播放 AVI 动画的编程技术。实例程序执行后，效果如图 32-1 所示。



图 32-1

实现方法

在 Windows 操作系统中提供了一个 API 函数专门用于播放短小的 AVI 动画文件，如查找过程 AVI 动画、复制过程 AVI 动画等。C++Builder 也提供了这样一个组件，专门用于播放这种短小的 AVI 动画文件，它既可以以文件方式指定 AVI 动画文件进行播放，也可以使用组件默认提供的系统动画进行播放。在实例程序中，实现使用 TAnimate 动画组件播放 AVI 动画文件的具体步骤如下：

（1）在组件面板的 Win32 标签页下选择 TAnimate 动画组件到 Form1 上。TAnimate 动画组件有一个 CommonAVI 属性，如果该属性设置为 aviNone，那么就要为 FileName 属性指定 AVI 动画文件名称；如果 CommonAVI 属性没有设置为 aviNone，则组件采用该属性值指定的动画文件（实例程序指定的是 aviRecycleFile）进行播放。CommonAVI 属性的取值如下：aviNone（没有指定公共动画文件，动画文件需要另外指定）、aviFindFolder（查找文件夹的动画）、aviFindFile（查找文件的动画）、aviFindComputer（查找计算机的动画）、aviCopyFiles（复制多个文件的动画）、aviCopyFile（复制单个文件的动画）、aviRecycleFile（删除文件到回收站的动画）、aviEmptyRecycle（清空回收站的动画）、aviDeleteFile（删除文件的动画）。

（2）激活 TAnimate 动画组件。TAnimate 动画组件只有在活动状态时才能播放动画。可以在 Object Inspector 属性对话框中设置 Active 属性为 true 来激活 TAnimate 动画组件，也可以按照以下代码方式激活 TAnimate 动画组件。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //开始播放动画
  this->Animate1->Active=true;
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//结束播放动画
  this->Animate1->Active=false;
}
```

最后，编译执行程序即可。此外，需要说明的是，TAnimate 动画组件不适宜播放大型的 AVI 电影文件。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 032 如何使用动画组件播放动画”。

实例 033 如何在状态栏上加载进度条

实例目的

此实例介绍了如何在状态栏上加载进度条的编程技术。实例程序执行后，当单击“启

动状态栏进度条"按钮，在程序状态栏上将立即演示进度条执行情况，效果如图 33-1 所示。



图 33-1

实现方法

在 C++Builder 中，状态条和进度条是作为一个独立的组件使用的，但是完全可以在状态栏上加载进度条。实例程序实现这种技术的示例步骤如下：

（1）从组件面板的 Win32 标签页中选择 TStatusBar 组件到 Form1 上。

（2）双击 Form1 上的 TStatusBar 组件，弹出 Editing StatusBar1->Panels 对话框，单击左上角的 Add New(Ins)按钮，在下面的编辑栏中将显示一个状态栏窗格，如 0-TStatusPannel，选择它，在 Object Inspector 属性对话框中设置其 Text 属性为"显示当前进度："，Width 属性为"100"；再单击左上角的 Add New(Ins)按钮，在下面的编辑栏中将显示一个状态栏窗格，如 1-TStatusPannel，如图 33-2 所示，选择它，在 Object Inspector 属性对话框中设置其 Style 属性为 psOwnerDraw。

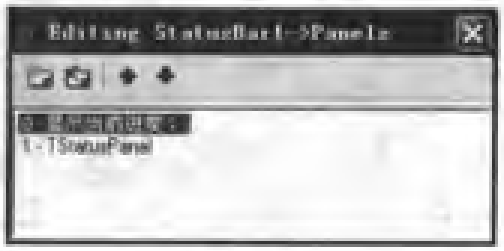


图 33-2

（3）在 Form1 上添加一个按钮，并按照如下所示的代码编写其实现代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //启动状态栏进度条
  TProgressBar *pBar=new TProgressBar(this);
  pBar->Top=2;
  pBar->Left=100;
  pBar->Width=270;
  pBar->Height=15;
  pBar->Visible=true;
  pBar->Parent=this->StatusBar1;
  pBar->Min=0;
```

```
    pBar->Max=270;
    pBar->Step=1;
    for(int i=1;i<pBar->Max;i+=pBar->Step)
    {
      pBar->StepIt();
      Sleep(5);
    }
    delete pBar;
}
```

最后，编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 033 如何在状态栏上加载进度条”。

实例 034 如何在编辑框内显示图像

实例目的

此实例介绍了如何在编辑框内显示图像的编程技术。实例程序执行后，效果如图 34-1 所示。



图 34-1

实现方法

一般情况下，在编辑框内是不能显示图像的，但是如果有这种特别需要，也是可以做到的。可以在取得编辑框（其他组件也可以采取类似的方法实现）的设备上下文后，调用 BitBlt()函数来显示一幅图像。BitBlt()函数的声明如下：

```
BOOL BitBlt(
  int x,              //目标矩形的左上角 X 坐标
  int y,              //目标矩形的左上角 Y 坐标
  int nWidth,         //目标矩形的宽度
  int nHeight,        //目标矩形的高度
```

```
    CDC* pSrcDC,        //源位图设备
    int xSrc,           //拷贝位图的左上角 X 坐标
    int ySrc,           //拷贝位图的左上角 Y 坐标
    DWORD dwRop         //光栅操作码
);
```

实例程序使用该函数在编辑框内显示图像的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  HDC DC=GetDC(this->RichEdit1->Handle);
  BitBlt(DC,0,0,this->Image1->Width,this->Image1->Height,
this->Image1->Picture->Bitmap->Canvas->Handle,0,0,SRCCOPY);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 034 如何在编辑框内显示图像”。

实例 035　如何在列表框中增加水平滚动条

实例目的

此实例介绍了如何在列表框中增加水平滚动条的编程技术。实例程序运行后，效果如图 35-1 所示。



图 35-1

实现方法

通常所见的 C++Builder 中的 ListBox 组件，只有垂直滚动条，没有水平滚动条。然而在列表项目超宽时，这种限制就会给操作带来一定的困难。要解决这一问题，可以通过向组件发送消息的方式实现。实例程序实现在列表框中增加水平滚动条的示例代码如下：

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{//添加水平滚动条
  SendMessage(this->ListBox1->Handle,LB_SETHORIZONTALEXTENT,500,0);
```

```
}
```

其中，500 为水平滚动条的滚动范围，可以通过计算列表项文字的最大长度决定此值。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 035 如何在列表框中增加水平滚动条”。

实例 036 如何在列表框中同时显示图像和文字

实例目的

此实例介绍了如何在列表框中同时显示图像和文字的编程技术。实例程序运行后，效果如图 36-1 所示。



图 36-1

实现方法

在 Windows 操作系统的文件夹浏览器中，文件和文件夹既可以图标形式显示，也可以详细资料等多种形式显示，这样使文件和文件夹在显示时既生动又形象。实际上，用户完全可以在 C++Builder 中使用 ListBox 组件实现类似的功能。具体步骤如下：

（1）在组件面板的 Standard 标签页中选择 ListBox 组件，并设置 ListBox 组件的 Style 属性为 lbOwnerDrawFixed，然后根据实际显示图像的大小设置 ListBox 组件的 ItemHeight 值。

（2）在组件面板的 Additional 标签页中选择 Image 组件，并设置图像，然后设置 Image 组件的 Visiable 属性为 false。根据实际在列表框中显示的列表项的多少决定 Image 组件的个数。

（3）为 ListBox 组件的 OnDrawItem 事件添加如下所示的实现代码。

```
void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
                                         TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBmp;
  int     iPos = 2;   //说明文字显示位置值
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect);
  pBmp = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
```

```
  if (pBmp)
  {
pCanvas->CopyRect(Bounds(Rect.Left + iPos, Rect.Top, pBmp->Width,
pBmp->Height), pBmp->Canvas,Bounds(0, 0, pBmp->Width, pBmp->Height));
    iPos += pBmp->Width + 4;
  }
  pCanvas->TextOut(Rect.Left + iPos, Rect.Top+20,
((TListBox *)Control)->Items->Strings[Index]);
}
```

（4）按照如下所示的代码为 ListBox 组件添加图像和文字列表项。

```
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
  this->ListBox1->Clear();
  this->ListBox1->Items->AddObject("第 1 幅图像",
this->Image1->Picture->Bitmap);
  this->ListBox1->Items->AddObject("第 2 幅图像",
this->Image2->Picture->Bitmap);
  this->ListBox1->Items->AddObject("第 3 幅图像",
this->Image3->Picture->Bitmap);
  this->ListBox1->Items->AddObject("第 4 幅图像",
this->Image4->Picture->Bitmap);
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 036 如何在列表框中同时显示图像和文字”。

实例 037　如何设置属性表组件标签图像

实例目的

通常情况下，属性表标签页标题是没有图像的，此实例主要介绍如何在属性表标签页的标题栏中加入图像，以增强程序的视觉效果。实例程序启动后，显示属性表界面，属性表的每一个标签上分别加载有图像。效果如图 37-1 所示。



图 37-1

实现方法

C++Builder 的属性表组件支持在每一个标签上放置图像。它通过重新处理操作系统 API 消息 TCM_SETIMAGELIST 和 TCM_SETITEM 实现在属性表控件标签上加载图像，主要功能通过处理主表单的 OnPaint 事件实现。实例程序实现这种效果的实现代码如下所示：

```
void __fastcall TForm1::OnPaint(TObject *Sender)
{//主表单 OnPaint 事件
  TabControl1->Perform(TCM_SETIMAGELIST,0,ImageList1->Handle);
  TCITEM tcitem;
  tcitem.mask=TCIF_IMAGE;
  for(int i=0;i<3;i++)
  {
    tcitem.iImage=i;
    TabControl1->Perform(TCM_SETITEM,i,(LPARAM)(LPTCITEM)&tcitem);
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 037 如何设置属性表组件标签图像”。

实例 038 如何控制 Memo 组件的文字边界

实例目的

此实例介绍了如何控制 Memo 组件内的文字边界的编程技术。实例程序执行后，效果如图 38-1 所示。

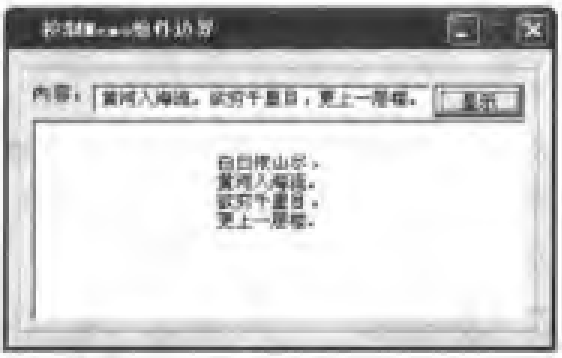


图 38-1

实现方法

通常情况下，C++Builder 中的 Memo 组件的文字是根据默认的窗口自动滚动的，如果需要控制文字在 Memo 组件窗口的指定区域显示，可以采用以下方式实现：首先发送消息 SendMessage(this->Memo1->Handle,EM_GETRECT,0,Longint(&MyRect))取得

Memo 组件的边界值，然后指定区域大小，再将指定区域以消息形式进行发送 SendMessage (this->Memo1->Handle,EM_SETRECT,0,Longint(&MyRect))即可。其中，消息 EM_GETRECT 的定义如下：

```
EM_GETRECT
wParam = 0;                          //保留参数，必须为 0
lParam (LPARAM) (LPRECT) lprc;       //指向 RECT 结构的指针
```

消息 EM_SETRECT 的定义如下：

```
EM_SETRECT
wParam = 0;                          //保留参数，必须为 0
lParam (LPARAM) (LPRECT) lprc;       //指向 RECT 结构的指针
```

以下是实例程序实现控制 Memo 组件文字边界的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  TRect MyRect;
  SendMessage(this->Memo1->Handle,EM_GETRECT,0,Longint(&MyRect));
  MyRect.left=120;
  MyRect.Top=20;
  MyRect.right=MyRect.right-120;
  MyRect.Bottom=MyRect.Bottom-20;
  SendMessage(this->Memo1->Handle,EM_SETRECT,0,Longint(&MyRect));
  this->Memo1->Text=this->Edit1->Text;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 038 如何控制 Memo 组件的文字边界”。

实例 039　如何使用代码控制文字滚动

实例目的

此实例介绍了如何使用代码控制编辑框中的文字实现类似键盘上的 PageUp 和 PageDown 键功能的编程技术。实例程序运行后，单击“文字向上滚动”按钮，编辑框中的文字将向上滚动一页，单击“文字向下滚动”按钮，编辑框中的文字将向下滚动一页，效果如图 39-1 所示。



图 39-1

实现方法

字幕滚动和屏幕图像滚动是多媒体程序经常出现的一种特效，这种特效通常是通过重画屏幕实现的。在此实例中，将通过使用 SendMessage()函数触发键盘的 PageUp 和 PageDown 键消息，实现文字的翻页滚动。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //文字向上滚动
  SendMessage(this->Memo1->Handle,WM_KEYDOWN,VK_PRIOR,0);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //文字向下滚动
  SendMessage(this->Memo1->Handle,WM_KEYDOWN,VK_NEXT,0);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 039 如何使用代码控制文字滚动”。

实例 040 如何纵向显示字符串

实例目的

此实例介绍了如何使字符串纵向显示的编程技术。实例程序执行后，效果如图 40-1 所示。



图 40-1

实现方法

在 C++Builder 中，除了可以通过修改字体的旋转角度和方向实现字符串纵向显示外，也可以通过在字符串中添加换行符的方式实现字符串纵向显示。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  AnsiString StrShow=this->Label1->Caption;
```

```
    AnsiString StrVer="";
    int iLength=StrShow.Length();
    unsigned char StrTemp;
    for(int i=0;i<iLength;i++)
    {
      StrTemp=StrShow.c_str()[i];
      if(StrTemp>0xa0)
      {
        StrVer+=StrShow.SubString(i+1,2)+"\n";
        i++;
      }
      else
      {
        StrVer+=StrShow.SubString(i+1,1)+"\n";
      }
    }
    this->Label1->Caption=StrVer;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分　界面设计\实例040 如何纵向显示字符串”。

实例041　如何旋转显示字符串

实例目的

此实例介绍了如何旋转显示字符串的编程技术。实例程序执行后，单击“旋转显示文字”按钮，将出现如图41-1所示的效果。



图41-1

实现方法

字符串的旋转显示可以通过对字符串文字的字体属性设置实现。字体的属性主要由创

建字体时所使用的 LOGFONT 结构决定，LOGFONT 结构中的 lfEscapement 成员指定了文字行与 X 轴的夹角，通过对夹角的设置就可以实现字符串的旋转。注意：夹角的角度单位是十分之一度而不是通常的 1 度，如 lf.lfEscapement = 450，表示旋转 45 度。LOGFONT 结构的定义如下：

```
typedef struct tagLOGFONT {
   LONG lfHeight;                    //字体高度（逻辑单位）
   LONG lfWidth;                     //字符平均宽度（逻辑单位）
   LONG lfEscapement;                //文本行角度
   LONG lfOrientation;               //字符的角度
   LONG lfWeight;                    //字符粗细度
   BYTE lfItalic;                    //倾斜
   BYTE lfUnderline;                 //下划线
   BYTE lfStrikeOut;                 //删除线
   BYTE lfCharSet;                   //字符集
   BYTE lfOutPrecision;              //字体输出结果和要求的匹配程度
   BYTE lfClipPrecision;             //如何裁剪落于剪裁区之外的字符
   BYTE lfQuality;                   //字符属性匹配的精确程度
   BYTE lfPitchAndFamily;            //字体间距和字体族
TCHAR lfFaceName[LF_FACESIZE];       //字体名称
} LOGFONT, *PLOGFONT;
```

其中，成员 BYTE lfClipPrecision 表示裁剪精度，可能取值包括 OUT_CHARACTER_PRECIS、OUT_DEFAULT_PRECIS、OUT_DEVICE_PRECIS、OUT_OUTLINE_PRECIS、OUT_RASTER_PRECIS、OUT_STRING_PRECIS、OUT_STROKE_PRECIS、OUT_TT_ONLY_PRECIS、OUT_TT_PRECIS。

由于 C++Builder 的 VCL 图形组件封装了 Windows 图形设备接口（GUI），因此许多人可能在 Visual C++程序中经常看到这类实例，但是在 C++Builder 中却很少看到这类实例。实际上，VCL 图形组件设计得的确很优秀，用户只需要明白 VCL 图形组件设计的 3 个层次，就可以使用比 Visual C++更简单的方法实现文字旋转。这 3 个层次分别是：最高层——绘制线条和图形，如：MoveTo，LineTo，Rectangle，Ellipse，TextOut，TextHeight，TextWidth，TextRect，FillRect 和 FloodFill 等文本和图形输出；中层——定制输出的文本和图形，通过控制以下属性：Pen，Brush，Font，Pixels，Draw，StretchDraw，BrushCopy，CopyRect 及 CopyMode 属性复制以及合并图形；底层——直接调用 Windows 的 GDI 函数，VCL 图形组件中提供了一个 Handle 属性，通过它就能控制一切。在 C++Builder 中通过 Handle 句柄传递旋转显示字符串的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  LOGFONT lf;
  memset(&lf,0,sizeof(LOGFONT));
  ::GetObject((HFONT)GetStockObject(DEFAULT_GUI_FONT),sizeof(lf),&lf);
  lf.lfHeight = -14;
  lf.lfWeight = FW_NORMAL;
  lf.lfClipPrecision = CLIP_LH_ANGLES;
  Canvas->Font->Handle = CreateFontIndirect (&lf);
  Canvas->Brush->Style = bsClear;
```

```
    Canvas->Pen->Color=clRed;
    for (int i=0;i<3600;i+=150)
    {
      lf.lfEscapement = i;
      Canvas->Font->Handle = CreateFontIndirect (&lf);
  this->Canvas->TextOutA(this->Width/2, this->Height/2,
  ".........China is a great country");
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例041 如何旋转显示字符串”。

实例042　如何设置显示文字分辨率

实例目的

此实例介绍了如何以不同分辨率显示文字的编程技术。实例程序运行后，单击不同的分辨率按钮，文字将以不同的分辨率显示，效果如图42-1所示。



图42-1

实现方法

通常人们只认为图像才具有分辨率，文字是没有分辨率的。而在C++Builder中，同样字号、同样字体的文字，如果其分辨率设置不同，则显示的文字大小和效果是不一样的。以下是在实例程序中通过改变文字分辨率显示不同效果文字的实现代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //以每英寸48像素显示
  this->Canvas->Font->PixelsPerInch=48;
  this->Canvas->Font->Size=50;
  this->Canvas->TextOutA(30,80,"中华人民共和国万岁");
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //以每英寸96像素显示文字
  this->Canvas->Font->PixelsPerInch=96;
  this->Canvas->Font->Size=50;
```

```
    this->Canvas->TextOutA(30,80,"中华人民共和国万岁");
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//以每英寸 180 像素显示文字
    this->Canvas->Font->PixelsPerInch=180;
    this->Canvas->Font->Size=50;
    this->Canvas->TextOutA(30,80,"中华人民共和国万岁");
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{////以每英寸 360 像素显示文字
    this->Canvas->Font->PixelsPerInch=360;
    this->Canvas->Font->Size=50;
    this->Canvas->TextOutA(30,80,"中华人民共和国万岁");
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{ //以每英寸 720 像素显示文字
    this->Canvas->Font->PixelsPerInch=720;
    this->Canvas->Font->Size=50;
    this->Canvas->TextOutA(30,80,"中华人民共和国万岁");
}
void __fastcall TForm1::BitBtn6Click(TObject *Sender)
{ //以每英寸 1440 像素显示文字
    this->Canvas->Font->PixelsPerInch=1440;
    this->Canvas->Font->Size=50;
    this->Canvas->TextOutA(30,80,"中华人民共和国万岁");
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第一部分 界面设计\实例 042 如何设置显示文字分辨率”。

第二部分　程序设置

实例 043　如何使用消息框

实例目的

此实例主要介绍了如何通过对参数的设置，实现消息框显示风格的多样化。当实例程序启动后，单击“消息框风格一”按钮，弹出一个提示信息框，单击该信息框的一个按钮，程序标题栏将显示刚才单击的按钮名称，单击其他按钮，将弹出与此不同风格的消息框，效果如图 43-1 所示。



图 43-1

实现方法

快速显示不同风格的消息框是编写程序时需要掌握的一个编程技巧。它经常用于提示用户或请用户作出选择，以决定下一步的操作。由 Windows 操作系统默认提供的快速显示消息框函数 MessageBox()，其风格由图标类型和按钮类型组合而成，可以使用按位运算符“|”或者加号运算符“+”实现图标和按钮的组合。消息框的图标风格包括 MB_ICONHAND、MB_ICONSTOP、MB_ICONINFORMATION、MB_ICONQUESTION、MB_ICONEXCLAMATION、MB_ICONWARNING、MB_ICONASTERISK 和 MB_ICONERROR。消息框的按钮风格包括 MB_OK（确定）、MB_ABORTRETRYIGNORE（放弃，重试，忽略）、MB_OKCANCEL（确定，取消）、MB_RETRYCANCEL（重试，取消）、MB_YESNO（是，否）、MB_YESNOCANCEL（是，否，取消）等。消息框的返回值包括 IDABORT（中断）、IDCANCEL（取消）、IDIGNORE（忽略）、IDNO（否）、IDOK（确定）、IDRETRY（重试）、IDYES（是）。函数 MessageBox()的声明如下：

```
int MessageBox(
    HWND hWnd,                  //窗口句柄
    LPCTSTR lpszText,           //消息字符串
```

```
    LPCTSTR lpszCaption = NULL,    //消息框标题
    UINT nType = MB_OK             //消息框风格
);
```

实例程序使用函数 MessageBox()显示不同风格消息框的示例代码如下：

```
AnsiString __fastcall TForm1::GetMessageBoxReturn(int i)
{//获取消息框返回值
  AnsiString StrRtn="";
  switch(i)
  {
  case IDABORT:
    StrRtn="中断";
    break;
  case IDCANCEL:
    StrRtn="取消";
    break;
  case IDIGNORE:
    StrRtn="忽略";
    break;
  case IDNO:
    StrRtn="否";
    break;
  case IDOK:
    StrRtn="确定";
    break;
  case IDRETRY:
    StrRtn="重试";
    break;
  case IDYES:
    StrRtn="是";
    break;
  }
  return StrRtn;
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"需要退出程序吗？","消息框风格一",MB_YESNOCANCEL);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"干得好！","消息框风格二",MB_ICONWARNING);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"请看帮助文件","消息框风格三",MB_ICONQUESTION);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
```

```
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"操作违规","消息框风格四",MB_ICONERROR);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"请看帮助文件","消息框风格五",MB_HELP);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
}
void __fastcall TForm1::BitBtn6Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"刚才的操作正确吗？","消息框风格六",MB_OKCANCEL);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
}
void __fastcall TForm1::BitBtn7Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"是否重试一次？","消息框风格七",MB_RETRYCANCEL);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
}
void __fastcall TForm1::BitBtn8Click(TObject *Sender)
{
  int i=0;
  i=MessageBox(NULL,"是否重试一次？","消息框风格八",MB_YESNO);
  this->Caption="您刚才选择了'"+this->GetMessageBoxReturn(i)+"'按钮";
}
```

此外，C++Builder 还提供了 MessageDlg()函数和 ShowMessage()函数来快速显示消息框，MessageDlg()函数的声明如下：

```
extern PACKAGE int __fastcall MessageDlg(const AnsiString Msg,
TMsgDlgType DlgType, TMsgDlgButtons Buttons, int HelpCtx);
```

其中，参数 TMsgDlgType DlgType 的可能值包括 mtWarning（显示警告信息）、mtError（显示错误信息）、mtInformation（显示说明信息）、mtConfirmation（显示确认信息）、mtCustom（显示自定义信息）。参数 TMsgDlgButtons Buttons 的可能值包括 mbOK（显示“确定”按钮）、mbCancel（显示“取消”按钮）、mbYes（显示“是”按钮）、mbNo（显示“否”按钮）、mbAbort（显示“中断”按钮）、mbRetry（显示“重试”按钮）、mbIgnore（显示“忽略”按钮）、mbAll（显示“全部”按钮）等。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 *程序设置*\实例 043 如何使用消息框”。

实例 044 如何显示简单关于对话框

实例目的

此实例主要介绍了如何通过一个非常简单的方法实现显示比较漂亮的关于对话框（该关于对话框可显示程序图标和系统资源使用情况等）的编程技术。实例程序执行后，效果如图 44-1 所示。



图 44-1

实现方法

一个完整的 Windows 应用程序通常都包含一个“关于”对话框，以显示应用程序版本信息等。在大多数情况下，程序员是通过定制一个对话框作为“关于”对话框，这样的“关于”对话框能够显示更多信息，甚至可包括超链接等，具有较强的灵活性。但如果只需要显示应用程序的简单版本信息，Windows Shell Library 函数 ShellAbout()就已足够了。ShellAbout()函数的声明如下：

```
int ShellAbout(
HWND hWnd,                    //父窗口句柄
LPCTSTR szApp,                //标题栏和第一行文字内容
LPCTSTR szOtherStuff,         //其他对话框文字
HICON hIcon                   //显示图标
);
```

实例程序使用函数 ShellAbout()制作简单关于对话框的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//显示简单关于对话框
   AnsiString StrTitle=Application->Title;
   AnsiString StrContent="唯佳全球有限公司，版权所有\n2001-2005";
   ShellAbout(Handle,StrTitle.c_str(),
StrContent.c_str(),Application->Icon->Handle);
```

}

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分　程序设置\实例 044 如何显示简单关于对话框”。

实例 045　如何在程序中增加热键

实例目的

此实例介绍了如何在程序中增加热键的编程技术。实例程序执行后，效果如图 45-1 所示。



图 45-1

实现方法

在 C++Builder 中，在应用程序中增加热键通常要经过以下 3 个过程：①向系统注册热键；②处理热键消息响应函数；③程序结束时向系统注销热键。以下是在实例程序中增加热键的主窗口的头文件和实现文件的具体内容。

```
//filename Mainform.h
#ifndef MainformH
#define MainformH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TImage *Image1;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
private:        // User declarations
```

```
public:         // User declarations
    __fastcall TForm1(TComponent* Owner);
    BEGIN_MESSAGE_MAP
      MESSAGE_HANDLER(WM_HOTKEY,TMessage,OnHotKey)
    END_MESSAGE_MAP(TForm)
    void __fastcall TForm1::OnHotKey(TMessage &Message); //响应热键消息
};
extern PACKAGE TForm1 *Form1;
#endif

//filename:Mainform.cpp
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
void __fastcall TForm1::FormCreate(TObject *Sender)
{  //注册系统热键
   if(!RegisterHotKey(Handle,2000,MOD_CONTROL,VK_F1))
        //这里注册的热键是 Ctrl+F1
        ShowMessage("热键注册失败");
}
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{//退出程序时注销热键
   UnregisterHotKey(Handle, 2000);
}
void __fastcall TForm1::OnHotKey(TMessage &Message)
{
    TForm::Dispatch(&Message);
    MessageBox(Handle,"这是通过系统热键产生的信息！","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 045 如何在程序中增加热键”。

实例 046 如何在程序中自定义消息

实例目的

此实例主要介绍了如何在程序中创建自定义消息，并通过自定义消息调用实现程序功

能的编程技术。实例程序独立执行后，效果如图 46-1 所示。



图 46-1

实现方法

Windows 应用程序是基于消息与事件的应用程序，其大部分功能实现均是通过消息触发而完成的。在一个 Windows 应用程序中，除了系统消息（如键盘消息、鼠标消息、定时器消息等）外，用户也可以在程序中自定义特定的消息。C++Builder 消息映射的定义形式如下：

```
BEGIN_MESSAGE_MAP
 MESSAGE_HANDLER(< message >,< message structure >,< message handler >)
 MESSAGE_HANDLER(< message >,< message structure >,< message handler >)
 ...
END_MESSAGE_MAP(ClassName)
```

其中，ClassName 是基础类名，message 参数是要捕获的 Windows 消息，message structure 参数用于传递 VCL 剖析消息后用于传递参数的消息剖析结构名，其中包含处理消息时所需要的全部参数，不同的消息其消息结构是不同的，一般性的消息结构定义如下：

```
struct TMessage {
        unsigned int Msg;          //消息
        int WParam;                //字参数
        int LParam;               //长字参数
        int Result;                //消息结果
    };
```

一般情况下，在应用程序中创建自定义消息包括以下步骤：

（1）定义消息。由于消息代号实际上就是一个整型数字，并且其中有部分数字已经被系统占用，因此默认情况下，应该在 WM_USER 这个宏的基础上增加消息定义，否则可能会产生冲突。示例如下：

```
#define WM_MYCUSTOMMESSAGE (WM_USER+1)
```

（2）定义消息映射。消息必须与消息实现代码关联起来，才能实现消息所触发的功能。消息映射通常通过消息映射宏实现。代码如下所示：

```
BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(WM_MYCUSTOMMESSAGE, TMessage, ChangePicture)
```

```
END_MESSAGE_MAP(TForm)
```

（3）定义消息映射函数和触发消息。触发消息有两个函数：PostMessage()和SendMessage()，其中，PostMessage()函数会立即返回一个表示消息是否发送成功的BOOL值。下面是实例程序实现整个自定义消息的完整过程代码。

```
//主窗体类头文件
#ifndef MainformH
#define MainformH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#define WM_MYCUSTOMMESSAGE (WM_USER+1)
class TForm1 : public TForm
{
__published:   // IDE-managed Components
        TImage *Image1;
        TBitBtn *BitBtn1;
        void __fastcall BitBtn1Click(TObject *Sender);
private:       // User declarations
public:        // User declarations
        bool bChange;
        __fastcall TForm1(TComponent* Owner);
        void __fastcall ChangePicture(TMessage Message);
        //消息映射宏
        BEGIN_MESSAGE_MAP
        MESSAGE_HANDLER(WM_MYCUSTOMMESSAGE, TMessage, ChangePicture)
        END_MESSAGE_MAP(TForm)
};
extern PACKAGE TForm1 *Form1;
#endif
//主窗体类实现文件
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  bChange=false;
}
void __fastcall TForm1::ChangePicture(TMessage Message)
{//消息响应函数
  if(!bChange )
  {
```

```
    this->Image1->Picture->LoadFromFile('luobin02.bmp');
    bChange=!bChange;
   }
   else
   {
    this->Image1->Picture->LoadFromFile('luobin01.bmp');
    bChange=!bChange;
   }
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//发送消息
//   SendMessage(Handle,WM_MYCUSTOMMESSAGE,NULL,NULL);
  PostMessage(Handle,WM_MYCUSTOMMESSAGE,NULL,NULL);
}
```

说明：自定义的消息处理函数末尾最好加一句 TForm::Dispatch(&Message)，其作用是让消息继续传递下去。如果没有这一句，消息将被完全拦截，VCL 类可能由于得不到消息而无法实现正常功能。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 046 如何在程序中自定义消息”。

实例 047　如何在程序中自定义系统菜单

实例目的

此实例主要介绍了如何在程序的系统菜单中添加自定义菜单的编程技术。实例程序执行后，右击程序标题栏，弹出一个下拉菜单，选择“通过系统菜单切换图片”命令，显示图片将被交换，效果如图 47-1 所示。



图 47-1

实现方法

毫无疑问，C++Builder 是快速开发 Windows 应用程序的利器，特别是信息管理程序。但是，也可能在编程过程中经常发现它的一些局限性。例如，对应用程序系统菜单的事件维护。其实，在 Windows API 编程中，特别是在 Visual C++中，对应用程序系统菜单的处理是一件非常简单的事情，但是在 C++Builder 中，这一过程却变得相对复杂起来。因为所有消息定义、消息映射都要由用户亲自处理。下面是在应用程序中添加系统菜单的示例步骤。

（1）定义消息。示例如下：

```
#define WM_MYCUSTOMMESSAGE (WM_USER+1)
```

（2）定义消息映射。如下面的代码所示：

```
BEGIN_MESSAGE_MAP
MESSAGE_HANDLER(WM_MYCUSTOMMESSAGE, TMessage, ChangePicture)
END_MESSAGE_MAP(TForm)
```

（3）定义消息响应函数。

（4）在窗体创建时添加系统菜单（OnCreate 事件）。

下面是实例程序实现以上步骤的所有代码。

```
//主窗体类头文件
#ifndef MainformH
#define MainformH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
//消息定义
#define WM_MYCUSTOMMESSAGE (WM_USER+1)
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TImage *Image1;
        void __fastcall OnCreate(TObject *Sender);
private:        // User declarations
public:         // User declarations
        bool bChange;
        __fastcall TForm1(TComponent* Owner);
        void __fastcall ChangePicture(TMessage Message);
        //消息映射宏
        BEGIN_MESSAGE_MAP
        MESSAGE_HANDLER(WM_SYSCOMMAND,TMessage,ChangePicture)
        END_MESSAGE_MAP(TForm)
};
extern PACKAGE TForm1 *Form1;
#endif
//主窗体类实现文件
```

```
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  bChange=false;
}
void __fastcall TForm1::ChangePicture(TMessage Message)
{//消息响应函数
   if(Message.WParam==WM_MYCUSTOMMESSAGE)
   {
     if(!bChange )
     {
      this->Image1->Picture->LoadFromFile("luobin02.bmp");
      bChange=!bChange;
     }
     else
     {
       this->Image1->Picture->LoadFromFile("luobin01.bmp");
       bChange=!bChange;
       }
   }
   TForm::Dispatch(&Message);
}
void __fastcall TForm1::OnCreate(TObject *Sender)
{//在系统菜单中添加新菜单项
  HMENU hMenu=GetSystemMenu(Handle,false);
  AppendMenu(hMenu,MF_STRING,WM_MYCUSTOMMESSAGE,"通过系统菜单切换图片");
}
```

AppendMenu()是一个系统 API 函数，该函数的声明如下：

```
BOOL AppendMenu(
    HMENU hMenu,                    // 要改变的菜单句柄
    UINT uFlags,                    // 菜单对象标志
    UINT uIDNewItem,                // 菜单标识
LPCTSTR lpNewItem                   //菜单内容
);
```

其中，参数 UINT nFlags 指定增加到菜单中的新菜单项状态的有关信息，可能值包括：MF_CHECKED（替换菜单项前的检测标记）、MF_UNCHECKED（删除菜单项前的检测标记）、MF_DISABLED（使菜单项无效，使它不能被选择，但菜单项不变灰）、MF_ENABLED（使菜单项有效，使它能够被选择，并从灰色状态中恢复原样）、MF_GRAYED（使菜单项无效，使它不能被选择，同时使菜单项变灰）、MF_MENUBARBREAK（在静态菜单里的新行中或弹出菜单的新列中放置菜单项，新的弹出菜单列与老的菜单列将由垂直分割线分开）、MF_MENUBREAK（在静态菜单里的新行中或弹出菜单的新列中放置菜单项，列

与列之间没有分割线）、MF_OWNERDRAW（指定菜单项为一个拥有者描绘的项，该选择项对于顶层菜单项无效）、MF_POPUP（指定菜单项具有与之相关联的弹出菜单，参数 UINT_PTR nIDNewItem 指定了与菜单项相关联的弹出菜单的句柄，它用于增加顶层弹出菜单项或用于增加弹出菜单项的下一级弹出菜单）、MF_SEPARATOR（绘制一条水平的分割线，它仅能用于弹出菜单项，该线不能变灰、无效或高亮度显示，其他的参数将被忽略）、MF_STRING（指定菜单项为一字符串）。

参数 UINT_PTR nIDNewItem 指定了新菜单项的命令 ID 号，或如果参数 UINT nFlags 设置为 MF_POPUP，该参数指定弹出菜单的菜单句柄 HMENU。如果参数 UINT nFlags 设置为 MF_SEPARATOR，那么参数 UINT_PTR nIDNewItem 将被忽略。

参数 LPCTSTR lps2NewItem 指定了新菜单项的内容。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 047 如何在程序中自定义系统菜单”。

实例 048 如何在执行程序间进行数据通信

实例目的

此实例介绍了如何通过内存映射在两个执行程序间进行数据通信的编程技术。实例程序执行后（启动此实例程序两次），效果如图 48-1 所示。



图 48-1

实现方法

在 C++Builder 中，在程序间进行数据通信通常使用 DDE 数据通信组件，但是也可以通过内存映射机制实现两个执行程序间的数据通信。通常将文件映射到内存需要 CreateFileMapping()函数和 MapViewOfFile()函数的支持，这两个函数的声明如下：

```
HANDLE CreateFileMapping(
  HANDLE hFile,                          // 映射文件句柄
  LPSECURITY_ATTRIBUTES lpAttributes,    // 安全描述符
```

```
  DWORD flProtect,                    // 保护方式
  DWORD dwMaximumSizeHigh,            // 文件映射对象的最大长度的高 32 位
  DWORD dwMaximumSizeLow,             // 文件映射对象的最大长度的低 32 位
  LPCTSTR lpName                      // 文件映射名
);
LPVOID MapViewOfFile(
  HANDLE hFileMappingObject,          // 映射文件对象句柄
  DWORD dwDesiredAccess,              // 访问模式
  DWORD dwFileOffsetHigh,             // 文件偏移地址的高 32 位
  DWORD dwFileOffsetLow,              // 文件偏移地址的低 32 位
  SIZE_T dwNumberOfBytesToMap         // 映射视图的大小
);
```

实例程序实现在两个执行程序间进行数据通信的具体代码如下所示：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//发送数据
  if(this->Memo1->Text.Length()<1)
     return;
  //创建文件映射对象
  HANDLE hMapping;
  LPSTR StrData;
  hMapping=CreateFileMapping((HANDLE)0xFFFFFFFF,NULL,
PAGE_READWRITE,0,0x100,"COMMUNICATION");
  if(hMapping==NULL)
  {
     MessageBox(Handle,"创建文件映射对象失败！","信息提示",MB_OK);
     return;
  }
  //将文件映射到一个进程的地址空间上
  StrData=(LPSTR)MapViewOfFile(hMapping,FILE_MAP_ALL_ACCESS,0,0,0);
  if(StrData==NULL)
  {
     MessageBox(Handle,"文件映射失败","信息提示",MB_OK);
     return;
  }
  //向映射内存写数据
  strcpy(StrData,this->Memo1->Text.c_str());
  //释放映射内存
  UnmapViewOfFile(StrData);
  MessageBox(Handle,"向接收程序发送数据操作成功！","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//接收数据
  //创建文件映射对象
  HANDLE hMapping;
  LPSTR StrData;
  hMapping=CreateFileMapping((HANDLE)0xFFFFFFFF,NULL,
PAGE_READWRITE,0,0x100,"COMMUNICATION");
  if(hMapping==NULL)
  {
    MessageBox(Handle,"创建文件映射对象失败！","信息提示",MB_OK);
```

```
    return;
  }
  //将文件映射到一个进程的地址空间上
  StrData=(LPSTR)MapViewOfFile(hMapping,FILE_MAP_ALL_ACCESS,0,0,0);
  if(StrData==NULL)
  {
    MessageBox(Handle,"文件映射失败","信息提示",MB_OK);
    return;
  }
  //获取映射内存的数据
  this->Memo2->Text=StrData;
  //释放映射内存
  UnmapViewOfFile(StrData);
  MessageBox(Handle,"接收发送程序发来的数据操作成功！","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 048 如何在执行程序间进行数据通信”。

实例 049 如何使用 OLE 技术启动画图

实例目的

此实例介绍了如何使用 OLE 技术启动画图的编程技术。实例程序运行后，效果如图 49-1 所示。



图 49-1

实现方法

OLE（Object Linking and Embedding）对象的链接与嵌入技术，是微软早在 20 世纪 90

年代初推出的技术。OLE 通常分为 OLE 服务程序和 OLE 客户程序，如操作系统自带的画图程序就是一个典型的 OLE 服务程序，而 Word 则是一个同时具有 OLE 服务和客户功能的程序。C++Builder 在组件模板的 System 标签页下提供了一个 OleContainer OLE 容器组件，就是一个 OLE 客户程序模板，它能够调用许多 OLE 服务程序。同时，OleContainer 组件提供了文件和流的存取功能，它能够将服务程序的结果保存为.ole 扩展名的文件，供以后反复使用。其包括的主要函数有：

（1）void __fastcall CreateObjectFromFile(AnsiString FileName, bool Iconic)。该函数能够读取指定的文件，产生一个嵌入的对象，参数 AnsiString FileName 指定文件名，参数 bool Iconic 指示 OLE 对象是否用图标显示。

（2）void __fastcall LoadFromFile(AnsiString FileName)。该函数从指定的文件载入 OLE 对象，参数 AnsiString FileName 为文件名。

（3）void __fastcall LoadFromStream(Classes::TStream* Stream)。该函数从指定的流中载入 OLE 对象，参数 Classes::TStream* Stream 为指定的流对象。

（4）void __fastcall SaveToFile(const AnsiString FileName)。该函数保存 OLE 对象到指定的文件中，参数 const AnsiString FileName 为保存的 OLE 文件名。

（5）void __fastcall SaveToStream(Classes::TStream* Stream)。该函数保存 OLE 对象到指定的流中，参数 Classes::TStream* Stream 为流对象。

关于 OLE 的文件保存与打开问题可以作如下解释：OLE 文件是一个整体文件，它不会因为服务程序是画图，则文件类型就是画图文件；服务程序是 Word，则文件类型就是 Word 文档；对 OLE 内容的编辑是所链接的另一个 OLE 服务程序的功能；OLE 整体是一个数据包，可以作为数据流或文件保存与读取，也可以被其他应用程序使用。

以下是实例程序使用 OleContainer 容器组件调用操作系统自带的画图程序的具体步骤：

（1）在组件模板的 System 标签页下选择 OleContainer 容器组件到 Form1 上，设置其 AutoActivate 为 aaGetFocus。然后双击 OleContainer 容器组件，弹出“插入对象”对话框，在“对象类型”列表中选择“画笔图片”，如图 49-2 所示。单击“确定”按钮返回。



图 49-2

（2）在组件模板的 Standard 标签页下选择 Mainmenu 组件到 Form1 上，保留所有默

认设置，以装入画图软件的菜单。

（3）按照如下所示的代码添加 OleContainer 容器组件的文件装入代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//浏览 OLE 文件
  if(this->OpenDialog1->Execute())
  {
    AnsiString StrFileName=this->OpenDialog1->FileName;
    this->OleContainer1->LoadFromFile(StrFileName);
    this->OleContainer1->Run();
    this->OleContainer1->SetFocus();
  }
}
```

（4）按照如下所示的代码添加 OleContainer 容器组件的文件保存代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//保存 OLE 文件
  if(this->SaveDialog1->Execute())
  {
    AnsiString StrFileName=this->SaveDialog1->FileName;
    this->OleContainer1->SaveToFile(StrFileName+".ole");
  }
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 049 如何使用 OLE 技术启动画图”。

实例 050 如何在程序中添加 Word 文档表格

实例目的

此实例介绍了如何在程序中调用 Word 绘制表格的方法，其中包括表格的生成、单元格的控制与操作的编程技术。实例程序执行后，效果如图 50-1 所示。



图 50-1

实现方法

C++Builder 的 WordDocument 组件能够实现 Word 文档的大多数功能，它不仅能够定制文字，也能够定制表格，包括单元格的合并和拆分等。以下是使用组件模板 Office2K 下的 WordApplication 和 WordDocument 两个组件实现在 Word 文档中添加表格的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  try
  {
     //连接 Word
     WordApplication1->Connect();
     //建立一个新文档
     WordApplication1->Documents->Add();
     //将 WordDocument 对象与 WordApplication 对象进行连接
     WordDocument1->ConnectTo(WordApplication1->Documents->Item(Variant(1)));
     //在 Word 中添加一个 4*8 表格
     WordDocument1->Tables
->Add(WordDocument1->Range(EmptyParam,EmptyParam),4,8);
     //在 Word 文档末尾添加一个 4*8 表格
     //WordDocument1->Tables->Add(WordDocument1->Range(Variant(WordDocument1
->Range(EmptyParam,EmptyParam)->End-1),EmptyParam),4,8);
     //第 3 行第 5 列的单元格需要插入一段文本
     WordDocument1->Tables->Item(Variant(1))->Cell(Variant(3),
Variant(5))->Range->set_Text(WideString("单元格内容"));
     //将表格中的(1,1)与(1,2)单元格合并
     WordDocument1->Tables->Item(Variant(1))->Cell(Variant(1),Variant(1))
->Merge(WordDocument1->Tables->Item(Variant(1))->Cell(Variant(1),
Variant(2)));
     //将表格中的(3,5)单元格拆分为 2 行 3 列的单元格
     WordDocument1->Tables->Item(Variant(1))->Cell(Variant(3),
Variant(5))->Split(Variant(2),Variant(3));
     WordApplication1->set_Visible(true);
  }
  catch(...)
  {
    MessageBox(Handle,"系统可能没有安装 Word!","信息提示",MB_OK);
    WordApplication1->Disconnect();
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 050 如何在程序中添加 Word 文档表格”。

实例 051 如何在程序中设置 Word 文档的字体风格

实例目的

此实例介绍了如何在程序中使用 OLE 技术设置 Word 文档的字体风格的编程技术。实例程序执行后，效果如图 51-1 所示。

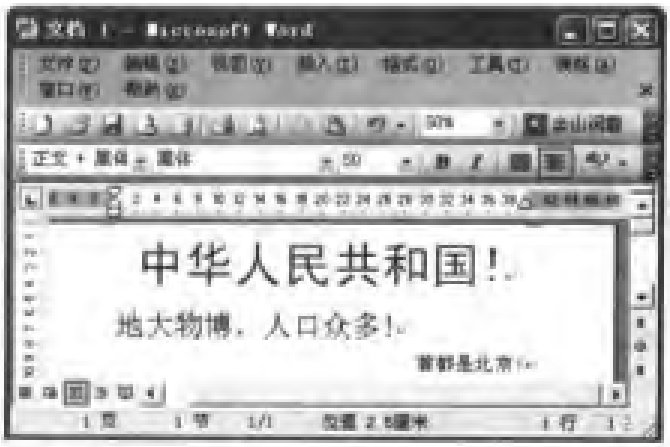


图 51-1

实现方法

在 C++Builder 组件模板 Office2K 标签页下，系统提供了 6 个 Word 组件控制对 Word 文档的操作：WordGlobal（对 Word 的最高级属性和方法进行调用）；WordApplication（Word 应用程序组件，完成对 Word 应用程序的调用，是实现 Word 调用的核心组件）；WordDocument（Word 文档处理组件，用于处理文档的具体内容）；WordFont（用于设置 Word 中的字体、字号等文字设置信息）；WordParagraphFormat（用于对 Word 中的段落格式进行处理）；WordLetterContent（对由信函向导创建的信函元素进行调用的组件）。以下是实例程序使用组件模板 Office2K 标签页下的 WordApplication 和 WordDocument 组件实现定制 Word 文档字体的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  try
  {  //连接 Word
     WordApplication1->Connect();
     //建立一个新文档
     WordApplication1->Documents->Add();
     //将 WordDocument 对象与 WordApplication 对象进行连接
     WordDocument1->ConnectTo(WordApplication1->Documents->Item(Variant(1)));
     //添加一个文字段
     WordDocument1->Paragraphs->Add();
     //设置该段文本字体
```

```
        WordDocument1->Paragraphs->Item(Variant(1))
    ->Range->Font->set_Name(WideString("黑体"));
        //设置字号
        WordDocument1->Paragraphs->Item(Variant(1))->Range->Font->Size=50;
        //设置文字颜色
        WordDocument1->Paragraphs->Item(Variant(1))
    ->Range->Font->Color=RGB(0,0,200);
        //设置该段文字内容
        WordDocument1->Paragraphs->Item(Variant(1))
    ->Range->set_Text(WideString("中华人民共和国!"));
        //设置该段文字居中
        WordDocument1->Paragraphs->Item(Variant(1))->Alignment=1;
        //添加一个文字段
        WordDocument1->Paragraphs->Add();
        //设置该文字段字体
        WordDocument1->Paragraphs->Item(Variant(2))
    ->Range->Font->set_Name(WideString("宋体"));
        //设置字号
        WordDocument1->Paragraphs->Item(Variant(2))->Range->Font->Size=30;
        //设置该文字段内容
        WordDocument1->Paragraphs->Item(Variant(2))
    ->Range->set_Text(WideString("地大物博，人口众多!"));
        //设置该段文字居左
        WordDocument1->Paragraphs->Item(Variant(2))->Alignment=0;
        //添加一个文字段
        WordDocument1->Paragraphs->Add();
        //设置该段文本字体
        WordDocument1->Paragraphs->Item(Variant(3))
    ->Range->Font->set_Name(WideString("宋体"));
        //设置字号
        WordDocument1->Paragraphs->Item(Variant(3))->Range->Font->Size=20;
        //设置当前段文字内容
        WordDocument1->Paragraphs->Item(Variant(3))
    ->Range->set_Text(WideString("首都是北京!"));
        //设置该段文字居右
        WordDocument1->Paragraphs->Item(Variant(3))->Alignment=2;
        WordApplication1->set_Visible(true);
      }
      catch(...)
      {
        MessageBox(Handle,"系统可能没有安装 Word!","信息提示",MB_OK);
        WordApplication1->Disconnect();
      }
    }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 051 如何在程序中设置 Word 文档的字体风格”。

实例 052　如何使用 OnIdle 事件

实例目的

此实例主要介绍了不使用定时器，而使用 OnIdle 事件随时监视程序状态变化的技术，当系统剪贴板有文本内容时，“粘贴”按钮处于活动状态，否则处于禁止状态；当编辑栏中的内容有变化时，“保存文件”按钮处于活动状态，否则处于禁止状态，以及对“复制”、“剪切”按钮状态的监视等。实例程序执行后，效果如图 52-1 所示。



图 52-1

实现方法

OnIdle 事件是大多数 Windows 应用程序的属性，它用于处理 Windows 应用程序在空闲状态时需要处理的事情。在 C++Builder 中，它的使用步骤如下：

（1）在窗体的 OnCreate()事件处理函数中声明 OnIdle 事件的对应函数，代码如下所示：

```
void __fastcall TForm1::OnCreate(TObject *Sender)
{
  Application->OnIdle=OnIdle;
}
```

（2）为 OnIdle 事件的对应函数添加实现代码，如下所示：

```
void __fastcall TForm1::OnIdle(TObject* Sender,bool& Done)
{
  if(this->Memo1->SelLength>0)
  {  //允许使用“复制”按钮
     this->BitBtn5->Enabled=true;
     //允许使用“剪切”按钮
     this->BitBtn6->Enabled=true;
  }
  else
  {  //禁止使用“复制”按钮
```

```
        this->BitBtn5->Enabled=false;
        //禁止使用“剪切”按钮
        this->BitBtn6->Enabled=false;
    }
    if(this->Memo1->Modified)
    {  //允许使用“保存”按钮
       this->BitBtn3->Enabled=true;
    }
    else
    {  //禁止使用“保存”按钮
       this->BitBtn3->Enabled=false;
    }
    if(this->Memo1->Lines->Count>0)
    { //允许使用“全选”按钮
      this->BitBtn8->Enabled=true;
    }
    else
    { //禁止使用“全选”按钮
      this->BitBtn8->Enabled=false;
    }
    if(Clipboard()->HasFormat(CF_TEXT))
    { //允许使用“粘贴”按钮
      this->BitBtn7->Enabled=true;
    }
    else
    { //禁止使用“粘贴”按钮
      this->BitBtn8->Enabled=false;
    }
}
```

最后，编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 052 如何使用 OnIdle 事件”。

实例 053　如何使程序在循环时响应外界事件

实例目的

此实例介绍了如何使程序在循环时响应外界事件的编程技术。实例程序运行后，单击“循环时不响应外界事件”按钮，程序在循环时不能执行拖动等操作；单击“循环时响应外界事件”按钮，程序在循环时能够执行拖动等操作，效果如图 53-1 所示。



图 53-1

实现方法

在 C++Builder 的 Application 对象中有一个 ProcessMessage()方法，它能够中断应用程序的执行，以便能够响应外界事件和消息。此技术对于解决循环的独占问题特别有用。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //循环时不响应外界事件
  int x, y;
  for (int i = 0; i < 24000; i++)
  {
    Randomize();
    for (int j = 0; j < 24000; j++)
      y = random(j);
    x = random(i);
  }
  MessageBox(Handle,'循环结束！','信息提示',MB_OK);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//循环时响应外界事件
  int x, y;
  for (int i = 0; i < 24000; i++)
  {
    Randomize();
    for (int j = 0; j <24000; j++)
    {
      y = random(j);
      Application->ProcessMessages();
    }
    x = random(i);
  }
  MessageBox(Handle,'循环结束！','信息提示',MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序

设置\实例 053 如何使程序在循环时响应外界事件”。

实例 054　如何设置程序在系统启动时自动运行

实例目的

此实例主要介绍了如何设置程序在系统启动时自动运行的编程技术。实例程序执行后，单击“浏览执行文件”按钮选择一个执行文件后，再单击“设置为启动自运行程序”按钮，重新启动计算机，可以发现，选择的执行文件在系统启动时自动运行，效果如图 54-1 所示。



图 54-1

实现方法

Windows 启动时通常会有许多程序自动运行。普通用户可以在“开始/程序/启动”菜单中加入自己的系统启动时自动运行的程序。实际上，让 Windows 系统启动时程序自动运行的途径很多，通常有两个文件夹和 8 个注册表子键可以设置程序在系统启动时自动运行。

两个文件夹分别是：①当前用户专有的启动文件夹。这是许多应用软件自动运行的常用位置。Windows 自动启动放入该文件夹的所有快捷方式。用户启动文件夹一般在\Documents and Settings\用户名字\开始菜单\程序\启动，其中“用户名字”是当前登录的用户账户名称。②对所有用户有效的启动文件夹。这是寻找自动运行程序的第二个重要位置，不管用户用什么身份登录系统，放入该文件夹的快捷方式总是自动运行。该文件夹一般在\Documents and Settings\All Users\开始菜单\程序\启动。

8 个系统启动时自动运行程序的注册表专用子键分别是：①Load 子键。Load 子键位于 HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion \Windows\load 。②Userinit 子键。Userinit 子键位于 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\Current Version\Winlogon\ Userinit。通常该子键下有一个 userinit.exe，但该子键允许指定用逗号分隔多个程序，例如：“userinit.exe,OSA.exe”（不含引号）。③Explorer\Run 子键。与 load、Userinit 子键不同，Explorer\Run 子键在 HKEY_CURRENT_USER 和 HKEY_LOCAL_MACHINE 下都有，具体位置在 HKEY_CURRENT_USER\Software\

Microsoft\Windows\CurrentVersion\Policies\Explorer\Run 和 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run。④RunOnce\Setup 子键。RunOnce\Setup 子键指定了用户登录之后运行的程序，它的位置是 HKEY_CURRENT_USER\Software\Microsoft\ Windows\CurrentVersion\ RunOnce\Setup 和 HKEY_LOCAL_MACHINE\Software\Microsoft\ Windows\CurrentVersion\ RunOnce\Setup。⑤RunOnce 子键。需要在安装过程中重新启动计算机的安装程序，通常使用 RunOnce 子键自动运行程序。它的位置在 HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Windows\CurrentVersion\ RunOnce 和 HKEY_CURRENT_USER\ Software\Microsoft\Windows\CurrentVersion\RunOnce。HKEY_ LOCAL_MACHINE 下面的 RunOnce 子键会在用户登录之后立即运行程序，其系统启动自动运行程序的运行时机在其他 Run 子键指定的程序之前。HKEY_CURRENT_USER 下面的 RunOnce 子键的系统启动自动运行程序在操作系统处理其他 Run 子键以及“启动”文件夹的内容之后运行。如果是 Windows XP 操作系统，还需要检查一下 HKEY_LOCAL_MACHINE\SOFTWARE\ Microsoft\Windows\CurrentVersion\RunOnceEx 子键。⑥CurrentVersion\Run 子键。CurrentVersion\Run 子键是自动运行程序最常用的子键，位置在 HKEY_CURRENT_USER \Software\Microsoft\Windows\CurrentVersion\Run 和 HKEY_LOCAL_MACHINE \SOFTWARE\ Microsoft\Windows\CurrentVersion\Run 。HKEY_CURRENT_USER 下面的 Run 子键紧接 HKEY_LOCAL_MACHINE 下面的 Run 子键运行，但两者都在处理“启动”文件夹之前。⑦RunServiceOnce 子键。RunServiceOnce 子键用来启动服务程序，启动时间在用户登录之前，而且先于其他通过注册表子键启动的程序。RunServiceOnce 子键的位置是 HKEY_CURRENT_USER\ Software\Microsoft\Windows\CurrentVersion\ RunServiceOnce 和 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ RunService Once。⑧RunService 子键。RunService 子键指定的程序紧接 RunServiceOnce 子键指定的程序之后运行，但两者都在用户登录之前。RunService 子键的位置是 HKEY_CURRENT_USER\ Software\ Microsoft\Windows\CurrentVersion\RunService 和 HKEY_ LOCAL_MACHINE\ SOFTWARE\ Microsoft\Windows\CurrentVersion\RunService。通常系统启动自动运行程序存放在 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\ Explorer\ Run 下。例如，如果要使系统启动时自动运行 MyAutoRunProgram.exe 程序，可以打开注册表子键 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run，在该子键下新建一个键值“我的自动运行程序”并赋值为“(全路径) MyAutoRunProgram.exe”，关闭注册表，重新启动计算机，该程序将会自动运行。

下面是实例程序实现设置程序在系统启动时自动运行的示例代码。

```
#include "Registry.hpp"
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  if(this->Edit1->Text.Length()<1)
    return;
  TRegistry* Reg=new TRegistry();
```

```
    Reg->RootKey=HKEY_LOCAL_MACHINE;
    if(Reg->OpenKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run",false))
    {
      Reg->WriteString("我的自动运行程序",this->Edit1->Text.c_str());
      Reg->CloseKey();
      MessageBox(Handle,"设置程序在系统启动时自动运行操作成功！",
"信息提示",MB_OK);
    }
    delete Reg;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 054 如何设置程序在系统启动时自动运行”。

实例 055　如何创建多线程程序

实例目的

此实例主要介绍了使用 C++Builder 的 TThread 类创建新线程实例的编程技巧。实例程序运行后，效果如图 55-1 所示。



图 55-1

实现方法

在 Windows 操作系统中，实现多线程应用程序的编程需要调用一系列的 API 函数，如 CreateThread()、ResumeThread()、CreateSemaphore()、CreateEvent()、WaitForSingleObject()、ReleaseSemaphore()、SetEvent()等，它需要程序员对多线程编程的深刻理解和线程间关系的充分理解，比较麻烦而且容易出错。C++Builder 提供了 TThread 对象，使多线程应用程序的编写变得非常简便易用。实例程序使用 TThread 对象创建多线程应用程序的示例步骤如下：

（1）在 C++Builder 的集成开发环境中选择菜单 File\New\Others...，弹出 New Items

对话框，在 New 标签页中选择对象 Thread Object（如图 55-2 所示），单击 OK 按钮，弹出 New Thread Object 对话框，按照如图 55-3 所示填写有关内容，单击 OK 按钮后，将自动产生一个新线程类，其头文件和实现文件的内容如下所示：



图 55-2



图 55-3

```
//新线程头文件
#ifndef MyThreadH
#define MyThreadH
#include <Classes.hpp>
class CMyThread : public TThread
{
  typedef struct tagTHREADNAME_INFO
  {
    DWORD dwType;          // must be 0x1000
    LPCSTR szName;         // pointer to name (in user addr space)
    DWORD dwThreadID;      // thread ID (-1=caller thread)
    DWORD dwFlags;         // reserved for future use, must be zero
  } THREADNAME_INFO;
private:
  void SetName();
protected:
        void __fastcall Execute();
public:
        __fastcall CMyThread(bool CreateSuspended);
};
//新线程实现文件
#include <vcl.h>
```

```
#pragma hdrstop
#include "MyThread.h"
#pragma package(smart_init)
//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall CMyThread::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
__fastcall CMyThread::CMyThread(bool CreateSuspended)
        : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void CMyThread::SetName()
{
        THREADNAME_INFO info;
        info.dwType = 0x1000;
        info.szName = "MyThread";
        info.dwThreadID = -1;
        info.dwFlags = 0;
        __try
        {
                RaiseException( 0x406D1388, 0,
sizeof(info)/sizeof(DWORD),(DWORD*)&info );
        }
        __except (EXCEPTION_CONTINUE_EXECUTION)
        {
        }
}
//---------------------------------------------------------------------------
void __fastcall CMyThread::Execute()
{
        SetName();
        //---- Place thread code here ----
        MessageBox(NULL,"这是由新线程产生的消息！","信息提示",MB_OK);
}
```

（2）通常用户在新线程中的实现代码放在新线程实现文件的 Execute()函数中，在调用新线程时，需要动态创建一个 TMyThread 对象，在构造函数中使用 Resume()方法，则程序中就增加了一个新自定义的线程 TMyThread。代码如下所示：

```
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#include "MyThread.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
    CMyThread* MyThread=new CMyThread(true);
    MyThread->Resume();
}
```

最后编译执行程序即可。

说明：如果要加载更多的线程，只要继续创建需要数量的 TMyThread 对象即可。此外，由于 C++Builder 编程是建立在 VCL 类库的基础上的，在程序中经常需要访问 VCL 对象的属性和方法。但是，VCL 类库并不保证其中对象的属性和方法是线程访问安全的（Thread_safe），访问 VCL 对象的属性或调用其方法可能会访问到不被别的线程所保护的内存区域而产生错误。因此，TThread 对象提供了一个 Synchronize 方法，当需要在线程中访问 VCL 对象属性或调用方法时，通过 Synchronize 方法来访问属性或调用方法就能避免冲突，使各个线程之间协调而不会产生意外的错误。此部分内容在联机文档中有相关的说明。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 055 如何创建多线程程序”。

实例 056 如何创建可独立执行程序

实例目的

此实例主要介绍了如何在 C++Builder 中创建一个不依赖于其他辅助文件和环境的可执行程序的技术。实例程序独立执行后，效果如图 56-1 所示。

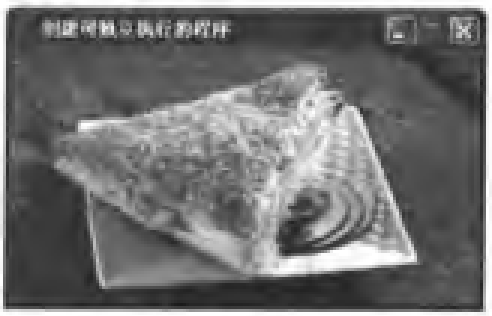


图 56-1

实现方法

默认情况下，C++Builder 编译链接的可执行程序相当小，但是这种程序在非 C++Builder

开发环境下往往不能执行，原因何在呢？实际上，这种可执行程序需要其他几个C++Builder默认提供的文件的支持才能执行，如果为了一个这样的执行程序而附带几个文件，有时实在是得不偿失。不过，C++Builder也能够编译不依赖于其他文件的可独立执行的程序，当然，这种程序就比前面所说的程序大多了。实例程序创建这种不依赖于其他文件的可独立执行的程序的具体步骤如下：

（1）选择Project\Options...菜单，弹出Project Options for ...对话框，单击Linker标签页，在Linking组框中去掉Use dynamic RTL选项，如图56-2所示。

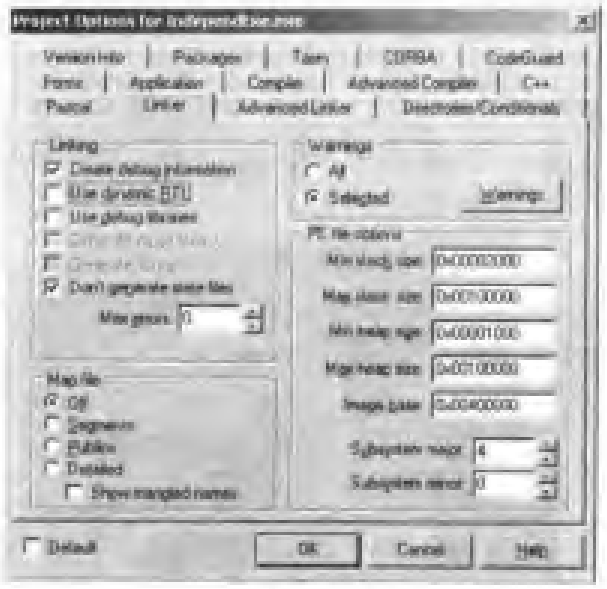


图56-2

（2）在图56-2中单击Packages标签，在Runtime packages组框中去掉Build with runtime packages选项，如图56-3所示。

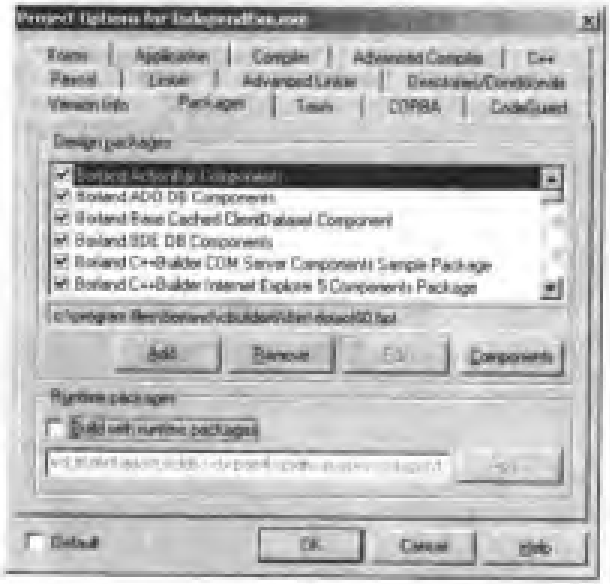


图56-3

重新编译、连接即可生成能独立运行的可执行文件。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 程序设置\实例 056 如何创建可独立执行程序”。

实例 057 如何编写单文档应用程序

实例目的

默认情况下，C++Builder 自动产生一个类似于对话框类型的应用程序，此实例主要解决如何在 C++Builder 中生成一个标准的单文档应用程序。当实例程序启动后，将执行一个标准单文档程序的操作。效果如图 57-1 所示。



图 57-1

实现方法

C++Builder 在组件库中提供了一个标准单文档程序的模板，它能够使用户不用编写一行代码即可自动产生标准的单文档程序框架。操作步骤如下：选择菜单 File\New\Other...，弹出 New Items 对话框，选择标签页 Projects 中的 SDI Application 图标，如图 57-2 所示，单击 OK 按钮，最后编译执行程序即可。



图 57-2

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 057 如何编写单文档应用程序”。

实例 058　如何编写多文档应用程序

实例目的

此实例主要解决如何在 C++Builder 中生成一个标准的多文档类型的应用程序。当实例程序启动后，将执行一个标准多文档程序的操作，在默认情况下，它能够读入或新建一个文本文件。效果如图 58-1 所示。



图 58-1

实现方法

MDI 多文档应用程序是 Windows 应用程序开发中使用频率较高的一类应用程序。通常，一个 MDI 应用程序包含以下 3 个基本窗口：①Frame Window。该窗口是应用程序的主窗口，上面有标题栏、主菜单栏和系统标准菜单及相关按钮。②Client Window。该窗口是 MDI 应用程序的管理员，它必须处理所有与 MDI 有关的命令，还要管理所属的 Child Window。③Child Window。该窗口是真正放置 MDI 文件的地方，它可以是文本、数据库、图形及其他类型的文件。它和主窗口一样，也有标题栏、菜单栏等，Child Window 是和主窗口结合在一起的，它的显示范围不会超出 Client Window。

C++Builder 在组件库中提供了一个标准多文档程序的模板，它能够使用户不用编写一行代码即可自动产生标准的多文档程序框架，并在默认情况下支持文本文件的新建和打开操作。操作步骤如下：选择菜单 File\New\Other...，弹出 New Items 对话框，选择标签页 Projects 中的 MDI Application 图标，如图 58-2 所示，单击 OK 按钮，最后编译执行程序即可。

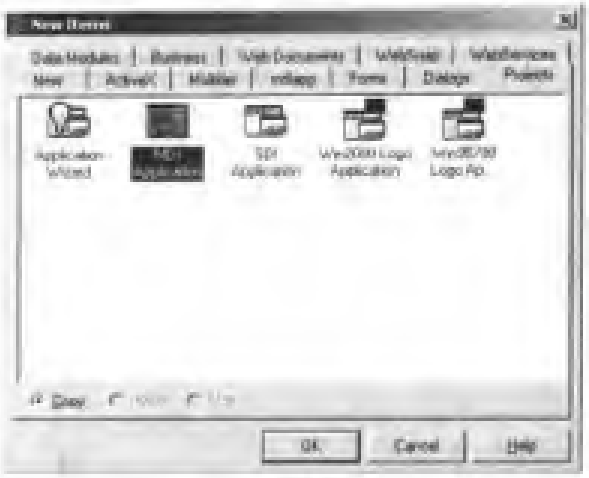


图 58-2

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 058 如何编写多文档应用程序”。

实例 059 如何编写 COM 服务器程序

实例目的

此实例介绍了如何在 C++Builder 中创建 COM 服务器程序的编程技术，实例程序必须与 COM 客户端程序配合使用。

实现方法

COM（Component Object Model），中文通常翻译为组件对象模型，允许对象向其他组件和宿主应用程序公开其功能，既定义对象如何公开自身，又定义此公开如何跨越多个进程和网络进行工作，COM 还定义对象的生命周期；同时它又是定义对象模块之间标准的二进制接口的规范，是与语言无关的、系统级的对象模型。使用 COM 意味着采用何种具体编程语言来搭建程序无关紧要，因为任何程序设计语言在二进制这一层面上没有什么区别。因此 COM 在以 Microsoft 技术为中心的环境中扮演着双重的角色：一方面，它是编写在基于组件的系统中运行的可重用软件的规范；另一方面，它为 Windows 架构的各个组成部分提供了一个统一的、可扩展的、面向对象的通信协议，是允许客户端和对象跨进程和计算机边界进行通信的复杂结构。微软在其应用程序、开发工具和操作系统中广泛使用了 COM 技术。C++Builder 简化了使用 SDK 开发 COM 组件的技术，它基本上不让程序员去理解 COM 组件的类、对象、接口和 GUID 等概念，只需要几个简单的步骤即可创建一个 COM 程序。具体步骤如下：

（1）启动 C++Builder，默认情况下，C++Builder 将创建一个默认应用程序。保留该

默认应用程序。

（2）选择菜单 File\New\Other...，单击 ActiveX 标签页，在该标签页中选择 Automation Object 自动化对象，如图 59-1 所示。在图中单击 OK 按钮，弹出 New Automation Object 对话框，在 CoClass Name 编辑栏中输入 COM 类名，如 MyComClass，其余保留默认值即可，如图 59-2 所示。



图 59-1

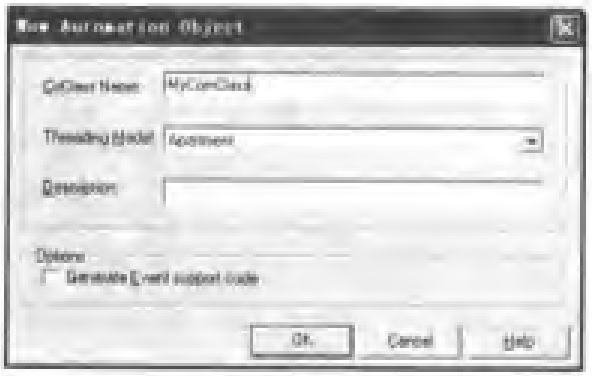


图 59-2

（3）在图 59-2 中单击 OK 按钮，C++Builder 将自动生成该 COM 类并产生 GUID 等，如图 59-3 所示。

（4）在图 59-3 中右击左边的 IMyComClass，在弹出的右键菜单中选择 New\Property，会产生 COM 类的属性。一般情况下，一个 COM 类的属性具有两个功能：一个是 Get（用于从 COM 客户端取得数据），一个是 Put（用于向 COM 客户端输出数据）。

（5）在图 59-4 中单击右边的 Parameters 标签页，在下面的列表框中修改 Value 的类型为 BSTR，如图 59-5 所示。然后，单击工具栏的 Refresh Implementation 按钮保存修改结果。COM 类编辑器除了产生以 Impl（即 Implementation）结尾的 CPP/H 文件外，还有两个分别是以_ATL 和_TLB 结尾的。_ATL 是自动产生的，通常不用改变它，当然如果有特别的要求并且对 ATL 又很熟悉，可以修改它；_TLB 是由 Type Library Editor 自动维护，无须修改，改了也没用，任何修改都会在 Refresh Implementation 时被改回来。

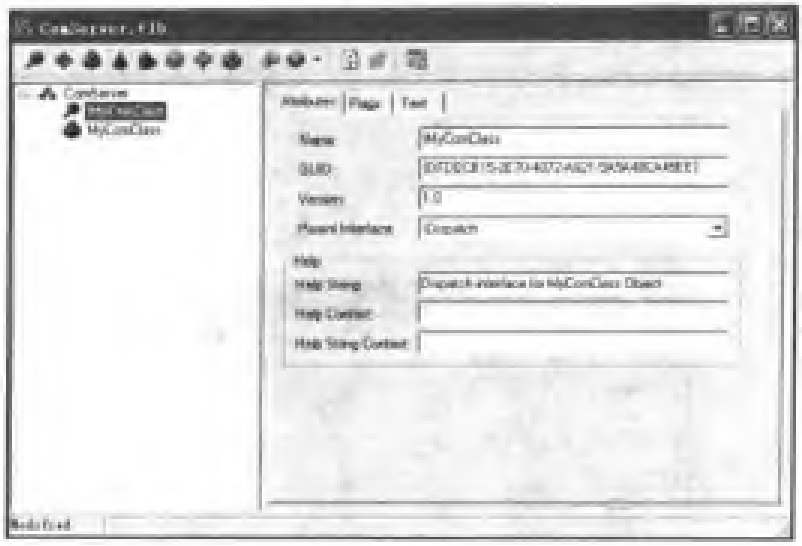


图 59-3

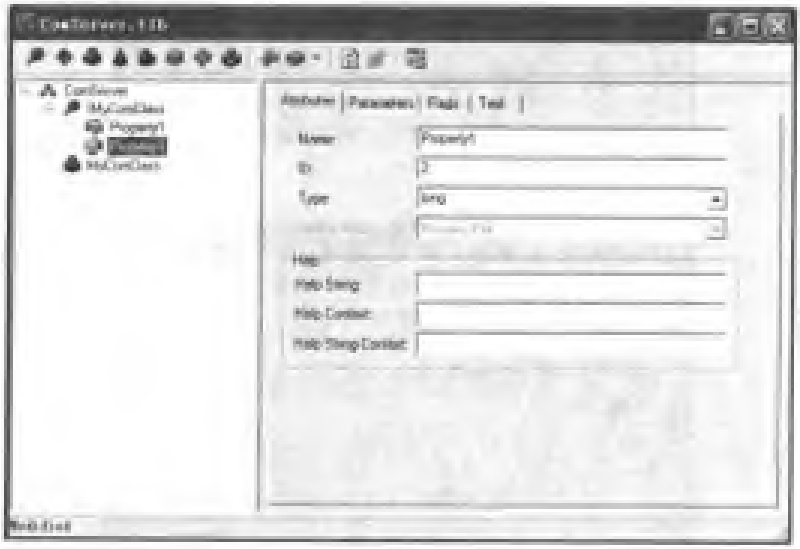


图 59-4

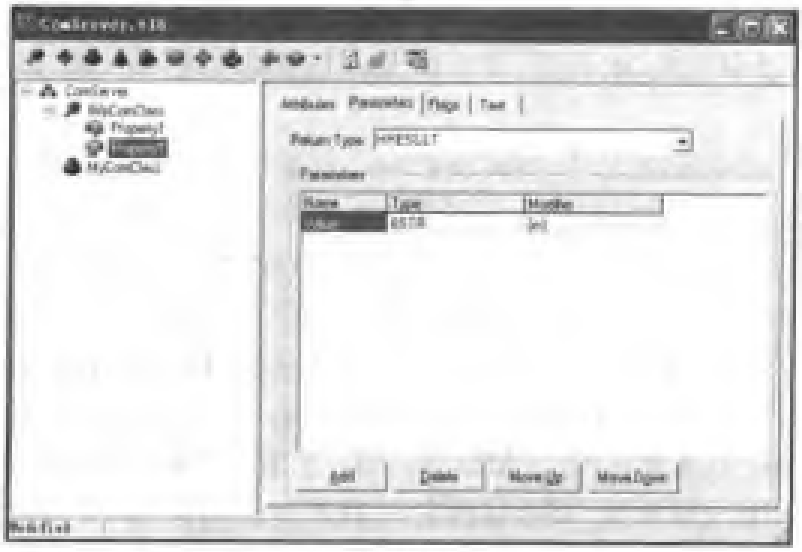


图 59-5

（6）打开 MYCOMCLASSIMPL.H 文件，在最后加上如下所示的内容。

```
class ATL_NO_VTABLE TMyComClassImpl :
  public CComObjectRootEx<CComSingleThreadModel>,
  public CComCoClass<TMyComClassImpl, &CLSID_MyComClass>,
  public IDispatchImpl<IMyComClass, &IID_IMyComClass, &LIBID_ComServer>
{
 ...
// IMyComClass
public:

  STDMETHOD(get_Property1(BSTR* Value));
  STDMETHOD(set_Property1(BSTR Value));
  WideString InputString;
);
```

（7）打开 MYCOMCLASSIMPL.CPP 文件，按照如下所示的内容修改代码。

```
STDMETHODIMP TMyComClassImpl::get_Property1(BSTR* Value)
{
  try
  {
   *Value = InputString; //新增

  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IMyComClass);
  }
  return S_OK;
);
STDMETHODIMP TMyComClassImpl::set_Property1(BSTR Value)
{
  try
  {
   InputString=Value;  //新增
  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IMyComClass);
  }
  return S_OK;
);
```

最后，编译执行程序。如果需要从 COM 客户端使用该 COM 服务器程序，必须首先在系统注册表中进行注册。可以在命令行下进行注册，例如：ComServer/RegServer，如图 59-6 所示。如果需要从注册表注销该 COM 服务器程序，则在命令行下输入 ComServer/UnRegServer 即可。

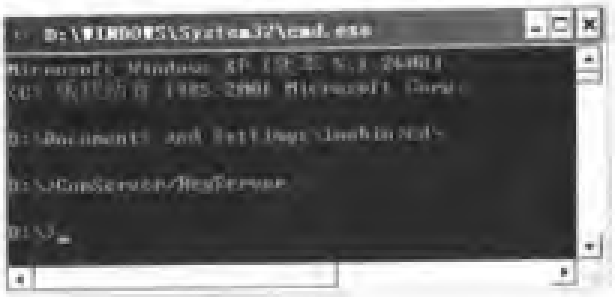


图 59-6

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 059 如何编写 COM 服务器程序”。

实例 060 如何编写 COM 客户端程序

实例目的

此实例介绍了如何在 C++Builder 中创建 COM 客户端程序调用 COM 服务器程序方法的编程技术，实例程序必须与 COM 服务端程序配合使用，实例执行后，将启动 COM 服务器程序 ComServer，并从该服务器程序中取出有关值，效果如图 60-1 所示。

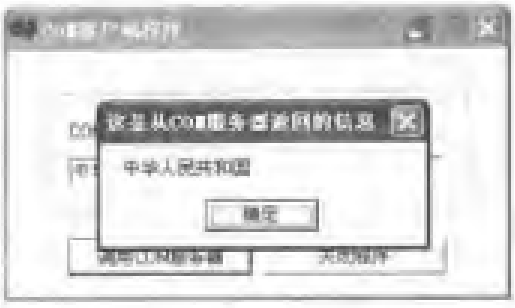


图 60-1

实现方法

使用 COM 服务器程序的方法有两种，第一种为前期联编法，该方法实际上就是根据 COM 服务器的 COM 类源文件直接调用其中的方法，作为 COM 技术本身来说，这种方法可用性不大；第二种方法称为后期联编法，该方法是通过直接访问 COM 接口操作服务器的方法，是 COM 技术的精髓所在。示例代码如下（请结合“实例 059 如何编写 COM 服务器程序”）：

```
#include <vcl/ComObj.hpp>
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->Edit1->Text.Length()<1)
```

```
    return;
    Variant v = CreateOleObject( 'ComServer.MyComClass' );
    v.Exec( PropertySet( 'Property1' ) << this->Edit1->Text.c_str() );
   AnsiString StrInfo = v.Exec( PropertyGet( 'Property1' ) );
    MessageBox(Handle,StrInfo.c_str(),"这是从 COM 服务器返回的信息",MB_OK);
}
```

最后，编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 060 如何编写 COM 客户端程序”。

实例 061　如何编写 DDE 服务器程序

实例目的

此实例介绍了如何使用 DDE 服务组件编写 DDE 服务器程序的编程技术。实例程序运行后（需要与 DDE 客户端程序配合使用），效果如图 61-1 所示。

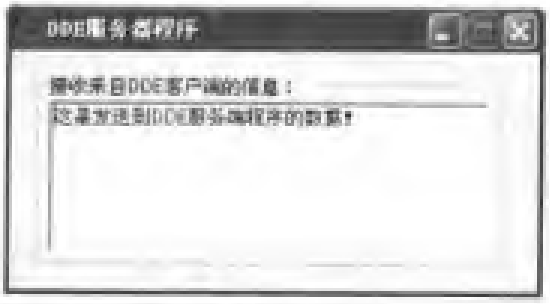


图 61-1

实现方法

在 C++Builder 中，提供了一组 DDE（动态数据交换）组件，它能够在不同程序（即 DDE 客户程序和 DDE 服务程序）间交换数据。DDE 服务器程序主要实现三大功能：①接收 DDE 客户端数据；②处理数据；③向 DDE 客户端程序反馈数据。编写 DDE 服务器程序主要使用了 TDDEServerConv 和 TDDEServerItem 组件，TDDEServerConv 组件完成服务器端有关链接的工作，TDDEServerItem 组件指定动态数据交换时传递的数据。如果使用程序的窗口作为数据交换对话主题，则 TDDEServerConv 组件可以省略不用，但是必须保证窗口标题在程序运行时始终保持不变。以下是实例程序编写 DDE 服务器程序的具体步骤。

（1）在组件模板的 System 标签页下选择 DdeServerConv 和 DdeServerItem 组件到 Form1 上，设置 DdeServerItem 组件的 ServerConv 属性为 DdeServerConv1，即 DdeServerConv

组件的名称。

（2）为 DdeServerConv 组件的 OnExecuteMacro 事件添加如下所示的代码：

```
void __fastcall TForm1::DdeServerConv1ExecuteMacro(TObject *Sender,
    TStrings *Msg)
{//响应 DDEServerConv1 组件的 OnExecuteMacro 事件
  this->Memo1->Clear();
  this->Memo1->Text=Msg->Strings[0];
}
```

（3）为 DdeServerItem 组件的 OnChange 事件添加如下所示的代码：

```
void __fastcall TForm1::DdeServerItem1Change(TObject *Sender)
{//响应 DDEServerItem1 组件的 OnChange 事件
  this->Memo1->Clear();
  this->Memo1->Text=this->DdeServerItem1->Text;
}
```

（4）为 DdeServerItem 组件的 OnPokeData 事件添加如下所示的代码：

```
void __fastcall TForm1::DdeServerItem1PokeData(TObject *Sender)
{//响应 DDEServerItem1 组件的 OnPokeData 事件
  this->Memo1->Clear();
  this->Memo1->Lines=this->DdeServerItem1->Lines;
}
```

最后，编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 061 如何编写 DDE 服务器程序”。

实例 062 如何编写 DDE 客户端程序

实例目的

此实例介绍了如何使用 DDE 客户端组件编写 DDE 客户端程序的编程技术。实例程序运行后（需要与 DDE 服务器程序配合使用），效果如图 62-1 所示。

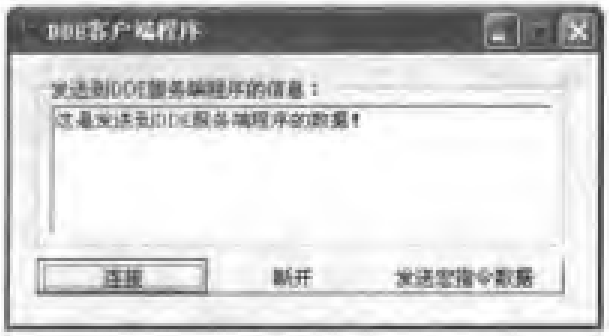


图 62-1

实现方法

在 C++Builder 中，提供了一组 DDE（动态数据交换）组件，它能够在不同程序（即 DDE 客户程序和 DDE 服务程序）间交换数据。编写 DDE 客户端程序主要使用了 TDDEClientConv 和 TDDEClientItem 组件，TDDEClientConv 组件用于与作为 DDE 服务器的应用程序建立一个链接，TDDEClientItem 组件获取动态数据交换时传递到客户端的数据。DDE 客户端与 DDE 服务器的连接可以在程序设计时实现，也可以在程序运行时动态实现，总的原则是 DDE 服务器必须先运行。在程序运行时实现服务器与客户端连接需要使用 TDDEClientConv 组件的 SetLink()、OpenLink()和 CloseLink()方法。在进行 DDE 数据通信前，需要指定数据通信的 3 个特性：①DDE 服务，通常 DDE 服务指的是 DDE 服务器的应用程序名称，也可以由服务器自行确定一个名称，用户可以设置 DdeClientConv 组件的 DdeService 属性来确定 DDE 服务；②DDE 主题，DDE 主题是一个字符串，可以设定为 DDE 服务器的应用程序名称，也可以是服务器应用程序中的 DdeServerConv 组件的对象名，用户可以设置 DdeClientConv 组件的 DdeTopic 属性来确定 DDE 主题；③DDE 项，DDE 项也是一个字符串，可以设定为 DDE 服务器应用程序中 DdeServerItem 组件的对象名，用户可以设置 DdeServerItem 组件的 DdeItem 属性来确定 DDE 项。以下是实例程序编写 DDE 客户端程序的具体步骤。

（1）在组件模板的 System 标签页下选择 DdeClientConv 和 DdeClientItem 组件到 Form1 上，修改 DdeClientConv 组件的 ConnectMode 属性为 ddeManual，单击 DdeClientConv 组件的 DdeService 属性右边的“...”按钮，按照如图 62-2 所示的内容修改该属性。其中，DdeService 属性输入的是 DDE 服务器执行文件的名称，可以包括路径，可以不写扩展名.exe；DdeTopic 属性输入的是对话主题，只要输入 DDE 服务器设计时的 DdeServerConv 组件的名称即可。



图 62-2

（2）在 Form1 上增加一个按钮，用于连接 DDE 服务器程序，其实现代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//连接 DDE 服务端程序
  if(this->DdeClientConv1->OpenLink())
  {
    MessageBox(Handle,"成功连接 DDE 服务端程序！","信息提示",MB_OK);
  }
  else
  {
```

```
    MessageBox(Handle,"连接 DDE 服务端程序失败！","信息提示",MB_OK);
  }
}
```

（3）在 Form1 上增加一个按钮，用于断开与 DDE 服务器程序的连接，其实现代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//断开与 DDE 服务端程序的连接
  this->DdeClientConv1->CloseLink();
}
```

（4）在 Form1 上增加一个按钮，用于向 DDE 服务器程序发送数据，其实现代码如下：

```
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//向 DDE 服务端程序发送宏指令数据
   if(this->DdeClientConv1->ExecuteMacro(this->Memo1->Text.c_str(),true))
   {
     MessageBox(Handle,"发送宏指令数据操作成功！","信息提示",MB_OK);
   }
   else
   {
     MessageBox(Handle,"发送宏指令数据操作失败！","信息提示",MB_OK);
   }
}
```

最后，编译执行程序即可。注意：有连接必有断开。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 062 如何编写 DDE 客户端程序”。

实例 063 如何编写 DLL 服务器程序

实例目的

此实例介绍了如何使用向导编写 DLL 服务程序的编程技术。实例程序需要与 DLL 客户端程序配合使用才能显示效果。

实现方法

DLL（动态链接库）是 Windows 操作系统最重要的组成要素，Windows 中的许多新功能、新特性都是通过 DLL 来实现的，而实际上 Windows 操作系统本身就是由许多 DLL 组成的，它最基本的三大组成模块 Kernel、GDI/GDI+和 User 都是 DLL，它所有的库模块也被设计成 DLL。DLL 技术具有许多优点：①通常多个应用程序调用同一个 DLL 时，在内存中只有一个实例，能高效经济地使用内存；②DLL 实现的代码封装性好，使得程序简洁明晰；③DLL 的编制与具体的编程语言及编译器无关，只要遵守 DLL 的开发规范和编程策略，并安排正确的调用接口，不管用何种编程语言编制的 DLL 都具有通用性。DLL 技

术实现通常包括 DLL 服务程序和 DLL 客户端调用程序两种，而 DLL 客户端程序调用方式也可以分为静态调用和动态调用两种。下面介绍如何在 C++Builder 中编写 DLL 服务程序，具体步骤如下：

（1）选择菜单 File\New\Other...，弹出 New Items 对话框，在该对话框中单击 New 标签页，选择 Dll Wizard 项目，如图 63-1 所示。单击 OK 按钮，弹出 DLL Wizard 对话框，如图 63-2 所示，保留默认值，单击 OK 按钮，向导即自动产生一个 DLL 的框架文件。



图 63-1



图 63-2

（2）选择菜单 File\New\Form，C++Builder 将自动产生一个表单，在该表单上放置一幅图像，如图 63-3 所示。然后将表单文件保存为 Image.cpp 即可。



图 63-3

（3）按照如下所示的代码修改 DLL 向导自动产生的文件。

```
#include <vcl.h>
#include <windows.h>
#pragma hdrstop
#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*
lpReserved)
{
      return 1;
}
#include "Image.h"  //表单头文件
extern "C" void _stdcall ShowImage(void);
void _export _stdcall ShowImage(void)
{
  Form1 = new TForm1(NULL);
  Form1->ShowModal();
  delete Form1;
}
```

最后，编译程序即可产生一个动态链接库文件：DllServer.dll。该 DLL 需要与 DLL 客户端程序配合使用。其中，extern "C" void _stdcall ShowImage(void)语句中的 extern "C" 是用来告诉编译器，以 C 的方式来命名，而不要以 C++的命名法，因为 C++的命名法会在函数名称后面加上参数型态等修饰字，以避免造成其他程序如 VC++、VB 等无法使用该 DLL 的困扰。另外_stdcall 用来表示它使用的参数传入方法。

注意：在 Windows 操作系统的 DLL 开发规范中，DLL 程序也有自己的主函数，但是在 C++Builder 中，这个主函数（程序的入口点和出口点）就是向导默认产生的 int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)函数。该函数的声明如下：

```
BOOL WINAPI DllEntryPoint(
   HINSTANCE hinstDLL,          //DLL 模块句柄
   DWORD fdwReason,//调用说明
   LPVOID lpvReserved           //保留字
  );
```

其中，参数 DWORD fdwReason 的取值如下：DLL_PROCESS_ATTACH（指示 DLL 已经将地址空间附加到了当前进程，这个结果可能是由于调用了 LoadLibrary()函数而引起的，这时，可以对 DLL 进行初始化）、DLL_THREAD_ATTACH（指示当前进程正在创建一个新的线程，系统调用了 DllEntryPoint，这时也可以对 DLL 进行初始化）、DLL_THREAD_DETACH（线程正在退出，可以调用释放函数释放内存，为 DLL 退出作准备）、DLL_PROCESS_DETACH（从进程地址空间中分离，这个结果可能是系统调用了 FreeLibrary()函数，此时可以调用释放函数释放内存，为 DLL 退出作准备）。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 063 如何编写 DLL 服务器程序”。

实例 064　如何编写 DLL 客户端程序

实例目的

此实例介绍了如何编写 DLL 客户端程序的编程技术。实例程序运行后，单击“显示 DLL 服务程序图像”按钮，即可显示出 DLL 服务程序中的图像，效果如图 64-1 所示。



图 64-1

实现方法

DLL 服务程序的功能必须通过 DLL 客户端程序调用才能表现出来。不论使用何种语言对编译好的 DLL 进行调用，基本上都有两种调用方式，即静态调用方式和动态调用方式。静态调用方式由编译系统完成对 DLL 的加载和应用程序结束时 DLL 卸载的编码，该调用方式简单实用，但不够灵活，只能满足一般要求，当用同一种开发语言开发的 DLL 服务程序和客户程序时，可以使用静态调用方式，当用其他语言开发 DLL 客户程序时，则需要使用动态调用方式。动态调用方式是由编程者用 API 函数加载和卸载 DLL 来达到调用 DLL 的目的，在使用上较复杂，但能更加有效地使用内存，是编制大型应用程序时的重要方式。在使用动态方式调用 DLL 时，首先是指定 DLL 服务程序的路径名作为参数调用 LoadLibrary() 函数，LoadLibrary() 函数将返回一个 HINSTANCE 参数；然后在 GetProcAddress()函数中使用该参数，转换到 DLL 服务程序中的方法所在的地址，调用 DLL 服务程序的方法即可。最后调用 FreeLibrary()函数卸载 DLL 服务程序，以释放占用的系统资源。LoadLibrary()函数的声明如下：

```
HINSTANCE LoadLibrary(
LPCTSTR lpLibFileName          //DLL 服务程序文件名
);
```

其中，参数 LPCTSTR lpLibFileName 指定要装载的 DLL 服务程序文件名，如果参数 LPCTSTR lpLibFileName 没有指定一个具体的路径，则 Windows 将按照下述顺序查找：①当前程序所在的目录；②Windows 目录，利用函数 GetWindowDirectory()可以取得系统的这一目录所在的路径；③系统目录，利用函数 GetSystemDirectory()可以取得

这一目录；④包含当前任务可执行文件的目录，利用函数 GetModuleFileName()可以返回这一目录所在的路径；⑤列在 PATH 环境变量中的目录；⑥网络映射目录列表。若 DLL 客户端程序在使用 LoadLibrary()函数调用 DLL 服务程序时，其他的 DLL 客户端程序已经将该模块装入内存，则 LoadLibrary()函数并不会装载该模块的另一个实例，而是使该模块的“引用计数”加 1。

函数 GetProcAddress()用于获取 DLL 服务程序中的方法所在的地址，该函数的声明如下：

```
FARPROC GetProcAddress(
    HMODULE hModule,              //DLL 模块的句柄
    LPCSTR lpProcName             //DLL 服务程序中的函数名
   );
```

函数 FreeLibrary()用于动态卸载 DLL 动态链接库，该函数的声明如下：

```
BOOL FreeLibrary(
    HMODULE hLibModule            //DLL 模块的句柄
   );
```

下面是实现程序调用本书中的“如何编写 DLL 服务器程序”中的方法的具体实现代码。

```
void (*ShowImage)(void);
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  HINSTANCE hInst=NULL;
  hInst = LoadLibrary("DLLServer.DLL");
  if(hInst)
  {
    (FARPROC &)ShowImage=GetProcAddress(hInst,"ShowImage");
     ShowImage();
    FreeLibrary(hInst);
 }
 else
 {
   MessageBox(Handle,"无法加载 DllServer.dll 文件","信息提示",MB_OK);
   FreeLibrary(hInst);
  }
}
```

注意：在使用动态方式调用 DLL 服务程序时，可能存在一个时间滞后的问题，例如，用户调用了 DLL 服务程序中的某一个函数，这个函数还没有来得及运行，而用户又采用了释放 DLL 的语句将 DLL 服务程序释放了，此时程序就会产生错误。因此，要注意释放 DLL 服务程序的时机。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 064 如何编写 DLL 客户端程序”。

实例 065　如何注册 DLL 服务器程序

实例目的

此实例介绍了如何在系统中注册 DLL、OCX 服务程序的编程技术。实例程序运行后，效果如图 65-1 所示。



图 65-1

实现方法

在 Windows 操作系统中，为了减少程序占用的存储空间，提高程序的利用率，大量使用了 DLL 动态链接库。DLL 文件通常放置在 Windows 的系统目录下，有的 DLL 文件需要进行注册才能使用，有的 DLL 文件则不需要注册即可直接使用；注册 DLL 文件一般在第一次使用或安装时进行，一次性注册即可永久使用。注册 DLL 文件操作可以在命令行下使用命令 REGSVR32 进行，也可以在程序中注册 DLL 文件。在程序中注册 DLL 文件的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//注册 DLL、OCX 服务程序
  if(this->Edit1->Text.Length()<1)
    return;
  HINSTANCE hInstance=LoadLibrary(this->Edit1->Text.c_str());
  FARPROC lpDP;
  if(hInstance<(HINSTANCE)HINSTANCE_ERROR)
  {
    MessageBox(Handle,"没法取得实例句柄","信息提示",MB_OK);
    return;
  }
  lpDP=GetProcAddress(hInstance,"DllRegisterServer");
  if(lpDP!=NULL)
  {
    if(FAILED((*lpDP)()))
    {
      MessageBox(Handle,"调用 DllRegisterServer 失败! ","信息提示",MB_OK);
      FreeLibrary(hInstance);
      return;
    }
```

```
        else
        {
          MessageBox(Handle,"注册 DLL、OCX 服务程序操作成功","信息提示",MB_OK);
          return;
        }
      }
      else
      {
          MessageBox(Handle,"调用 DllRegisterServer 失败! ","信息提示",MB_OK);
          FreeLibrary(hInstance);
          return;
      }
  }
  void __fastcall TForm1::BitBtn3Click(TObject *Sender)
  {//取消注册 DLL、OCX 服务程序
    if(this->Edit1->Text.Length()<1)
      return;
    HINSTANCE hInstance=LoadLibrary(this->Edit1->Text.c_str());
    FARPROC lpDP;
    if(hInstance<(HINSTANCE)HINSTANCE_ERROR)
    {
      MessageBox(Handle,"没法取得实例句柄","信息提示",MB_OK);
      return;
    }
    lpDP=GetProcAddress(hInstance,"DllUnregisterServer");
    if(lpDP!=NULL)
    {
      if(FAILED((*lpDP)()))
      {
        MessageBox(Handle,"调用 DllUnregisterServer 失败!","信息提示",MB_OK);
        FreeLibrary(hInstance);
        return;
      }
      else
      {
        MessageBox(Handle,"取消注册 DLL、OCX 服务程序操作成功","信息提示",MB_OK);
        return;
      }
    }
    else
    {
        MessageBox(Handle,"调用 DllUnregisterServer 失败!","信息提示",MB_OK);
        FreeLibrary(hInstance);
        return;
    }
  }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序

设置\实例 065 如何注册 DLL 服务器程序”。

实例 066　如何制作聊天服务器程序

实例目的

此实例主要介绍了如何使用基于 WinSock 的技术创建点对点的聊天室服务器程序。实例程序执行后，设置通信端口，启动“监听”功能，就可以允许与聊天室客户端连接，并可以接收从聊天室客户端发来的信息，以及从聊天室服务器向聊天室客户端发送信息，效果如图 66-1 所示。

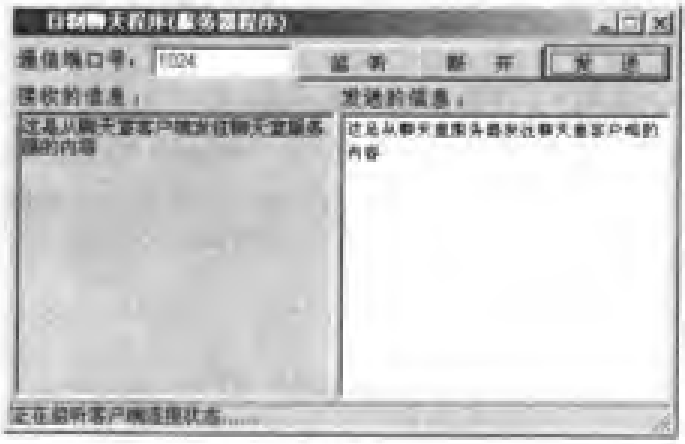


图 66-1

实现方法

网络通信的基础是套接字 Socket。Socket 是通信端点的一种概念抽象，它提供了相连的各个通信端点通过网络进行数据发送和接收的机制。Socket 最初是美国加州大学伯克利分校为在 UNIX 操作系统下实现 TCP/IP 网络通信而开发的一个 API（应用程序接口），随着 UNIX 操作系统的广泛使用，Socket 成为当前最流行的网络通信应用程序接口之一。最初的伯克利 Socket 只能用于 UNIX 操作系统，不能在 DOS 和 Windows 操作系统下使用。在 Sun、Intel、SGI、Informix、Novell 等电脑巨头的支持下，微软推出了以伯克利 Socket 为基础的 Windows Sockets（WinSock），并经过不断完善，使之成为 Windows 网络编程事实上的标准，成为 WOSA（Microsoft Windows Open System Architecture）的一部分。Windows Sockets 规范是一套开放的、支持多种协议的 Windows 操作系统下的网络编程 API，它不仅包含了伯克利 Socket 风格的库函数，也包含了一组针对 Windows 操作系统的扩展库函数（如异步函数），使之能够充分发挥 Windows 的消息驱动机制。常用的 Socket API 函数包括 accept()、bind()、closesocket()、connect()、getpeername()、getsockname()、getsockopt()、htonl()、htons()、inet_addr()、inet_ntoa()、ioctlsocket()、listen()、ntohl()、ntohs()、recv()、recvfrom()、select()、send()、sendto()、setsockopt()、shutdown()、socket()等。

由于在应用程序中直接使用这些 Socket API 函数编程，需要对这些函数和网络运行机制的深刻理解，给程序设计人员带来了极大的麻烦。在 C++Builder 中，Borland 公司将有关的 Socket API 函数以事件的形式进行了封装，集中于 ClientSocket 和 ServerSocket 两个组件中，实际使用时，只需要直接调用这两个组件即可。虽然这两个控件用于管理服务器和客户端的连接，它们本身并不是 Socket 对象，操纵 Socket 对象的是 TCustomWinSocket 及其派生类，如 TClientWinSocket、TServerWinSocket、TServerClientWinSocket 等。

网络聊天工具的基础协议是 TCP/IP 协议，因此可以把网络聊天程序分为服务器和客户端两部分，其中服务器把程序转换成一个虚拟的 TCP/IP 服务器，并且和指定的客户端建立连接，在连接成功后，向指定的客户端发送和接收数据；客户端把程序转换成一个虚拟的 TCP/IP 客户端，并且向指定的服务器发出连接信号，在连接成功后，向指定的服务器发送和接收数据。下面是在 C++Builder 中使用 ServerSocket 组件创建聊天室服务器程序的示例步骤和代码。

（1）ServerSocket 组件位于组件模板的 Internet 标签页下，使用时直接拖动到 Form1 上即可。使用 ServerSocket 组件时，必须设置 ServerSocket 组件的端口号，它是与客户端进行通信与连接的通道，客户端必须以相同的端口号才能连接。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//监听
   this->ServerSocket->Port=StrToInt(this->Edit1->Text);
   this->ServerSocket->Active=true;
   this->StatusBar1->SimpleText="正在监听客户端连接状态.....";
}
```

如果在客户端与服务器建立连接后又断开，将断开客户端与服务器的连接，服务器端将触发 OnClientDisconnect 事件，而客户端则会触发 OnDisconnect 事件，这时服务器端应回到监听状态，等待用户的连接；而客户端将返回到连接前的状态，等待用户再次建立连接，如果不止一个服务器，可以选择连接到其他的服务器上。在服务器接受了客户的请求后会触发 OnAccept 事件。

（2）在 ServerSocket 组件需要重新设置端口号时，应该首先断开与客户端的连接，然后再设置端口号并重新启用。示例代码如下：

```
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//断开
   this->ServerSocket->Close();
   this->StatusBar1->SimpleText="已经断开与客户端的连接";
}
```

（3）如果需要使用 ServerSocket 组件发送信息，直接使用其方法 SendText()即可。示例代码如下：

```
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//发送
  this->ServerSocket->Socket->Connections[0]->SendText(this->RichEdit1->Text);
}
```

（4）当 ServerSocket 组件收到客户端发来的信息时，它会启用 OnClientRead 事件，

要显示相应的信息内容时，需要自行处理该事件，示例代码如下：

```
void __fastcall TForm1::ServerOnClientRead(TObject *Sender,
                                           TCustomWinSocket *Socket)
{
  this->Memo1->Lines->Add(Socket->ReceiveText());
}
```

最后编译执行程序即可。注意：该实例程序需要与“实例 067　如何制作聊天客户端程序”配合使用。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 066 如何制作聊天服务器程序”。

实例 067　如何制作聊天客户端程序

实例目的

此实例主要介绍了如何使用基于 WinSock 的技术创建点对点的聊天室客户端程序。实例程序执行后，设置通信端口和服务器名或 IP 地址，启动“连接”功能，就可以允许与聊天室服务器连接，并可以接收从聊天室服务器发来的信息，以及从聊天室客户端向聊天室服务器发送信息，效果如图 67-1 所示。

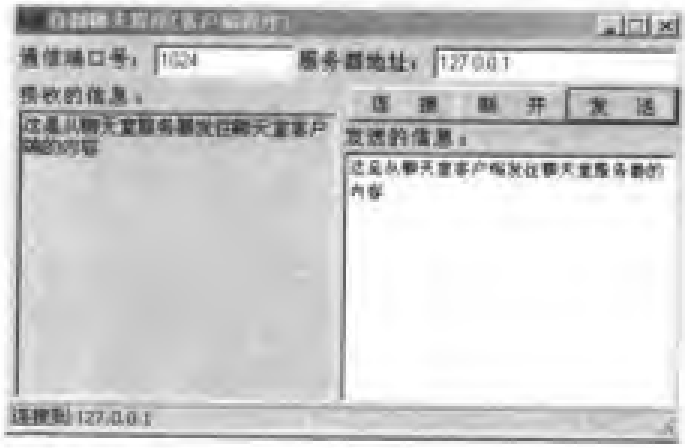


图 67-1

实现方法

网络通信的基础是套接字 Socket。Socket 是通信端点的一种概念抽象，它提供了相连的各个通信端点通过网络进行数据发送和接收的机制。Socket 之间的连接可以分为 3 种类型：客户端连接、监听连接以及服务器端连接。所谓客户端连接，是指由客户端 Socket 提出连接请求，连接目标是服务器端的 Socket。为此，客户端 Socket 必须首先描述它要连接

的服务器端 Socket（主要是指服务器端 Socket 的地址和端口号），然后再定位所要连接的服务器端 Socket，找到以后，就向服务器端 Socket 请求连接。当然，服务器端 Socket 此时未必正好处于准备好状态，不过，服务器端的 Socket 会自动维护客户请求连接的队列，然后在它认为合适的时候向客户端 Socket 发出“允许连接”(Accept)信号，这时客户端 Socket 与服务器端 Socket 的连接就建立了。所谓监听连接，服务器端 Socket 并不定位具体的客户端 Socket，而是处于等待连接的状态。当服务器端 Socket 监听到客户端 Socket 的连接请求后，它就响应客户端 Socket 的请求建立一个新的 Socket 句柄并与客户端连接，而服务器端 Socket 继续处于监听状态，还可以接收其他客户端 Socket 的连接请求。所谓服务器端连接，是指当服务器端 Socket 接收到客户端 Socket 的连接请求后，就把服务器端 Socket 的描述发给客户端，一旦客户端确认了此描述，连接就建立了。要通过 Socket 传输数据，至少需要一对 Socket，一个 Socket 在客户端，另一个 Socket 在服务端。C++Builder 提供了两个与 Socket 有关的组件：ClientSocket 和 ServerSocket，这两个组件其实并不是 Socket 对象，其属性 Socket 将返回各自的 Socket 对象。TClientSocket 组件用来处理客户端到服务器端之间的 socket 连接，TServerSocket 用来处理由客户端发来的 socket 连接，一旦客户端和服务器端都接通了 socket，客户端和服务器端就可以互相通信了。下面是实例程序使用 ClientSocket 组件创建聊天室客户端的示例步骤和代码。

（1）ClientSocket 组件位于组件模板的 Internet 标签页下，使用时直接拖动到 Form1 上即可。使用 ClientSocket 组件时，必须设置 ClientSocket 组件的端口号，这个端口号必须与服务器所设置的端口号一致，否则它不能与服务器进行连接，并且需要设置连接的服务器的 IP 地址。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//连接
this->ClientSocket1->Port=StrToInt(this->Edit1->Text);
this->ClientSocket1->Host=this->Edit2->Text;
this->ClientSocket1->Active=true;
this->StatusBar1->SimpleText="连接服务器.....";
}
```

（2）ClientSocket 组件需要重新与其他服务器连接时，应该首先断开与原服务器的连接，然后再设置端口号和服务器 IP 地址并重新连接。在活动状态时，不允许连接新的服务器地址。示例代码如下：

```
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//断开
  this->ClientSocket1->Close();
  this->StatusBar1->SimpleText="已经断开与服务器的连接";
}
```

（3）当 ClientSocket 组件与服务器连接成功时，会产生 OnConnect 事件，可以处理该事件显示相关的内容。示例代码如下：

```
void __fastcall TForm1::ClientOnConnect(TObject *Sender,
     TCustomWinSocket *Socket)
{
this->StatusBar1->SimpleText="连接到:"+this->Edit2->Text;
```

```
RichEdit1->Lines->Clear();
}
```

（4）如果需要使用 ClientSocket 组件发送信息，直接使用其方法 SendText()即可。示例代码如下：

```
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//发送
this->ClientSocket1->Socket->SendText(this->RichEdit1->Text);
}
```

（5）当 ClientSocket 组件收到服务器端发来的信息时，它会启用 OnRead 事件，要显示相应的信息内容时，需要自行处理该事件，示例代码如下：

```
void __fastcall TForm1::ClientOnRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
  Memo1->Lines->Add(this->ClientSocket1->Socket->ReceiveText());
}
```

最后编译执行程序即可。注意：该实例程序需要与“实例 066　如何制作聊天服务器程序”配合使用。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 067 如何制作聊天客户端程序”。

实例 068　如何获取程序名称和路径

实例目的

此实例主要介绍了如何获取当前执行程序所在的路径和程序名称，即使程序改变名称也能够正确取得。实例程序执行后，效果如图 68-1 所示。



图 68-1

实现方法

C++Builder 的 VCL 的 TApplication::ExeName 属性记录着当前执行程序的全路径名称，读取该属性即可得知程序的全路径名称。而函数 extern PACKAGE AnsiString __fastcall

ExtractFilePath (const AnsiString FileName)则可以从一个 TApplication::ExeName 属性中析取出不包括文件名称的全路径名。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  this->Edit1->Text=ExtractFilePath(Application->ExeName);
  this->Edit2->Text=Application->ExeName;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 068 如何获取程序名称和路径”。

实例 069 如何获取程序命令行参数

实例目的

此实例主要介绍了如何获取应用程序在命令行状态下启动时参数的编程技术。实例程序执行后，效果如图 69-1 所示。



图 69-1

实现方法

在 DOS 操作系统下，应用程序通常通过命令行参数决定应用程序的启动方式；Windows 应用程序虽然不显示应用程序的命令行调用，但实际上也是遵循这一方式进行程序启动或调用的。最常见的就是一些系统的定制安装，它可以在 DOS 窗口下，在 Setup 命令行下附加参数，进行不同类别和层次的安装。开发这种带参数的应用程序时，可以在程序中通过系统函数 GetCommandLine()获取用户在命令行方式下输入的参数，GetCommandLine()函数的声明如下：

```
LPTSTR GetCommandLine(VOID)
```

该函数在调用后，直接返回用户的命令行输入，如果没有带参数，返回的就是应用程序的全路径名。遗憾的是，用户在使用该函数时必须按照字符串的方式分解其中的参数。如果使用 VCL 的 extern PACKAGE AnsiString __fastcall ParamStr(int Index)函数，则可以直

接取得程序启动的命令行参数。ParamStr()函数需要一个整数参数并且返回一个 AnsiString 对象。若参数为 0，ParamStr()函数将返回可执行文件的全路径；若参数为 1，将返回程序名及第一个命令行参数；若参数为 2，将返回第二个参数，以此类推。此外，ParamStr()函数能够对目录中的空格进行智能判断，把生成的 EXE 文件复制到 Program Files 目录下再运行，就会看到 ParamStr(0)返回全路径，并包含空格。实例程序使用 ParamStr()函数和 GetCommandLine()函数获取程序命令行参数的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  this->Edit1->Text=AnsiString(GetCommandLine());
  for(int i=0;ParamStr(i).Length()>0;i++)
  {
    this->ListBox1->Items->Add(ParamStr(i));
  }
}
```

说明，在测试时，可以从 C++Builder 菜单中选择 Run\Parameters 菜单。输入参数(-luo -bin)，如图 69-2 所示，再次运行程序，即可得到如图 69-1 所示的效果。



图 69-2

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 069 如何获取程序命令行参数”。

实例 070　如何析取动态链接库中的图标

实例目的

此实例主要介绍了如何在动态链接库文件、执行文件、图标文件中读取并保存图标的编程技术。实例程序执行后，效果如图 70-1 所示。

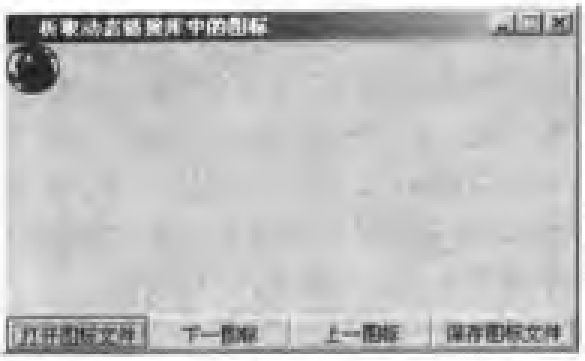


图 70-1

实现方法

Windows 操作系统的图标通常存放在文件 Shell32.dll 中，系统中的大多数程序的图标均来源于此，通过 API 函数 ExtractIcon()可以从中分别析取出这些图标。API 函数 ExtractIcon()也能从指定的 exe、icon 及其他含有图标的文件中析取图标，该函数的原型如下：

```
HICON ExtractIcon(
    HINSTANCE hInst,              //进程句柄
    LPCTSTR lpszExeFileName,      //含有图标的文件名
    UINT nIconIndex               //要提取的图标索引值，若 nIconIndex=-1，
                                  //则函数返回此文件所含的图标数目
);
```

实例程序使用 API 函数 ExtractIcon()从含有图标的文件中析取图标的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  this->OpenDialog1->Options.Clear();
  this->OpenDialog1->Filter= "执行文件 (*.exe)|*.exe|
动态链接库文件(*.dll)|*.dll|图标文件(*.ico)|*.ico";
  this->OpenDialog1->InitialDir="C:\\";
  if(OpenDialog1->Execute())
  { //获得打开的文件名
    StrFileName=OpenDialog1->FileName;
    //得到此文件所含的图标数目
    iCount=(int)ExtractIcon(HInstance,StrFileName.c_str(),-1);
    //根据 index 值，得到当前的图标，将索引图标的句柄赋给 Image1 显示此 icon
    Image1->Picture->Icon->Handle=
ExtractIcon(HInstance,StrFileName.c_str(),iIndex);
  }
}
```

实例程序保存从含有图标的文件中析取的图标的示例代码如下：

```
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//保存图标为图标文件
 if(this->SavePictureDialog1->Execute())
 {
   AnsiString StrPath=SavePictureDialog1->FileName;
   AnsiString StrFile=StrPath+".ico";
```

```
        Image1->Picture->SaveToFile(StrFile);
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 070 如何析取动态链接库中的图标”。

实例 071　如何设置 Edit 组件的文本选择范围

实例目的

此实例介绍了如何设置 Edit 组件的文本选择范围的编程技术。实例程序执行后，效果如图 71-1 所示。



图 71-1

实现方法

在 C++Builder 中使用 Edit 组件时，当把焦点转移到该组件时，组件中的内容将自动被全部选中，以蓝底白字显示；如果再按一个字符键，则选中的内容将被全部删除，这就是 Edit 组件的自动选择文本功能。这个功能是很有用的，它可以减少许多按键输入操作；如果需要取消这种自动选中文本功能，可以将该组件的 AutoSelect 属性值设置为 false。如果需要将该组件中的部分文本设置为选择状态（即蓝底白字显示），则必须处理 Edit 组件的 OnEnter 事件，通过代码的方式实现。示例代码如下：

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{  //设置从第四个字符开始的 16 个字符为选中状态
   this->Edit1->SelStart=3;
   this->Edit1->SelLength=16;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序

设置\实例 071 如何设置 Edit 组件的文本选择范围"。

实例 072 如何将小写金额转换成大写金额

实例目的

此实例介绍了如何将小写金额转换为大写金额的编程技术。实例程序执行后，效果如图 72-1 所示。



图 72-1

实现方法

在 C++Builder 中，金额的大小写转换实际上就是一个字符串转换的问题，只不过需要在转换的过程中注意数据类型的选择。例如：如果将数据类型设置为 Float，则在超过 10 万时，数据就可能出现不正确的情况，因此需要选择数据类型为 Double。以下是实例程序进行金额大小写转换的示例代码：

```
AnsiString TransferNumberToChinese(double fNumber,
 bool DisplayZero, int SpaceLength)
{  //将数值转换为大写金额
   //18 个单位
const String StrScale[18] = {"分","角","元","拾","佰","仟",
"万","拾","佰","仟","亿","拾","佰","仟","兆","拾","佰","仟"};
   //10 个数字
const String StrNumber[10] = {"零","壹","贰","叁","肆","伍","陆","柒","捌","
玖"};
      AnsiString StrTemp;
      StrTemp=FloatToStr(fNumber);
      StrTemp.Delete(StrTemp.Pos("."), 1);    //去掉小数点
      AnsiString StrRusult = "";
      for (int i = StrTemp.Length(); i > 0; i--)
      {
         if ( !DisplayZero && i > 2 && StrTemp[StrTemp.Length() - i + 1] - 48
== 0 )
            continue;
         StrRusult += StrNumber[StrTemp[StrTemp.Length()-i+1]-48];
         StrRusult += StrScale[i-1];
         StrRusult += AnsiString::StringOfChar(' ', SpaceLength);
      }
```

```
    return StrRusult;
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->Edit1->Text.Length()<1)
  {
    MessageBox(Handle,"请输入金额小写数字！","信息提示",MB_OK);
    return;
  }
  if(this->Edit1->Text.Length()>17)
  {
    MessageBox(Handle,"数据太大！","信息提示",MB_OK);
    return;
  }
  double fNumber=StrToFloat(this->Edit1->Text);
  this->Edit2->Text=TransferNumberToChinese(fNumber,true,0);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 072 如何将小写金额转换成大写金额”。

实例 073　如何将全角字符串转换成半角字符串

实例目的

此实例介绍了如何将全角字符串转换成半角字符串的编程技术。实例程序执行后，效果如图 73-1 所示。



图 73-1

实现方法

在 Windows 操作系统中，中文字符和全角字符均占两个字节，并且使用了 ASCII Chart 2 (codes 128-255)表示。据此可以判断用户输入的字符是否是中文字符或全角字符。实际上，全角字符的第一个字节总是被置为 163，而第二个字节则是相同半角字符编码加上 128（不包括空格）。如半角‘A’为 65，则全角‘A’则是 163（第一个字节）、193（第二个字节，128+65）。实例程序据此将全角字符串转换成半角字符串的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//将全角字符串转换成半角字符串
  int iLength=this->Edit1->Text.Length();
  if (iLength<1)
    return;
  AnsiString Str="";
  char *CharTemp=new char[iLength+1];
  StrPCopy(CharTemp,this->Edit1->Text);
  BYTE c1,c2;
  int i;
  for (i=0;i<iLength;i++)
  {
    c1=CharTemp[i];
    c2=CharTemp[i+1];
    if (c1==163)//判断是否为全角字符
    {
      Str=Str+AnsiString((char)(c2-128));
      i++;
      continue;
    }
    if (c1>163)//判断是否为文字
    {
      Str=Str+AnsiString((char)c1);
      Str=Str+AnsiString((char)c2);
      i++;
      continue;
    }
    if ((c1==161) && (c2==161))
    {//全角空格是一个特例，另加处理
      Str=Str+" ";
      i++;
      continue;
     }
    Str=Str+ AnsiString(char(c1));
  }
  this->Edit2->Text=Str;
  delete CharTemp;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 073 如何将全角字符串转换成半角字符串”。

实例 074 如何按照指定的有效位数转换数字

实例目的

此实例介绍了如何根据输入的数字和规定的有效位数进行四舍五入转换数字的编程

技术。实例程序执行后，效果如图 74-1 所示。

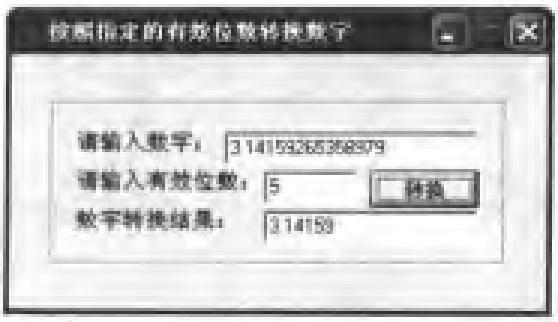


图 74-1

实现方法

在进行浮点数运算和显示时，通常会涉及到数字小数点后位数的保留问题。实例程序通过使用 C++流格式控制巧妙地解决了这个问题。它能够根据用户的要求实现保留规定的小数点后数字的长度，示例代码如下：

```
#include "strstream.h"
#include "fstream.h"
#include "iomanip.h"
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//按照指定的有效位数转换数字
  try
  {
    int iLength=this->Edit1->Text.Length();
    if(iLength<1||StrToInt(this->Edit2->Text)>12)
      return;
    char *StrBuffer=new char[iLength];
    ostrstream ofstr(StrBuffer,iLength);
    ofstr<<setiosflags(ios::fixed|ios::showpoint)<<setprecision(StrToInt
(this->Edit2->Text))<<StrToFloat(this->Edit1->Text)<<ends;
    this->Edit3->Text=StrBuffer;
  }
  catch(...)
  {
    MessageBox(Handle,"数字输入错误","信息提示",MB_OK);
    return;
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 074 如何按照指定的有效位数转换数字”。

实例 075 如何获取汉字的拼音索引

实例目的

此实例介绍了如何获取一个字符串汉字拼音索引的编程技术。实例程序执行后，效果如图 75-1 所示。

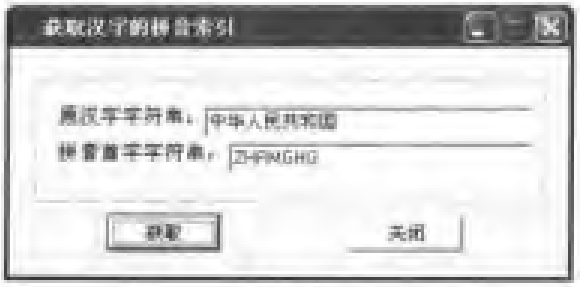


图 75-1

实现方法

一个汉字通常由两个字节组成，通过对这两个字节的预定义拼音值的判断，可以获取该汉字的拼音首字符。实例程序据此获取一个字符串汉字拼音索引的示例代码如下：

```
bool __fastcall In(int LowPosition, int HighPosition,int Value)
{
  return ((Value<=HighPosition)&&(Value>=LowPosition));
}
char __fastcall GetPYFirstChar(AnsiString HZ)
{
   WORD High =WORD(HZ[1])<<8;
   WORD Low =BYTE(HZ[2]);
   int n=High +Low ;
   if (In(0xB0A1,0xB0C4,n))
     return 'A';
   if (In(0XB0C5,0XB2C0,n))
     return 'B';
   if (In(0xB2C1,0xB4ED,n))
     return 'C';
   if (In(0xB4EE,0xB6E9,n))
     return 'D';
   if (In(0xB6EA,0xB7A1,n))
     return 'E';
   if (In(0xB7A2,0xB8C0,n))
     return 'F';
   if (In(0xB8C1,0xB9FD,n))
     return 'G';
```

```
    if (In(0xB9FE,0xBBF6,n))
      return 'H';
    if (In(0xBBF7,0xBFA5,n))
      return 'J';
    if (In(0xBFA6,0xC0AB,n))
      return 'K';
    if (In(0xC0AC,0xC2E7,n))
      return 'L';
    if (In(0xC2E8,0xC4C2,n))
      return 'M';
    if (In(0xC4C3,0xC5B5,n))
      return 'N';
    if (In(0xC5B6,0xC5BD,n))
      return 'O';
    if (In(0xC5BE,0xC6D9,n))
      return 'P';
    if (In(0xC6DA,0xC8BA,n))
      return 'Q';
    if (In(0xC8BB,0xC8F5,n))
      return 'R';
    if (In(0xC8F6,0xCBF0,n))
      return 'S';
    if (In(0xCBFA,0xCDD9,n))
      return 'T';
    if (In(0xCDDA,0xCEF3,n))
      return 'W';
    if (In(0xCEF4,0xD188,n))
      return 'X';
    if (In(0xD1B9,0xD4D0,n))
      return 'Y';
    if (In(0xD4D1,0xD7F9,n))
      return 'Z';
    return char(0);
}
AnsiString __fastcall GetPYString(AnsiString InputHZString)
{
   AnsiString TempString, ReturnString;
   AnsiString Str =InputHZString.Trim();
   for (int i=1; i<=Str.Length(); i++)
   {
     if (isprint(Str[i]))
     {
       ReturnString = ReturnString + Str[i];
     }
     else
     {
       TempString += Str[i];
```

```
            if (TempString.Length() == 2)
            {
              ReturnString = ReturnString + GetPYFirstChar(TempString);
              TempString = "";
            }
          }
        }
        return ReturnString;
}

void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->Edit1->Text.Length()<1)
    return;
  this->Edit2->Text=GetPYString(this->Edit1->Text);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 075 如何获取汉字的拼音索引”。

实例 076 如何设置输入方法

实例目的

此实例主要解决在应用程序的不同编辑栏中，程序会自动根据编辑栏的特殊需要输入模式，决定当前的输入方法，而不需要用户进行手工切换。当实例程序启动后，在用户英文名编辑栏中，程序关闭输入法；在用户中文名编辑栏中，程序打开五笔型输入法；在身份证号码编辑栏中，程序又关闭输入法。实例程序执行后，效果如图 76-1 所示。

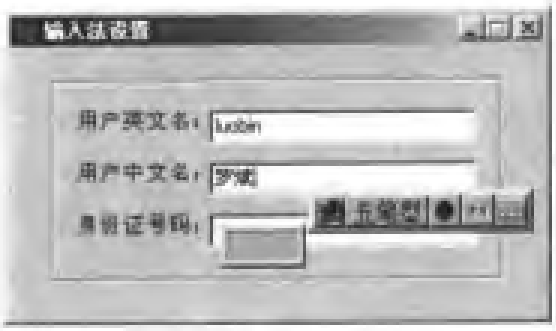


图 76-1

实现方法

在 C++Builder 的所有文字输入组件中，在属性面板上都已经加上了 ImeMode 输入模

式和 ImeName 输入法名称这两个属性，用来控制用户在数据输入时，使用的是中文输入模式，还是其他文种输入模式。实际上，在程序中也可以通过代码进行控制，示例代码如下：

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  this->Edit1->Text="";
  this->Edit2->Text="";
  this->Edit3->Text="";
  this->Edit1->ImeMode=imClose;
  this->Edit3->ImeMode=imClose;
  this->Edit2->ImeMode=imChinese;
  this->Edit2->ImeName="五笔型";
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 076 如何设置输入方法”。

实例 077　如何控制数据输入格式

实例目的

通常编辑栏允许接受键盘的任意输入，此编程技巧能够实现在编辑栏中对键盘的输入进行控制，例如只能输入数字，或者只能输入字母，或者只能输入一定长度和格式的字母数字等。当实例程序启动后，在用户姓名栏中可以输入任何字符；在出生日期栏中只能输入数字，并且年份只能输入 4 位数，月、日只能输入 2 位数；在电话号码栏中只能在规定的长度内输入数字，在身份证号码栏中只能输入小于 18 位的数字，最后单击“保存”按钮可以将输入结果输出。实例程序执行后，效果如图 77-1 所示。

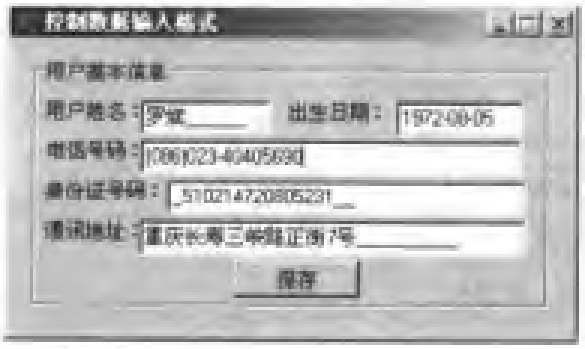


图 77-1

实现方法

在 C++Builder 中，实现以上控制数据输入格式的功能不用手工编写任何代码，完全依

靠 MaskEdit 控件实现。MaskEdit 控件在组件模板的 Additional 页面上。操作时，可以直接在表单上添加一个 MaskEdit 控件，然后单击属性编辑器的 EditMask 项右边的“...”按钮，弹出 Input Mask Editor 对话框，如图 77-2 所示，然后按照需要的格式进行设置即可。

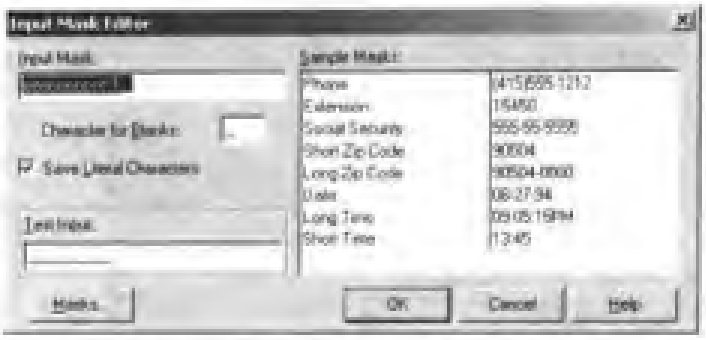


图 77-2

常用的 Mask 控制字符的含义如表 77-1 所示。

表 77-1

Mask 控制字符	意义
!	去掉输入数据开头的空白字符
>	使输入的字母都变成大写，直到遇到 Mask 字符<
<	使输入的字母都变成小写，直到遇到 Mask 字符>
<>	不限制输入字符的大小写形式
\	在需要添加特殊字符的前面加上此 Mask，如需要添加(), 可以改写成\(\)
L	允许输入英文字母，并且一定要输入
l	允许输入英文字母，不一定要输入
A	仅允许输入英文字母和数字，并且一定要输入
a	仅允许输入英文字母和数字，不一定要输入
C	允许输入任何字符，并且一定要输入
c	允许输入任何字符，不一定要输入
0	仅允许输入阿拉伯数字，并且一定要输入
9	仅允许输入阿拉伯数字，不一定要输入
#	允许输入阿位伯数字和正、负符号
:	分隔时间数据中的时、分、秒
/	分隔日期数据中的年、月、日
;	分隔 Mask 属性

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 077 如何控制数据输入格式”。

实例 078　如何控制编辑框中的输入法

实例目的

此实例主要介绍了如何在程序中控制编辑栏等输入栏位的输入模式和输入法。实例程序执行后，将自动搜索系统已经安装的输入法，当选择输入模式和中文输入法后，在“测试输入法”编辑栏中输入文字时，程序将使用选择的中文输入法进行输入，效果如图 78-1 所示。

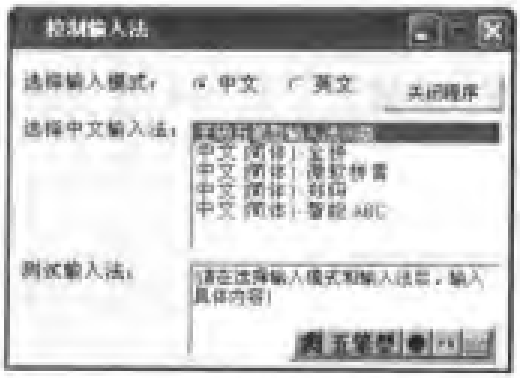


图 78-1

实现方法

在使用 C++Builder 开发应用程序时，常常需要根据业务的实际情况确定编辑栏的内容，据此确定输入法和输入模式，从而减少用户在各种输入法和输入模式间的切换次数。实际上，C++Builder 的大多数组件都具有 ImeMode 和 ImeName 两个属性，其中 ImeMode 表明当前的输入法，与中文有关的几个值分别是 imDisable、imClose、imOpen、imDontCare、imChinese。若将 ImeMode 属性设置为 imChinese，表明此控件的输入法为中文，而 ImeName 属性则反映的是何种中文输入法；将 ImeMode 设置为 imClose，则可以关闭已经打开的中文输入法，回到英文输入法状态。此外，C++Builder 有一个全局变量 Screen，其属性 Ime 反映的是系统已经安装的输入法，可以据此直接获取系统已经安装的输入法。下面是实例程序获取系统输入法和设置编辑栏输入法的实现代码。

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{//获取系统已经安装的输入法
  this->ListBox1->Items->Assign(Screen->Imes);
}
void __fastcall TForm1::OnEnterSetIme(TObject *Sender)
{ //设置编辑栏的输入法
  if(this->RadioButton1->Checked)
  {//中文输入法
   this->Memo1->ImeMode=imChinese;
   int iIndex=this->ListBox1->ItemIndex;
```

```
        AnsiString StrItems=this->ListBox1->Items->Strings[iIndex];
        this->Memo1->ImeName=StrItems.c_str();
      }
      if(this->RadioButton2->Checked)
      {//英文输入法
        this->Memo1->ImeMode=imClose;
      }
    }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 078 如何控制编辑框中的输入法”。

实例 079 如何设置回车键后的焦点

实例目的

此实例主要解决当用户按下回车键后，程序焦点自动落入下一个目标位置中。当实例程序启动后，在用户英文名编辑栏中按下回车键后，光标自动跳到用户中文名编辑栏；在用户中文名编辑栏中按下回车键后，光标自动跳到身份证号码编辑栏，在身份证号码编辑栏中按下回车键后，光标自动跳到用户英文名编辑栏。实例程序执行后，效果如图 79-1 所示。

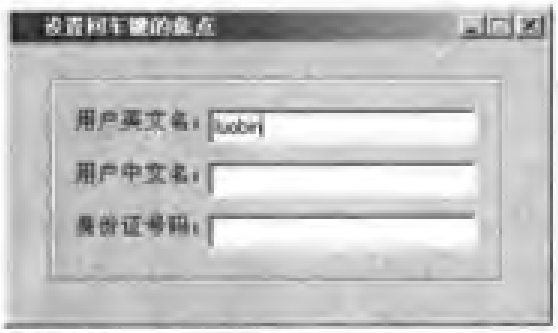


图 79-1

实现方法

在 DOS 操作系统下，每个输入框都必须键入回车键后才转移到下一输入框（鼠标单击除外），而在 Windows 操作系统下，这种输入框焦点的转移，默认情况下是通过 Tab 键来实现的。如果 Windows 应用程序的输入框焦点转换需要通过回车键实现，则通常应该处理表单的 OnKeyPress 事件。实例程序实现此技术的示例代码如下：

```
void __fastcall TForm1::OnKeyPress(TObject *Sender, char &Key)
{//处理 OnKeyPress 事件
  if(Key==13)
```

```
    {
      if(this->Edit1->Focused())
      {
        this->Edit2->SetFocus();
        return;
      }
      if(this->Edit2->Focused())
      {
        this->Edit3->SetFocus();
        return;
      }
      if(this->Edit3->Focused())
      {
        this->Edit1->SetFocus();
        return;
      }
    }
}
```

注意：需要在表单的构造函数或 OnCreate 事件中设置 KeyPreview 属性为 true。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 079 如何设置回车键后的焦点”。

实例 080　如何将项目保存到对象库

实例目的

此实例主要介绍了如何将对象添加到 C++Builder 对象库和如何从 C++Builder 对象库中提取对象，实现组件重复多次使用的目的。实例程序执行后，效果如图 80-1 所示。

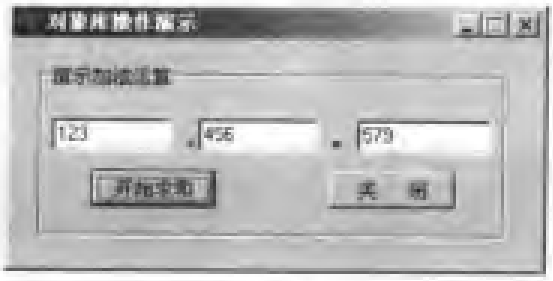


图 80-1

实现方法

C++Builder 允许用户将已经制作完成的 Forms、Dialogs、Projects、Data Modules

作为一个独立的对象保存在对象库中，供以后多次重复使用。它的操作主要分为两个阶段：一是将对象保存到对象库中；二是从对象库中提取对象。

将对象保存到对象库中的过程如下（以 Forms 为例）：在 Forms 上右击，在右键菜单中选择 Add to Repository...菜单项，弹出 Add to Repository...对话框，如图 80-2 所示。如果 Forms 列表框中有多个 Forms，则可以进行 Forms 选择，在 Title:栏中输入对象名称，在 Description:栏中输入对添加对象的描述，在 Page:下拉列表中选择添加对象的类型（包括 Forms、Dialogs、Projects、Data Modules 四种），在 Author:栏中输入作者名称，单击 OK 按钮即可。

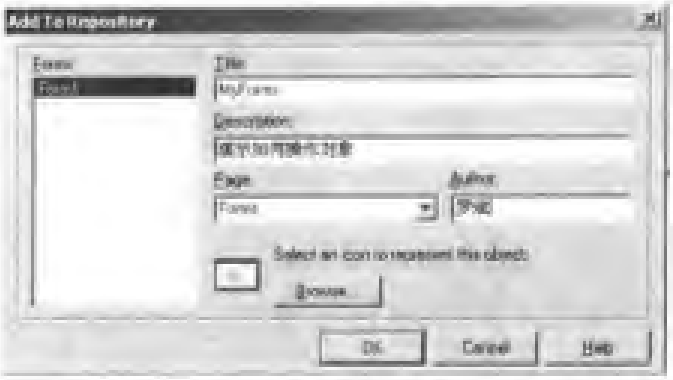


图 80-2

从对象库中提取对象的过程如下：选择菜单 File\New\Other...，弹出 New Item 对话框，单击 Forms 标签页，前一过程添加的对象 MyForms 出现在对象库中，如图 80-3 所示。双击 MyForms 对象，则该对象自动将上一过程创建的表单添加到当前项目中。



图 80-3

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序

设置\实例 080 如何将项目保存到对象库”。

实例 081　如何检测鼠标信息

实例目的

此实例介绍了如何检测鼠标信息的编程技术。实例程序执行后，效果如图 81-1 所示。



图 81-1

实现方法

在使用 Windows API 编程时，可以非常轻松地识别鼠标的左右键，然而在 C++Builder 中，这种事情反而不那么容易。因为 C++Builder 已经把 Alt、Ctrl、Shift 键和鼠标按键封装在一个 TShiftState 类型的集合中，要检测这些信息，必须处理 TShiftState 集合，该集合中取值的意义如下：ssShift（Shift 键被按下）；ssAlt（Alt 键被按下）；ssCtrl（Ctrl 键被按下）；ssLeft（鼠标左键被按下）；ssRight（鼠标右键被按下）；ssMiddle（鼠标中键被按下）；ssDouble（鼠标双击）。以下是实例程序解析 TShiftState 集合检测鼠标信息的示例代码。

```
void __fastcall TForm1::Image1MouseDown(TObject *Sender,
                TMouseButton Button, TShiftState Shift, int X, int Y)
{//解析TShiftState集合检测鼠标信息
 AnsiString StrInfo;
 StrInfo.printf("鼠标当前位置：(%d，%d)，同时",X,Y);
 if(Shift.Contains(ssShift))
 {
   StrInfo+="Shift键被按下！";
   MessageBox(Handle,StrInfo.c_str(),"信息提示",MB_OK);
 }
 if(Shift.Contains(ssCtrl))
 {
```

```
        StrInfo+="Ctrl 键被按下！";
        MessageBox(Handle,StrInfo.c_str(),"信息提示",MB_OK);
    }
    if(Shift.Contains(ssLeft))
    {
        StrInfo+="鼠标左键被按下！";
        MessageBox(Handle,StrInfo.c_str(),"信息提示",MB_OK);
    }
    if(Shift.Contains(ssRight))
    {
        StrInfo+="鼠标右键被按下！";
        MessageBox(Handle,StrInfo.c_str(),"信息提示",MB_OK);
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 081 如何检测鼠标信息”。

实例 082 如何读取鼠标信息

实例目的

此实例主要介绍如何侦测用户的鼠标输入情况，以便应用程序能够根据用户鼠标的不同输入作出不同的处理。当实例程序启动后，在图像上移动鼠标时，在编辑栏中会显示鼠标当前在图像上的位置，当按下或抬起鼠标左键、中键、右键时，在编辑栏中会给出相应的提示。实例程序执行后，效果如图 82-1 所示。



图 82-1

实现方法

当 C++Builder 应用程序在侦测到鼠标左键、中键、右键按下和抬起以及鼠标移动等动作时，通常会调用 OnMouseMove、OnMouseDown、OnMouseUp 等消息处理函数（对于鼠标单击、双击以及滚轮操作与此同理），在编程时通过对这些事件的处理，可以达到控制鼠

标的目的。鼠标移动、按下和抬起消息处理函数的声明如下：

```
void __fastcall TForm1::GetMouseMove(TObject *Sender,
 TShiftState Shift,int X, int Y)
void __fastcall TForm1::GetMouseDown(TObject *Sender,
 TMouseButton Button,TShiftState Shift, int X, int Y)
void __fastcall TForm1::GetMouseUp(TObject *Sender,
TMouseButton Button,TShiftState Shift, int X, int Y)
```

其中，参数 TObject *Sender 为鼠标动作所依存的组件；TMouseButton Button 为鼠标在按下或抬起时所对应的按钮（如 mbLeft、mbRight、mbMiddle）；TShiftState Shift 为鼠标在发生以上动作时，使用者又同时按下了键盘上的 Alt 、Ctrl 或 Shift 键；int X 为鼠标当前所在位置的横坐标；int Y 为鼠标当前所在位置的纵坐标。实例程序根据以上 3 类消息获取鼠标信息的示例代码如下：

```
void __fastcall TForm1::GetMouseDown(TObject *Sender,
TMouseButton Button,TShiftState Shift, int X, int Y)
{//处理 OnMouseDown 事件
   AnsiString StrMouseKey;
   switch(Button)
   {
   case mbLeft:
      StrMouseKey="鼠标左键被按下";
      break;
   case mbMiddle:
      StrMouseKey="鼠标中键被按下";
      break;
   case mbRight:
      StrMouseKey="鼠标右键被按下";
      break;
   }
   this->Edit1->Text=StrMouseKey;
}
void __fastcall TForm1::GetMouseUp(TObject *Sender,
TMouseButton Button,TShiftState Shift, int X, int Y)
{//处理 OnMouseUp 事件
   AnsiString StrMouseKey;
   switch(Button)
   {
   case mbLeft:
      StrMouseKey="鼠标左键抬起";
      break;
   case mbMiddle:
      StrMouseKey="鼠标中键抬起";
      break;
   case mbRight:
      StrMouseKey="鼠标右键抬起";
      break;
   }
   this->Edit1->Text=StrMouseKey;
}
```

```
void __fastcall TForm1::GetMouseMove(TObject *Sender,
TShiftState Shift,int X, int Y)
{//处理 OnMouseMove 事件
  AnsiString StrMove="横坐标: ";
  StrMove+=IntToStr(X);
  StrMove+="\t 纵坐标: ";
  StrMove+=IntToStr(Y);
  this->Edit1->Text=StrMove;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 082 如何读取鼠标信息”。

实例 083 如何实现鼠标拖动操作

实例目的

此实例主要介绍了如何在组件间实现鼠标拖动效果的编程技术。当实例程序启动后，在左边的编辑栏中拖动文字，则这段文字将自动从左边的编辑栏移动到右边的编辑栏，或者在右边的编辑栏中拖动文字，则这段文字将自动从右边的编辑栏移动到左边的编辑栏，效果如图 83-1 所示。

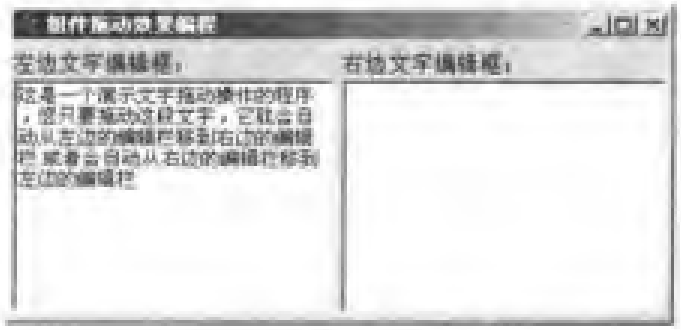


图 83-1

实现方法

C++Builder 组件库中的大多数可视化组件，几乎都具备鼠标拖动操作所需要的 DragMode 属性、OnDragDrop 事件和 OnDragOver 事件处理程序。DragMode 属性用来控制拖动组件时组件的反应，如果 DrapMode 属性设置成 dmAutomatic，则当用户在源组件上按下鼠标左键时，就会自动进入拖动的状态；OnDragOver 程序用来决定是否接受源组件的拖放项目，它使用 Accept 参数来判断接受与否，如果 Accept 设成 true，则接受；OnDrapDrop 事件是编写拖动完成后实际需要处理的动作，是真正的处理程序。OnDragOver、OnDrapDrop 事件的声明如下：

```
void __fastcall TForm1::OnDrapDrop(TObject *Sender,
 TObject *Source,int X, int Y)
void __fastcall TForm1::OnDragOver(TObject *Sender,
     TObject *Source, int X, int Y, TDragState State, bool &Accept)
```

实例程序实现文字从一个编辑栏拖动到另一个编辑栏的示例代码如下：

```
void __fastcall TForm1::TextFromRightToLeft(TObject *Sender,
     TObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept=true;
}
void __fastcall TForm1::FromRightToLeftText(TObject *Sender,
               TObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept=true;
}
void __fastcall TForm1::OnDragpDrop(TObject *Sender,
 TObject *Source,int X, int Y)
{
  if(this->Memo2->Text.Length()>0)
  {
    this->Memo1->Text=this->Memo2->Text;
    this->Memo2->Text="";
  }
}
void __fastcall TForm1::TwoDrapDrop(TObject *Sender,
TObject *Source,int X, int Y)
{
  if(this->Memo1->Text.Length()>0)
  {
    this->Memo2->Text=this->Memo1->Text;
    this->Memo1->Text="";
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 083 如何实现鼠标拖动操作”。

实例 084　如何限制鼠标活动范围

实例目的

此实例介绍了如何限制鼠标活动范围的编程技术。实例程序执行后，单击“限制鼠标

活动范围”按钮，则鼠标只能在第二个按钮上移动；单击“取消鼠标活动范围限制”按钮，则鼠标能够在所有范围自由移动，效果如图 84-1 所示。



图 84-1

实现方法

操作系统提供的 API 函数 ClipCursor(CONST RECT *lpRect)能够实现限制鼠标的活动范围，当参数 lpRect 为一个具体的矩形区域时，使用该函数就能够限制鼠标在该限制区域内活动，当参数 lpRect 为 0 时，则鼠标恢复自由活动。ClipCursor()函数的声明如下：

```
BOOL ClipCursor(
    CONST RECT *lpRect              //限制区域
  );
```

以下是实例程序使用该函数实现限制鼠标的活动范围的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //限制鼠标活动范围
  TRect MyArea=this->BitBtn2->BoundsRect;
  MapWindowPoints(Handle,0,(LPPOINT)(&MyArea),2);
  ClipCursor(&MyArea);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{  //取消鼠标活动范围限制
  ClipCursor(0);
}
```

注意，由于光标是一个共享资源，如果使用了 ClipCursor()函数限制了鼠标的活动范围，在程序退出时应该调用 ClipCursor(0)命令释放光标资源，以便于其他程序使用光标。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 084 如何限制鼠标活动范围”。

实例 085　如何读取键盘信息

实例目的

此实例主要介绍了如何侦测用户的键盘输入操作，以便程序根据用户的键盘输入作出相应的处理。当根据此技巧编写的实例程序启动后，在键盘上按下任一键，程序将给出按键的 ASCII 码，如果是复合键，程序会指出复合键名。实例程序执行后，效果如图 85-1 所示。

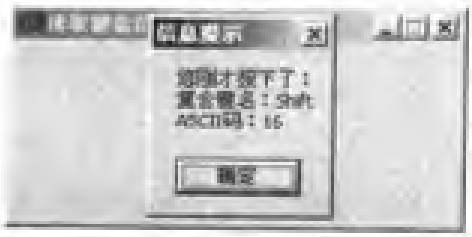


图 85-1

实现方法

C++Builder 在每个组件内都提供了 3 个事件来处理键盘的信息，它们分别是 OnKeyDown、OnKeyPress 和 OnKeyUp。OnKeyDown、OnKeyPress 对应键盘的按键被按下时的消息，OnKeyUp 对应按键抬起时的消息。这 3 个函数的声明如下：

```
void __fastcall TForm1::OnKeyDown(TObject *Sender, WORD &Key,TShiftState Shift)
void __fastcall TForm1::KeyPress(TObject *Sender, char &Key)
void __fastcall TForm1::OnKeyUp(TObject *Sender, WORD &Key,TShiftState Shift)
```

其中，参数 WORD &Key 是用户按下的键盘扫描码，参数 TShiftState Shift 用来判断用户是否同时按下 Shift、Ctrl 和 Alt 键，参数 TShiftState Shift 的可能值包括 ssShift、ssAlt、ssLeft、ssRight、ssMiddle、ssDouble。

注意，如果将 TForm 组件上的 KeyPreview 属性设置成 true，当 TForm 上任何可视化组件被激活且用户有按键操作时，将首先执行 TForm 的 OnKeyDown、OnKeyPress 和 OnKeyUp 消息响应函数，然后才去执行 TForm 上的被激活组件的 OnKeyDown、OnKeyPress 和 OnKeyUp 消息响应函数。

实例程序读取键盘信息的示例代码如下：

```
void __fastcall TForm1::FormOnKeyDown(TObject *Sender,
 WORD &Key,TShiftState Shift)
{
   AnsiString StrInfo="您刚才按下了：\n 复合键名：";
   if(Shift.Contains(ssAlt))
     StrInfo+="Alt";
   if(Shift.Contains(ssShift))
```

```
        StrInfo+="Shift";
    if(Shift.Contains(ssCtrl))
        StrInfo+="Ctrl";
    StrInfo+="\nASCII 码: ";
    StrInfo+=IntToStr(Key);
    MessageBox(NULL,StrInfo.c_str(),"信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第二部分 程序设置\实例 085 如何读取键盘信息”。

第三部分　文件操作

实例 086　如何复制文件

实例目的

文件复制操作是编写应用程序的基本操作，特别是在制作安装程序时，此实例主要介绍原始的 API 文件复制操作方法。当实例程序启动后，单击“浏览”按钮，弹出文件“打开”对话框，选择需要复制的源文件，然后在目标文件全路径栏中输入复制文件的目的地，单击“复制”按钮即可。如果文件复制操作成功，将给出文件复制操作成功提示，否则给出文件复制操作失败提示，效果如图 86-1 所示。



图 86-1

实现方法

使用系统 API 函数 CopyFile()可以实现文件的复制操作，该函数的声明如下：

```
BOOL CopyFile(
LPCTSTR lpExistingFileName,        //源文件全路径名
LPCTSTR lpNewFileName,             //目标文件全路径名
    BOOL bFailIfExists             //目标文件已经存在时处理标志
  );
```

注意：如果目标文件已经存在，且参数 BOOL bFailIfExists 为 true，不进行文件复制操作，复制文件操作失败；如果目标文件已经存在，且参数 BOOL bFailIfExists 为 false，则已经存在的目标文件将被覆盖，复制文件操作成功。在进行文件复制时，如果文件名缺少路径，则默认为当前路径，也即在当前路径下查找源文件并产生目标文件。函数 CopyFile()在源文件被使用时也能够进行正常的复制操作。实例程序使用系统 API 函数 CopyFile()实现文件复制操作的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  if(this->Edit1->Text.IsEmpty()||this->Edit2->Text.IsEmpty())
```

```
    return;
  if(CopyFile(this->Edit1->Text.c_str(),this->Edit2->Text.c_str(),false))
  {
    MessageBox(NULL,"文件复制操作成功！","信息提示",MB_OK);
  }
  else
  {
    MessageBox(NULL,"文件复制操作失败！","信息提示",MB_OK);
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 086 如何复制文件”。

实例 087 如何定制文件复制操作

实例目的

此实例介绍了如何直接通过读写文件方式进行文件复制操作的编程技术。实例程序执行后，效果如图 87-1 所示。

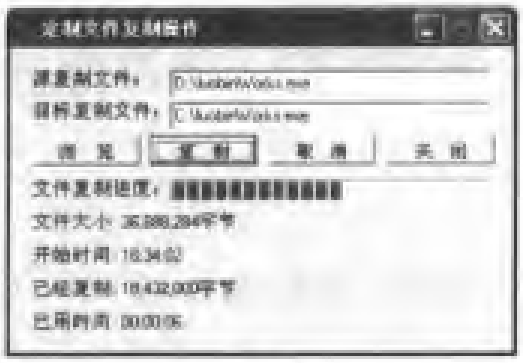


图 87-1

实现方法

Windows API 提供了 CopyFile()函数进行文件的复制操作，该函数可以直接进行单个文件的复制操作，但是并没有显示其文件复制操作过程。用户完全可以通过操作流式文件的方式编写出一个可以显示文件复制操作过程的 CopyFile()函数。流式文件操作有一个重要的 FILE 结构，该结构在 stdio.h 文件中的声明如下：

```
typedef struct {
int level;
unsigned flags;
char fd;
unsigned char hold;
```

```
int bsize;
unsigned char _FAR *buffer;
unsigned char _FAR *curp;
unsigned istemp;
short token;
} FILE;
```

FILE 结构包含了文件操作的基本属性，对文件的操作都要通过这个结构的指针进行，流式文件操作常用函数的功能如下：

fopen()	打开文件流
fclose()	关闭文件流
fputc()	写一个字符到文件流中
fgetc()	从文件流中读一个字符
fseek()	在文件流中定位到指定的字符
fputs()	写字符串到文件流
fgets()	从文件流中读一行或指定一个字符
fprintf()	按格式输出到文件流
fscanf()	从文件流中按格式读取
feof()	到达文件尾时返回真值
ferror()	发生错误时返回真值
rewind()	复位文件定位到文件开始处
remove()	删除文件
fread()	从流中读指定个数的字符
fwrite()	向流中写指定个数的字符
tmpfile()	生成一个临时文件流
tmpnam()	生成一个惟一的文件名

其中：fopen()函数的声明如下：

```
FILE *fopen(const char *filename,const char *mode);
```

参数 const char *filename 指向要打开的文件名；参数 const char *mode 表示打开状态的字符串，其取值如下："r" 以只读方式打开文件，"w" 以只写方式打开文件，"a" 以追加方式打开文件，"r+" 以读/写方式打开文件，如无文件出错，"w+" 以读/写方式打开文件，如无文件，则生成新文件。此函数返回一个 FILE 指针，所以声明一个 FILE 指针后不用初始化，而是用 fopen()函数来返回一个指针并与一个特定的文件相连，如果成功，返回 NULL。

fopen()函数可以使一个文本文件以文本模式或二进制模式打开，这两种方式的区别是：在文本模式中回车被当成一个字符 "\n"，而二进制模式认为它是两个字符 0x0D,0x0A；如果在文件中读到 0x1B，文本模式会认为这是文件结束符，也就是二进制模型不会对文件进行处理，而文本方式会按一定的方式对数据作相应的转换。系统默认的是以文本模式打开，可以修改全部变量_fmode 的值来修改这个设置，例如_fmode=O_TEXT；就设置默认打开

方式为文本模式；而_fmode=O_BINARY；则设置默认打开方式是二进制模式。也可以在模式字符串中指定打开的模式，如“rb”表示以二进制模式打开只读文件，“w+t”或“wt+”表示以文本模式打开读/写文件。

文件打开以后，在程序结束时一定要记得关闭打开的文件，否则可能会造成数据丢失。fclose()函数的功能就是用于关闭用 fopen()函数打开的文件，fclose()函数的声明如下：

```
int fclose(FILE *fp);
```

如果操作成功，返回值为 0，失败则返回 EOF。

fread()函数用于从文件流中读取指定个数的字符，该函数的声明如下：

```
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

参数 void *ptr 用于保存读取的数据，void*的指针可用任何类型的指针来替换，如 char*、int *等来替换；size_t size 是每块的字节数；size_t n 是读取的块数；FILE *stream 是字符串表示的文件名。如果成功，返回实际读取的块数（不是字节数），该函数一般用于二进制模式打开的文件中。

fwrite()函数用于向文件流中写指定的数据，该函数的声明如下：

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);
```

参数 const void *ptr 是要写入的数据指针，void*的指针可用任何类型的指针来替换，如 char*、int *等来替换；size_t size 是每块的字节数；size_t n 是要写的块数；FILE *stream 是字符串表示的文件名，如果操作成功，返回实际写入的块数（不是字节数），此函数一般用于二进制模式打开的文件中。

实例程序应用以上几个文件操作函数，通过流式方式直接读写文件进行文件复制操作的示例代码如下：

```
//FileName :Mainform.cpp
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#include <stdio.h>
#include <time.h>
#include "sys\stat.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
bool bExist;
const BufferSize=409600; //基本单位为 400KB
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->OpenDialog1->Execute())
  {
     AnsiString StrFileName=this->OpenDialog1->FileName;
     this->Edit1->Text=StrFileName;
  }
```

```
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   if(this->Edit1->Text.Length()<1||this->Edit2->Text.Length()<1)
    return;
   AnsiString FromFile=this->Edit1->Text;
   AnsiString ToFile=this->Edit2->Text;
   if (CopyFile(FromFile,ToFile))
   {
     MessageBox(Handle,"文件复制操作成功","信息提示",MB_OK);
   }
   else
   {
     bExist = false;
     DeleteFile(ToFile);
   }
}
bool TForm1::CopyFile(AnsiString FromFile,AnsiString ToFile)
{
   bool result(false);
   FILE *pFromFile,*pToFile;
   int NumRead,NumWritten;
   char buf[BufferSize];
   Screen->Cursor = crHourGlass;
   pToFile=fopen(ToFile.c_str(),"wb");
   pFromFile=fopen(FromFile.c_str(),"rb");
   try
   {
     AnsiString StartTime = Now();
     struct stat StatBuffer;
     fstat(fileno(pFromFile), &StatBuffer);
     int ModVal = StatBuffer.st_size % BufferSize;
     int MaxVal = StatBuffer.st_size / BufferSize;
     if (ModVal) MaxVal++;
     ProgressBar1->Max = MaxVal;
     float fSize = StatBuffer.st_size;
     TVarRec args[2] = {0,fSize};
     this->Label1->Caption = Format("文件大小: %.*n字节     ", args, 1);
     this->Label2->Caption = "开始时间: "
+FormatDateTime("hh:nn:ss",StartTime)+" ";
     bExist = false;
     ProgressBar1->Position = 0;
     do
     {
       if (ProgressBar1->Position<MaxVal-1)
       {
         NumRead = fread(&buf, sizeof(buf), 1, pFromFile);  //读文件数据
         NumWritten = fwrite(&buf, sizeof(buf), 1, pToFile);  //写文件数据
         fSize = BufferSize* (ProgressBar1->Position + 1);
       }
```

```
            else if (ModVal) {//当剩余的数据不足 400KB 时，需作特殊处理...
               char *Smallbuf;
               Smallbuf = (char *)SysGetMem(ModVal);
               NumRead = fread(Smallbuf, ModVal, 1, pFromFile);   // 读文件数据
               NumWritten = fwrite(Smallbuf, ModVal, 1, pToFile); // 写文件数据
               fSize += ModVal;
               SysFreeMem(Smallbuf);
               NumRead = 0;
            }
            ProgressBar1->Position++;
            TVarRec args[2] = {0,fSize};
            this->Label3->Caption = Format("已经复制: %.*n 字节       ", args, 1);
            this->Label4->Caption = "已用时间: "
 +FormatDateTime("hh:nn:ss",Now()-StartTime)+"    ";
            Application->ProcessMessages();
         }
         while (!bExist && (NumRead!=0 || NumRead==NumWritten));
         ProgressBar1->Position++;
         fclose(pToFile);
         fclose(pFromFile);
         result = !bExist;
      }
      catch(...)
      {
         fclose(pToFile);
         fclose(pFromFile);
         MessageDlg("在复制文件时出现错误，未能完成操作。",
 mtWarning, TMsgDlgButtons() << mbOK, 0);
      }
      Screen->Cursor = crDefault;
      return result;
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  if (MessageDlg("正在复制文件，是否要取消操作？", mtWarning,
 TMsgDlgButtons() << mbYes << mbNo, 0)==mrYes)
    bExist = true;
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
   this->Close();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 087 如何定制文件复制操作”。

实例 088　如何更名文件

实例目的

此实例主要介绍在 C++Builder 中如何进行文件更名操作。当实例程序启动后，单击"浏览"按钮，弹出文件"打开"对话框，选择需要更名的源文件，然后在新文件名栏中输入新文件名（包括路径），单击"更名"按钮即可。如果文件更名操作成功，将给出文件更名操作成功提示，否则给出文件更名操作失败提示。效果如图 88-1 所示。

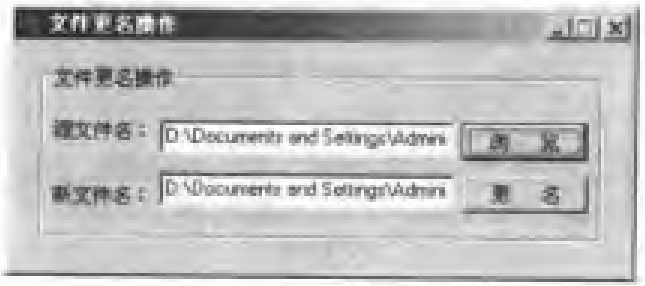


图 88-1

实现方法

在 C++Builder 中，使用函数 RenameFile()可以实现文件的更名操作，该函数的声明如下：

```
extern PACKAGE bool __fastcall RenameFile(
const AnsiString OldName,  //旧文件名
const AnsiString NewName   //新文件名
);
```

实例程序使用函数 RenameFile()实现文件更名操作的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  if(this->Edit1->Text.IsEmpty()||this->Edit2->Text.IsEmpty())
    return;
  if(RenameFile(this->Edit1->Text,this->Edit2->Text))
  {
    MessageBox(NULL,"文件更名操作成功!","信息提示",MB_OK);
  }
  else
  {
    MessageBox(NULL,"文件更名操作失败!","信息提示",MB_OK);
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘"源代码\第三部分 文件

操作\实例 088 如何更名文件"。

实例 089 如何更改文件扩展名

实例目的

此实例介绍了如何更改文件扩展名的编程技术。实例程序运行后，效果如图 89-1 所示。



图 89-1

实现方法

在计算机操作系统中，通常使用文件扩展名对文件分门别类，这样便于对文件进行集中管理。例如：对于以.txt 扩展名结尾的文件，用户一看就知道它是一个文本文件，因此可以在系统中约定，凡是以.txt 扩展名结尾的文件，都可以使用记事本程序打开；对于以.doc 扩展名结尾的文件，则可以使用写字板或 Word 程序打开。在 DOS 系统下，用户可以在命令行下通过更名文件实现对文件扩展名的更改，在 Windows 操作系统下，则可以使用 ChangeFileExt()函数来更改文件扩展名。ChangeFileExt()函数的声明如下：

```
extern PACKAGE AnsiString __fastcall ChangeFileExt(const AnsiString
FileName, const AnsiString Extension);
```

其中，参数 const AnsiString FileName 表示要更改扩展名的文件全路径名，参数 const AnsiString Extension 表示文件新的扩展名。

需要注意的是：在使用 ChangeFileExt()函数更改文件扩展名后，它并没有将文件扩展名真正地修改，而只是返回了一个文件名字符串，要想真正地将文件名更改，则需要再使用文件更名函数 RenameFile()。在实例程序中使用 ChangeFileExt()函数和 RenameFile()函数更改文件扩展名的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //更改文件扩展名
  if(this->Edit1->Text.Length()<1)
    return;
  AnsiString StrFileName=ChangeFileExt(this->Edit1->Text,this->Edit2->Text);
  if(RenameFile(this->Edit1->Text,StrFileName))
  {
    MessageBox(Handle,"文件扩展名修改操作成功！","信息提示",MB_OK);
```

```
    }
    else
    {
      MessageBox(Handle,"文件扩展名修改操作失败！","信息提示",MB_OK);
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分　文件操作\实例 089 如何更改文件扩展名”。

实例 090　如何删除文件

实例目的

此实例主要介绍了如何使用操作系统的 API 函数删除一个文件。当实例程序启动后，单击“浏览”按钮，弹出文件“打开”对话框，选择需要删除的源文件，单击“删除”按钮；如果文件删除操作成功，将给出文件删除操作成功提示，否则给出文件删除操作失败提示。效果如图 90-1 所示。



图 90-1

实现方法

使用系统 API 函数 DeleteFile()可以实现文件的删除操作，该函数的声明如下：

```
BOOL DeleteFile(
    LPCTSTR lpFileName              //删除文件名
  );
```

如果使用该函数删除文件操作失败，函数返回值为 0，否则函数返回值为一个非 0 数字。

此外，C++Builder 也提供了一个删除文件的函数 DeleteFile()，该函数的声明如下：

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

其中，参数 const AnsiString FileName 为删除文件名，如果不指定具体路径，则系统以当前路径作为该文件名的路径，如果文件删除操作成功，该函数的返回值为真，否则函数返回值为假。

实例程序使用函数 DeleteFile()实现文件删除操作的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   if(this->Edit1->Text.IsEmpty())
     return;
   if(DeleteFile(this->Edit1->Text))
   {
      MessageBox(NULL,"删除文件操作成功！","信息提示",MB_OK);
   }
   else
   {
      MessageBox(NULL,"删除文件操作失败！","信息提示",MB_OK);
   }
}
```

注意：如果文件为只读文件，则该文件不能被直接删除。需要使用_rtl_chmod()函数先将该文件的只读属性取消才能执行删除操作，关于_rtl_chmod()函数的使用说明请参考本书的其他部分。删除只读文件的示例代码如下：

```
#include "dos.h"
#include "io.h"
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   if(this->Edit1->Text.IsEmpty())
     return;
   if(DeleteFile(this->Edit1->Text))
   {
      MessageBox(NULL,"删除文件操作成功！","信息提示",MB_OK);
   }
   else
   {  //删除只读文件
     _rtl_chmod(this->Edit1->Text.c_str(),1,FA_ARCH);
      if(DeleteFile(this->Edit1->Text))
     {
       MessageBox(NULL,"删除文件操作成功！","信息提示",MB_OK);
     }
     else
     {
       MessageBox(NULL,"删除文件操作失败！","信息提示",MB_OK);
     }
   }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 090 如何删除文件”。

实例 091　如何删除文件夹

实例目的

此实例介绍了如何删除文件夹的编程技术。实例程序运行后，在删除文件夹名编辑栏中输入指定的文件夹，单击“删除文件夹”按钮，即可删除指定的文件夹，效果如图 91-1 所示。

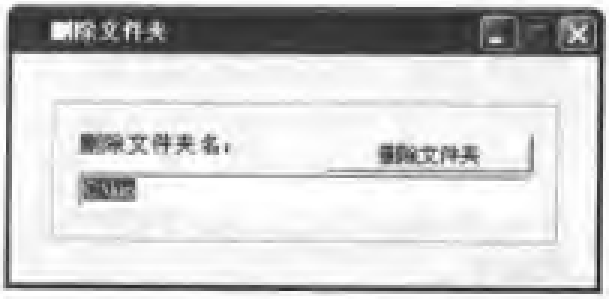


图 91-1

实现方法

在 C++Builder 中，删除文件夹使用函数 RemoveDir()，该函数的声明如下：

```
extern PACKAGE bool __fastcall RemoveDir(const AnsiString Dir);
```

其中，参数 const AnsiString Dir 为指定的文件夹名，如果删除操作成功，则该函数的返回值为 true，否则为 false。如果删除文件夹操作没有执行成功，则可能该文件夹下还有文件或子文件夹没有被删除，所以，要确保删除文件夹操作成功，必须确保删除文件夹为空。也可以像实例程序中那样使用 DOS 命令删除一个空的文件夹，示例代码如下：

```
void __fastcall TForm1::Bitbtn1Click(TObject *Sender)
{
  AnsiString StrFolder=this->Edit1->Text;
  if(StrFolder.Length()<1)
    return;
  if(!DirectoryExists(StrFolder))
  {
    MessageBox(Handle,"删除文件夹不存在","信息提示",MB_OK);
    return;
  }
  AnsiString StrOpt="rd  ";
  StrOpt+=StrFolder+"  ";
  int iResult=system(StrOpt.c_str());
  switch(iResult)
  {
    case ENOMEM:
      MessageBox(Handle,"系统内存或资源不足！","信息提示",MB_OK);
      break;
```

```
        case ENOENT:
          MessageBox(Handle,"路径或文件错误！","信息提示",MB_OK);
          break;
        case ENOEXEC:
          MessageBox(Handle,"执行文件格式错误！","信息提示",MB_OK);
          break;
        default:
          break;
      }
}
```

如果需要一次性删除多层文件夹或包括文件的文件夹，可以使用文件操作结构实现，关于文件操作结构的使用，请参考本书的其他部分，也可以使用 DOS 命令 deltree 一次性实现删除包含文件的多层文件夹。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 091 如何删除文件夹”。

实例 092　如何选择文件夹

实例目的

此实例介绍了如何在文件夹选择对话框中选择文件夹的编程技术。实例程序执行后，效果如图 92-1 所示。

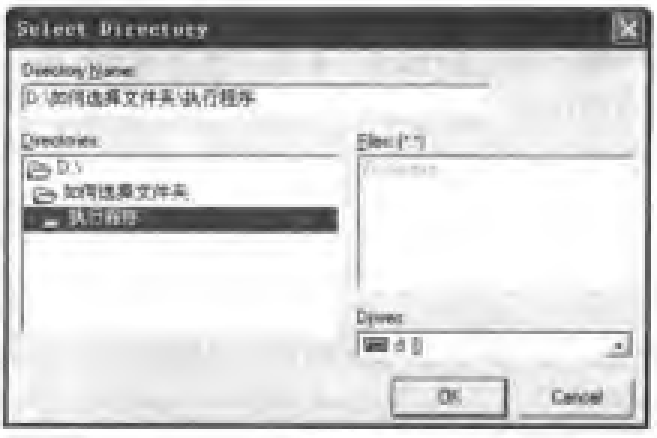


图 92-1

实现方法

在 Windows 操作系统的文件打开对话框中，一般情况下，用户只能选择文件而不能选择文件夹，而 C++Builder 提供的函数 SelectDirectory()则可以实现文件夹的选择，该函数有两种重载形式，其中一种重载形式的声明如下：

```
extern PACKAGE bool __fastcall SelectDirectory(AnsiString &Directory,
TSelectDirOpts Options, int HelpCtx);
```

实例程序使用函数 SelectDirectory()实现选择文件夹的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  AnsiString StrFolderName='';
  SelectDirectory(StrFolderName,
TSelectDirOpts() << sdAllowCreate << sdPerformCreate << sdPrompt,0);
  if(StrFolderName.Length()>0)
    MessageBox(Handle,StrFolderName.c_str(),'信息提示',MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 092 如何选择文件夹”。

实例 093　如何创建多层文件夹

实例目的

此实例介绍了如何一次性创建多层文件夹的编程技术。实例程序运行后，在多层文件夹名编辑栏中输入指定的多层文件夹，单击“创建多层文件夹”按钮，即可创建指定的多层文件夹，效果如图 93-1 所示。



图 93-1

实现方法

文件夹实质上就是通常所说的目录，在 Windows 操作系统出现之前，人们只有目录的概念，出现 Windows 操作系统之后，为了让人们更容易理解目录的含义，才取了一个文件夹的名词。在 C++Builder 中，可以使用函数 CreateDir()创建文件夹，该函数的声明如下：

```
extern PACKAGE bool __fastcall CreateDir(const AnsiString Dir);
```

其中，参数 const AnsiString Dir 为创建的文件夹名，如果不包含具体的路径，则系统以当前路径为参考路径。如果创建成功，该函数的返回值为 true，否则为 false。但是该函数只能创建一层文件夹，而不能创建多层文件夹。例如：要创建文件夹 C:\luo\luo01，则必

须使用代码 CreateDir("C:\luo")创建上一层文件夹后，才能使用代码 CreateDir("C:\luo\luo01")创建下一层文件夹。实例程序则巧妙地使用一个循环，使用函数 CreateDir()一次创建多层关系的文件夹。下面是在实例程序中一次性创建任意层次文件夹的实现代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->Edit1->Text.Length()<1)
    return;
  if(DirectoryExists(this->Edit1->Text))
  {
    MessageBox(Handle,"多层文件夹已经存在！","信息提示",MB_OK);
    return;
  }
   AnsiString StrFolder=this->Edit1->Text;
   AnsiString StrDir,StrTemp;
   for(int i=1;i<=StrFolder.Length();i++)
   {
      StrTemp=StrFolder.SubString(i,1);
      if(StrTemp=="\\")
      {
        if(StrDir.Length()>=1)
          CreateDir(StrDir);
      }
      StrDir+=StrTemp;
   }
   if(CreateDir(StrDir))
    MessageBox(Handle,"创建多层文件夹操作成功！","信息提示",MB_OK);
   else
    MessageBox(Handle,"创建多层文件夹操作失败！","信息提示",MB_OK);
}
```

注意：在使用函数 CreateDir()创建文件夹前，可以使用 DirectoryExists()函数判断要创建的文件夹是否已经存在，如果已经存在，则没有再创建的必要。此外，曾经在 DOS 系统下创建目录的函数 mkdir()，在 Windows 操作系统环境下同样可以使用，在应用时，需要包括头文件：#include"dir.h"。函数 mkdir()的声明如下：

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

其中参数 const char *path 为创建的目录，该函数在 C++Builder 中的变种为：extern PACKAGE void __fastcall MkDir(const AnsiString S);，使用效果完全相同。C++Builder 也提供了一个一次性创建多层文件夹的函数 ForceDirectories()，该函数的声明如下：

```
extern PACKAGE bool __fastcall ForceDirectories(AnsiString Dir);
```

不过，该函数的参数 AnsiString Dir 必须写成 C:\\luo\\luo01\\luo02 的形式才能创建，似乎不符合一般情况下人们的文件夹命名习惯。同时，使用函数 ForceDirectories()创建多层文件夹时，必须包含头文件：#include"FileCtrl.hpp"。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件

操作\实例 093 如何创建多层文件夹”。

实例 094　如何自动获取文件夹和文件

实例目的

此编程技巧主要介绍了不使用系统 API 函数等原始方式获取系统驱动器、文件夹和文件的方式而自动获取文件夹和文件的方法。当实例程序启动后，驱动器组合框中自动显示系统已经安装的驱动器，选择一个驱动器，在文件夹列表框中显示该驱动器的当前级次的所有文件夹，在文件夹列表框中单击其中一个文件夹，在文件列表框中显示该文件夹中的所有文件。效果如图 94-1 所示。



图 94-1

实现方法

C++Builder 组件模板的 Win3.1 标签页上，存放了获取系统驱动器（DriverComboBox）、文件夹（DirectoryListBox）和文件（FileListBox）的组合框和列表框组件，直接使用即可获取系统文件和文件夹的信息。示例代码如下：

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
   this->DirectoryListBox1->FileList=this->FileListBox1;
   this->DriveComboBox1->DirList=this->DirectoryListBox1;
}
void __fastcall TForm1::GetPath(TObject *Sender)
{
   AnsiString StrPath;
   StrPath=this->FileListBox1->FileName;
   MessageBox(NULL,StrPath.c_str(),'信息提示',MB_OK);
}
```

注意：使用时必须把这几个组件的关系连接起来，否则它使用默认的当前状态。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分　文件操作\实例 094 如何自动获取文件夹和文件”。

实例 095 如何搜索指定目录下的文件

实例目的

此实例介绍了如何在指定目录下搜索特定文件的编程技术。实例程序运行后，在“查找文件夹”编辑栏中输入指定的文件夹，在“查找文件”编辑栏中输入指定的文件名或部分文件名，单击“查找”按钮，即可显示出符合条件的文件，效果如图 95-1 所示。



图 95-1

实现方法

Windows 操作系统为用户搜索文件默认提供了一个查找功能，它能够执行文件、文件夹及用户的搜索，但它是在系统外部执行的，不能在用户程序中直接使用。而 C++Builder 则提供了 FindFirst()和 FindNext()两个函数和一个结构 TSearchRec，三者结合使用可以非常简单地实现在指定目录下搜索文件。结构 TSearchRec 的定义如下：

```
struct TSearchRec
{
    int Time;                          //日期
    int Size;                          //大小
    int Attr;                          //属性
    AnsiString Name;                   //名称
    int ExcludeAttr;                   //特别属性
    int FindHandle;                    //查找句柄
    _WIN32_FIND_DATAA FindData;        //附加信息
};
```

其中参数 int Attr 的可能取值包括：aReadOnly（只读文件）、faHidden（隐藏文件）、faSysFile（系统文件）、faVolumeID（卷标文件）、faDirectory（目录文件）、faArchive（归档文件）、faAnyFile（任意文件）。

实例程序使用 FindFirst()函数、FindNext()函数和结构 TSearchRec 实现在指定目录搜索文件的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  this->ListBox1->Clear();
  AnsiString StrFolder=this->Edit1->Text;
  AnsiString StrFile=this->Edit2->Text;
  if(StrFolder.Length()<1||StrFile.Length()<1)
    return;
  TSearchRec StrRecord;
  int iAttributes = 0;
  iAttributes |=faHidden;
  if (FindFirst(StrFolder+"\\"+StrFile, iAttributes, StrRecord) == 0)
  {
    this->ListBox1->Items->Add(StrFolder+"\\"+StrRecord.Name);
    while (FindNext(StrRecord) == 0)
    {
     this->ListBox1->Items->Add(StrFolder+"\\"+StrRecord.Name);
    }
    FindClose(StrRecord);
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 095 如何搜索指定目录下的文件”。

实例 096　如何搜索指定目录下的隐藏文件

实例目的

此实例介绍了如何在指定目录下搜索隐藏文件的编程技术。实例程序运行后，在查找文件夹编辑栏中输入指定的文件夹，单击“查找”按钮，即可显示出该文件夹下的隐藏文件，效果如图 96-1 所示。

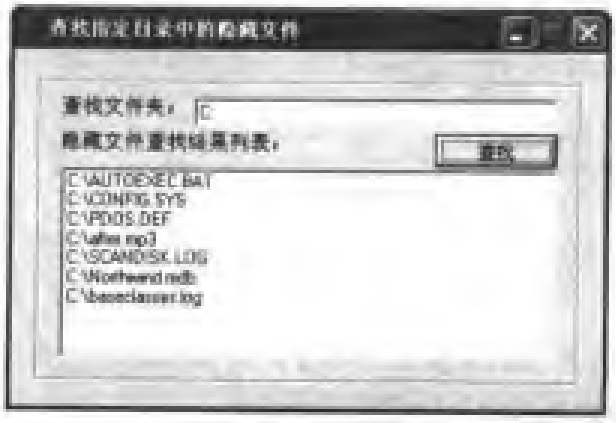


图 96-1

实现方法

隐藏文件是操作系统实现文件保护的一种特殊机制，但是在某种情况下，用户需要了解隐藏文件。显示系统中的隐藏文件有多种方法，实例程序是使用_rtl_chmod()实现在查找文件中确定隐藏文件。使用_rtl_chmod()函数需要包括 io.h 和 dos.h 两个头文件。该函数的声明如下：

```
int _rtl_chmod(const char *path, int func [, int attrib]);
```

其中，参数 const char *path 表示文件全路径名，参数 int func [, int attrib]表示获取或设置文件属性，如果参数 func 为 0 表示获取文件属性，如果参数 func 为 1 表示设置文件属性。参数 int attrib 或返回值代表文件的属性值，其值的含义如下：FA_RDONLY（只读属性）、FA_HIDDEN（隐藏属性）、FA_SYSTEM（系统属性）、FA_LABEL（卷标属性）、FA_DIREC（目录属性）、FA_ARCH（归档属性）。

实例程序使用 FindFirst()函数、FindNext()函数、_rtl_chmod()函数和结构 TSearchRec 实现在指定目录搜索隐藏文件的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  this->ListBox1->Clear();
  AnsiString StrFolder=this->Edit1->Text;
  if(StrFolder.Length()<1)
    return;
  TSearchRec StrRecord;
  int iAttributes = 0;
  iAttributes |=faHidden;
  if (FindFirst(StrFolder+"\\*.*", iAttributes, StrRecord) == 0)
  {
    if(_rtl_chmod(StrRecord.Name.c_str(),0)==FA_HIDDEN)
    {
      _rtl_chmod(StrRecord.Name.c_str(),1,FA_ARCH);
      this->ListBox1->Items->Add(StrFolder+"\\"+StrRecord.Name);
    }
    while (FindNext(StrRecord) == 0)
    {
      _rtl_chmod(StrRecord.Name.c_str(),1,FA_ARCH);
      this->ListBox1->Items->Add(StrFolder+"\\"+StrRecord.Name);
    }
    FindClose(StrRecord);
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 096 如何搜索指定目录下的隐藏文件”。

实例 097　如何搜索文本文件中的字符串

实例目的

此实例主要介绍了如何在不显示文本文件的情况下，对文本文件中的字符串进行搜索。当实例程序启动后，单击“浏览”按钮，选择一个文本文件打开，在“目标字符串”栏中输入要搜索的字符串，单击“搜索”按钮，程序立即执行搜索操作，并将结果显示在“搜索结果”栏中。效果如图 97-1 所示。



图 97-1

实现方法

C++Builder 的 Memo 组件提供了一个非常好的功能，它把获取的文本文件的内容存放在 Lines 中，而 Lines 是一个 TString 类型的对象，利用它包含的 Count 和 Strings 属性可以非常容易地取得文本文件的行数和每一行的字符串，通过比较字符串，就可以搜索到文本文件中的目标字符串。实例程序据此实现在不显示文本文件的情况下，对文本文件中的字符串进行搜索的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
 try
 {
  this->Memo1->Lines->LoadFromFile(this->Edit1->Text);
 }
 catch(Exception&e)
 {
   MessageBox(NULL,"文本文件不能正确打开","信息提示",MB_OK);
   return;
 }
 int iCount=0,i=0;
 bool bFlag=false;
 char p1[255],p2[255],*p3,*p4;
 for(i=0;i<this->Memo1->Lines->Count;i++)
 {
   StrPCopy(p1,this->Memo1->Lines->Strings[i]);
   p3=&(p1[0]);
```

```
        while((*p3)!='\0')
        {
          StrPCopy(p2,this->Edit2->Text);
          p4=&(p2[0]);
          if(strlen(p4)>strlen(p3))
             break;
          bFlag=true;
          while((*p4)!='\0'&&(*p3)!='\0')
          {
            if(*p4!=*p3)
            {
              bFlag=false;
              p3++;
              break;
            }
            p4++;
            p3++;
          }
          if(bFlag==true)
           iCount++;
       }
      }
      if(iCount)
      {
         AnsiString StrResult;
         StrResult="找到";
         StrResult+=IntToStr(iCount);
         StrResult+="个相同的字符串";
         this->Edit3->Text=StrResult;
      }
      else
      {
         this->Edit3->Text="没有找到相同的字符串";
      }
  }
```

说明：实例程序不支持中文查找。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 097 如何搜索文本文件中的字符串”。

实例 098 如何替换文本文件中的字符串

实例目的

此实例主要介绍了如何灵活地应用 AnsiString 提供的成员函数，实现对文本文件中字

符串的替换操作。当实例程序启动后，单击“选择文件”按钮，弹出文件“打开”对话框，选择一个文本文件打开，然后在“旧字符串”编辑栏中输入要替换的字符串，在“新字符串”编辑栏中输入目标字符串，单击“开始替换”按钮，即可实现替换功能。效果如图 98-1 所示。



图 98-1

实现方法

在 C++Builder 中，几乎每个 VCL 组件都有 String 类型的属性，然而由于 C++采用字符串指针的形式来处理字符串，它并没有 String 一样的数据类型，而是提供了一个 AnsiString 来模拟 String 类型，其原形定义在 C:\Program Files\Borland\CBuilder6\Include\VCL\ dstring.h 文件中。AnsiString 类型非常有用，它能够实现字符串的查找、定位、替换、删除等多种操作，常用的 AnsiString 方法（函数）的作用如表 98-1 所示。

表 98-1

函数原形	作用
Int AnsiCompare(const AnsiString&rhs)	与另一个字符串进行比较，如果相等则返回 0，小于则返回-1，大于则返回 1
int AnsiCompareIC(const AnsiString&rhs)	与 AnsiCompare()功能类似，该函数没有大小写之分
char* AnsiLast()	取出字符串中的最后一个字符
char* _fastcall c_str()	返回一个指向本字符串 AnsiString 的字符指针
void Delete(int index,int count)	从字符串的第 index 个字符开始，往后删除 count 个字符
void Insert(const AnsiString&str,int index)	从字符串的第 index 个字符后面开始，插入字符串 str
bool IsEmpty()	判断字符串是否为空，如果是则返回 true，否则返回 false
bool IsPathDelimiter(int index)	判断字符串的第 index 个字符是否为反斜线\字符，如果是则返回 true，否则返回 false
int LastDelimiter(const AnsiString&delimiters)	返回此字符串中最后一个反斜线\字符的位置

续表

函数原形	作用
int Length()	返回此字符串的长度
AnsiString LowerCase()	将字符串中的所有字符都转换成小写字母
int Pos(const AnsiString&substr)	返回字符串 substr 在此字符串中的位置
void SetLength(int newLength)	强制设定此字符串的长度为 newLength，如果 newLength 值比原字符串长，则多出来的位置会被补上空的；如果比原字符串短，则多出的字符串会被截掉
AnsiString StringofChar(char ch,int count)	此字符串将会被填入 count 个 ch 字符
AnsiString SubString(int index,int count)	从此字符串的第 index 个字符开始取出 count 个字符
double ToDouble()	把此字符串转换成 double 类型，此字符串不允许有数字及小数点以外的字符出现
int ToInt()	把此字符串转换成 int 类型，此字符串将不允许有数字以外的字符出现
int ToIntDef(int dv)	把此字符串转换成 int 类型，如果此字符串中有数字以外的字符出现，则返回数据 dv
AnsiString Trim()	把字符串头尾所包括的空白字符去掉
AnsiString TrimLeft()	把此字符串头部所包括的空白字符去掉
AnsiString TrimRight()	把此字符串尾部所包括的空白字符去掉
AnsiString UpperCase()	把字符串内的所有小写字母都转换成大写字母
AnsiString CurrToStr(Currency value)	把货币类型的数值数据 value 转换成字符串
AnsiString CurrToStrF(Currency value, TStringFloatFormat format,int digits)	把货币类型的数值数据 value 转换成 format 格式、小数点 digits 位的字符串
AnsiString FloatToStrF(long double value,TStringFloatFormat format,int precision,int digits)	把浮点运算 float 类型的数值转换成字符串
AnsiString LoadStr(int ident)	把 Resource File 的第 ident 段字符串截入到此字符串内

AnsiString 类型所支持的运算符的意义如表 98-2 所示。

表 98-2

运算符	意义	示例
=	赋值运算	Str="This is a string"
+	合并	Str3=Str1+Str2
==	逻辑运算符等于	if(Str1==Str2)…
!=	逻辑运算符不等于	if(Str1!=Str2)…
<	逻辑运算符小于	if(Str1<Str2)…
>	逻辑运算符大于	if(Str1>Str2)…
<=	逻辑运算符小于等于	if(Str<=Str2)…
>=	逻辑运算符大于等于	if(Str>=Str2)…
[]	字符串数组索引运算符	char c=Str1[10]

下面是实例程序进行字符串替换的示例代码：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   AnsiString StrOld=this->Edit2->Text;
   AnsiString StrNew=this->Edit1->Text;
   AnsiString Str=this->RichEdit1->Text;
   int iPos=Str.Pos(StrOld);
   Str.Delete(iPos,StrOld.Length());
   Str.Insert(StrNew,iPos);
   this->RichEdit1->Text=Str;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分　文件操作\实例 098 如何替换文本文件中的字符串”。

实例 099　如何将长文件名转换为短文件名

实例目的

此实例介绍了如何将长文件名转换为短文件名的编程技术。实例程序运行后，效果如图 99-1 所示。

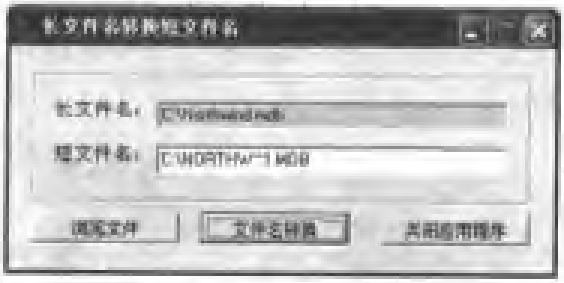


图 99-1

实现方法

在 DOS 操作系统下，文件名命名规定为 8.3 格式，即主文件名长度不超过 8 个字符，文件扩展名不超过 3 个字符。随着 Windows 操作系统的出现，文件名命名方式突破了 8.3 格式的制约，可以命名长达上百个字符的长文件名。然而，通过 Windows 操作系统的 DOS 窗口显示长文件名时，还是按照 DOS 的命名行方式显示文件，对于超过 8 个字符的长文件名，显示时则是在后面加一个“~”符号和一个数字符号显示。用户可以使用 Windows 的 API 函数 GetShortPathName()实现这种长文件名以短文件名形式显示的效果。GetShortPathName()函数的声明如下：

```
DWORD GetShortPathName(
    LPCTSTR lpszLongPath,          //长文件名
```

```
    LPTSTR lpszShortPath,          //短文件名
    DWORD cchBuffer                //缓冲大小
);
```

使用函数 GetShortPathName()将长文件名转换为短文件名显示的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//将长文件名转换为短文件名
  if(this->Edit1->Text.Length()<1)
    return;
  char ShortFileName[MAX_PATH];
  GetShortPathName(this->Edit1->Text.c_str(),ShortFileName,MAX_PATH);
  this->Edit2->Text=ShortFileName;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 099 如何将长文件名转换为短文件名”。

实例 100 如何在文件打开对话框中选择多个文件

实例目的

许多 MP3 播放器都可以一次性地选择多个 MP3 文件进行连续播放，此实例主要介绍这种一次性选择多个文件进行操作的原理。当实例程序启动后，单击“选择文件”按钮，弹出文件打开对话框，在文件打开对话框中选择多个文件并打开，这些文件名就显示在程序主界面中，效果如图 100-1 所示。



图 100-1

实现方法

在 C++Builder 中操作 Dialogs 标签页下的 OpenDialog 组件时，设置该组件的属性 ofAllowMultiSelect，就可以在文件打开对话框中选择多个文件，这些文件的文件名保存在 Strings 成员变量中，析取之即可取得选择的多个文件名。实例程序从文件打开对话框中提取多个文件的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  OpenDialog1->Options.Clear();
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist;
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2;
  if (OpenDialog1->Execute())
  {
    for (int I = 0; I < OpenDialog1->Files->Count; I ++)
    {
      this->ListBox1->Items->Add(OpenDialog1->Files->Strings[I]);
    }
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 100 如何在文件打开对话框中选择多个文件”。

实例 101　如何定制文件打开和关闭对话框

实例目的

在 C++Builder 中，文件打开和文件关闭对话框是以组件的形式提供，它们的属性可以在 Object Inspector 中进行设置，此实例主要介绍如何在程序中用代码的方式对文件打开和文件关闭对话框的风格样式进行定制。当实例程序启动后，单击“打开文件”按钮，弹出定制的文件打开对话框，在标题栏显示定制对话框的标题为“我的文件打开对话框”；初始化文件夹为 C:\，而非默认的“我的文档”文件夹；在“文件类型”下拉列表框中显示定制的两类文件 Text files 和 C++ files。在单击“文件保存”按钮后，也将显示定制的文件保存对话框标题，同时将保存文件的扩展名定义为 TXT，效果如图 101-1 所示。



图 101-1

实现方法

文件打开对话框和文件保存对话框属于操作系统默认提供的标准对话框，由操作系统自动支持。在 C++Builder 组件面板的 Dialogs 标签页下有 10 个这样的标准对话框，直接使用之即可。但是，也可以根据特定情况的需要，对这些标准对话框进行定制，例如，定制文件打开对话框的标题栏、文件名过滤器、初始定位文件夹等。下面是实例程序定制文件打开和保存对话框的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//定制文件打开对话框
   this->OpenDialog1->Options.Clear();
   this->OpenDialog1->Filter=  "Text  files  (*.txt)|*.TXT|C++  files
(*.cpp)|*.cpp";
   this->OpenDialog1->Title="我的文件打开对话框";
   this->OpenDialog1->InitialDir="C:\\";
   if(this->OpenDialog1->Execute())
   {
      this->RichEdit1->Lines->LoadFromFile(this->OpenDialog1->FileName);
   }
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//定制文件保存对话框
   this->SaveDialog1->Options.Clear();
   this->SaveDialog1->Title="我的文件保存对话框";
   this->SaveDialog1->InitialDir="C:\\";
   this->SaveDialog1->DefaultExt="txt";
   if(this->SaveDialog1->Execute())
   {
      AnsiString StrFileName=this->SaveDialog1->FileName;
      this->RichEdit1->Lines->SaveToFile(StrFileName);
   }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 101 如何定制文件打开和关闭对话框”。

实例 102 如何修改文件打开对话框的按钮标题

实例目的

此实例介绍了如何修改操作系统公共文件打开对话框按钮标题的编程技术。实例程序执行后，效果如图 102-1 所示。



图 102-1

实现方法

C++Builder 组件面板的 Dialogs 标签页上的组件封装了 Windows 常用的对话框。这些对话框在所有的应用程序中显示时，风格都是一致的。如果需要定制这些对话框的显示风格，可以通过属性对话框设置属性实现，也可以处理这些对话框的 OnShow()事件进行定制。以下是实例程序修改文件打开对话框按钮标题的示例代码。

```
void __fastcall TForm1::OpenDialog1Show(TObject *Sender)
{//修改文件打开对话框按钮标题
  HWND  hWndDialog=GetParent(this->OpenDialog1->Handle);
  HWND  hWndButton =GetDlgItem(hWndDialog, 1);
  SetWindowText(hWndButton, "新增");
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 102 如何修改文件打开对话框的按钮标题”。

实例 103　如何使用查找与替换对话框

实例目的

查找与替换对话框与一般的公共对话框不同，程序员必须对查找与替换对话框的有关事件进行处理后，才能实现它们所具有的功能。此实例主要介绍了如何在文本文件中实现完全意义上的查找与替换功能。实例程序执行后，打开一个文本文件，单击“替换”按钮后，弹出“替换”对话框，在“查找内容”与“替换为”编辑栏中输入相关内容后，单击“替换”按钮，程序将立即执行替换操作，效果如图 103-1 所示。



图 103-1

实现方法

查找与替换对话框属于 Windows 操作系统的公共对话框，直接调用之即可使用，而不需要创建 Form。但是在实现其查找与替换功能时，必须对其事件进行处理后，才能完成相应的功能。下面是实例程序实现查找与替换对话框的具体步骤（查找（FindDialog）与替换对话框（ReplaceDialog）组件位于组件模板的 Dialogs 标签页下）。

（1）在主窗体的头文件中定义两个宏和一个函数，用于处理查找与替换对话框和文本编辑器产生的消息。代码如下：

```
typedef enum {
    frtFind,
    frtReplace,
    frtRepAll
} FRT ;
#define GETSEL(ichS,ichE)
SendMessage(Memo1->Handle,EM_GETSEL,(WPARAM)&ichS,(LPARAM)&ichE)
#define SETSEL(ichS,ichE)
SendMessage(Memo1->Handle,EM_SETSEL, (WPARAM) ichS, (LPARAM) ichE)
BOOL FWholeWord(char* sz,char* szStart,char* szEnd,int cch)
{
if (sz!=szStart&&*(sz-1)!=' ' &&*(sz-1)!='\t'&&*(sz-1)!=','
&&*(sz-1)!='.'&&*(sz-1)!='\r')
    {
     return FALSE;
    }
if (sz+cch!=szEnd&&*(sz+cch)!=' '&&*(sz +cch)!='\t'
&&*(sz+cch)!=','&&*(sz+cch)!='.'&&*(sz+cch)!='\r')
    {
     return FALSE;
    }
    return TRUE;
}
```

（2）在查找对话框（FindDialog1）中添加 OnFind 事件处理函数，该函数及调用函数

的实现代码如下：

```
void __fastcall TForm1::OnFind(TObject *Sender)
{
  SetFocus();
  FindReplace(FindDialog1->FindText.c_str(), "", frtFind,
      FindDialog1->Options. Contains(frDown),
      FindDialog1->Options.Contains(frMatchCase),
      FindDialog1->Options.Contains(frWholeWord));
}
void TForm1::FindReplace(char * szFind, char * szReplace,
FRT frt, BOOL fDown, BOOL fMatchCase, BOOL fWholeWord)
{
  switch (frt)
  {
    case frtFind:
      Find(szFind, fDown, fMatchCase, fWholeWord,FALSE);
      break;
    case frtReplace:
      Find(szFind, fDown, fMatchCase, fWholeWord,FALSE);
      Replace (szReplace);
      break;
    case frtRepAll:
      ReplaceAll(szFind, szReplace, fMatchCase, fWholeWord);
   }
}
bool TForm1::Find(char * szFind, BOOL fDown, BOOL fMatchCase,
 BOOL fWholeWord, BOOL fNoScroll)
{
   char * szText;
   UINT ichStart;
   UINT ichEnd;
   UINT cch;
   int dch;
   int dchComp;
   char * szTerm;
   char * sz;
   UINT cchFind;
   BOOL fFound = FALSE;
   int dchOnFind;
   szText=Memo1->Lines->GetText();
   cchFind=strlen(szFind);
   GETSEL(ichStart,ichEnd);
   cch=strlen(szText);
   if(ichEnd-ichStart==cchFind&&!strnicmp(szText+ichStart,szFind,cchFind))
     dchOnFind=1;
   else
     dchOnFind=0;
   if(fDown)
   {
    dch=1;
```

```
        dchComp=0;
        sz=szText+ichStart+dchOnFind;
        szTerm=szText+cch;
       }
       else
       {
         dch=-1;
         dchComp=1;
         sz=szText+ichStart-dchOnFind;
         szTerm=szText;
       }
       for(;sz+dchComp!=szTerm; sz+=dch)
       {
         if(tolower(*sz)==tolower(*szFind)&&
    (fMatchCase?!strncmp(sz,szFind,cchFind):!strnicmp(sz,szFind,cchFind))
                        &&(!fWholeWord||FWholeWord(sz,szText,szTerm,cchFind)))
         {
           SendMessage(Memo1->Handle,EM_SETSEL,sz-szText,sz-szText+cchFind);
           if (!fNoScroll)
              SendMessage(Memo1->Handle,EM_SCROLLCARET,0,0L);
           fFound = TRUE;
           break;
         }
       }
       if(!fNoScroll&&!fFound)
          MessageBox(NULL, szFind, "字符串没有找到",MB_OK);
       return fFound;
    }
```

（3）在替换对话框（ReplaceDialog1）中添加 OnFind 事件处理函数，该函数的实现代码如下：

```
    void __fastcall TForm1::OnReplaceFind(TObject *Sender)
    {
      SetFocus();
      if(ReplaceDialog1->Options.Contains(frFindNext))
         FindReplace(ReplaceDialog1->FindText.c_str(),"", frtFind,
                        ReplaceDialog1->Options.Contains(frDown),
                        ReplaceDialog1->Options.Contains(frMatchCase),
                        ReplaceDialog1->Options.Contains(frWholeWord));

    }
```

（4）在替换对话框（ReplaceDialog1）中添加 OnReplace 事件处理函数，该函数及调用函数的实现代码如下：

```
    void __fastcall TForm1::Replace(TObject *Sender)
    {
     SetFocus ();
     if(ReplaceDialog1->Options. Contains (frReplace))
       FindReplace(ReplaceDialog1->FindText.c_str(),
```

```
                ReplaceDialog1->ReplaceText. c_str(),frtReplace,
                ReplaceDialog1->Options. Contains(frDown),
                ReplaceDialog1->Options. Contains(frMatchCase),
                ReplaceDialog1->Options. Contains(frWholeWord));
  else if(ReplaceDialog1->Options. Contains (frReplaceAll))//全部替换
    FindReplace(ReplaceDialog1->FindText.c_str(),
                ReplaceDialog1->ReplaceText.c_str(), frtRepAll,
                ReplaceDialog1->Options. Contains(frDown),
                ReplaceDialog1->Options. Contains(frMatchCase),
                ReplaceDialog1->Options. Contains(frWholeWord));
}
void TForm1::Replace(char * szReplace)
{
   SendMessage(Memo1->Handle,EM_REPLACESEL,0,(LPARAM)(LPSTR)szReplace);
}
void TForm1::ReplaceAll(char * szFind,
char * szReplace, bool fMatchCase, bool fWholeWord)
{
  UINT ichStart;
  UINT ichEnd;
  GETSEL(ichStart,ichEnd);
  SETSEL(0,0);
  while(Find(szFind, TRUE, fMatchCase, fWholeWord,TRUE))
  {
   Replace (szReplace);
  }
  SETSEL(ichStart, ichEnd);
}
```

（5）为“查找”按钮添加单击事件处理函数，代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  this->FindDialog1->Execute();
}
```

（6）为“替换”按钮添加单击事件处理函数，代码如下：

```
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  this->ReplaceDialog1->Execute();
}
```

最后，编译执行程序即可。注意：在以上代码中，所有查找与替换功能的实现代码是直接使用的 Win32 SDK 代码，其实在 Memo 和 RichEdit 组件中，提供了非常简便的查找与替换功能（在本书的其他部分中有所介绍），可以直接使用。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 103 如何使用查找与替换对话框”。

实例 104 如何获取和设置当前目录

实例目的

此实例介绍了在 C++Builder 中获取和设置当前工作目录的编程技术。实例程序执行后，效果如图 104-1 所示。



图 104-1

实现方法

Windows 操作系统 API 提供了 GetCurrentDirectory()和 SetCurrentDirectory()两个函数获取和设置当前程序的工作目录。这两个函数的声明如下：

```
DWORD GetCurrentDirectory(
    DWORD nBufferLength,          //当前工作目录字符串大小
    LPTSTR lpBuffer               //当前工作目录字符串
   );
BOOL SetCurrentDirectory(
    LPCTSTR lpPathName            //新的当前工作目录字符串
   );
```

在 C++Builder 提供的文件和目录操作函数中，它提供了以下几个函数对目录进行操作。

（1）DirectoryExists()函数用于判断目录是否已经存在。该函数的声明如下：

```
extern PACKAGE bool __fastcall
DirectoryExists(const System:: AnsiString Name);
```

其中，参数 const System:: AnsiString Name 为要检测的目录名。如果要检测的目录存在则返回 true，否则返回 false。

（2）CreateDir()函数用于建立子目录。该函数的声明如下：

```
extern PACKAGE bool __fastcall CreateDir(const System::AnsiString Dir);
```

其中，参数 const System::AnsiString Dir 为要创建的子目录名字。如果成功则返回 true，否则返回 false。

（3）RemoveDir()函数用于删除一个目录。该函数的声明如下：

```
extern PACKAGE bool __fastcall RemoveDir(const AnsiString Dir);
```

其中，参数 const AnsiString Dir 为要删除的目录名。如果成功则返回 true，否则返回 false。

（4）GetCurrentDir()函数用于获取当前的工作目录。该函数的声明如下：

```
extern PACKAGE AnsiString __fastcall GetCurrentDir();
```

该函数的返回值即为取得的当前目录名。

（5）SetCurrentDir()函数用于设置当前的工作目录。该函数的声明如下：

```
extern PACKAGE bool __fastcall SetCurrentDir(const AnsiString Dir);
```

其中，参数 const AnsiString Dir 为要新设置的当前目录。该函数如果操作成功则返回 true，否则返回 false。

以下是实例程序使用 SetCurrentDir()函数和 GetCurrentDir()函数设置和获取当前工作目录的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//获取当前工作目录
  this->Edit1->Text=GetCurrentDir();
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//设置当前工作目录
  if(this->Edit2->Text.Length()<1)
    return;
  SetCurrentDir(this->Edit2->Text);
  MessageBox(Handle,"设置当前目录操作成功！","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 104 如何获取和设置当前目录”。

实例 105　如何判断文件类型

实例目的

此实例介绍了如何判断文件类型的编程技术。实例程序执行后，效果如图 105-1 所示。



图 105-1

实现方法

通过系统 API 函数 SHGetFileInfo()和文件信息结构 SHFILEINFO 的综合使用能够对一

个系统注册文件的类型进行判断。系统 API 函数 SHGetFileInfo()能够获取文件、文件夹、目录、驱动器根目录的信息，该函数的声明如下：

```
DWORD_PTR SHGetFileInfo(
      LPCTSTR pszPath,              //文件/目录全路径名
      DWORD dwFileAttributes,       //文件属性
      SHFILEINFO *psfi,             //文件信息结构变量
      UINT cbFileInfo,              //文件信息结构变量大小
      UINT uFlags                   //特殊标志
);
```

该函数的返回值与 uFlags 标志有关，它能够决定文件是 Exe、Com、Bat 或其他非执行文件等。SHFILEINFO 结构的声明如下：

```
typedef struct _SHFILEINFO{
     HICON hIcon;                   //文件图标
     int   iIcon;                   //系统图像列表中的图标索引
     DWORD dwAttributes;            //文件属性
     TCHAR szDisplayName[MAX_PATH]; //文件名
     TCHAR szTypeName[80];          //文件描述
} SHFILEINFO;
```

以下是实例程序使用 SHGetFileInfo()和文件信息结构 SHFILEINFO 对文件的类型进行判断的示例代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //判断文件类型
  if(this->Edit1->Text.Length()<1)
    return;
  AnsiString StrTypeScript=this->Edit1->Text+"是一个: ";
  SHFILEINFO SFileInfo;
  SHGetFileInfo(this->Edit1->Text.c_str(),NULL,&SFileInfo,
sizeof(SHFILEINFO),SHGFI_TYPENAME);
  StrTypeScript+=SFileInfo.szTypeName;
  MessageBox(Handle,StrTypeScript.c_str(),"信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 105 如何判断文件类型”。

实例 106 如何获取系统所有文件类型

实例目的

此实例介绍了如何通过注册表查找系统所有注册文件类型的编程技术。实例程序执行后，效果如图 106-1 所示。



图 106-1

实现方法

在操作系统注册表的主键 HKEY_CLASSES_ROOT 下带“.”的子键为系统中所有注册文件的扩展名类型，通过查找这些子键就可以获取系统支持的文件类型。在 C++Builder 中，除了可以使用 Registry.hpp 中提供的 TRegistry 类读写注册表外，也可以直接使用系统 API 注册表操作函数操作注册表。常用的系统 API 操作注册表的函数如表 106-1 所示。

表 106-1

函数名	说明
RegCloseKey	关闭应用程序与特定注册表对象之间的连接
RegConnectRegistry	允许应用程序与远程计算机上的注册表建立连接
RegCreateKeyEx	创建一个新的注册表子键，可以进行安全性、选项及类别设置
RegDeleteKey	删除一个已经打开的注册表子键
RegDeleteValue	删除一个已经打开的注册表子键的指定键值
RegEnumKeyEx	从指定键或子键开始枚举所有的子键，每次调用返回一个对象，直到返回 ERROR_NO_MORE_ITEMS 为止。同时还检索类名、最近一次修改的时间以及对象的名字
RegEnumValue	枚举特定键或子键内的所有键值
RegFlushKey	将对注册表进行的任何修改立即写入注册表
RegGetKeySecurity	检索指定注册表对象的安全属性，如果用户有足够的权限也可以设置对象安全属性
RegLoadKey	在 HKEY_USERS 或 HKEY_LOCAL_MACHINE 中创建子键，新子键的信息保存在文件中，文件名以参数的形式传递给 RegLoadKey
RegNotifyChangeKeyValue	通知系统在指定对象被修改或者对象属性被修改时将消息通知此函数的调用者，如果对象被删除则不传递任何消息，调用者应用其中的一个事件句柄来处理消息表
RegOpenKeyEx	打开一个注册表对象，此函数返回的句柄将传递给其他需要注册表句柄做参数的注册表函数
RegQueryInfoKey	获取指定注册表键的信息
RegQueryMultipleValues	获取注册表键的键值数据和类型列表

续表

函数名	说明
RegQueryValueEx	获取注册表键的指定键值数据和类型
RegReplaceKey	这个函数主要用于备份、恢复注册表和故障恢复
RegRestoreKey	恢复多个注册表对象
RegSaveKey	将注册表键或子键的内容保存到文件
RegSetKeySecurity	设置指定注册表键的安全属性
RegSetValueEx	设置指定注册表键值的数据和类型
RegUnLoadKey	从注册表中删除指定对象

实例程序使用系统注册表 API 操作函数获取系统所有文件类型的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//获取系统所有文件类型
   this->ListBox1->Clear();
   HKEY hKey;
   if(ERROR_SUCCESS==RegOpenKeyEx(HKEY_CLASSES_ROOT,"",0,
KEY_ENUMERATE_SUB_KEYS,&hKey))
   {
     DWORD dwIndex=0;
     char Name[100];
     SHFILEINFO shSmall;
     while(RegEnumKey(hKey,dwIndex++,Name,100)==ERROR_SUCCESS)
     {
       if(Name[0]!='.')
         continue;
       ZeroMemory(&shSmall,sizeof(shSmall));
       SHGetFileInfo(Name,FILE_ATTRIBUTE_NORMAL,&shSmall,sizeof(shSmall),
SHGFI_USEFILEATTRIBUTES|SHGFI_DISPLAYNAME|
SHGFI_TYPENAME|SHGFI_ICON|SHGFI_SMALLICON);
       AnsiString StrFileType="";
       StrFileType+="文件扩展名：";
       StrFileType+=Name;
       StrFileType+="      文件类型：";
       StrFileType+=shSmall.szTypeName;
       this->ListBox1->Items->Add(StrFileType);
     }
     ::RegCloseKey(hKey);
   }
   MessageBox(Handle,"获取系统所有文件类型操作成功！","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 106 如何获取系统所有文件类型”。

实例 107　如何获取和设置文件属性

实例目的

此实例介绍了如何获取和设置文件属性的编程技术。实例程序执行后，效果如图 107-1 所示。



图 107-1

实现方法

通常一个文件的属性包括系统、隐藏、只读、归档等几大类。文件属性实际上是一个二进制的标志值，它可以进行二进制与（&）运算。通过适当的二进制运算就可以取得和设置文件属性。常用文件属性标志值意义如表 107-1 所示。

表 107-1

属性标志值	标志意义
FILE_ATTRIBUTE_ARCHIVE	归档文件，标明文件已经归档
FILE_ATTRIBUTE_HIDDEN	隐藏文件，平常在文件列表中看不见
FILE_ATTRIBUTE_NORMAL	表示文件没有设置其他属性，这个标志只能单独使用
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	表示文件不会被收录索引
FILE_ATTRIBUTE_OFFLINE	远程存储
FILE_ATTRIBUTE_READONLY	只读文件，不能被写入或删除
FILE_ATTRIBUTE_SYSTEM	系统文件，具有排他性
FIL FILE_ATTRIBUTE_TEMPORARY	临时存储文件

在 Windows API 函数中，通过 GetFileAttributes()函数和 SetFileAttributes()函数可以获取和设置文件属性。获取文件属性的 GetFileAttributes()函数的声明如下：

```
DWORD GetFileAttributes(
    LPCTSTR lpFileName          // 文件名
);
```

如果该函数操作成功，返回值为文件属性值，否则为错误代码，可以调用错误查看函数 GetLastError()进行查看。设置文件属性的 API 函数 SetFileAttributes()的声明如下：

```
BOOL SetFileAttributes(
       LPCTSTR lpFileName,          // 文件名
       DWORD dwFileAttributes       // 文件属性标志值
     );
```

以下是实例程序使用以上两个函数获取和设置文件属性的示例代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//获取文件属性
   this->CheckBox1->Checked=false;
   this->CheckBox2->Checked=false;
   this->CheckBox3->Checked=false;
   this->CheckBox4->Checked=false;
   AnsiString StrFileName=this->Edit1->Text;
   if(StrFileName.Length()<1)
     return;
  DWORD dwAttrs = GetFileAttributes(StrFileName.c_str());
  if (dwAttrs & FILE_ATTRIBUTE_READONLY)
  {
    this->CheckBox1->Checked=true;
  }
  if (dwAttrs & FILE_ATTRIBUTE_SYSTEM)
  {
    this->CheckBox2->Checked=true;
  }
  if (dwAttrs&FILE_ATTRIBUTE_HIDDEN)
  {
    this->CheckBox3->Checked=true;
  }
  if (dwAttrs & FILE_ATTRIBUTE_ARCHIVE)
  {
    this->CheckBox4->Checked=true;
  }
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//设置文件属性
   AnsiString StrFileName=this->Edit1->Text;
   if(StrFileName.Length()<1)
     return;
  DWORD dwAttrs = GetFileAttributes(StrFileName.c_str());
  SetFileAttributes(StrFileName.c_str(),0);
  if (this->CheckBox3->Checked)
  {
    SetFileAttributes(StrFileName.c_str(),dwAttrs|FILE_ATTRIBUTE_HIDDEN);
  }
  dwAttrs = GetFileAttributes(StrFileName.c_str());
  if (this->CheckBox1->Checked)
  {
   SetFileAttributes(StrFileName.c_str(),dwAttrs|FILE_ATTRIBUTE_READONLY);
  }
  dwAttrs = GetFileAttributes(StrFileName.c_str());
   if (this->CheckBox2->Checked)
```

```
    {
      SetFileAttributes(StrFileName.c_str(),dwAttrs|FILE_ATTRIBUTE_SYSTEM);
    }
    dwAttrs = GetFileAttributes(StrFileName.c_str());
     if (this->CheckBox3->Checked)
    {
      SetFileAttributes(StrFileName.c_str(),dwAttrs|FILE_ATTRIBUTE_HIDDEN);
    }
    dwAttrs = GetFileAttributes(StrFileName.c_str());
     if (this->CheckBox4->Checked)
    {
      SetFileAttributes(StrFileName.c_str(),dwAttrs|FILE_ATTRIBUTE_ARCHIVE);
    }
    MessageBox(Handle,'设置文件属性操作成功!','信息提示',MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 107 如何获取和设置文件属性”。

实例 108　如何获取和设置文件时间属性

实例目的

此实例介绍了如何获取和设置文件时间属性的编程技术。实例程序运行后，效果如图 108-1 所示。



图 108-1

实现方法

在 Windows 操作系统中，文件除了具有隐藏、归档、只读和系统属性外，还有文件创建时间、文件最近被访问时间及文件最近被修改时间等时间属性。在 C++Builder 中，可以使用函数 GetFileTime()来获取文件的时间属性，也可以使用函数 SetFileTime()设置文件的

时间属性。函数 GetFileTime()的声明如下：

```
BOOL GetFileTime(
    HANDLE hFile,                    //文件句柄
    LPFILETIME lpCreationTime,       //文件创建时间
    LPFILETIME lpLastAccessTime,     //文件最近被访问时间
    LPFILETIME lpLastWriteTime       //文件最近被修改时间
   );
```

其中时间结构 LPFILETIME 的定义如下：

```
typedef struct _FILETIME { // ft
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
} FILETIME;
```

由于使用函数 GetFileTime()获取文件的时间属性是以 FILETIME 表示的，因此在实际显示文件时间属性时可以使用 FileTimeToSystemTime()函数将其转换为 TSystemTime 形式，再将系统时间形式 TSystemTime 通过函数 SystemTimeToDateTime()转换为 TDateTime 形式显示。SetFileTime()函数的声明如下：

```
BOOL SetFileTime(
    HANDLE hFile,                          //文件句柄
    CONST FILETIME *lpCreationTime,        //文件创建时间
    CONST FILETIME *lpLastAccessTime,      //文件最近被访问时间
    CONST FILETIME *lpLastWriteTime        //文件最近被修改时间
   );
```

下面是在实例程序中使用 GetFileTime()函数和 SetFileTime()函数获取和设置文件时间属性的示例代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//获取文件时间属性
  AnsiString StrFile=this->Edit1->Text;
  if(StrFile.Length()<1)
     return;
  int FileHandle=FileOpen(StrFile,fmOpenRead);
  if(FileHandle==-1)
  {
    MessageBox(Handle,"文件打开不正确！","信息提示",MB_OK);
    return;
  }
  TFileTime CreateTime,AccessTime,WriteTime;
  if(GetFileTime((void*)FileHandle,&CreateTime,&AccessTime,&WriteTime))
  {
   TSystemTime SystemTime;
   FileTimeToSystemTime(&CreateTime,&SystemTime);
   this->Edit2->Text=StrToInt(SystemTime.wYear);
   this->Edit3->Text=StrToInt(SystemTime.wMonth);
   this->Edit4->Text=StrToInt(SystemTime.wDay);
   FileTimeToSystemTime(&AccessTime,&SystemTime);
   this->Edit5->Text=StrToInt(SystemTime.wYear);
   this->Edit6->Text=StrToInt(SystemTime.wMonth);
   this->Edit7->Text=StrToInt(SystemTime.wDay);
```

```
        FileTimeToSystemTime(&WriteTime,&SystemTime);
        this->Edit8->Text=StrToInt(SystemTime.wYear);
        this->Edit9->Text=StrToInt(SystemTime.wMonth);
        this->Edit10->Text=StrToInt(SystemTime.wDay);
      }
      else
      {
        MessageBox(Handle,"获取文件时间属性操作失败","信息提示",MB_OK);
      }
      FileClose(FileHandle);
  }
  void __fastcall TForm1::BitBtn3Click(TObject *Sender)
  {//设置文件时间属性
      AnsiString StrFile=this->Edit1->Text;
      if(StrFile.Length()<1||this->Edit2->Text.Length()<1)
         return;
      int FileHandle=FileOpen(StrFile,fmOpenWrite);
      if(FileHandle==-1)
      {
        MessageBox(Handle,"文件打开不正确！","信息提示",MB_OK);
        return;
      }
      TFileTime CreateTime,AccessTime,WriteTime;
      if(!GetFileTime((void*)FileHandle,&CreateTime,&AccessTime,&WriteTime))
      {
       MessageBox(Handle,"设置文件时间属性操作失败","信息提示",MB_OK);
      }
       TSystemTime SystemTime;
       SystemTime.wYear=StrToInt(this->Edit2->Text);
       SystemTime.wMonth=StrToInt(this->Edit3->Text);
       SystemTime.wDay=StrToInt(this->Edit4->Text);
       SystemTimeToFileTime(&SystemTime,&CreateTime);
       SystemTime.wYear=StrToInt(this->Edit5->Text);
        SystemTime.wMonth=StrToInt(this->Edit6->Text);
        SystemTime.wDay=StrToInt(this->Edit7->Text);
        SystemTimeToFileTime(&SystemTime,&AccessTime);
        SystemTime.wYear=StrToInt(this->Edit8->Text);
        SystemTime.wMonth=StrToInt(this->Edit9->Text);
        SystemTime.wDay=StrToInt(this->Edit10->Text);
        SystemTimeToFileTime(&SystemTime,&WriteTime);
        if(SetFileTime((void*)FileHandle,&CreateTime,&AccessTime,&WriteTime))
        {
          MessageBox(Handle,"设置文件时间属性操作成功","信息提示",MB_OK);
        }
        else
        {
           MessageBox(Handle,"设置文件时间属性操作失败","信息提示",MB_OK);
        }
     FileClose(FileHandle);
  }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 108 如何获取和设置文件时间属性”。

实例 109 如何对文件名函数进行操作

实例目的

此实例介绍了 C++Builder 中的文件名函数的使用方法。实例程序执行后，效果如图 109-1 所示。



图 109-1

实现方法

C++Builder 提供了一组文件名操作函数，这组函数可以对文件的名称、所在子目录、驱动器和扩展名等进行操作。这些函数及其功能说明如下：

（1）ExpandFileName()函数用于返回文件的全路径（含驱动器、路径）。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
__fastcall ExpandFileName(const AnsiString FileName);
```

（2）ExtractFileExt()函数用于从文件名中析取文件扩展名。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
__fastcall ExtractFileExt(const AnsiString FileName);
```

（3）ExtractFileName()函数用于从文件名中析取不含路径的文件名。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
__fastcall ExtractFileName(const AnsiString FileName);
```

（4）ExtractFilePath()函数用于从文件名中析取文件所在的路径名。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
```

```
__fastcall ExtractFilePath(const AnsiString FileName);
```

（5）ExtractFileDir()函数用于从文件名中析取目录名。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
__fastcall ExtractFileDir(const AnsiString FileName);
```

（6）ExtractFileDrive()函数用于从文件名中析取文件所在的驱动器名。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
__fastcall ExtractFileDrive(const AnsiString FileName);
```

（7）ChangeFileExt()函数用于改变文件的扩展名。它不是对真正的文件进行改名，只是对文件名这个字符串进行处理。如：ShowMessage(ChangeFileExt("c:\\luo.dat",".txt"))语句执行后将显示 c:\\luo.txt。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE System::AnsiString __fastcall ChangeFileExt(
  const System::AnsiString FileName, //文件名
  const System::AnsiString Extension  //文件新扩展名
 );
```

（8）ExpandUNCFileName()函数用于返回含有网络驱动器的文件全路径，其格式为“\\机器名\共享名\文件名”。如果 F 是映射的网络驱动器\\WINNT，则下列语句 ShowMessage(ExpandUNCFileName("F:\\luo.dat"))执行后将显示\\WINNT\luo.dat。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
__fastcall ExpandUNCFileName(const AnsiString FileName);
```

（9）ExtractRelativePath()函数用于从文件名中抽取相对路径信息，如..\WinDVD\Setup.exe 这种形式。该函数的声明如下：

```
extern PACKAGE AnsiString __fastcall ExtractRelativePath(
  const AnsiString BaseName, //基准文件名
  const AnsiString DestName  //目标文件名
 );
```

（10）ExtractShortPathName()函数用于把文件名转化为 DOS 的 8.3 格式。如语句 ShowMessage(ExtractShortPathName("E:\\Program Files\\luo.dat"));执行后将显示 "E:\Progra~1\ luo.dat"。该函数的声明如下（参数 const AnsiString FileName 为文件名）：

```
extern PACKAGE AnsiString
__fastcall ExtractShortPathName(const AnsiString FileName);
```

（11）MatchesMask()函数用于检查文件是否与指定的文件名格式匹配。该函数的声明如下：

```
extern PACKAGE bool
__fastcall MatchesMask(const AnsiString Filename, const AnsiString Mask);
```

其中，参数 const AnsiString Filename 为文件名，参数 const AnsiString Mask 为文件名格式，支持通配符。如语句 ShowMessage(MatchesMask("luo.exe","*.?x?"));执行后将显示 true。

以下是实例程序中使用文件名操作函数操作文件名的示例代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
```

```
    if(this->OpenDialog1->FileName.Length()<1)
      return;
    AnsiString StrFile=this->OpenDialog1->FileName;
    this->Edit2->Text=ExpandFileName(StrFile);
    this->Edit3->Text=ExtractFileExt(StrFile);
    this->Edit4->Text=ExtractFileName(StrFile);
    this->Edit5->Text=ExtractFilePath(StrFile);
    this->Edit6->Text=ExtractFileDir(StrFile);
    this->Edit7->Text=ExtractFileDrive(StrFile);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 109 如何对文件名函数进行操作”。

实例 110 如何读写文本文件

实例目的

此实例主要介绍了原始的文本文件读写操作的编程技术，以便编写文本文件类型的应用程序操作时进行更深入的开发。当实例程序启动后，单击“新建文件”按钮，在主窗口中输入文件内容，单击“保存文件”按钮，在“另存为”对话框中输入新文件名，新建文本文件将被保存；单击“打开文件”按钮，在“打开”对话框中选择一个文本文件打开，该文本文件内容将显示在主窗口中，修改其中内容，单击“保存文件”按钮，修改后的文本文件将被重新保存，原文件将以.Bak 后缀名保存，效果如图 110-1 所示。



图 110-1

实现方法

C++Builder 为了充分发挥其整体功能，重写了大部分已经在标准 C、C++中的文件输

入输出函数。这些函数和 C++Builder 关系紧密，能使用 C++Builder 中的 AnsiString 等数据类型，在 C++Builder 中用这种方式的文件操作是最方便的。常用的实现这类操作的函数及其功能的含义如下：

FileOpen()　打开文件
FileClose()　关闭文件
FileRead()　读文件
FileSeek()　文件定位
FileWrite()　写文件
FileCreate()　创建文件

其中，FileOpen()函数用于打开一个指定的文件，该函数的声明如下：

```
extern PACKAGE int __fastcall FileOpen(const AnsiString FileName, int Mode);
```

参数 const AnsiString FileName 用于指定要打开的文件名；int Mode 用于指定文件打开的方式，取值如下（可用"|"运算符连接）：fmOpenRead 或 0（以只读属性打开）；fmOpenWrite 或 1（以只写属性打开）；fmOpenReadWrite 或 2（以读/写属性打开）；fmShareCompat（兼容 FCB 方式）；fmShareExclusive 或 16（共享方式：以独占方式打开，在关闭以前，别人不能访问）；fmShareDenyWrite 或 32（共享方式：拒绝写访问）；fmShareDenyRead 或 48（共享方式：拒绝读访问）；fmShareDenyNone 或 64（共享方式：无限制，允许读写）。如果成功返回其句柄，否则返回-1。

FileClose()函数用于关闭打开的文件句柄，该函数的声明如下：

```
extern PACKAGE void __fastcall FileClose(int Handle);
```

参数 int Handle 指明要关闭的句柄。

FileCreate()函数用于创建一个文件句柄，该函数的声明如下：

```
extern PACKAGE int __fastcall FileCreate(const AnsiString FileName);
```

参数 const AnsiString FileName 为要创建的文件名。创建文件成功返回其句柄，否则返回-1。

FileSeek()函数用于移动文件读取指针，该函数的声明如下：

```
extern PACKAGE int __fastcall FileSeek(int Handle, int Offset, int Origin);
```

其中，int Handle 指定相关联的句柄；int Offset 为移动偏移量；int Origin 为起始位置，0=文件头，1=当前位置，2=文件尾。如果该函数操作成功，返回文件指针的位置，失败则返回-1。

使用 FileRead()函数可以根据文件名读取一个文件的具体内容，该函数的声明如下：

```
extern PACKAGE int __fastcall FileRead(
int Handle,          //文件句柄
void *Buffer,        //文件内容
unsigned Count       //文件大小
);
```

注意：int Handle 句柄必须首先由 FileOpen()函数或 FileCreate()函数创建。

使用 FileWrite()函数可以将一个文件的具体内容保存到磁盘中，该函数的声明如下：

```
extern PACKAGE int __fastcall FileWrite(
int Handle,          //文件句柄
```

```
const void *Buffer, //文件内容
unsigned Count      //文件大小
);
```

注意：int Handle 句柄必须首先由 FileOpen()函数或 FileCreate()函数创建。

实例程序使用 FileRead()函数读取一个文本文件内容的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      pszBuffer = new char[iFileLength+1];
      iBytesRead = FileRead(iFileHandle, pszBuffer, iFileLength);
      FileClose(iFileHandle);
      this->Memo1->Text=pszBuffer;
      delete [] pszBuffer;
    }
    catch(...)
    {
      Application->MessageBox("Can't perform one of the  file
read operation", "Alert", IDOK);
    }
  }
}
```

实例程序使用 FileWrite()函数新建或保存一个文本文件的示例代码如下：

```
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
 char szFileName[MAXFILE+4];
 int iFileHandle;
 int iLength;
 if (SaveDialog1->Execute())
 {
  if (FileExists(SaveDialog1->FileName))
  {
   fnsplit(SaveDialog1->FileName.c_str(), 0, 0, szFileName, 0);
   strcat(szFileName, ".BAK");
   RenameFile(SaveDialog1->FileName, szFileName);
  }
  iFileHandle = FileCreate(SaveDialog1->FileName);
```

```
    FileWrite(iFileHandle,this->Memo1->Text.c_str(),
this->Memo1->Text.Length());
    FileClose(iFileHandle);
  }
}
```

注意：在对文件进行读写操作时，通常需要经过以下 3 个步骤：①打开文件，取得文件句柄；②读写等操作文件；③使用完毕后，关闭文件。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 110 如何读写文本文件”。

实例 111　如何打印文本文件

实例目的

此实例主要介绍如何使用 TPrinter 组件打印文本文件的编程技术。实例程序执行后，效果如图 111-1 所示。



图 111-1

实现方法

C++Builder 在 VCL 组件库中提供了一个负责所有打印工作的组件，即 TPrinter 组件。C++Builder 已经预先说明了一个 Printer 变量在 Printers 程序库中，可以直接使用默认的 Printer 变量来控制程序中的所有打印操作，而不需要再重复声明一遍。不过，要使用默认的 Printer 变量，必须在程序前面加上包含文件：#include<VCL\Pritners.hpp>。如此才可以在程序中使用默认的Printer变量，否则在程序编译时将会出现Printer没有定义的错误信息。常用的 TPrinter 组件的属性和方法的意义如表 111-1 所示。

表 111-1

名称	属性或方法	功能说明
Canvas	属性	将图形传送到打印机打印出来
TextOut	方法	将文字填充在 Canvas 对应在内存的图形区域，这个区域最后会被送到打印机打印出来
BeginDoc	方法	开始一个打印工作
EndDoc	方法	结束一个打印工作
PageHeight	属性	传回报表的页高，以“点”为单位
NewPage	属性	传出一个跳页控制码到打印机上，并且将 Canvas 控制的画行坐标移到原点(0,0)
PageNumber	属性	传回目前正在打印中的报表页码

下面是在实例程序中使用 TPrinter 打印文本文件的示例代码：

```
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
  AnsiString PageNum;
  int Ctr,i,PHeight,LineSpace,x;
  Printer()->BeginDoc();
  x=0;
  Ctr=1;
  PHeight=Printer()->PageHeight;
  LineSpace=PHeight/60;
  PageNum=IntToStr(Printer()->PageNumber);
  this->Label1->Caption="现在正在打印第"+PageNum+"页的数据...";
  for(i=0;i<(Memo1->Lines->Count);i++)
  {
    Printer()->Canvas->TextOutA(0,x,Memo1->Lines->Strings[i]);
    x+=LineSpace;
    Ctr++;
    if(Ctr>59)
    {
      Printer()->NewPage();
      x=0;
      Ctr=0;
      PageNum=IntToStr(Printer()->PageNumber);
      this->Label1->Caption="现在正在打印第"+PageNum+"页的数据...";
    }
  }
  Printer()->EndDoc();
  this->Label1->Caption="打印完成，共打印了"+PageNum+"页";
}
```

从上面的代码可以看出，当使用 TPrinter 组件打印报表时，必须先执行 BeginDoc() 方法来启动一个打印工作，然后利用 Canvas 属性开始将文字和图形数据填入内存缓冲区内，最后再执行 EndDoc()方法结束打印工作。除了 TPrinter 组件外，VCL 中还有 TRichEdit、TForm、TPrintDialog 和 TPrinterSetupDialog 组件与打印有关。TRichEdit 和 TForm 组件提供了 Print()方法，可以直接将 TRichEdit、TForm 的数据传送到打印机上打印出来。因

此，如果不需要定制打印时的输出风格，请直接使用 TRichEdit、TForm 内附的打印功能打印数据。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 111 如何打印文本文件”。

实例 112　如何创建特大型文件

实例目的

此实例介绍了如何在程序中创建特大型文件的编程技术。实例程序执行后，效果如图 112-1 所示。



图 112-1

实现方法

通常在创建文件时，是根据文件实际占用存储空间的大小而分配相应的字节数，但是，如果在创建文件时移动文件指针，则可以随意创建一个上百兆甚至上千兆的特大型文件。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//创建特大型文件
  HANDLE hHugeFile;
  hHugeFile = CreateFile(this->Edit1->Text.c_str(),GENERIC_WRITE,0,
NULL,CREATE_NEW,FILE_ATTRIBUTE_NORMAL,NULL);
  if( hHugeFile == INVALID_HANDLE_VALUE )
  {// 处理出错信息
    LPVOID StrInfo;
    t::FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER|FORMAT_MESSAGE_FROM_
SYSTEM,0, GetLastError(),MAKELANGID(LANG_NEUTRAL,
SUBLANG_DEFAULT),(LPTSTR)&StrInfo, 0, NULL );
    ::MessageBox(0, (LPCTSTR)StrInfo, '信息提示',MB_OK|MB_ICONINFORMATION );
    ::LocalFree( StrInfo );
    return;
  }
  //特大型文件的大小
```

```
    DWORD dwPtr=SetFilePointer(hHugeFile,1024*1024*StrToInt(this->Edit2->Text),
NULL,FILE_BEGIN);
    if (dwPtr == 0xFFFFFFFF)
    {// 处理出错信息
      LPVOID StrInfo;
      ::FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
 FORMAT_MESSAGE_FROM_SYSTEM,0, GetLastError(),MAKELANGID(LANG_NEUTRAL,
 SUBLANG_DEFAULT),(LPTSTR)&StrInfo, 0, NULL );
      ::MessageBox(0, (LPCTSTR)StrInfo,"信息提示", MB_OK|MB_ICONINFORMATION );
      ::LocalFree( StrInfo );
      return;
     }
     SetEndOfFile(hHugeFile);
     ::CloseHandle(hHugeFile);
     MessageBox(Handle,"特大型文件创建成功！","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 112 如何创建特大型文件”。

实例 113 如何制作临时文件

实例目的

此实例介绍了如何产生临时文件的编程技术。实例程序执行后，效果如图 113-1 所示。



图 113-1

实现方法

在 Windows 操作系统中，应用程序产生或需要的临时文件可以放在系统指定的临时文件夹中进行操作。获取 Windows 操作系统的临时文件夹可以使用 API 函数 GetTempPath() 实现，该函数的声明如下：

```
DWORD GetTempPath(
    DWORD nBufferLength,          //临时文件夹路径名的尺寸大小
    LPTSTR lpBuffer               //临时文件夹路径名
   );
```

通过 API 函数 GetTempFileName()可以在指定的临时文件夹中创建一个临时文件，该

函数的声明如下：

```
UINT GetTempFileName(
    LPCTSTR lpPathName,          //临时文件夹路径名
    LPCTSTR lpPrefixString,      //临时文件名前缀
    UINT uUnique,                //创建临时文件的文件名时使用的一个数字
    LPTSTR lpTempFileName        //保存临时文件夹的文件名的字符串
   );
```

以下是实例程序使用这两个 API 函数产生临时文件的示例代码：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //产生临时文件
  if(this->Edit1->Text.Length()<1)
    return;
  char TempPath[MAX_PATH];
  GetTempPath(sizeof(TempPath),TempPath);
  char TempFile[MAX_PATH];
  char PreFix[3];
  strcpy(PreFix,this->Edit1->Text.c_str());
  GetTempFileName(TempPath,PreFix,0,TempFile);
  this->Edit2->Text=TempFile;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分　文件操作\实例 113 如何制作临时文件”。

实例 114　如何启动帮助文件

实例目的

此实例主要介绍了如何在应用程序界面上启动自制的.hlp 格式的帮助文件。实例程序执行后，单击“查看帮助文件”按钮，弹出帮助文件对话框，效果如图 114-1 所示。

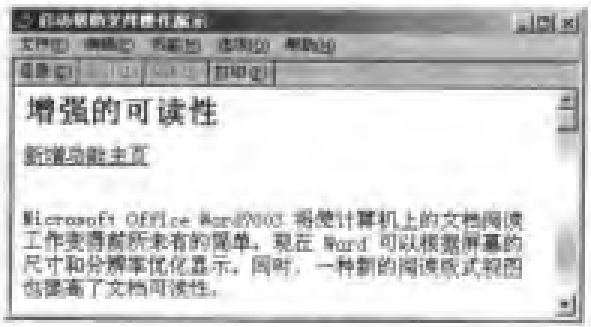


图 114-1

实现方法

一个完整的应用程序一般均有在线帮助，以方便用户操作。在应用程序中制作在线帮

助并不难，只需要调用.hlp 格式的帮助文件即可，主要工作集中在制作帮助文件.hlp 操作中。C++Builder 提供了一个制作帮助文件的编译器 HCW，它负责将.rtf 文件编译成.hlp 文件，该程序通常存放在 C:\Program Files\Borland\CBuilderX\Help\Tools 目录下。实例程序中制作和启动帮助文件的示例步骤如下：

（1）启动 Word 或写字板编辑.rtf 格式的文档，即帮助文件的内容，如图 114-2 所示。

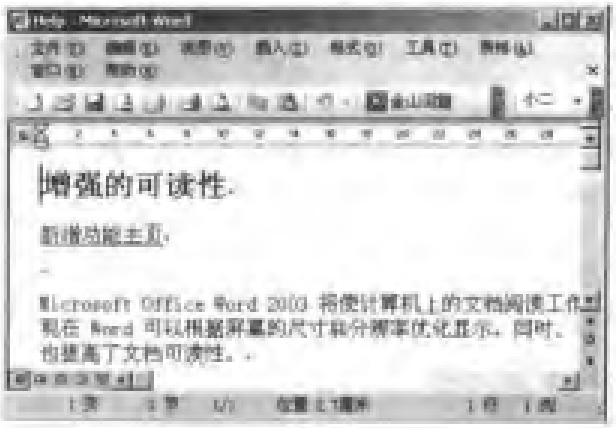


图 114-2

（2）启动 HCW 帮助文件编译程序，选择菜单 New，弹出 New 对话框，在该对话框的列表框中选择 Help Project 项，如图 114-3 所示，单击 OK 按钮，弹出“保存”对话框保存帮助文件项目。



图 114-3

（3）单击 HCW 主界面右侧的 Files 按钮，弹出 Topic Files 对话框，再单击 Add...按钮增加刚才编辑的 rtf 格式的帮助文件。如图 114-4 所示。



图 114-4

（4）单击 HCW 主界面右下角的 Save and Compile 按钮，HCW 程序将自动生成一个以.hlp 结尾的帮助文件。

（5）在应用程序中调用.hlp 文件的示例代码如下所示。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  //指定帮助文件
  Application->HelpFile="Help.hlp";
  //打开帮助窗体
  Application->HelpCommand(HELP_CONTENTS,0);
}
```

最后编译执行程序即可。此外，操作系统也提供了一个 API 函数 WinHelp()来启动帮助文件，该函数的声明如下：

```
BOOL WinHelp(
    HWND hWndMain,              // 需要调用帮助文件的窗口句柄
    LPCTSTR lpszHelp,           // 帮助文件
    UINT uCommand,              // 帮助类型
    DWORD dwData                // 根据帮助文件类型设置的附加数据
  );
```

其中，参数 UINT uCommand 的可能取值包括 HELP_COMMAND、HELP_CONTENTS、HELP_CONTEXT、HELP_CONTEXTPOPUP、HELP_FORCEFILE、HELP_HELPONHELP、HELP_INDEX、HELP_KEY、HELP_MULTIKEY、HELP_PARTIALKEY、HELP_QUIT、HELP_SETCONTENTS、HELP_SETINDEX、HELP_SETWINPOS。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 114 如何启动帮助文件”。

实例 115　如何使用组件在主机间传送文件

实例目的

此实例主要介绍了如何直接使用 NMStrm 和 NMStrmServ 组件在主机间传送文件的编程技术。实例程序运行后，在传送文件名编辑栏中选择要传送的文件全路径名，在目标主机名编辑栏中输入目标主机的 IP 地址，单击“发送”按钮，如果发送成功，将给出操作成功提示，否则会给出一个操作失败提示；目标主机在收到发送的文件后，将弹出一个文件保存对话框，提示保存接收到的文件（注意：使用时，必须在发送主机和接收主机间运行同一程序）。效果如图 115-1 所示。



图 115-1

实现方法

C++Builder 在组件模板的 FastNet 标签页中提供了 NMStrm 和 NMStrmServ 组件，直接使用即可在两个主机间传送文件。发送文件使用的组件是 NMStrm，其发送文件的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//发送文件
  if(this->Edit1->Text.Length()<1)
    return;
  if(this->Edit2->Text.Length()<1)
    return;
  TFileStream *MyFStream;
  MyFStream=new TFileStream(this->Edit1->Text,fmOpenRead);
  try
  {
    this->NMStrm1->Host=this->Edit2->Text;
    this->NMStrm1->FromName="NetMasters";
    this->NMStrm1->PostIt(MyFStream);
    MessageBox(Handle,"发送文件到目标主机操作成功！","信息提示",MB_OK);
  }
  catch(...)
  {
    MessageBox(Handle,"发送文件到目标主机操作失败！","信息提示",MB_OK);
  }
  MyFStream->Free();
}
```

接收文件使用的组件是 NMStrmServ，它是通过事件 OnMSG 来感知发送主机发送的文件，要处理接收到的文件，必须处理该事件，其接收文件的示例代码如下：

```
void __fastcall TForm1::NMStrmServOnMSG(TComponent *Sender,
                                        const AnsiString sFrom, TStream *strm)
{//接收文件
  if(this->SaveDialog1->Execute())
  {
    if(FileExists(this->SaveDialog1->FileName))
    {
      if(MessageBox(Handle,"目标文件已经存在，是否删除之？",
"信息提示",MB_YESNO)==IDNO)
      {
        return;
```

```
        }
        DeleteFile(this->SaveDialog1->FileName);
      }
      TFileStream *MyFStream;
      //建立文件
      MyFStream=new TFileStream(this->SaveDialog1->FileName,fmCreate);
      try
      {
        MyFStream->CopyFrom(strm,strm->Size);
        MessageBox(Handle,"接收目标主机发来的文件操作成功！","信息提示",MB_OK);
      }
      catch(...)
      {
        MessageBox(Handle,"接收目标主机发来的文件操作失败！","信息提示",MB_OK);
      }
      MyFStream->Free();
    }
  }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分　文件操作\实例 115 如何使用组件在主机间传送文件”。

实例 116　如何对文件进行编码和解码

实例目的

此实例介绍了如何使用 NMUUProcessor 组件对文件进行编码和解码的编程技术。实例程序运行后，编码效果如图 116-1 所示。

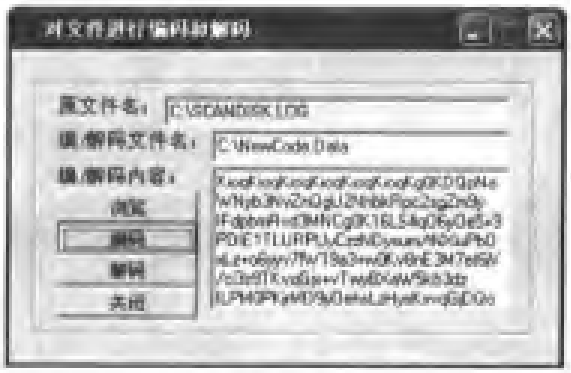


图 116-1

实现方法

在 C++Builder 组件模板的 FastNet 标签页下有一个 NMUUProcessor 组件，它能够非常

方便地对文件进行编码和解码，这对于网络传输加强数据保密具有相当的重要性。以下是实例程序使用该组件对文件进行编码和解码的实现代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//对文件进行编码
  if(this->Edit1->Text.Length()<1||this->Edit2->Text.Length()<1)
    return;
  TFileStream *InputStream=new TFileStream(this->Edit1->Text,fmOpenRead);
  TFileStream *OutputStream=
new TFileStream(this->Edit2->Text,fmCreate|fmOpenReadWrite);
  this->NMUUProcessor1->Method==uuMime;
  this->NMUUProcessor1->InputStream=InputStream;
  this->NMUUProcessor1->OutputStream=OutputStream;
  this->NMUUProcessor1->Encode();
  OutputStream->Seek(soFromBeginning,0);
  char *WriteBuffer=new char[OutputStream->Size];
  OutputStream->Read(WriteBuffer,OutputStream->Size);
  this->Memo1->Clear();
  this->Memo1->Text=WriteBuffer;
  delete WriteBuffer;
  delete OutputStream;
  delete InputStream;
  MessageBox(Handle,"编码文件操作成功","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//对文件进行解码
  if(this->Edit1->Text.Length()<1||this->Edit2->Text.Length()<1)
    return;
  TFileStream *InputStream=new TFileStream(this->Edit1->Text,fmOpenRead);
  TFileStream *OutputStream=
new TFileStream(this->Edit2->Text,fmCreate|fmOpenReadWrite);
  this->NMUUProcessor1->Method==uuMime;
  this->NMUUProcessor1->InputStream=InputStream;
  this->NMUUProcessor1->OutputStream=OutputStream;
  this->NMUUProcessor1->Decode();
  OutputStream->Seek(soFromBeginning,0);
  char *WriteBuffer=new char[OutputStream->Size];
  OutputStream->Read(WriteBuffer,OutputStream->Size);
  this->Memo1->Clear();
  this->Memo1->Text=WriteBuffer;
  delete WriteBuffer;
  delete OutputStream;
  delete InputStream;
  MessageBox(Handle,"解码文件操作成功","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 116 如何对文件进行编码和解码”。

实例 117　如何设置 RichEdit 组件的存盘格式

实例目的

此实例主要介绍了如何以不同的方式保存 RichEdit 组件中的内容。当实例程序启动后，在主界面中输入文字内容（可以通过单击“设置文字格式”按钮设置文字的字体及颜色），然后单击“保存为 TXT 格式”按钮，弹出文件“另存为”对话框，输入保存文件名，则该部分文字内容将保存为 TXT 形式文件；如果单击“保存为 RTF 格式”按钮，弹出文件“另存为”对话框，输入保存文件名，则该部分文字内容将保存为 RTF 形式文件。效果如图 117-1 所示。

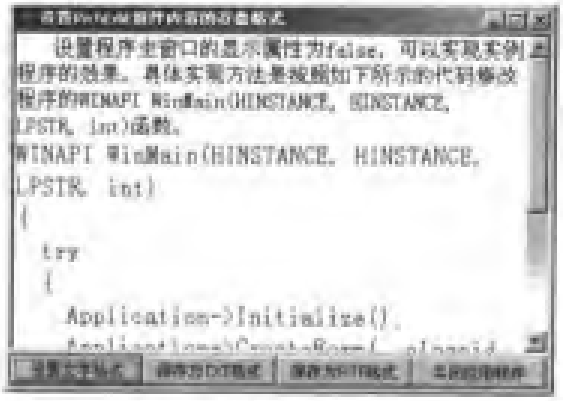


图 117-1

实现方法

C++Builder 的 RichEdit 组件可以将内容保存为两种不同的文件格式。当设置 RichEdit 组件的 PlainText 属性为 true 时，则 RichEdit 组件在调用 Lines 属性的 SaveToFile()函数保存文件时，将把 RichEdit 组件的内容保存为文本文件；当设置 RichEdit 组件的 PlainText 属性为 false 时，则 RichEdit 组件在调用 Lines 属性的 SaveToFile()函数保存文件时，将把 RichEdit 组件的内容保存为 RTF 文件格式。示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  if(this->SaveDialog1->Execute())
  {
    this->RichEdit1->PlainText=true;
    this->RichEdit1->Lines->SaveToFile(this->SaveDialog1->FileName+".txt");
  }
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  if(this->SaveDialog1->Execute())
  {
```

```
        this->RichEdit1->PlainText=false;
        this->RichEdit1->Lines->SaveToFile(this->SaveDialog1->FileName+".rtf");
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 117 如何设置 RichEdit 组件的存盘格式”。

实例 118 如何在 WIN.INI 文件中保存程序信息

实例目的

此实例主要介绍了如何使用 Win.ini 文件而非使用注册表保存应用程序信息的编程技巧。实例程序执行后，在主界面中输入如图 118-1 所示的数据，单击“设置”按钮，再打开 Win.ini 文件，在 Win.ini 文件中保存的内容如图 118-2 所示。

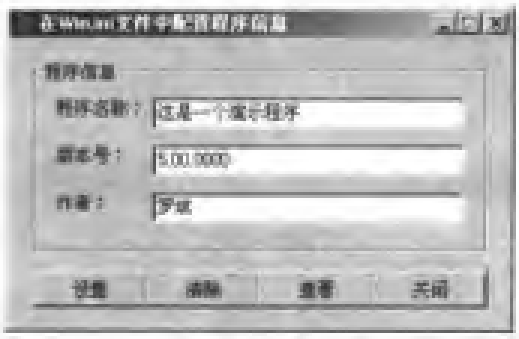


图 118-1

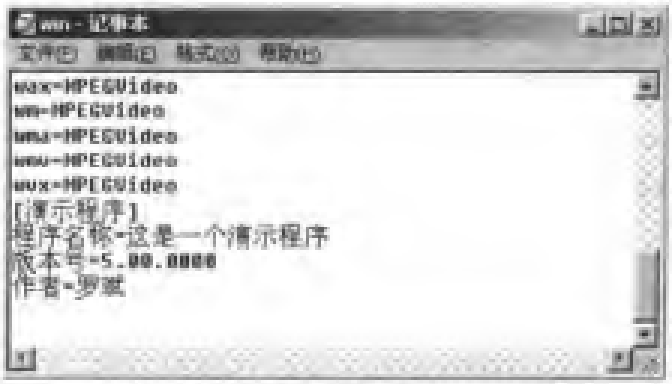


图 118-2

实现方法

在 Windows 95 操作系统出现以前，应用程序在系统中的配置信息大多存放在一个系统配置文件 Win.ini 中，只有在 Windows 95 操作系统出现以后，应用程序在系统中的配置

信息才大部分移植到注册表中。不过，在 Windows 95 以后的操作系统中，用户也完全可以将应用程序的配置信息存放在系统配置文件 Win.ini 中，读写系统配置文件 Win.ini 的 API 函数 WriteProfileString()和 GetProfileString()的声明如下：

```
BOOL WriteProfileString(
    LPCTSTR lpAppName,          //Win.ini 文件中的段名
    LPCTSTR lpKeyName,          //Win.ini 文件中的键名
    LPCTSTR lpString            //Win.ini 文件中的键值
   );
DWORD GetProfileString(
    LPCTSTR lpAppName,          // Win.ini 文件中的段名
    LPCTSTR lpKeyName,          // Win.ini 文件中的键名
    LPCTSTR lpDefault,          // Win.ini 文件中的默认值
    LPTSTR lpReturnedString,    // Win.ini 文件中的键值
    DWORD nSize                 // Win.ini 文件中的键值字符串长度
   );
```

实例程序读写 Win.ini 文件内容的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//写 Win.ini 文件
  WriteProfileString("演示程序","程序名称",this->Edit1->Text.c_str());
  WriteProfileString("演示程序","版本号",this->Edit2->Text.c_str());
  WriteProfileString("演示程序","作者",this->Edit3->Text.c_str());
  MessageBox(Handle,"信息提示","应用程序信息设置成功！",MB_OK);
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//读 Win.ini 文件
  char Temp[MAX_PATH];
  AnsiString Str,StrInfo="程序名称:";
  GetProfileString("演示程序","程序名称","",Temp,MAX_PATH);
  Str.printf("%s",Temp) ;
  StrInfo+=Str+"\n 版本号:";
  GetProfileString("演示程序","版本号","",Temp,MAX_PATH);
  Str.printf("%s",Temp) ;
  StrInfo+=Str+"\n 作者:";
  GetProfileString("演示程序","作者","",Temp,MAX_PATH);
  Str.printf("%s",Temp) ;
  StrInfo+=Str;
  MessageBox(Handle,StrInfo.c_str(),"查询结果",MB_OK);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //清空 Win.ini 文件中的段名
  this->Edit1->Text="";
  this->Edit2->Text="";
  this->Edit3->Text="";
  WriteProfileString("演示程序",NULL,NULL);
}
```

此外，C++Builder 还提供了一个 TIniFile 对象，它提供的方法能够更简单地实现对 INI 文件的读写。如：ReadString()、WriteString()、ReadBool()、ReadDate()、ReadDateTime()、ReadFloat()、ReadInteger()、WriteTime()、WriteBool()、WriteDate()、WriteDateTime()、

WriteFloat()、WriteInteger()和 WriteTime()等。其中 ReadString()函数的声明如下：

```
virtual AnsiString __fastcall ReadString(const AnsiString Section,
 const AnsiString Ident, const AnsiString Default);
```

参数 const AnsiString Section 表示段名；参数 const AnsiString Ident 表示名称；参数 const AnsiString Default 表示默认项。

WriteString()函数的声明如下：

```
virtual void __fastcall WriteString(const AnsiString Section,
const AnsiString Ident, const AnsiString Value);
```

参数 const AnsiString Section 表示段名；参数 const AnsiString Ident 表示名称；参数 const AnsiString Value 表示设置的字符串值。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 118 如何在 WIN.INI 文件中保存程序信息”。

实例 119 如何将数据转换成 Word 文档格式

实例目的

此实例主要介绍了如何直接将应用程序中的数据导入到 Word 文档，并启动 Word 应用程序。实例程序执行后，效果如图 119-1 所示。

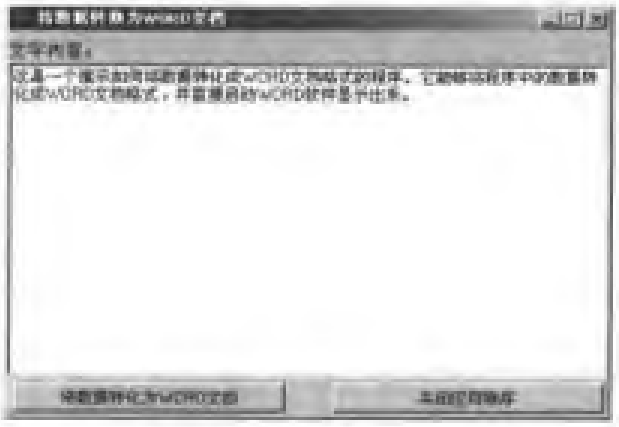


图 119-1

实现方法

在 C++Builder 中，通常使用 OLE 自动化技术调用 Word，要掌握这一技术，首先应该了解 Word 的自动化对象及 Microsoft Word Visual Basic 帮助文件中关于 Word 的对象方法和属性的介绍。Word 对象代表一个 Word 的元素，如文档、段落、书签或单个的字符。集合是一个对象，该对象包含多个其他对象，通常这些对象属于相同的类型，例如，一个集

合对象中可包含文档中的所有书签对象。通过使用这些属性和方法，可以修改单个的对象，也可修改整个的对象集合。属性是对象的一个特性或者该对象操作的一个方面。例如，文档属性包含名称、内容、保存状态，以及是否启用修订等。要更改一个对象的属性，可以修改该属性的值。方法则是对象可以进行的动作。代表 Word 应用程序的自动化对象有 Word.Application 和 Word.Basic，通过 Application 对象的 WordBasic 属性可以允许它们之间进行变换赋值。在 C++Builder 中，通过 CreateOleObject("Word.Basic")就可启动 Word 并获得句柄，将其赋给 Variant 变量，再通过属性和方法操作即可实现数据向 Word 文档的转换。示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  Variant Var;
  Var=CreateOleObject("Word.Basic");
  Var.Exec(Procedure("AppShow"));
  Var.Exec(Procedure("FileNew")<<"Normal");
  for(int i=0;i<this->Memo1->Lines->Count;i++)
  {
     Var.Exec(Procedure("Insert")<<Memo1->Lines->Strings[i]+"\r\n");
  }
}
```

注意：要使用 CreateOleObject()函数，应该在源代码中包含#include"Comobj.hpp"文件。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘："源代码\第三部分 文件操作\实例 119 如何将数据转换成 Word 文档格式"。

实例 120　如何直接从 Excel 文件导入数据

实例目的

此实例主要介绍了如何从 Excel 文件导入数据的编程技巧。实例程序运行后，效果如图 120-1 和图 120-2 所示。

实现方法

OLE（对象的链接与嵌入）是微软于 20 世纪 90 年代初提出并大力推广和不断发展的技术，是应用程序间交换数据、相互操作的一种有效方式，微软的 Office 产品提供了很强的 OLE 服务功能，Visual Basic、Delphi 及 C++Builder 等语言都可以客户方式向 Excel 提交命令，以存取其中的数据，其中 Visual Basic 和 Delphi 在操作上最为简单，在语法上采用类 VBA 的语句；而 C++Builder 则采用间接的办法，利用变体类 Variant 所提供的 4 个"方法"向 OLE 服务程序提交操纵命令。

```
Variant  Variant::OlePropertyGet(属性名,参数….);      // 取对象属性
```

```
Void     Variant::OlePropertySet(属性名,参数…,);      // 设置对象属性
Variant  Variant::OleFunction(函数名,参数…,);         // 运行对象的函数
void     Variant::OleProcedure(过程名,参数…,);        // 运行对象的过程
```

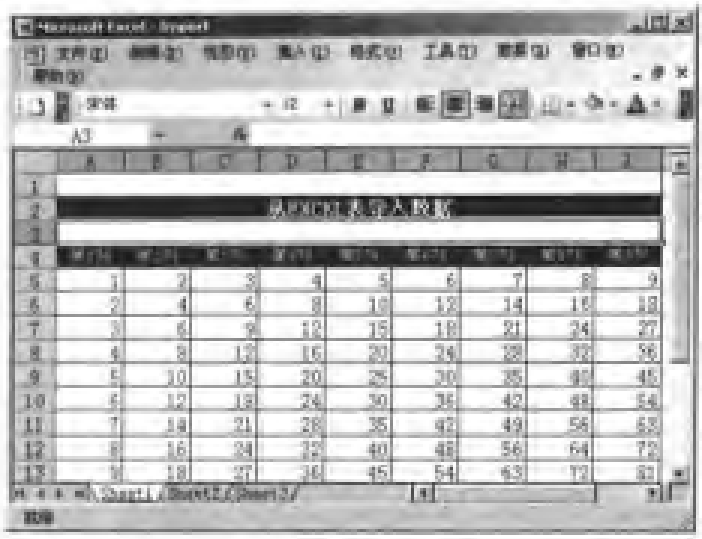


图 120-1



图 120-2

在 C++Builder 中，使用 OLE 方式操作 Excel 表通常包括以下过程：创建应用程序对象；创建工作簿对象；创建工作表对象；打开当前工作表；操作工作表（如设置单元格值，获取单元格值，设置单元格字体，设置单元格格式，重命名工作表，设置列宽，设置行高及插入删除等操作）；保存工作簿；退出 Excel 程序。以下是实例程序通过 OLE 方式从 Excel 文件导入数据的示例代码。

```
#pragma link "Excel_2K_SRVR"
#include "Excel_2K_SRVR.h"
#include <OleServer.hpp>
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  AnsiString StrFileName=GetCurrentDir()+"\\Import.xls";
  if(!FileExists(StrFileName))
  {
    MessageBox(Handle,"Excel 表文件 Import.xls 不存在,无法打开! ",
"信息提示",MB_ICONSTOP|MB_OK);
```

```
    return;
  }
  Variant ExcelApp;   //对象
  Variant WorkBook1;  //工作簿
  Variant WorkSheet1; //工作表
  try
  {
    ExcelApp=Variant::CreateObject("Excel.Application");
  }
  catch(...)
  {
   MessageBox(Handle,"无法启动 Excel! ","信息提示",MB_ICONSTOP|MB_OK);
   return;
  }
  ExcelApp.OlePropertySet("Visible",true);//设置 Excel 为可见
  //打开指定的 Excel 文件,文件中最好只设定一个 Sheet
  ExcelApp.OlePropertyGet("WorkBooks").
OleProcedure("Open",StrFileName.c_str());
  WorkBook1=ExcelApp.OlePropertyGet("ActiveWorkBook");
  WorkSheet1=WorkBook1.OlePropertyGet("ActiveSheet");//获得当前默认的 Sheet
  if(MessageBox(Handle,"是否将当前的 EXCEL 文件导入程序? ",
"信息提示",MB_YESNO)==IDNO)
    return;
  int nRow=WorkSheet1.OlePropertyGet("UsedRange").
OlePropertyGet("Rows").OlePropertyGet("Count");
  int nCol=WorkSheet1.OlePropertyGet("UsedRange").
OlePropertyGet("Columns").OlePropertyGet("Count");
 //从 Excel 表获取数据
  for(int i=5;i<=nRow;i++)
   {
     for(int j=1;j<=nCol;j++)
     {
        AnsiString Str;
        Str=WorkSheet1.OlePropertyGet("Cells",i,j).OlePropertyGet("Value");
        this->StringGrid1->Cells[i-5][j-1]=Str;
     }
  }
  WorkBook1.OleProcedure("Close");//关闭表格
  ExcelApp.OleFunction("Quit");//退出 Excel
}
```

说明：Excel 文件 Import.xls 为事先制作的包括数据的样表。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 120 如何直接从 Excel 文件导入数据”。

实例 121 如何直接将数据导入到 Excel 文件

实例目的

此实例主要介绍了如何将数据导出到 Excel 文件的编程技巧。实例程序运行后，效果如图 121-1 和图 121-2 所示。



图 121-1

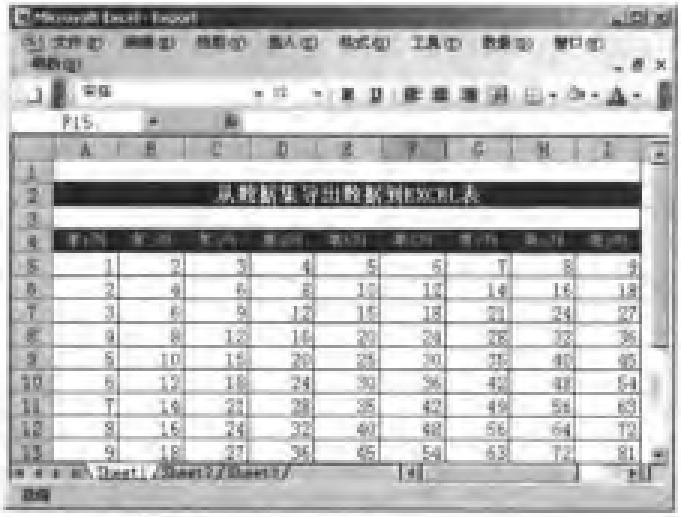


图 121-2

实现方法

操作 Excel 文件的方式多种多样，但功能最为强大、最全面的应为 OLE 方式，它基本涵盖了操作 Excel 文件的所有内容。通常在 C++Builder 中使用 OLE 方式操作 Excel 文件时，必须深刻理解 Excel 文件的工作簿、工作表、当前活动状态的对象及应用程序对象等。实例程序通过 OLE 方式将数据直接导入 Excel 文件的示例代码如下：

```
#pragma link 'Excel_2K_SRVR'
#include 'Excel_2K_SRVR.h'
#include <OleServer.hpp>
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
```

```
{
  AnsiString StrFileName=GetCurrentDir()+"\\Export.xls";
  if(!FileExists(StrFileName))
  {
    MessageBox(Handle,"Excel 文件 Export.xls 不存在！","信息提示",MB_OK);
    return;
  }
  Variant ExcelApp;        //对象
  Variant WorkBook1;       //工作簿
  Variant WorkSheet1;      //工作表
  Variant Range;           //范围
  Variant Borders;         //边框
  try
  {
    ExcelApp=Variant::CreateObject("Excel.Application");
  }
  catch(...)
  {
   MessageBox(Handle,"无法启动 Excel！","信息提示",MB_ICONSTOP|MB_OK);
   return;
  }
  ExcelApp.OlePropertySet("Visible",false);//设置 Excel 为不可见
  //打开指定的 Excel 文件,文件中最好只设定一个 Sheet
  ExcelApp.OlePropertyGet("WorkBooks").OleProcedure("Open",
StrFileName.c_str());
  WorkBook1=ExcelApp.OlePropertyGet("ActiveWorkBook");
  WorkSheet1=WorkBook1.OlePropertyGet("ActiveSheet");//获得当前默认的 Sheet
  //清空 EXCEL 表的数据
  int nRow=WorkSheet1.OlePropertyGet("UsedRange").
OlePropertyGet("Rows").OlePropertyGet("Count");
  int nCol=WorkSheet1.OlePropertyGet("UsedRange").
OlePropertyGet("Columns").OlePropertyGet("Count");
  for(int i=5;i<=nRow+1;i++)
     WorkSheet1.OlePropertyGet("Rows",i).OlePropertySet("Value","");
 //给 Excel 表赋值
   for(int j=1;j<10;j++)
   {
     for(int i=1;i<10; i++)
     {
       AnsiString StrValue=IntToStr(i*j);
       WorkSheet1.OlePropertyGet("Cells",i+4,j).
OlePropertySet("Value",StrValue.c_str());
     }
   }
 AnsiString StrRange="A"+IntToStr(1)+":I"+IntToStr(nRow);//获取操作范围
 Range=WorkSheet1.OlePropertyGet("Range",StrRange.c_str());
 Borders=Range.OlePropertyGet("Borders");//获取边框对象
// Borders.OlePropertySet("linestyle",xlNone); //去掉边框线
 //画表格边框
 Borders.OlePropertySet("linestyle",xlContinuous);
```

```
    Borders.OlePropertySet('weight',xlThin);
    Borders.OlePropertySet('colorindex',xlAutomatic);
    WorkBook1.OleProcedure('Save');//保存表格
    WorkBook1.OleProcedure('Close');//关闭表格
    ExcelApp.OleFunction('Quit');//退出 Excel
    char StrCurDir[MAX_PATH+1];
    GetCurrentDirectory(MAX_PATH,StrCurDir);
    char exeStr[]='Export.xls';
    SHELLEXECUTEINFO Info;
    memset (&Info, 0, sizeof(Info)) ;
    Info.cbSize=sizeof (Info) ;
    Info.lpVerb='open' ;
    Info.lpFile=exeStr;
    Info.lpParameters=NULL;
    Info.fMask=SEE_MASK_NOCLOSEPROCESS ;
    Info.nShow=SW_SHOWDEFAULT ;
    if (! ShellExecuteEx (&Info))
       MessageBox(Handle,'打开 Excel 文件 Export.xls 失败!','信息提示',MB_OK);
    SetCurrentDirectory(StrCurDir);
}
```

说明：Excel 文件 Export.xls 为事先制作的不包括数据的样表。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 121 如何直接将数据导入到 Excel 文件”。

实例 122 如何发送电子邮件（一）

实例目的

此实例主要解决在应用程序中如何调用 MAPIMessage，以便在应用程序中以最简单的方式发送电子邮件。当实例程序启动后，在主界面中输入邮件内容，然后单击“发送”按钮，程序将启动系统已经安装的邮件发送程序，如图 122-1 所示。在这里，邮件内容已经填写完毕，将收件人地址、抄送地址、主题填写完毕后就可以发送邮件了。

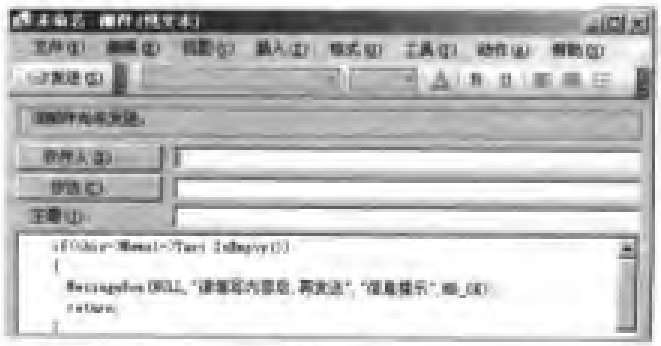


图 122-1

实现方法

因特网的盛行使得电子邮件成了网民通信与交流的主要方式之一。通常网民发送电子邮件大多喜欢采用 Outlook、Foxmail 以及各个网站上自带的邮件发送系统。要真正实现电子邮件的独立运行，则需要深入理解 MAPI。MAPI（Message Application Programming Interface）是微软公司提供的用于处理消息发送和接收的应用程序编程接口，它是一组复杂的 API 函数，利用它不仅能发送普通的电子邮件，而且可以处理传真及语音邮件。MAPI 内置于操作系统中，接口分为两个部分：一个用于前端客户应用程序，一个用于后端服务器应用程序。MAPI 前端客户程序大致分为 4 种使用情况：①简单 MAPI（主要用于与以前的程序兼容）；②公共消息调用（建立在核心 MAPI 子系统的顶端，它独立于操作系统）；③扩展 MAPI（扩展 MAPI 是基于 COM 组件的，所有的消息或其他对象都被定义为 COM 对象，这些 COM 对象封装了属性、方法及事件）；④OLE 消息（通过 OLE 自动化接口导出 MAPI 对象）。简单 MAPI 中共有 12 个函数，它们提供了信息的发送、寻址、接收和响应的应用程序接口（具体作用如表 122-1 所示）。

表 122-1

函数名	意义
MAPIDeleteMail	删除邮件
MAPIDetails	显示邮件消息的详细信息对话框
MAPIFindNext	返回特定邮件消息类型的下一个消息 ID
MAPIFreeBuffer	释放邮件系统分配的内存资源
MAPILogoff	结束一个会话
MAPILogon	建立一个会话
MAPIReadMail	读取一个消息
MAPIResolveName	显示对话框，以解答不明确的接收
MAPISaveMail	保存邮件信息
MAPISendDocuments	发送标准邮件信息
MAPISendMail	发送消息，功能强于 MAPISendDocuments()
MAPIAddress	建立或修改地址表

简单 MAPI 调用使用操作系统的 MAPI32.DLL 库，其具体使用步骤为：①使用 LoadLibrary()函数装载 MAPI32.DLL；②调用 GetProcAddress()函数获取 MAPISendMail 的入口地址；③填充信息结构体 MAPIMessage；④调用函数 MAPISendMail()发送邮件。 在使用简单 MAPI 函数时，注意应该包含头文件 MAPI.H。文件 MAPI.H 包含了以上这些函数的定义以及使用简单 MAPI 所涉及的几个数据结构，下面是几个常用 API 数据结构的定义。

```
typedef struct{
    ULONG ulReserved;                    //保留值，为 0
    LPTSTR lpszSubject;                  //描述消息主题
    LPTSTR lpszNoteText;                 //消息正文
```

```
    LPTSTR lpszMessageType;                    //消息类型
    LPTSTR lpszDateReceived;                   //消息被接收的日期
    LPTSTR lpszConversationID;                 //识别一个消息所属的会话线程
    FLAGS  flFlags;                            //消息状态标识
    lpMapiRecipDesc lpOriginator;              //描述发送人信息
    ULONG nRecipCount;                         //MapiRecipDesc 结构的数量
    lpMapiRecipDesc lpRecips;                  //指向一组 MapiRecipDesc
    ULONG nFileCount;                          //MapiFileDesc 结构的数量
    lpMapiFileDesc lpFiles;                    //指向 MapiFileDesc 结构的指针
     }MapiMessage,FAR *lpMapiMessage;

typedef struct{
    ULONG ulReserved;                          //保留值，为 0
    ULONG flFlags;                             //附件类型标志
    ULONG nPosition;                           //附件的存放位置
    LPTSTR lpszPathName;                       //附件文件名及全路径
    LPTSTR lpszFileName;                       //可选，收件人见到的附件文件名
    LPTSTR lpFileType;                         //附件文件的类型
}MapiFileDesc,FAR *lpMapiFileDesc;

typedef struct{
     ULONG ulReserved;                         //保留值，为 0
  ULONG ulRecipClass;                          //收件人的类型
  LPTSTR lpszName;                             //收件人或发件人
  LPTSTR lpszAddress;                          //收件人或发件人地址
  ULONG ulEIDSize;                             //lpEntryID 的大小
  LPVOID lpEntryID;                            //收件人或发件人条目的标识符
}MapiRecipDesc, FAR *lpMapiRecipDesc;
```

实际上，使用 C++Builder 编写此类程序时，完全不必如此劳师动众。在 C++Builder 中，MAPIMessage 已经被封装为 TMapiMessage，在需要将特定的内容作为电子邮件发送时，只需要填充这个信息结构即可。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->Memo1->Text.IsEmpty())
  {
   MessageBox(NULL,"请填写内容后,再发送","信息提示",MB_OK);
   return;
  }
  AnsiString StrContent=this->Memo1->Text;
  TMapiMessage MapiMessage;
  Cardinal MError;
  MapiMessage.ulReserved = 0;
  MapiMessage.lpszSubject = NULL;
  MapiMessage.lpszNoteText =StrContent.c_str();
  MapiMessage.lpszMessageType = NULL;
  MapiMessage.lpszDateReceived = NULL;
  MapiMessage.lpszConversationID = NULL;
  MapiMessage.flFlags = 0;
  MapiMessage.lpOriginator = NULL;
```

```
    MapiMessage.nRecipCount = 0;
    MapiMessage.lpRecips = NULL;
    MapiMessage.nFileCount = 0;
    MapiMessage.lpFiles = NULL;
    MError = MapiSendMail(0, reinterpret_cast<unsigned int>(Application->Handle),
                MapiMessage, MAPI_DIALOG | MAPI_LOGON_UI | MAPI_NEW_SESSION, 0);
    if (MError)
        MessageBox(NULL,"电子邮件发送不成功!","信息提示",MB_OK);
}
```

注意：在使用 TMapiMessage 时，需要包括头文件：#include "MAPI.hpp"。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分　文件操作\实例 122 如何发送电子邮件（一）”。

实例 123　如何发送电子邮件（二）

实例目的

此实例主要介绍了通过 SMTP 直接发送电子邮件的编程技术。实例程序执行后，效果如图 123-1 所示。

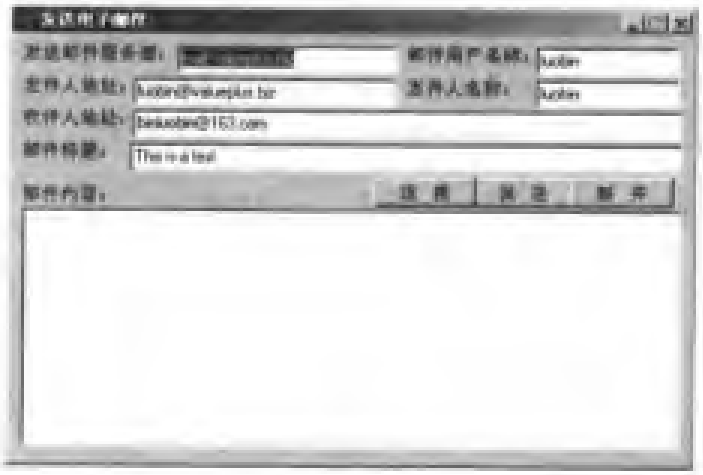


图 123-1

实现方法

在因特网协议中，与电子邮件密切相关的两个协议是 POP3 和 SMTP，POP3 用于从 POP3 邮件服务器中接收电子邮件，SMTP 是 Simple Mail Transfer Protocol 的缩写，用于通过 SMTP 服务器发送电子邮件。这两个协议本身是很复杂的，但 C++Builder 从 NetManage 公司引进了 NMPOP、NMSMTP 组件，直接使用即可。NMSMTP 组件位于组件面板的 FastNet 标签页下，它们的属性和方法包括如下内容。

（1）Host 属性：用于指定 SMTP 服务器的主机名。

（2）Port 属性：用于指定 SMTP 服务器的端口号，一般设为 25。

（3）UserId 属性：用于指定登录 SMTP 服务器时的用户名。

（4）PostMessage 属性：此属性非常重要，可以设置发件人的姓名、地址、收件人地址等，以下是它的子属性。

- FromName 属性：用于指定发件人的名称。
- FromAddress 属性：用于指定发件人的电子邮件地址。
- ToAddress 属性：用于指定收件人的电子邮件地址。
- Body 属性：用于设置邮件正文。
- Subject 属性：用于设置邮件主题。

（5）Connect 方法：用于登录到 SMTP 服务器，使用此函数前，Host、Port、UserId 等属性必须已设好。

（6）Disconnect 方法：用于断开与 SMTP 服务器的连接。

以下是实例程序使用该组件实现发送邮件的示例代码：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//发送邮件
  if (NMSMTP1->Connected)
  {
    NMSMTP1->PostMessage->FromAddress = Edit5->Text;
    NMSMTP1->PostMessage->FromName = Edit7->Text;
    NMSMTP1->PostMessage->ToAddress->Text = Edit3->Text;
    NMSMTP1->PostMessage->Body->Text = Memo1->Text;
    NMSMTP1->PostMessage->Subject = Edit5->Text;
    NMSMTP1->PostMessage->LocalProgram = "SendMail";
    NMSMTP1->SendMail();
    MessageBox(Handle,"邮件发送完成","信息提示",MB_OK);
  }
 else
   MessageBox(Handle,"请首先连接发送邮件服务器","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //连接发送邮件服务器
  if(NMSMTP1->Connected)
    NMSMTP1->Disconnect();
  else
  {
    NMSMTP1->Host = Edit1->Text;
    NMSMTP1->UserID = Edit2->Text;
    NMSMTP1->Connect();
  }
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//断开与发送邮件服务器的连接
```

```
  if(NMSMTP1->Connected)
    NMSMTP1->Disconnect();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第三部分 文件操作\实例 123 如何发送电子邮件（二）”。

第四部分　图像处理

实例 124　如何浏览图像

实例目的

此实例主要介绍了在 C++Builder 中如何通过文件打开对话框任意浏览图像的编程技术。当实例程序启动后，显示默认的图像文件。单击“浏览”按钮，弹出文件“打开”对话框，选择一个图像文件后打开，该图像文件将显示在主窗口中，效果如图 124-1 所示。



图 124-1

实现方法

C++Builder 提供的 Image 控件除了能够在属性中设置图像，也可以通过代码的方式动态浏览不同的图像文件。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->OpenDialog1->Execute())
  {
    this->Image1->Picture->LoadFromFile(this->OpenDialog1->FileName);
  }
}
```

此外，需要说明的是，浏览图像文件通常使用组件模板 Dialog 标签页下的 OpenPictureDialog 组件来打开文件。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像

处理\实例 124 如何浏览图像”。

实例 125　如何滚动浏览大图像

实例目的

此实例主要介绍了在 C++Builder 中，如何使用 TImage 控件浏览超过程序界面的大图像的编程技术。实例程序执行后，效果如图 125-1 所示。



图 125-1

实现方法

在 C++Builder 中，可以利用 TImage 组件轻松地读入图像文件并显示图像，但 TImage 组件本身不具备水平和垂直滚动条，不能显示超出程序界面的大图像。但是，通过与 TScrollBox 组件的结合，可以比较容易地解决这一问题。具体步骤如下：

（1）在组件模板的 Additional 标签页下，拖动 TScrollBox 组件到程序窗体上，设置 TScrollBox 组件的 AutoScroll 属性为 True 并调整好尺寸。

（2）在组件面板的 Additional 标签页下，拖动 TImage 组件到前一步骤放置在程序界面上的 TScrollBox 组件上，设置 TImage 组件的 AutoSize 属性为 True，Left 属性为 0，Top 属性为 0。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 125 如何滚动浏览大图像”。

实例 126 如何拖动图像

实例目的

此实例主要介绍了在 C++Builder 中，如何使用 TImage 组件实现图像拖动效果的编程技术。当实例程序执行后，使用鼠标拖动图像，图像立即处于浮动状态，效果如图 126-1 所示。



图 126-1

实现方法

在 C++Builder 中，在使用 TImage 组件显示图像时，通过对该组件的 OnMouseDown 和 OnMouseMove 消息的重写，可以实现图像的拖动效果。具体步骤和代码如下：

（1）在组件模板的 Additional 标签页中，拖动 TImage 组件到程序窗体上，并设置好默认图像。

（2）在主窗体的头文件中声明两个整型变量 int iXStart, iYStart，用于保存鼠标的落点位置，并在构造函数中初始化值为 0。

（3）按照如下所示的代码重写 TImage 组件的 OnMouseDown 消息处理函数。

```
void __fastcall TForm1::ImageMouseDown(TObject *Sender,
                TMouseButton Button, TShiftState Shift, int X, int Y)
{
  iXStart=X;
  iYStart=Y;
}
```

（4）按照如下所示的代码重写 TImage 组件的 OnMouseMove 消息处理函数。

```
void __fastcall TForm1::ImageMouseMove(TObject *Sender,
                    TShiftState Shift, int X, int Y)
{
  int iX, iY;
  if(Shift.Contains(ssLeft))
  {
```

```
        iX=X-iXStart;
        iY=Y-iYStart;
        if(iY)
        Image1->Left+=iX;
        if(iY)
          Image1->Top+=iY;
    }
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 126 如何拖动图像”。

实例 127　如何动态显示图像

实例目的

此实例主要介绍了通过画布变换实现图像动态显示的编程技术。实例程序执行后，效果如图 127-1 所示。



图 127-1

实现方法

图像动态显示能够明显增加软件的吸引力，特别是多媒体软件，在许多软件的片头片尾部分都或多或少地有一些图像通过位移或变换来增强图像的动感。在 DOS 时代，程序员可以通过调用中断编写出一些特殊效果的图像显示，以增强程序显示效果。然而，在 Windows 操作系统环境中，由于 Windows 直接管理中断调用，想用原来的方法来实现一些特殊的图像显示效果变得越来越困难了。那么，在 Windows 时代程序员如何实现图像的特殊效果显示呢？实际上，由于整个 Windows 是一个图形显示系统，因此对系统而言，程序

的每一个界面、窗口都是系统的一个画布，可以通过画布的变换来实现图像的特殊显示。在本部分中，介绍了通过画布变化实现图像动态显示的 10 种效果，包括：图像从中间往左右分出、图像从中间往上下分出、图像从左刷新屏幕显示、图像从上刷新屏幕显示、图像从上往下流水、图像从下往上流水、图像圆形展出、图像左右错移、图像上下错移、图像从右移入等。示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//图像从中间往左右分出
  BITMAP bm;
  Graphics::TBitmap *Bitmap;
  //定义一个图像句柄
  Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
  //取得显示窗口的颜色属性
  if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
  {
    GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
    //如果特效显示的图像颜色属性为 256 色
    if(bm.bmBitsPixel==8)
    {  //从特效显示的颜色画板中提取配色方案
      SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
      //在显示窗口中设置配色方案
      RealizePalette(Form1->Canvas->Handle);
    }
    Bitmap->HandleType=bmDDB;
  }
   Image1->Refresh( );
   int WideHalf=Bitmap1->Width/2+Bitmap1->Width%2;
   for(int i=0;i<=WideHalf; i++)
   {
     TRect DestRect=Rect(WideHalf-i,0,WideHalf+i,Bitmap1->Height);
     TRect SourceRect=Rect(WideHalf-i,0,WideHalf+i,Bitmap1->Height);
     Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
     Sleep(10);
   }
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
   if(this->OpenDialog1->Execute())
   {
     AnsiString StrFile=this->OpenDialog1->FileName;
     this->Image1->Picture->LoadFromFile(StrFile);
   }
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{// 图像从中间往上下分出
  BITMAP bm;
  Graphics::TBitmap *Bitmap;
  Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
  if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
```

```
    {
      GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
      if(bm.bmBitsPixel==8)
      {
        SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
        RealizePalette(Form1->Canvas->Handle);
      }
      Bitmap->HandleType=bmDDB;
    }
     Form1->Refresh( );
     int HeightHalf=Bitmap1->Height/2+Bitmap1->Height%2;
     for(int i=0;i<=HeightHalf;i++)
     {
      TRect DestRect=Rect(0,0,Bitmap1->Width,i);
      TRect SourceRect=Rect(0,HeightHalf-i,Bitmap1->Width,HeightHalf);
      TRect Dest2=Rect(0,Bitmap1->Height-i,Bitmap1->Width,Bitmap1->Height);
      TRect Source2=Rect(0,HeightHalf,Bitmap1->Width,HeightHalf+i);
      Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
      Form1->Canvas->CopyRect(Dest2,Bitmap1->Canvas,Source2);
      Sleep(10);
     }
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//图像从左刷新屏幕显示
  BITMAP bm;
  Graphics::TBitmap *Bitmap;
  Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
  if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
  {
    GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
    if(bm.bmBitsPixel==8)
    {
      SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
      RealizePalette(Form1->Canvas->Handle);
    }
    Bitmap->HandleType=bmDDB;
  }
   Form1->Refresh( );
   for(int i=0;i<=Bitmap1->Width;i++)
   {
    TRect DestRect =Rect(0,0,i,Bitmap1->Height);
    TRect SourceRect =Rect(0,0,i,Bitmap1->Height);
    Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
    Sleep(10);
   }
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{//图像从上刷新屏幕显示
  BITMAP bm;
  Graphics::TBitmap *Bitmap;
```

```
    Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
    if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
    {
       GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
       if(bm.bmBitsPixel==8)
       {
          SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
          RealizePalette(Form1->Canvas->Handle);
       }
       Bitmap->HandleType=bmDDB;
    }
     Form1->Refresh( );
     for(int i=0;i<=Bitmap1->Height;i++)
     {
       TRect DestRect=Rect(0,0,Bitmap1->Width+0,i+0);
       TRect SourceRect=Rect(0,0,Bitmap1->Width,i);
       Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
       Sleep(10);
     }
  }
  void __fastcall TForm1::BitBtn6Click(TObject *Sender)
  { //图像从上往下流水
    BITMAP bm;
    Graphics::TBitmap *Bitmap;
    Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
    if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
    {
       GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
       if(bm.bmBitsPixel==8)
       {
          SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
          RealizePalette(Form1->Canvas->Handle);
       }
       Bitmap->HandleType=bmDDB;
    }
     Form1->Refresh( );
     for(int i=Bitmap1->Height;i>=0;i--)
     {
       TRect SourceRect=Rect(0,i,Bitmap1->Width,i+1);
       for(int j1=0;j1<=i;j1++)
       {
         TRect DestRect=Rect(0 ,j1,Bitmap1->Width+0,j1+1);
         Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
       }
     }
  }
  void __fastcall TForm1::BitBtn7Click(TObject *Sender)
  {//图像从下往上流水
    BITMAP bm;
    Graphics::TBitmap *Bitmap;
```

```
    Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
    if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
    {
      GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
      if(bm.bmBitsPixel==8)
      {
        SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
        RealizePalette(Form1->Canvas->Handle);
      }
      Bitmap->HandleType=bmDDB;
    }
     Form1->Refresh( );
     for(int i=0;i<=Bitmap1->Height;i++)
     {
      TRect SourceRect=Rect(0,i,Bitmap1->Width,i+1);
      for(int j1=Bitmap1->Height-1;j1>=i;j1--)
      {
       TRect DestRect=Rect(0,j1,Bitmap1->Width,j1+1);
       Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
      }
     }
 }
 void __fastcall TForm1::BitBtn8Click(TObject *Sender)
 {//图像圆形展出
   BITMAP bm;
   Graphics::TBitmap *Bitmap;
   Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
   if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
   {
     GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
     if(bm.bmBitsPixel==8)
     {
       SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
       RealizePalette(Form1->Canvas->Handle);
     }
     Bitmap->HandleType=bmDDB;
   }
    Form1->Refresh( );
    int j1=sqrt(Bitmap1->Width*Bitmap1->
Width+Bitmap1->Height*Bitmap1->Height)/2+4;
    for (int i=1;i<j1;i++)
    {
      HRGN MyRgn=CreateEllipticRgn(Bitmap1->Width/2-i,
Bitmap1->Height/2-i,Bitmap1->Width/2+i,Bitmap1->Height/2+i);
      SelectObject(Canvas->Handle,MyRgn);
      Canvas->Draw(0,0,Bitmap1);
      DeleteObject(MyRgn);
    }
}
void __fastcall TForm1::BitBtn9Click(TObject *Sender)
```

```
{//图像左右错移
  BITMAP bm;
  Graphics::TBitmap *Bitmap;
  Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
  if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
  {
    GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
    if(bm.bmBitsPixel==8)
    {
      SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
      RealizePalette(Form1->Canvas->Handle);
    }
    Bitmap->HandleType=bmDDB;
  }
  Form1->Refresh( );
  int HeightHalf=Bitmap1->Height/2+Bitmap1->Height%2;
  TRect SourceRect=Rect(0,0,Bitmap1->Width,HeightHalf);
  TRect Source2=Rect(0,HeightHalf,Bitmap1->Width,Bitmap1->Height);
  for(int i=0;i<=Bitmap1->Width;i++)
  {
    TRect DestRect=Rect(0-Bitmap1->Width+i,0,i,0+HeightHalf);
    TRect Dest2=Rect(Bitmap1->Width-i,HeightHalf,
Bitmap1->Width+Bitmap1->Width-i,Bitmap1->Height+0);
    Form1->Canvas->MoveTo(0-Bitmap1->Width+i-1,0);
    Form1->Canvas->LineTo(0-Bitmap1->Width+i-1,HeightHalf);
    Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
    Form1->Canvas->MoveTo(Bitmap1->Width+Bitmap1->Width-i,0);
    Form1->Canvas->LineTo(Bitmap1->Width+Bitmap1->Width-i,Bitmap1->Height);
    Form1->Canvas->CopyRect(Dest2,Bitmap1->Canvas,Source2);
  }
}
void __fastcall TForm1::BitBtn10Click(TObject *Sender)
{//图像上下错移
  BITMAP bm;
  Graphics::TBitmap *Bitmap;
  Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
  if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
  {
    GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
    if(bm.bmBitsPixel==8)
    {
      SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
      RealizePalette(Form1->Canvas->Handle);
    }
    Bitmap->HandleType=bmDDB;
  }
  Form1->Refresh( );
  int WideHalf=Bitmap1->Width/2+Bitmap1->Width%2;
  TRect SourceRect=Rect(0,0,WideHalf,Bitmap1->Height);
  TRect Source2=Rect(WideHalf,0,Bitmap1->Width,Bitmap1->Height);
```

```
    for(int i=0;i<=Bitmap1->Height;i++)
    {
      TRect DestRect=Rect(0,0-Bitmap1->Height+i,WideHalf,0+i);
      TRect Dest2=Rect(WideHalf,Bitmap1->Height-i,
Bitmap1->Width,Bitmap1->Height);
      Form1->Canvas->LineTo(WideHalf,0-Bitmap1->Height+i-1);
      Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
      Form1->Canvas->MoveTo(WideHalf,Bitmap1->Height+Bitmap1->Height-i);
      Form1->Canvas->LineTo(Bitmap1->Width,Bitmap1->Height+Bitmap1->Height-i);
      Form1->Canvas->CopyRect(Dest2,Bitmap1->Canvas,Source2);
    }
}
void __fastcall TForm1::BitBtn11Click(TObject *Sender)
{//图像从右移入
  BITMAP bm;
  Graphics::TBitmap *Bitmap;
  Graphics::TBitmap *Bitmap1=this->Image1->Picture->Bitmap;
  if(GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL)==8)
  {
    GetObject(Bitmap1->Handle,sizeof(BITMAP),(LPSTR)&bm);
    if(bm.bmBitsPixel==8)
    {
      SelectPalette(Form1->Canvas->Handle,Bitmap1->Palette,FALSE);
      RealizePalette(Form1->Canvas->Handle);
    }
    Bitmap->HandleType=bmDDB;
  }
   Form1->Refresh( );
   for(int i=0;i<=Bitmap1->Width;i++)
   {
     TRect DestRect=Rect(Bitmap1->Width-i,0,Bitmap1->Width,Bitmap1->Height);
     TRect SourceRect=Rect(0,0,i,Bitmap1->Height);
     Form1->Canvas->CopyRect(DestRect,Bitmap1->Canvas,SourceRect);
   }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 127 如何动态显示图像”。

实例 128　如何维护多个图像

实例目的

此实例主要介绍在程序中有多个图像需要进行调度时，如何进行管理的编程技术。当实例程序启动后，单击“增加图像”按钮，弹出图像文件打开对话框，选择一个 BMP 格

式的图像文件，然后打开，则该图像文件将在左上角的列表框中出现，单击它则在下方显示出来。在图像列表框中选择一个图像文件，单击“减少图像”按钮，则该图像消失，效果如图 128-1 所示。



图 128-1

实现方法

C++Builder 的 ImageList 组件主要用来存放多个相同图像文件格式的图像数据。实际上，ImageList 组件是由一系列 TBitmap 类型的组件组成，因此用户可以声明一个 TBitmap 类型的指针变量，利用它把 ImageList 组件中所存放的图像取出，再指定给 Image 组件显示。通常用户可以直接将 ImageList 组件拖放到 Form 上，然后连续单击 ImageList 组件两次，系统会弹出一个 ImageList 编辑窗口，利用这个编辑窗口就可以把图像文件加入到 ImageList 组件内，不过用户也可以使用该组件的 Add()、Delete()、Clear()方法增加或减少 ImageList 组件的图像。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//增加图像
  this->OpenPictureDialog1->Filter="Bmp Files(*.bmp)|*.bmp";
  if(this->OpenPictureDialog1->Execute())
  {
    pic->LoadFromFile(this->OpenPictureDialog1->FileName);
    ImageList1->Height=pic->Height;
    ImageList1->Width=pic->Width;
    ImageList1->Add(pic,NULL);
    ListBox1->Items->Add(this->OpenPictureDialog1->FileName);
  }
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//减少图像
  this->ImageList1->Delete(this->ListBox1->ItemIndex);
  this->ListBox1->Items->Delete(ListBox1->ItemIndex);
  this->Image1->Picture->Assign(NULL);
}
void __fastcall TForm1::ShowImage(TObject *Sender)
```

```
{//显示图像
    this->ImageList1->GetBitmap(this->ListBox1->ItemIndex,pic);
    Image1->Picture->Assign(pic);
    Image1->Width=pic->Width;
    Image1->Height=pic->Height;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 128 如何维护多个图像”。

实例 129　如何绘制图形

实例目的

此实例主要演示了图形的绘制、复制、剪切、粘贴以及图像文件的打开及保存的编程技术。当实例程序执行后，效果如图 129-1 所示。



图 129-1

实现方法

在 Windows 操作系统环境下，如果要在程序窗口上绘制图形，必须直接调用 Windows 操作系统提供的 GDI 函数来实现，这些绘图元素包括画笔、画刷、字体、位图等。C++Builder 把这些 GDI 元素封装成 TCanvas，以解决图形的绘制问题，在 VCL 中，许多组件均包含有 TCanvas 属性，如 TForm、TPaintBox、TListBox、TImage 等。基本上，C++Builder 的 TCanvas 封装了 Windows 操作系统的 3 个等级的函数。它们分别是 High、Intermediate、Low。High 等级内的函数用来画线、造型及绘制文字，相对应的函数有 LineTo()、Ellipse()、TextOut()等；Intermediate 等级内的函数则用来管理图像，如 Pen、Brush、Font、CopyRect、BrushRect；Low 等级内的函数则直接调用 Windows 的 GDI 函数，使用的是 TCanvas 的 Handle。TCanvas 内也有 4 个重要的属性，分别是画线用的画笔（Tpen）、填图用的画刷

（Tbrush）、写字用的字体（Tfont）、用来显示图形的像素数组（Pixel Array）。画笔 TPen 具有 4 个属性，分别是颜色（Color）、线宽（Width）、线条样式（Style）及显示模式（Mode）。画刷 TBrush 也包括 3 个属性，分别是颜色（Color）、样式（Style）和位图（Bitmap），颜色（Color）是画刷填充颜色；样式（Style）用于设置画刷填入图案的样式；位图（Bitmap）是在画刷中填充的图形或图像。实际上，当利用各种包含 TCanvas 属性的组件绘制图形时，可以把每个 TCanvas 想象成一张一张的画布，所有图形操作方法都是在该画布上进行的。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//new
   this->Image1->Canvas->Pen->Color=clRed;
   this->Image1->Canvas->Pen->Width=5;
   this->Image1->Canvas->Brush->Color=clBlue;
   this->Image1->Canvas->Brush->Style=0;
   this->Image1->Canvas->Ellipse(20,20,400,200);

}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//open
   if(this->OpenPictureDialog1->Execute())
   {
     this->Image1->Picture->LoadFromFile(this->OpenPictureDialog1->FileName);
   }
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//save
  if(this->SavePictureDialog1->Execute())
  {
    Image1->Picture->SaveToFile(this->SavePictureDialog1->FileName+".bmp");
  }
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//Copy
   Clipboard()->Assign(this->Image1->Picture);
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{//cut
   Clipboard()->Assign(this->Image1->Picture);
   TRect ARect;
   this->Image1->Canvas->CopyMode=cmWhiteness;
   ARect=Rect(0,0,Image1->Width,Image1->Height);
   this->Image1->Canvas->CopyRect(ARect,Image1->Canvas,ARect);
   this->Image1->Canvas->CopyMode=cmSrcCopy;
}
void __fastcall TForm1::BitBtn6Click(TObject *Sender)
{//Paste
  if(Clipboard()->HasFormat(CF_BITMAP))
```

```
    {
      Graphics::TBitmap *Bitmap;
      Bitmap=new Graphics::TBitmap();
      try
      {
        Bitmap->Assign(Clipboard());
        this->Image1->Canvas->Draw(0,0,Bitmap);
        delete Bitmap;
      }
      catch(...)
      {
        delete Bitmap;
      }
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 129 如何绘制图形”。

实例 130　如何以不规则方式裁剪图像

实例目的

此实例介绍了如何以不规则方式显示图像的编程技术。实例程序运行后，效果如图 130-1 所示。



图 130-1

实现方法

图像通常是以矩形的规则方式显示的，但是在 C++Builder 中，完全可以通过改变复制模式实现图像的不规则显示。示例步骤如下：

（1）准备一张源图像，如图 130-2 所示，用 Image1 组件载入。



图 130-2

（2）在画图中制作一幅裁剪轮廓图，如图 130-3 所示，用 Image2 组件载入，设置 Image2 组件的 Visible 属性为 false。

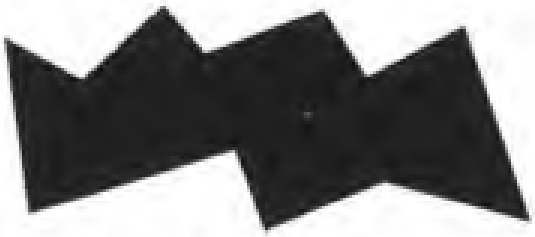


图 130-3

（3）将裁剪轮廓图以 cmSrcPaint 方式复制到源图像，实现代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  TRect MyRect;
  MyRect.left=0;
  MyRect.top=0;
  MyRect.right=this->Image1->Width;
  MyRect.Bottom=this->Image1->Height;
  this->Image1->Canvas->CopyMode=cmSrcPaint;
  this->Image1->Canvas->CopyRect(MyRect,this->Image2->Canvas,MyRect);
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 130 如何以不规则方式裁剪图像”。

实例 131 如何以任意角度旋转显示图像

实例目的

此实例介绍了如何以任意角度旋转显示图像的编程技术。实例程序执行后，效果如图 131-1 所示。

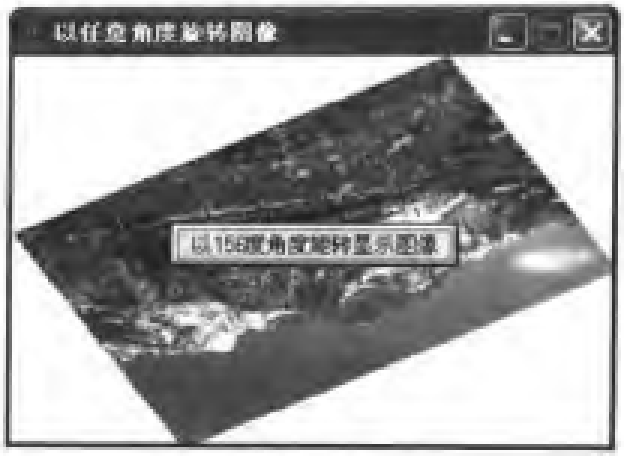


图 131-1

实现方法

通过使用数学函数变换图像像素点的显示位置能够实现图像以任意角度显示。实例程序实现此类技术的示例代码如下：

```
#include "math.h"
void RotatePicture(Graphics::TBitmap *SourcePicture,
Graphics::TBitmap *DestPicture,int angle)
{//以任意角度旋转图像
  if(angle>180)
    angle=360-angle;
  if(angle<-180)
    angle=360+angle;
  float radians=(2*3.1416*angle)/360;
  float cosine=(float)cos(radians);
  float sine=(float)sin(radians);
  float Point1x=(-SourcePicture->Height*sine);
  float Point1y=(SourcePicture->Height*cosine);
  float Point2x=(SourcePicture->Width*cosine-SourcePicture->Height*sine);
  float Point2y=(SourcePicture->Height*cosine+SourcePicture->Width*sine);
  float Point3x=(SourcePicture->Width*cosine);
  float Point3y=(SourcePicture->Width*sine);
  float minx=0,miny=0,maxx=0,maxy=0;
  minx=Point2x<Point3x?Point2x:Point3x;
  minx=minx<Point1x?minx:Point1x;
  minx=minx<0?minx:0;
  miny=Point2y<Point3y?Point2y:Point3y;
  miny=miny<Point1y?miny:Point1y;
  miny=miny<0?miny:0;
  maxx=Point2x>Point3x?Point2x:Point3x;
  maxx=maxx>Point1x?maxx:Point1x;
  maxx=maxx>0?maxx:0;
  maxy=Point2y>Point3y?Point2y:Point3y;
  maxy=maxy>Point1y?maxy:Point1y;
```

```
    maxy=maxy>0?maxy:0;
    int DestBitmapWidth,DestBitmapHeight;
    if(angle>90&&angle<180)
      DestBitmapWidth=(int)ceil(-minx);
    else
      DestBitmapWidth=(int)ceil(maxx-minx);
    if(angle>-180&&angle<-90)
      DestBitmapHeight=(int)ceil(-miny);
    else
      DestBitmapHeight=(int)ceil(maxy-miny);
    DestPicture->Height=DestBitmapHeight;
    DestPicture->Width=DestBitmapWidth;
    for(int x=0;x<DestBitmapWidth;x++)
    {
     for(int y=0;y<DestBitmapHeight;y++)
     {
      int SrcBitmapx=(int)((x+minx)*cosine+(y+miny)*sine);
      int SrcBitmapy=(int)((y+miny)*cosine-(x+minx)*sine);
      if(SrcBitmapx>=0&&SrcBitmapx<SourcePicture->
Width&&SrcBitmapy>=0&&SrcBitmapy<SourcePicture->Height)
      {
       DestPicture->Canvas->Pixels[x][y]=
SourcePicture->Canvas->Pixels[SrcBitmapx][SrcBitmapy];
      }
     }
    }
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  Graphics::TBitmap *pBmp=new Graphics::TBitmap();
  RotatePicture(this->Image1->Picture->Bitmap,pBmp,150);
  this->Image1->Picture->Bitmap=pBmp;
  delete pBmp;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 131 如何以任意角度旋转显示图像”。

实例 132 如何在桌面上显示图像

实例目的

此实例主要介绍了直接在桌面上显示图像的编程技巧。实例程序运行后，效果如图 132-1 所示。



图 132-1

实现方法

要在桌面上显示图像或绘制图形，必须取得操作桌面的句柄，一旦取得了句柄，就可以按照一般操作进行图像显示或图形绘制操作。GetDesktopWindow()函数可以直接返回桌面的句柄。下面是实例程序使用桌面句柄显示图像的示例代码：

```
void __fastcall TForm1::Bitbtn1Click(TObject *Sender)
{
  Application->Minimize();
  HDC MyDC;
  TRect MyRect;
  MyDC=GetWindowDC(GetDesktopWindow());
StretchBlt(MyDC,0,0,Screen->Width,Screen->Height,
this->Image1->Canvas->Handle,0,0,this->Image1->Picture->Width,
this->Image1->Picture->Height,SRCCOPY);
}
```

说明：图像显示函数 StretchBlt()主要是对原始图像进行拉伸操作，该函数的声明如下。

```
BOOL StretchBlt(
HDC hdcDest,                    //目标图像显示句柄
int nXOriginDest,               //目标图像左上角横坐标
    int nYOriginDest,           //目标图像左上角纵坐标
    int nWidthDest,             //目标图像宽度
    int nHeightDest,            //目标图像高度
    HDC hdcSrc,                 //源图像显示句柄
    int nXOriginSrc,            //源图像左上角横坐标
    int nYOriginSrc,            //源图像左上角纵坐标
    int nWidthSrc,              //源图像宽度
    int nHeightSrc,             //源图像高度
    DWORD dwRop                 //光栅操作码
   );
```

通常情况下，StretchBlt()成员函数的光栅操作码选用 SRCCOPY，当然还有许多其他的光栅操作码，通过这些光栅操作码的使用可以使图像达到一些特殊的显示效果。常用的光栅操作码的意义如表 132-1 所示。

表 132-1

光栅操作码	意义
BLACKNESS	用物理调色板上的 0 索引颜色填充目标矩形，默认为黑色
CAPTUREBLT	只适用于 Windows 98/Me、Windows 2000/XP
DSTINVERT	反转目标矩形
MERGECOPY	使用 AND 操作符合并当前选择的画刷和源矩形的颜色
MERGEPAINT	使用 OR 操作符合并反转源矩形和目标矩形的颜色
NOMIRRORBITMAP	只适用于 Windows 98/Me、Windows 2000/XP
NOTSRCCOPY	复制反转的源矩形到目标矩形
NOTSRCERASE	使用 OR 操作符组合源和目标矩形颜色后，然后反转
PATCOPY	复制当前画刷到目标位图
PATINVERT	使用 XOR 操作符组合当前选择的画刷颜色和目标矩形颜色
PATPAINT	使用 OR 操作符组合当前选择的画刷颜色和反转的源矩形颜色
SRCAND	使用 AND 操作符组合源矩形和目标矩形颜色
SRCCOPY	直接复制源矩形到目标矩形
SRCERASE	使用 AND 操作符组合源矩形颜色和目标矩形颜色的反转色
SRCINVERT	使用 XOR 操作符组合源矩形和目标矩形的操作符
SRCPAINT	使用 OR 操作符组合源矩形和目标矩形的颜色
WHITENESS	使用物理调色板索引 1 的颜色填充目标矩形

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 132 如何在桌面上显示图像”。

实例 133 如何拷贝当前屏幕

实例目的

此实例介绍了如何拷贝当前屏幕以及如何将拷贝结果保存为位图文件的编程技术。实例程序执行后，效果如图 133-1 所示。



图 133-1

实现方法

C++Builder 的 TCanvas 类提供了一个函数 CopyRect()，它能够将一个画布上指定区域的内容复制到另一个画布上。该函数的声明如下：

```
void __fastcall CopyRect(const TRect &Dest, TCanvas* Canvas, const TRect &Source);
```

其中，参数 const TRect &Dest 表示目标画布的图像大小，参数 TCanvas* Canvas 表示源画布，参数 const TRect &Source 表示源画布的图像大小。

以下是实例程序使用该函数拷贝当前屏幕并保存为图像文件的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  Graphics::TBitmap *MyBmp=new Graphics::TBitmap();
  TCanvas* MyCanvas=new TCanvas();
  HDC MyDC=GetWindowDC(NULL);
  MyCanvas->Handle=MyDC;
  TRect MyRect=Rect(0,0,Screen->Width,Screen->Height);
  MyBmp->Width=MyRect.right;
  MyBmp->Height=MyRect.Bottom;
  MyBmp->Canvas->CopyRect(MyRect,MyCanvas,MyRect);
  ReleaseDC(0,MyDC);
  MyCanvas->Handle=0;
  this->Image1->Picture->Bitmap=MyBmp;
  delete MyCanvas;
  delete MyBmp;
  MessageBox(Handle,"拷贝当前屏幕图像操作成功！","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   if(this->SavePictureDialog1->Execute())
   {
      AnsiString StrFileName=this->SavePictureDialog1->FileName;
      StrFileName+=".bmp";
      this->Image1->Picture->SaveToFile(StrFileName);
      MessageBox(Handle,"保存当前屏幕图像操作成功！","信息提示",MB_OK);
   }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 133 如何拷贝当前屏幕”。

实例 134　如何打印图像文件

实例目的

此实例主要介绍了使用 TPrinter 组件打印图像文件的编程技术。实例程序执行后，效

果如图 134-1 所示。



图 134-1

实现方法

在 C++Builder 的 VCL 组件库中提供了一个负责所有打印工作的组件即 TPrinter。C++Builder 已经预先说明了一个 Printer 变量在 Printers 程序库中，可以直接使用默认的 Printer 变量来控制程序中的所有打印操作，而不需要再重复声明一遍。不过，要使用默认的 Printer 变量，必须在程序前面加上包含文件：#include<VCL\Pritners.hpp>。这样才可以在程序中使用默认的 Printer 变量，否则在程序编译时将会出现 Printer 没有定义的错误信息。通过对 TPrinter 的 Canvas 属性的设置，可以实现对打印字体、颜色甚至图像的处理。下面是在实例程序中使用 TPrinter 组件打印图像文件的示例代码。

```
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  TRect PrnRect,ImgRect;
  int CenterX,CenterY,X1,Y1,X2,Y2,SDiv;
  this->Label1->Caption="正在打印图像...";
  Printer()->BeginDoc();
  int PHeight=Printer()->PageHeight;
  int PWidth=Printer()->PageWidth;
  CenterX=PWidth/2;
  CenterY=PHeight/2;
  SDiv=2;//放大2倍打印
  X1=CenterX-(PWidth/(SDiv*2));
  Y1=CenterY-(PHeight/(SDiv*2));
  X2=CenterX+(PWidth/(SDiv*2));
  Y2=CenterY-(PHeight/(SDiv*2));
  PrnRect=Rect(X1,Y2,X2,Y2);
  ImgRect=Rect(0,0,Image1->Width,Image1->Height);
  Printer()->Canvas->CopyRect(PrnRect,Form1->Canvas,ImgRect);
  Printer()->EndDoc();
  this->Label1->Caption="打印完毕！";
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 134 如何打印图像文件”。

实例 135　如何制作马赛克图像



实例目的

此实例介绍了如何制作马赛克图像效果的编程技术。实例程序执行后，效果如图 135-1 所示。



图 135-1

实现方法

马赛克效果实际上就是产生一些随机像点，以达到类似马赛克效果的图像。随机像点可以通过随机数函数 Randomize()和 Rand()产生，像点设置可以使用 SetPixel()函数实现。SetPixel()函数的声明如下：

```
COLORREF SetPixel(
    HDC hdc,                    //像素所在设备的句柄
    int X,                      //像素的 X 坐标
    int Y,                      //像素的 Y 坐标
    COLORREF crColor            //像素颜色
   );
```

以下是实例程序实现马赛克效果的代码：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //产生马赛克效果
  HDC MyDC=GetDC(this->Handle);
  Randomize();
  for(int i=0;i<this->Width;i+=5)
    for(int j=0;j<this->Height;j+=5)
    {
      int Red=rand()%255;
      int Green=rand()%255;
      int Blue=rand()%255;
```

```
        for(int k=i;k<i+4;k++)
         for(int l=j;l<j+4;l++)
         {
          SetPixel(MyDC,k,l,RGB(Red,Green,Blue));
         }
      }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 135 如何制作马赛克图像”。

实例 136 如何实现图像中心扩散效果

实例目的

此实例介绍了通过对图像显示位置进行变换实现图像从中心向外逐渐扩散显示的编程技术。实例程序执行后，单击“中心扩散”按钮，图像将从中心逐渐向外扩散显示，效果如图 136-1 所示。



图 136-1

实现方法

在一些多媒体软件中，经常可以看到图像的各种特技显示，这些图像经过多种变换，动感显示图像，往往给人一种赏心悦目的感觉。在 C++Builder 中，Image 组件提供了很好的图像支持功能，特别是在图像的刷新上，显示非常平滑，使用该组件可以非常简单地实现图像由中心向外扩散显示的效果。示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  int iLeft,iTop,iWidth,iHeight;
  iLeft =this->Image1->Left;
```

```
    iTop =this->Image1->Top;
    iWidth =this->Image1->Width;
    iHeight =this->Image1->Height;
    for (int i = 0; i<=iWidth; i+=2)
    { //通过图像的坐标选定图像的一部分
      this->Image1->Left = iLeft + (iWidth - i)/2;
      this->Image1->Top =iTop + iHeight/2 - i*iHeight/iWidth/2;
      this->Image1->Width = i;
      this->Image1->Height = i*iHeight/iWidth;
      this->Image1->Refresh();
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 136 如何实现图像中心扩散效果”。

实例 137　如何实现图像百叶窗效果

实例目的

此实例介绍了如何以水平或垂直百叶窗方式显示图像的编程技术。实例程序执行后，单击“以垂直百叶窗方式显示图像”按钮，图像将以垂直百叶窗方式显示；单击“以水平百叶窗方式显示图像”按钮，图像将以水平百叶窗方式显示，效果如图 137-1 所示。



图 137-1

实现方法

百叶窗显示图像特效技术是一种迷人的图像处理技术，具有非常强烈而形象的动感性。在 C++Builder 中，使用画布的 CopyRect()函数结合定时器，可以非常容易地实现这种效果。CopyRect()函数的声明如下：

```
    void __fastcall CopyRect(const TRect &Dest, TCanvas* Canvas, const
TRect &Source);
```

其中，参数 const TRect &Dest 为目标矩形大小，参数 TCanvas* Canvas 为源图像的画布，参数 const TRect &Source 为源矩形大小。

以下是实例程序实现水平和垂直方向百叶窗显示图像的代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //以垂直百叶窗方式显示图像
   this->Refresh();
   Graphics::TBitmap *pBmp=new Graphics::TBitmap;
   pBmp->LoadFromFile("Show.bmp");
   int iWidth=this->Width/20;
   for(int i=0;i<=iWidth;i++)
   {
     for(int j=0;j<=20;j++)
     {
       TRect MyRect=TRect(j*iWidth,0,j*iWidth+i,Height);
       Form1->Canvas->CopyMode=cmSrcCopy;
       Form1->Canvas->CopyRect(MyRect,pBmp->Canvas,MyRect);
     }
     Sleep(100);
   }
   delete pBmp;
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//以水平百叶窗方式显示图像
   this->Refresh();
   Graphics::TBitmap *pBmp=new Graphics::TBitmap;
   pBmp->LoadFromFile("Show.bmp");
   int iHeight=this->Height/20;
   for(int i=0;i<=iHeight;i++)
   {
     for(int j=0;j<=20;j++)
     {
       TRect MyRect=TRect(0,j*iHeight,Width,j*iHeight+i);
       Form1->Canvas->CopyMode=cmSrcCopy;
       Form1->Canvas->CopyRect(MyRect,pBmp->Canvas,MyRect);
     }
     Sleep(100);
   }
}
```

此外，使用画布的 CopyRect()函数时，可以设置 CopyMode 属性，以达到一些特殊效果。常用的 CopyMode 属性包括：cmBlackness（用黑色填充目标区域）、cmDstInvert（忽略源图像，只对目标区域取反）、cmMergeCopy（用 AND 方式组合位图）、cmMergePaint（源位图取反，用 OR 方式组合位图）、cmNotSrcCopy（源位图取反后复制到目标图像）、cmNotSrcErase（以 OR 方式组合位图，然后对结果取反）、cmPatCopy（复制源位图至目标图像）、cmPatInvert（以 XOR 方式组合位图）、cmPatPaint（源位图取反，然后按 OR 方式组合）、CmSrcAnd（通过 AND 方式组合位图）、cmSrcCopy（直接复制）、cmSrcErase（目标位图取反，与源位图进行 AND 操作）、cmSrcInvert（组合位图，按 XOR 方式）、cmSrcPaint

（组合位图，按 OR 方式）、cmWhiteness（用白色填充目标区）、cmCreateMask（以掩码方式操作源位图）。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 137 如何实现图像百叶窗效果”。

实例 138　如何实现图像底片化效果

实例目的

此实例介绍了如何使图像以底片化效果显示的编程技术。实例程序执行后，效果如图 138-1 所示。



图 138-1

实现方法

在操作系统自带的画笔程序中，有一个反色的菜单项，能够实现图像的底片化效果，即图像的反色。图像的反色就是取每一个像素点的对立颜色，例如：如果图像的一个像素点的 RGB 值为 RGB(128,52,38)，则它的反色值就为 RGB(127,203,217)。实际上，相片就是根据这个原理制成的。以下是实例程序实现图像底片化效果显示的示例代码：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//底片化图像
  BYTE *ptr;
  for(int y=0;y<this->Image1->Picture->Bitmap->Height;y++)
  {
    ptr=(BYTE*)this->Image1->Picture->Bitmap->ScanLine[y];
    for(int x=0;x<this->Image1->Picture->Bitmap->Width*3;x+=3)
    {  //取 RGB 的反向值
      ptr[x]=BYTE(255-ptr[x]);
      ptr[x+1]=BYTE(255-ptr[x+1]);
```

```
            ptr[x+2]=BYTE(255-ptr[x+2]);
        }
      }
      this->Image1->Refresh();
    }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 138 如何实现图像底片化效果”。

实例 139 如何实现图像翻页效果

实例目的

此实例介绍了如何以翻页效果显示图像的编程技术。实例程序执行后，效果如图 139-1 所示。



图 139-1

实现方法

在 C++Builder 中，除了可以使用 StretchBlt()之类的函数实现图像的缩放显示，也可以参考下面的方法实现图像的缩放显示，以达到类似翻页的效果。具体步骤如下：

（1）在组件面板的 Additional 标签页中拖动 TImage 组件到 Form1 上，选择一幅图像载入该组件中。

（2）在组件面板的 System 标签页中拖动 TTimer 组件到 Form1 上，为该组件的 OnTimer 事件添加如下所示的代码。

```
int State;
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{//以翻页效果显示图像
  if(State==0)
  {
    if(this->Image1->Width>0)
```

```
    {
      this->Image1->Width=this->Image1->Width-10;
    }
    else
      State=1;
  }
  else if(this->Image1->Width<250)
  {
    this->Image1->Width=this->Image1->Width+10;
  }
  else
    State=0;
}
```

最后，编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 139 如何实现图像翻页效果”。

实例 140　如何实现图像镜像效果

实例目的

此实例介绍了如何使图像产生镜像效果的编程技术。实例程序执行后，效果如图 140-1 所示。



图 140-1

实现方法

在 C++Builder 中进行图像处理时，人们经常使用 Canvas 对象的 Pixels 属性访问图像的像素，使用这种方法有一个很大的缺点，就是程序运行速度较慢。如果使用 TBitmap 的 ScanLine 属性，则图像像素的读取速度将大大加快。下面是实例程序通过使用 TBitmap 的 ScanLine 属性读取图像并产生图像镜像效果的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //镜像图像
  Graphics::TBitmap *pBmp1=new Graphics::TBitmap();
  Graphics::TBitmap *pBmp2=new Graphics::TBitmap();
  Byte *pB1,*pB2;
  pBmp1->Assign(this->Image1->Picture->Bitmap);
  pBmp2->Assign(this->Image1->Picture->Bitmap);
  for(int y=0;y<pBmp1->Height;y++)
  {
    pB1=(Byte*)pBmp1->ScanLine[y];
    pB2=(Byte*)pBmp2->ScanLine[y];
    for(int x=0;x<pBmp1->Width*3/2;x++)
    {
      pB2[x]=pB1[pBmp1->Width*3-1-x];
      pB2[pBmp1->Width*3-1-x]=pB1[x];
    }
  }
  this->Image1->Canvas->Draw(0,0,pBmp2);
  delete pBmp1;
  delete pBmp2;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 140 如何实现图像镜像效果”。

实例 141 如何柔化处理图像

实例目的

此实例主要介绍了通过高斯模板柔化图像的图像特效处理技巧。实例程序运行后，效果如图 141-1 所示。



图 141-1

实现方法

图像的柔化特效处理主要是通过减少图像中边缘值间的剧烈变化来实现的。当将当前像素的颜色值设为以该像素为中心的像素块中所有像素颜色的平均值时，如果当前像素和周围相邻像素的颜色差别不大时，则取平均值不会产生显著的影响；如果差别比较大时，则取平均值后，就会使当前像素点的颜色值趋于一致，这样就达到了柔化图像的目的。影响柔化图像的因素是像素块的大小及设定的系数。由于这种算法是通过高斯函数得到的，因此也称为高斯模板。实例程序实现图像柔化特效显示的示例代码如下。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  this->Label1->Caption="正在进行图像柔化处理...                ";
  this->Label1->Update();
  int iHeight=this->Image1->Picture->Width;
  int iWidth=this->Image1->Picture->Height;
  //高斯模板
  int Gauss[9]={1,2,1,2,4,2,1,2,1};
  for(int x=1;x<iWidth-1;x++)
    for(int y=1;y<iHeight-1;y++)
    {
       int r=0,g=0,b=0;
       int Index=0;
       for(int col=-1;col<=1;col++)
        for(int row=-1;row<=1;row++)
            {
              COLORREF pixel=this->Image1->Canvas->Pixels[x+row][y+col];
              r+=GetRValue(pixel)*Gauss[Index];
            g+=GetGValue(pixel)*Gauss[Index];
            b+=GetBValue(pixel)*Gauss[Index];
            Index++;
          }
       r/=16;
       g/=16;
       b/=16;
       //处理颜色值溢出
       r=r>255?255:r;
       r=r<0?0:r;
       g=g>255?255:g;
       g=g<0?0:g;
       b=b>255?255:b;
       b=b<0?0:b;
       this->Image1->Canvas->Pixels[x-1][y-1]=RGB(r,g,b);
    }
  this->Label1->Caption="图像柔化处理完毕！";
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 141 如何柔化处理图像”。

实例 142 如何锐化处理图像

实例目的

此实例主要介绍了通过拉普拉斯模板锐化图像的图像特效处理技巧。实例程序运行后，效果如图 142-1 所示。



图 142-1

实现方法

图像的锐化特效处理就是突出图像中有关形体的边缘。所谓形体的边缘就是图像像素点的颜色值发生显著变化的地方，在图像的平淡区，这种颜色值的变化比较平缓，而在图像的边缘区域这种变化则相当明显。也就是说，在平缓区，相邻两像素点颜色值的差值较小，而在边缘区域，相邻两像素点颜色值的变化则陡得多。因此如果在边缘区域处理这个数值则可能使这种突出效果更加突出，而在非边缘区域则变得较暗，即图像的锐化。通常对图像进行锐化处理使用拉普拉斯模板。实例程序使用拉普拉斯模板实现图像锐化特效的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
 this->Label1->Caption="正在锐化处理图像...          ";
 this->Label1->Update();
 int iHeight=this->Image1->Picture->Height;
 int iWidth=this->Image1->Picture->Width;
 //拉普拉斯模板
 int Laplacian[9]={-1,-1,-1,-1,9,-1,-1,-1,-1};
 for(int x=1;x<iWidth-1;x++)
```

```
    for(int y=1;y<iHeight-1;y++)
    {
      int r=0,g=0,b=0;
      int Index=0;
      for(int col=-1;col<=1;col++)
        for(int row=-1;row<=1;row++)
        {
         COLORREF pixel=this->Image1->Canvas->Pixels[x+row][y+col];
        r+=GetRValue(pixel)*Laplacian[Index];
        g+=GetGValue(pixel)*Laplacian[Index];
        b+=GetBValue(pixel)*Laplacian[Index];
        Index++;
        }
      //处理颜色值溢出
      r=r>255?255:r;
      r=r<0?0:r;
      g=g>255?255:g;
      g=g<0?0:g;
      b=b>255?255:b;
      b=b<0?0:b;
      this->Image1->Canvas->Pixels[x-1][y-1]=RGB(r,g,b);
    }
  this->Label1->Caption="锐化处理图像完毕!   ";
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 142 如何锐化处理图像”。

实例 143　如何雾化处理图像

实例目的

此实例介绍了如何雾化处理图像的编程技术。实例程序运行后，单击“图像雾化显示”按钮，效果如图 143-1 所示。



图 143-1

实现方法

图像的雾化处理不是基于像素之间的运算，而是在图像中引入一定的随机性，使图像具有毛玻璃带水雾般的效果。影响雾化效果的一个重要因素是图像中像素块的确定，所选取的像素块越大，产生的效果越明显。实例程序实现图像雾化特效的示例代码如下：

```
void __fastcall TForm1::Bit8tn1Click(TObject *Sender)
{//雾化显示图像
  for(int x=1;x<this->Image1->Picture->Width;x++)
     for(int y=1;y<this->Image1->Picture->Height;y++)
     {
      COLORREF pixel;
      double f=RAND_MAX;
      int k=(int)(24*(rand()/f));
      int dx=x+k%15;
      int dy=y+k%15;
      if(dx>=Width)
              dx=Width-1;
      if(dy>=Height)
         dy=Height-1;
        pixel=this->Image1->Canvas->Pixels[dx][dy];
        this->Image1->Canvas->Pixels[x][y]=pixel;
     }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 143 如何雾化处理图像”。

实例 144 如何将彩色图像转换成黑白图像

实例目的

此实例介绍了如何将彩色图像转换为黑白图像的编程技术。实例程序运行后，效果如图 144-1 所示。

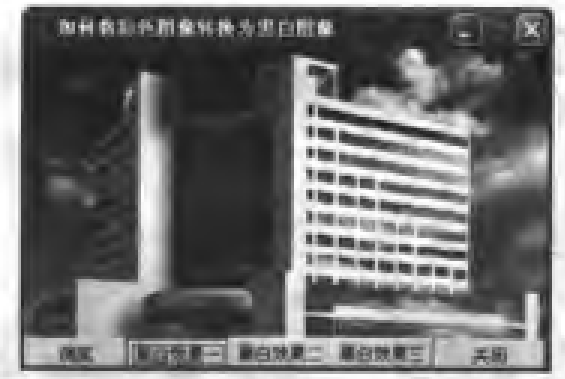


图 144-1

实现方法

利用光栅操作码能够对彩色图像进行一定程度的黑白效果处理，但真正意义上的彩色图像黑白化处理通常有3种方式：最大值法、平均值法、加权平均值法。这3种方法可以根据图像黑白处理的具体要求进行选择。但通常采用平均值法。3种方法的原理如下。

（1）最大值法：最大值法使每个像素点的R、G、B值等于原像素点的RGB值中最大的一个，即：R=G=B=MAX(GetRValue(pixel)，GetGValue(pixel),GetBValue(pixel))。最大值法会形成亮度很高的黑白图像。

（2）平均值法：平均值法使每个像素点的R、G、B值等于原像素点的颜色值中RGB值的平均值，即：R=G=B=(GetRValue(pixel)+GetGValue(pixel)+GetBValue(pixel))/3。平均值法会形成比较柔和的黑白图像。

（3）加权平均值法：加权平均值法根据需要指定每个像素点R、G、B值的权数，并取其加权平均值，即：R=G=B=(Wr*GetRValue(pixel)+Wg*GetGValue(pixel)+Wb*GetBValue(pixel))/3；Wr+Wg+Wb=3。其中：Wr代表该像素点R的权数，大于0；Wg代表该像素点G的权数，大于0；Wb代表该像素点B的权数，大于0。通过对Wr、Wg、Wb取不同的值可以获得不同的效果。

以下是实例程序采用以上3种方式实现彩色图像黑白化效果处理的示例代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//平均值法将彩色图像转换为黑白图像
   for(int i=0;i<this->Image1->Width;i++)
     for(int j=0;j<this->Image1->Height;j++)
     {
        int Pixel=this->Image1->Picture->Bitmap->Canvas->Pixels[i][j];
        int ColorRed=GetRValue(Pixel);
        int ColorGreen=GetGValue(Pixel);
        int ColorBlue=GetBValue(Pixel);
        int NewPixel=((ColorRed+ColorGreen+ColorBlue)/3);
        this->Image1->Picture->Bitmap->Canvas->Pixels[i][j]=
RGB(NewPixel,NewPixel,NewPixel);
     }
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//最大值法将彩色图像转换为黑白图像
   for(int i=0;i<this->Image1->Width;i++)
     for(int j=0;j<this->Image1->Height;j++)
     {
        int Pixel=this->Image1->Picture->Bitmap->Canvas->Pixels[i][j];
        int ColorRed=GetRValue(Pixel);
        int ColorGreen=GetGValue(Pixel);
        int ColorBlue=GetBValue(Pixel);
        int NewPixel=ColorRed>ColorGreen?ColorRed:ColorGreen;
        NewPixel=NewPixel>ColorBlue?NewPixel:ColorBlue;
        this->Image1->Picture->Bitmap->Canvas->Pixels[i][j]=
RGB(NewPixel,NewPixel,NewPixel);
```

```
    }
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{//加权平均值法将彩色图像转换为黑白图像
  for(int i=0;i<this->Image1->Width;i++)
    for(int j=0;j<this->Image1->Height;j++)
    {
      int Pixel=this->Image1->Picture->Bitmap->Canvas->Pixels[i][j];
      int ColorRed=GetRValue(Pixel);
      int ColorGreen=GetGValue(Pixel);
      int ColorBlue=GetBValue(Pixel);
      int NewPixel=ColorRed*0.6+ColorGreen*0.3+ColorBlue*0.1;
      this->Image1->Picture->Bitmap->Canvas->Pixels[i][j]=
        RGB(NewPixel,NewPixel,NewPixel);
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 144 如何将彩色图像转换成黑白图像”。

实例 145 如何将 BMP 图像文件转换为 JPG 图像文件

实例目的

此实例介绍了如何将 BMP 图像文件转换为 JPG 图像文件的编程技术。实例程序运行后，效果如图 145-1 所示。



图 145-1

实现方法

BMP 格式图像文件是 Windows 操作系统的标准图像文件格式，这种格式的图像文件未经过压缩处理，失真度小，能够比较真实地反映图像的本来面貌，缺点是保存时占用存

储空间较大。实际上，任何格式的图像文件在 Windows 操作系统上显示时，都是以 BMP 格式显示的，只是在保存为具体的图像文件时，才有压缩和未压缩之分。JPEG 格式图像文件是一种经过压缩处理的图像文件，这种格式的图像文件的最大优点是文件尺寸较小，压缩比较高，一幅同样大小的图像文件，BMP 格式文件尺寸大小与 JPEG 格式文件尺寸大小相差几倍甚至数十倍。C++Builder 提供了 TJPEGImage 对象，能够非常简单地处理 JPEG 格式的图像文件，但是在使用时必须包括头文件：#include"jpeg.hpp"。以下是在实例程序中将图像保存为 JPEG 格式文件的示例代码。

```
#include"jpeg.hpp"
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  if(this->SavePictureDialog1->Execute())
  {
    TJPEGImage* Jpeg=new TJPEGImage();
    Jpeg->Assign(this->Image1->Picture->Bitmap);
    Jpeg->SaveToFile(this->SavePictureDialog1->FileName+".jpg");
    MessageBox(Handle,"图像保存为JPG格式文件操作成功！","信息提示",MB_OK);
    delete Jpeg;
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 145 如何将 BMP 图像文件转换为 JPG 图像文件”。

实例 146　如何全屏播放 AVI 文件

实例目的

此实例主要介绍了如何全屏播放 AVI 文件的编程技术。实例程序执行后，单击“全屏播放 AVI 文件”按钮，实例程序将全屏播放测试用的 AVI 文件，效果如图 146-1 所示。



图 146-1

实现方法

使用 MCI 控制播放 AVI 文件通常有两种方式：发送命令和发送字符串，分别对应的两个函数 mciSendString()和 mciSendCommand()，它们能够实现播放、暂停、记录、保存、停止等声音操作功能。mciSendString()函数的声明如下：

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

其中，参数 LPCTSTR lpszCommand 为命令字符串，参数 HANDLE hwndCallback 为窗口句柄。

通常使用 mciSendString()函数实现有窗口播放的代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
 mciSendString("open Clock.avi type avivideo alias video", NULL,0, 0);
  mciSendString("set video time format ms", NULL,0, 0);
  mciSendString("play video from 0", NULL, 0, 0);
}
```

如果需要实现全屏幕播放 AVI 文件，则可以参考以下代码：

```
//Filename: Mainform.hpp
#ifndef MainformH
#define MainformH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "MMSystem.h"
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
class TForm1 : public TForm
{
__published:   // IDE-managed Components
       TBitBtn *BitBtn1;
       TImage *Image1;
       void __fastcall BitBtn1Click(TObject *Sender);
private:   // User declarations
public:         // User declarations
   __fastcall TForm1(TComponent* Owner);
   void virtual __fastcall mciNotify(TMessage& Msg);
   BEGIN_MESSAGE_MAP
   MESSAGE_HANDLER(MM_MCINOTIFY,TMessage,mciNotify)
   END_MESSAGE_MAP(TForm)
};
extern PACKAGE TForm1 *Form1;
#endif
```

```
//Filename: Mainform.cpp
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  mciSendString("open Clock.avi alias video", NULL,0, 0);
  mciSendString("window video handle 0", NULL,0, 0);
  mciSendString("window video state show maximized", NULL,0, 0);
  mciSendString("play video notify", NULL, 0, Handle);
}
void __fastcall TForm1::mciNotify(TMessage& Msg)
{
  mciSendString("close video", NULL,0, 0);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 146 如何全屏播放 AVI 文件”。

实例 147　如何制作媒体播放器

实例目的

此实例主要演示了使用 TMediaPlayer 组件编写媒体播放器的编程技术。当实例程序启动后，效果如图 147-1 所示。



图 147-1

实现方法

C++Builder 的组件库内包括一个多媒体播放组件 TMediaPlayer，这个组件允许播放 WAV 声音文件、MIDI 乐器声音文件、AVI 视频文件。在 C++Builder 中使用该组件创建媒体播放器时，只需先设定其 Display 属性确定视频的显示位置，设定 FileName 属性确定播放的媒体文件，最后调用 Open()函数直接播放媒体文件即可。具体步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 System 标签页下拖动 MediaPlayer1:TMediaPlayer 组件到 Form1 上。

（2）在组件模板的 Standard 标签页下拖动 Panel1:TPanel 组件到 Form1 上。

（3）在组件模板的 Dialogs 标签页下拖动 OpenDialog1:TOpenDialog 组件到 Form1 上。

（4）在组件模板的 Additional 标签页下拖动 BitBtn1:TBitBtn 组件到 Form1 上，在 Object Inspector\Events 中添加 OnClick 事件函数，该函数的实现代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
   this->OpenDialog1->Options.Clear();
   this->OpenDialog1->Filter= "Voice files (*.wav)|*.wav|
Midi files (*.mid)|*.mid|Avi files (*.avi)|*.avi";
   this->OpenDialog1->Title="选择播放文件";
   this->OpenDialog1->InitialDir="C:\\";
  if(this->OpenDialog1->Execute())
  {
    this->MediaPlayer1->FileName=this->OpenDialog1->FileName;
    this->MediaPlayer1->Display=this->Panel1;
    this->MediaPlayer1->Open();
  }
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第四部分 图像处理\实例 147 如何制作媒体播放器”。

第五部分　系统编程

实例 148　如何获取键盘类型

实例目的

此实例主要用于取得用户使用的键盘类型。当实例程序启动后，单击“获取”按钮，即可取得当前用户所使用的键盘类型，效果如图 148-1 所示。



图 148-1

实现方法

C++Builder 允许直接在程序中调用 WIN32 API 函数，这是 C++Builder 一项非常强大的功能，有了这项功能，C++Builder 就变得无所不能。在 C++Builder 中直接调用 WIN32 API 函数编写应用程序时，必须先找出会使用到的 API 是哪一类型的 API，然后找出这些 API 函数的功能描述及函数格式，C++Builder 的在线帮助就包含有 WIN32 API 函数的有关说明，位置在 C:\Program Files\Borland\CBuilder6\Help\Win32.hlp，直接调用之即可。下面是获取键盘类型的 API 函数的声明：

```
int GetKeyboardType(int nTypeFlag);
```

其中，参数 int nTypeFlag 表示获取信息的类型：0 表示键盘类型；1 表示键盘子类型；2 表示键盘的按钮数目。该函数的返回值表示的意义如表 148-1 所示。

表 148-1

返回值	意义
1	IBM PC/XT 或兼容（83 键）键盘
2	Olivetti "TCO"（102 键）键盘
3	IBM PC/AT（84 键）或同类键盘
4	IBM 增强（101 键或 102 键）键盘
5	诺基亚 1050 和同类键盘
6	诺基亚 9140 和同类键盘
7	日式键盘

实例程序使用 GetKeyboardType()函数获取键盘类型的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//获取键盘类型
   int iType=GetKeyboardType(0);
   switch(iType)
   {
    case 1:
      this->Edit1->Text="IBM PC/XT or compatible (83-key) keyboard";
      break;
    case 2:
      this->Edit1->Text="Olivetti 'ICO' (102-key) keyboard";
      break;
    case 3:
      this->Edit1->Text="IBM PC/AT (84-key) or similar keyboard";
      break;
    case 4:
      this->Edit1->Text="IBM enhanced (101- or 102-key) keyboard";
      break;
    case 5:
      this->Edit1->Text="Nokia 1050 and similar keyboards";
      break;
    case 6:
      this->Edit1->Text="Nokia 9140 and similar keyboards";
      break;
    case 7:
      this->Edit1->Text="Japanese keyboard";
      break;
   }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 148 如何获取键盘类型”。

实例 149 如何获取声卡配置

实例目的

此实例主要用于获取系统已经安装的声卡的类型。当实例程序执行后，效果如图 149-1 所示。



图 149-1

实现方法

在设计多媒体程序时，程序常常会使用声音文件以增强效果。通常在安装这种程序时，安装程序首先应该检测和配置声卡，以防止程序因缺少硬件而出现不必要的麻烦。对声卡的检测，可以分别通过 API 函数 waveOutGetNumDevs()和 midiOutGetNumDevs()检测波形设备和 MIDI 设备，再利用 waveOutGetDevCaps()和 midiOutGetDevCaps()获得声音设备的细节资料。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
    int wavedevice,mididevice;
    WAVEOUTCAPS wavecap;
    MIDIOUTCAPS midicap;
    wavedevice=(int)waveOutGetNumDevs();
    mididevice=(int)midiOutGetNumDevs();
    if (wavedevice==0)
      this->Edit1->Text="没有发现波形设备";
    else
    {
      waveOutGetDevCaps(0,&wavecap,sizeof(WAVEOUTCAPS));
      this->Edit1->Text=String(wavecap.szPname);
    }
    if (mididevice==0)
      this->Edit2->Text="没有发现 MIDI 设备";
    else
    {
      midiOutGetDevCaps(0,&midicap,sizeof(MIDIOUTCAPS));
      this->Edit2->Text=String(midicap.szPname);
    }
}
```

注意：以上获取声卡信息的函数需要头文件#include <mmsystem.h>的支持。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 149 如何获取声卡配置”。

实例 150 如何获取处理器信息

实例目的

此实例主要介绍了获取处理器相关信息的编程技术。当实例程序启动后，单击“获取”按钮，即可取得当前用户所使用处理器的相关信息，效果如图 150-1 所示。

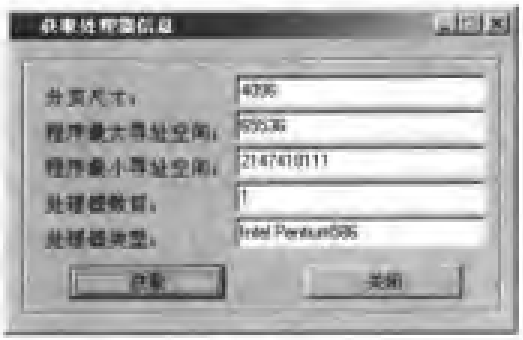


图 150-1

实现方法

使用硬件信息结构 SYSTEM_INFO 和 API 函数 GetSystemInfo()可以查询计算机系统处理器的相关信息。API 函数 GetSystemInfo()的声明如下：

```
VOID GetSystemInfo(
    LPSYSTEM_INFO lpSystemInfo  // 系统信息结构变量
);
```

其中，参数 SYSTEM_INFO 结构包含有当前计算机系统的处理器类型的架构、处理器的数量、页大小等。该结构的声明如下：

```
typedef struct _SYSTEM_INFO {
    union {
    DWORD  dwOemId;                          //测试早期版本的成员，现已不用
    struct {
        WORD wProcessorArchitecture;         //处理器架构
        WORD wReserved;                      //保留值
        };
    };
    DWORD  dwPageSize;                       //页大小
    LPVOID lpMinimumApplicationAddress;      //可访问的最低内存地址
    LPVOID lpMaximumApplicationAddress;      //可访问的最高内存地址
    DWORD_PTR dwActiveProcessorMask;         //掩码标志，位 0 是处理器 0
    DWORD dwNumberOfProcessors;              //处理器个数
    DWORD dwProcessorType;                   //处理器类型，测试早期版本的成员，现已不用
    DWORD dwAllocationGranularity;           //分配虚内存的间隔
    WORD wProcessorLevel;                    //处理器等级
    WORD wProcessorRevision;                 //处理器版本
} SYSTEM_INFO;
```

实例程序使用硬件信息结构 SYSTEM_INFO 和 API 函数 GetSystemInfo()查询系统处理器信息的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  SYSTEM_INFO si;
  GetSystemInfo(&si);
  this->Edit1->Text=IntToStr(si.dwPageSize);
  this->Edit2->Text=IntToStr(Integer(si.lpMinimumApplicationAddress));
  this->Edit3->Text=IntToStr(Integer(si.lpMaximumApplicationAddress));
  this->Edit4->Text=IntToStr(si.dwNumberOfProcessors);
  switch(si.dwProcessorType)
  {
    case 386:
    case 486:
      this->Edit5->Text='Intel 80'+IntToStr(si.dwProcessorType);
      break;
    case 586:
      this->Edit5->Text='Intel Pentium'+IntToStr(si.dwProcessorType);
      break;
    default:
      this->Edit5->Text='非 Intel 处理器';
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 150 如何获取处理器信息”。

实例 151　如何获取系统元素颜色

实例目的

此实例主要介绍了如何获取系统元素颜色的编程技术。实例程序执行后，单击“获取 3D 按钮阴影颜色”按钮，效果如图 151-1 所示。



图 151-1

实现方法

通过使用操作系统提供的 API 函数 GetSysColor()可以获取系统元素的颜色，该函数的声明如下：

```
DWORD GetSysColor(
    int nIndex  //显示元素
   );
```

其中，参数 int nIndex 的取值包括 COLOR_3DDKSHADOW、COLOR_3DFACE、COLOR_BTNFACE、COLOR_3DHILIGHT、COLOR_3DHIGHLIGHT、COLOR_BTNHILIGHT、COLOR_BTNHIGHLIGHT、COLOR_3DLIGHT、COLOR_3DSHADOW、COLOR_BTNSHADOW、COLOR_ACTIVEBORDER、CGLOR_ACTIVECAPTION、COLOR_APPWORKSPACE、COLOR_BACKGROUND、COLOR_DESKTOP、COLOR_BTNTEXT、COLOR_CAPTIONTEXT、COLOR_GRAYTEXT、COLOR_HIGHLIGHT、COLOR_HIGHLIGHTTEXT、COLOR_INACTIVEBORDER、COLOR_INACTIVECAPTION、COLOR_INACTIVECAPTIONTEXT、COLOR_INFOBK、COLOR_INFOTEXT、COLOR_MENU、COLOR_MENUTEXT、COLOR_SCROLLBAR、COLOR_WINDOW、COLOR_WINDOWFRAME、COLOR_WINDOWTEXT。

也可以使用 API 函数 SetSysColors()设置系统元素的颜色，函数 SetSysColors()的声明如下：

```
BOOL WINAPI SetSysColors(
    int cElements,                  //显示元素的个数
    CONST INT *lpaElements,         //显示元素数组
    CONST COLORREF *lpaRgbValues    //颜色数组
   );
```

实例程序使用 API 函数 GetSysColor()获取系统部分元素颜色的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  this->Color=GetSysColor(COLOR_WINDOW);
  MessageBox(Handle,"程序窗口显示的颜色即为系统设置的窗口背景颜色","信息提示
",MB_OK);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  this->Color=GetSysColor(COLOR_BACKGROUND);
  MessageBox(Handle,"程序窗口显示的颜色即为系统设置的桌面背景颜色","信息提示
",MB_OK);
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
  this->Color=GetSysColor(COLOR_ACTIVECAPTION);
  MessageBox(Handle,"程序窗口显示的颜色即为系统设置的活动窗口标题颜色",
"信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
```

```
  this->Color=GetSysColor(COLOR_INACTIVECAPTION);
  MessageBox(Handle,"程序窗口显示的颜色即为系统设置的非活动窗口标题颜色",
"信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{
  this->Color=GetSysColor(COLOR_MENU);
  MessageBox(Handle,"程序窗口显示的颜色即为系统设置的菜单颜色",
"信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn6Click(TObject *Sender)
{
  this->Color=GetSysColor(COLOR_BTNSHADOW);
  MessageBox(Handle,"程序窗口显示的颜色即为系统设置的3D按钮阴影颜色",
"信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分　系统编程\实例151 如何获取系统元素颜色”。

实例152　如何获取显示元素信息

实例目的

此实例主要用于获取系统的屏幕分辨率和像素值。当实例程序执行后，效果如图152-1所示。



图152-1

实现方法

屏幕分辨率和像素值是程序员在编写程序时必须事先考虑的问题，它对程序界面的规划和设计有着相当重要的影响。实际上，在使用C++Builder编写程序时，屏幕分辨率的求法很简单，直接调用Screen对象的属性就行了。而像素值则要利用API函数GetDeviceCaps获得。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
```

```
{
    int tcs;
    long int bpp,cp,tc;
    this->Edit1->Text=String(Screen->Width)+"*"+String(Screen->Height);
    bpp=GetDeviceCaps(Form1->Canvas->Handle,BITSPIXEL);
    this->Edit2->Text=String(bpp);
}
```

注意：以上代码需要头文件#include"math.h"的支持。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 152 如何获取显示元素信息”。

实例 153 如何获取系统支持的显示模式

实例目的

此实例介绍了如何获取系统支持的显示模式的编程技术。实例程序执行后，效果如图 153-1 所示。



图 153-1

实现方法

通过系统 API 函数 EnumDisplaySettings()可以获取指定显示设备支持的显示模式。该函数的声明如下：

```
BOOL EnumDisplaySettings(
    LPCTSTR lpszDeviceName,     //指定显示设备
    DWORD iModeNum,             //显示模式索引
    LPDEVMODE lpDevMode         //包含显示模式信息的结构
);
```

其中，当参数 LPCTSTR lpszDeviceName 为 NULL 时表示当前显示设备。当参数 DWORD iModeNum 为 ENUM_CURRENT_SETTINGS 时表示获取当前显示模式的信息；当参数 DWORD iModeNum 为 ENUM_REGISTRY_SETTINGS 时表示获取注册表中保存的

显示模式的信息；可以采用将此值首先设置为0，然后逐渐加1，直到函数的返回值为FALSE的方法来获取系统所有的显示模式。实例程序使用 API 函数 EnumDisplaySettings()获取指定显示设备支持的显示模式的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  while(this->ListBox1->Items->Count>0)
    this->ListBox1->Items->Delete(0);
  DEVMODE *lpDevMode;
  lpDevMode=new DEVMODE;
  int  i=0,nIndex;
  BOOL Result=EnumDisplaySettings(NULL,i,lpDevMode);
  AnsiString StrDisplayMode="系统支持的显示模式包括：\n";;
  while(Result)
  {
    AnsiString Str;
    Str+=IntToStr(lpDevMode->dmPelsWidth);
    Str+="x";
    Str+=IntToStr(lpDevMode->dmPelsHeight);
    Str+="x";
    Str+=IntToStr(lpDevMode->dmBitsPerPel);
    StrDisplayMode+=Str;
    this->ListBox1->Items->Add(Str);
    lpDevMode=new DEVMODE;
    Result=EnumDisplaySettings(NULL,++i,lpDevMode);
    delete lpDevMode;
  }
  delete lpDevMode;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 153 如何获取系统支持的显示模式”。

实例 154　如何动态设置系统的显示模式

实例目的

此实例介绍了如何动态设置系统显示模式的编程技术。实例程序执行后，效果如图154-1所示。

实现方法

通过系统 API 函数 ChangeDisplaySettings()可以设置指定显示设备的显示模式。该函数的声明如下：

```
LONG ChangeDisplaySettings(
```

```
        LPDEVMODE lpDevMode,        //显示模式
        DWORD dwflags               //显示模式选项
);
```



图 154-1

其中，参数 LPDEVMODE lpDevMode 包含将要设置的显示模式信息的 DEVMODE 结构的指针，可以设置为下列成员：dmBitsPerPel、dmPelsWidth、dmPelsHeight、dmDisplayFlags、dmDisplayFrequency、dmPosition。参数 DWORD dwflags 指定 DEVMODE 结构中的哪些成员将用于改变显示模式的设置。可以取下列值之一：0（动态改变当前屏幕的显示模式）、CDS_FULLSCREEN（临时设置）、CDS_GLOBAL（设置结果对所有用户有效）、CDS_NORESET（显示模式设置将被保存在注册表中，但不影响当前的显示模式，该标志必须与 CDS_UPDATEREGISTRY 标志一起使用）、CDS_RESET（表示显示模式设置将被改变，即使所请求的显示模式和当前的设置相同）、CDS_SET_PRIMARY（指定显示设备将成为主显示设备）、CDS_TEST（测试所请示的显示模式是否可以被设置）、CDS_UPDATEREGISTRY（动态改变当前屏幕的显示模式，并把该显示模式保存到注册表的用户配置信息中）。返回值的可能值包括：DISP_CHANGE_SUCCESSFUL（设置成功）、DISP_CHANGE_BADDUALVIEW（设置成功，Windows XP 专用）、DISP_CHANGE_BADFLAGS（参数 flags 设置错误）、DISP_CHANGE_BADMODE（不支持的显示模式）、DISP_CHANGE_BADPARAM（参数设置有误）、DISP_CHANGE_FAILED（设置失败）、DISP_CHANGE_NOTUPDATED（无法更新注册表，仅适用于 Windows NT/2000/XP）、DISP_CHANGE_RESTART（必须重新启动计算机才能使设置生效）。实例程序使用系统函数 ChangeDisplaySettings()设置指定显示设备的显示模式的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(MessageBox(Handle,"是否将当前显示模式设置为：1024*768*32 ?",
"信息提示",MB_YESNO)==IDNO)
    return;
  DEVMODE *lpDevMode;
  lpDevMode=new DEVMODE;
  int  i=0,nIndex;
  BOOL Result=EnumDisplaySettings(NULL,i,lpDevMode);
  while(Result)
```

```
    {
      if(lpDevMode->dmPelsWidth==1024&&lpDevMode->dmPelsHeight==768&&
 lpDevMode->dmBitsPerPel==32)
      {
            lpDevMode->dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;
            LONG lResult=ChangeDisplaySettings(lpDevMode,0);
            if(lResult==DISP_CHANGE_SUCCESSFUL)
            MessageBox(Handle,"设置显示模式成功","信息提示",MB_OK);
          else if(lResult==DISP_CHANGE_RESTART)
            MessageBox(Handle,"设置显示模式成功","信息提示",MB_OK);
          else
            MessageBox(Handle,"设置显示模式失败","信息提示",MB_OK);
      }
      lpDevMode=new DEVMODE;
      Result=EnumDisplaySettings(NULL,++i,lpDevMode);
    }
    delete lpDevMode;
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
    if(MessageBox(Handle,"是否将当前显示模式设置为：800*600*32?",
"信息提示",MB_YESNO)==IDNO)
      return;
    DEVMODE *lpDevMode;
    lpDevMode=new DEVMODE;
    int  i=0,nIndex;
    BOOL Result=EnumDisplaySettings(NULL,i,lpDevMode);
    while(Result)
    {
      if(lpDevMode->dmPelsWidth==800&&lpDevMode->dmPelsHeight==600&&
lpDevMode->dmBitsPerPel==32)
      {
            lpDevMode->dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;
            LONG lResult=ChangeDisplaySettings(lpDevMode,0);
            if(lResult==DISP_CHANGE_SUCCESSFUL)
             MessageBox(Handle,"设置显示模式成功","信息提示",MB_OK);
          else if(lResult==DISP_CHANGE_RESTART)
             MessageBox(Handle,"设置显示模式成功","信息提示",MB_OK);
          else
             MessageBox(Handle,"设置显示模式失败","信息提示",MB_OK);
      }
      lpDevMode=new DEVMODE;
      Result=EnumDisplaySettings(NULL,++i,lpDevMode);
    }
    delete lpDevMode;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统

编程\实例 154 如何动态设置系统的显示模式”。

实例 155 如何获取内存空间信息

实例目的

此实例主要用于获取系统内存的空间大小信息，包括实际内存总大小、可用的实际内存大小、虚拟内存大小、可用的虚拟内存大小等。当实例程序启动后，程序会随时自动更新系统内存的状态，效果如图 155-1 所示。

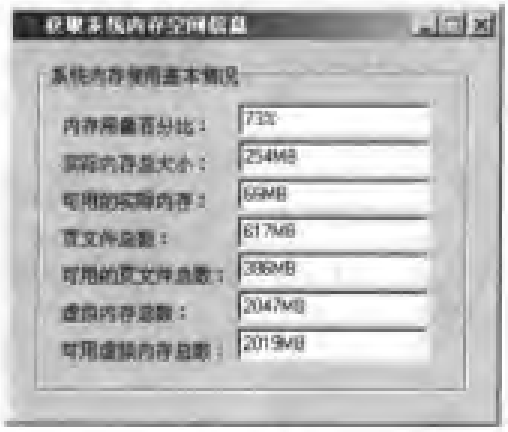


图 155-1

实现方法

使用系统 API 函数 GlobalMemoryStatus()可以获取系统的总内存、已用内存、可用内存的情况。该函数的声明如下：

```
VOID GlobalMemoryStatus(
  LPMEMORYSTATUS lpBuffer   //内存状态结构
);
```

其中，内存状态结构的声明如下：

```
typedef struct _MEMORYSTATUS {
  DWORD dwLength;                    //结构长度
  DWORD dwMemoryLoad;                //使用内存百分比
  SIZE_T dwTotalPhys;                //总物理内存
  SIZE_T dwAvailPhys;                //可用内存
  SIZE_T dwTotalPageFile;            //总页面文件
  SIZE_T dwAvailPageFile;            //可用页面文件数
  SIZE_T dwTotalVirtual;             //总虚拟内存
  SIZE_T dwAvailVirtual;             //可用虚拟内存
} MEMORYSTATUS, *LPMEMORYSTATUS;
```

实例程序使用函数 GlobalMemoryStatus()获取系统内存基本情况的示例代码如下：

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

```
{
    TMemoryStatus vm;
    vm.dwLength=sizeof(TMemoryStatus);
    GlobalMemoryStatus(&vm);
    this->Edit1->Text=IntToStr(vm.dwMemoryLoad)+"%";
    this->Edit2->Text=IntToStr(vm.dwTotalPhys/1024/1024)+"MB";
    this->Edit3->Text=IntToStr(vm.dwAvailPhys/1024/1024)+"MB";
    this->Edit4->Text=IntToStr(vm.dwTotalPageFile/1024/1024)+"MB";
    this->Edit5->Text=IntToStr(vm.dwAvailPageFile/1024/1024)+"MB";
    this->Edit6->Text=IntToStr(vm.dwTotalVirtual/1024/1024)+"MB";
    this->Edit7->Text=IntToStr(vm.dwAvailVirtual/1024/1024)+"MB";
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 155 如何获取内存空间信息”。

实例 156　如何获取磁盘类型

实例目的

此实例主要用于判断系统指定磁盘驱动器的类型，当实例程序启动后，效果如图 156-1 所示。



图 156-1

实现方法

Windows 操作系统通常按照 A:、B:（为软驱）、C:（为硬盘），C:盘以后的存储器为硬盘、光驱等顺序排列。通过操作系统 API 函数 GetDriveType()可以对系统磁盘驱动器的类型进行判断，该函数的声明如下：

```
UINT GetDriveType(
    LPCTSTR lpRootPathName   //根目录或文件夹
);
```

根据该函数的返回值可以对磁盘类型进行判断，返回值的意义如表 156-1 所示。

表 156-1

返回值	说明
DRIVE_UNKNOWN	未知的驱动器
DRIVE_NO_ROOT_DIR	根目录无效
DRIVE_REMOVABLE	可移动磁盘
DRIVE_FIXED	硬盘
DRIVE_REMOTE	网络驱动器
DRIVE_CDROM	光驱
DRIVE_RAMDISK	RAM 磁盘

如果返回值为 0，表示找不到驱动器类型。实例程序使用 GetDriveType()函数判断系统指定磁盘类型的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  char cDrive=this->DriveComboBox1->Drive;
  AnsiString StrDrive=cDrive;
  StrDrive+=":";
  AnsiString  StrDisktype;
  switch( ::GetDriveType(StrDrive.c_str()) )
  {
   case 1:
     StrDisktype="路径未知";
   case DRIVE_REMOVABLE:
     StrDisktype="软驱";
     break;
   case DRIVE_FIXED:
     StrDisktype="固定硬盘";
     break;
   case DRIVE_REMOTE:
     StrDisktype="网络驱动器";
     break;
   case DRIVE_CDROM:
     StrDisktype="光驱";
     break;
   case DRIVE_RAMDISK:
     StrDisktype="RAM 驱动器";
     break;
   default:
     StrDisktype="未知";
     break;
  }
  this->Edit1->Text=StrDisktype;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分　系统编程\实例 156 如何获取磁盘类型”。

实例 157　如何获取磁盘卷信息

实例目的

此实例主要用于取得系统磁盘的卷及其相关信息。当实例程序启动后，单击“获取”按钮，即可取得系统磁盘的卷及其相关信息，效果如图 157-1 所示。



图 157-1

实现方法

每个磁盘都有一个惟一标识的序列号，用户可以通过检测磁盘驱动器中磁盘的卷标号是否变化来确定磁盘是否被切换过。使用系统 API 函数 GetVolumeInformation()可以获得这些相关信息，该函数的声明如下：

```
BOOL GetVolumeInformation(
    LPCTSTR lpRootPathName,          // 根目录字符串
    LPTSTR lpVolumeNameBuffer,       // 卷标名
    DWORD nVolumeNameSize,           // 卷标名长度
    LPDWORD lpVolumeSerialNumber,    // 磁盘序列号
    LPDWORD lpMaximumComponentLength,// 系统最大文件名长度
    LPDWORD lpFileSystemFlags,       // 文件系统标志
    LPTSTR lpFileSystemNameBuffer,   // 文件系统名称
    DWORD nFileSystemNameSize        // 文件系统名字符串长度
);
```

其中，参数 LPCTSTR lpRootPathName 若为 NULL，则使用当前目录的根路径，该参数也可以是一个 UNC 名，此时必须以“\”结束，例如：\\MyServer\\MyShare\。

参数 LPDWORD lpFileSystemFlags 的可能值包括 FS_CASE_IS_PRESERVED、FS_CASE_SENSITIVE、FS_UNICODE_STORED_ON_DISK、FS_PERSISTENT_ACLS、FS_FILE_COMPRESSION、FS_VOL_IS_COMPRESSED、FILE_NAMED_STREAMS、

FILE_READ_ONLY_VOLUME 、 FILE_SUPPORTS_ENCRYPTION 、 FILE_SUPPORTS_OBJECT_IDS、FILE_SUPPORTS_REPARSE_POINTS、FILE_SUPPORTS_SPARSE_FILES、FILE_VOLUME_QUOTAS。

实例程序使用系统函数 GetVolumeInformation()获取系统指定磁盘卷信息的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  char cDrive=this->DriveComboBox1->Drive;
  AnsiString StrDrive=cDrive;
  StrDrive+=":";
  char VolName[200],FSName[50];
  DWORD dwVolSer,lpMaxComLen,lpFSFlags;
 if( GetVolumeInformation(StrDrive.c_str(),VolName,200,
&dwVolSer,&lpMaxComLen,&lpFSFlags,FSName,50))
 {
  if(dwVolSer==0)
     this->Edit1->Text="";
  else
    this->Edit1->Text=IntToStr(Integer(dwVolSer));;
  this->Edit2->Text=VolName;
  this->Edit3->Text=FSName;
 }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 157 如何获取磁盘卷信息”。

实例 158 如何获取磁盘空间信息

实例目的

此实例主要用于获取系统磁盘的空间大小信息，包括总空间大小、已经使用的空间大小、未使用的空间大小。当实例程序启动后，单击“获取”按钮，即可取得系统指定磁盘的空间大小信息，效果如图 158-1 所示。



图 158-1

实现方法

通过系统 API 函数 GetDiskFreeSpaceEx()可以获取指定磁盘及文件夹的空间大小，该函数的声明如下：

```
BOOL GetDiskFreeSpaceEx(
  LPCTSTR lpDirectoryName,                 //文件夹名称
  PULARGE_INTEGER lpFreeBytesAvailable,    //可用自由空间大小
  PULARGE_INTEGER lpTotalNumberOfBytes,    //总空间大小
  PULARGE_INTEGER lpTotalNumberOfFreeBytes //总自由空间大小
);
```

此外，在C++Builder中提供的DiskFree()和DiskSize()函数也可用于获取磁盘空间大小。DiskFree()函数用于检测磁盘剩余空间，该函数的声明如下：

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

其中，参数 Byte Drive 为磁盘的代号，0 表示当前盘，1=A，2=B，3=C，以此类推，如果指定的磁盘无效，返回-1。函数的返回值是以字节为单位表示的磁盘剩余空间。例如，执行语句 ShowMessage(DiskFree(0));将显示当前盘的剩余空间。

DiskSize()函数用于检测磁盘容量，该函数的声明如下：

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

其中，参数 Byte Drive 为磁盘的代号，0 表示当前盘，1=A，2=B，3=C 以此类推，如果指定的磁盘无效，返回-1。函数的返回值是以字节为单位表示的磁盘空间。例如，执行语句 ShowMessage(DiskSize(0));将显示当前盘的空间。

实例程序使用 GetDiskFreeSpaceEx()函数获取系统指定磁盘空间大小的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
   char cDrive=this->DriveComboBox1->Drive;
   AnsiString StrDrive=cDrive;
   StrDrive+=":";
   ULARGE_INTEGER nTotalBytes, nTotalFreeBytes, nTotalAvailable;
  if( ::GetDiskFreeSpaceEx(StrDrive.c_str(),
&nTotalAvailable,&nTotalBytes, &nTotalFreeBytes))
  {
     Edit1->Text=IntToStr((int)(nTotalBytes.QuadPart/1024/1024))+"MB";
     Edit3->Text=IntToStr((int)(nTotalFreeBytes.QuadPart/1024/1024))+"MB";
     this->Edit2->Text=IntToStr((int)(nTotalBytes.QuadPart/1024/1024)-
(int)(nTotalFreeBytes.QuadPart/1024/1024))+"MB";
  }
  else
  {
     this->Edit1->Text="";
     this->Edit3->Text="";
     this->Edit2->Text="";
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 158 如何获取磁盘空间信息”。

实例 159 如何在我的电脑中隐藏磁盘

实例目的

此实例介绍了如何在“我的电脑”中隐藏驱动器的编程技术。实例程序执行后，在选择需要隐藏的驱动器后，单击“隐藏磁盘驱动器”按钮，重启或注销计算机后该驱动器将不会在我的电脑中出现；单击“显示磁盘驱动器”按钮，则会取消所有隐藏的驱动器，效果如图 159-1 所示。



图 159-1

实现方法

通过对注册表的修改，可以在“我的电脑”和“资源管理器”中隐藏选择的驱动器。例如，要隐藏 C 盘，注册表修改操作如下：打开注册表子键 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer，新建一个键值 NoDrives 并设置其二进制值为 04000000，关闭注册表编辑器，重新启动计算机后，可以发现“我的电脑”中没有 C 盘了。表 159-1 是隐藏磁盘驱动器的相关键值数据。

表 159-1

键值	数据类型	数据	说明
NoDrives	二进制值	00000000	不隐藏任何盘
		01000000	隐藏 A 盘
		02000000	隐藏 B 盘
		04000000	隐藏 C 盘
		08000000	隐藏 D 盘
		10000000	隐藏 E 盘
		20000000	隐藏 F 盘

续表

键值	数据类型	数据	说明
		40000000	隐藏 G 盘
		80000000	隐藏 H 盘
		00010000	隐藏 I 盘
		00020000	隐藏 J 盘
		00040000	隐藏 K 盘
		00080000	隐藏 L 盘
		00100000	隐藏 M 盘
		00200000	隐藏 N 盘
		00400000	隐藏 O 盘
		00800000	隐藏 P 盘
		00000100	隐藏 Q 盘
		00000200	隐藏 R 盘
		00000400	隐藏 S 盘
		00000800	隐藏 T 盘
		00001000	隐藏 U 盘
		00002000	隐藏 V 盘
		00004000	隐藏 W 盘
		00008000	隐藏 X 盘
		00000001	隐藏 Y 盘
		00000002	隐藏 Z 盘
		FFFFFFFF	隐藏所有驱动器

如果将这些数据进行累加，则所有累加数据所代表的驱动器都将会被隐藏。下面是实例程序实现在“我的电脑”和“资源管理器”中隐藏磁盘驱动器的具体代码。

```
#include "Registry.hpp"
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//隐藏磁盘驱动器
  long NoDrives=0;
  switch(this->DriveComboBox1->Drive)
  {
   case 'a':
     NoDrives=1;
     break;
   case 'b':
     NoDrives=2;
     break;
   case 'c':
     NoDrives=4;
     break;
```

```
        case 'd':
          NoDrives=8;
          break;
        case 'e':
          NoDrives=16;
          break;
        case 'f':
          NoDrives=32;
          break;
        case 'g':
          NoDrives=64;
          break;
        case 'h':
          NoDrives=128;
          break;
        case 'i':
          NoDrives=256;
          break;
        case 'j':
          NoDrives=512;
          break;
        case 'k':
          NoDrives=1024;
          break;
        case 'l':
          NoDrives=2048;
          break;
        case 'm':
          NoDrives=4096;
          break;
        case 'n':
          NoDrives=8192;
          break;
        case 'o':
          NoDrives=16384;
          break;
        case 'p':
          NoDrives=32768;
          break;
        case 'q':
          NoDrives=65536;
          break;
        case 'r':
          NoDrives=131072;
          break;
        case 's':
          NoDrives=262144;
          break;
        case 't':
          NoDrives=524288;
```

```
            break;
         case 'u':
            NoDrives=1084576;
            break;
         case 'v':
            NoDrives=2169152;
            break;
         case 'w':
            NoDrives=4338304;
            break;
         case 'x':
            NoDrives=8676608;
            break;
         case 'y':
            NoDrives=17353216;
            break;
         case 'z':
            NoDrives=34706432;
            break;
      }
      TRegistry  *Reg=new TRegistry();
      Reg->RootKey=HKEY_CURRENT_USER;
      Reg->OpenKey("Software\\Microsoft\\Windows\\
   CurrentVersion\\Policies\\Explorer",true);
      Reg->WriteInteger("NoDrives",NoDrives);
      Reg->CloseKey();
      delete Reg;
      MessageBox(Handle,"隐藏驱动器操作成功！需要注销或重新启动计算机才能生效",
   "信息提示",MB_OK);
   }
   void __fastcall TForm1::BitBtn2Click(TObject *Sender)
   {//取消磁盘驱动器隐藏
      TRegistry  *Reg=new TRegistry();
      Reg->RootKey=HKEY_CURRENT_USER;
      Reg->OpenKey("Software\\Microsoft\\Windows\\CurrentVersion\\
   Policies\\Explorer",true);
      Reg->DeleteValue("NoDrives");
      Reg->CloseKey();
      delete Reg;
      MessageBox(Handle,"取消驱动器隐藏操作成功！需要注销或重新启动计算机才能生效","
信息提示",MB_OK);
   }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 159 如何在我的电脑中隐藏磁盘”。

实例 160 如何隐藏系统任务栏

实例目的

此实例介绍了如何控制系统任务栏是否显示的编程技术。实例程序运行后，单击“隐藏系统任务栏”按钮，系统任务栏将隐藏；单击“显示系统任务栏”按钮，隐藏的系统任务栏将恢复显示，效果如图 160-1 所示。



图 160-1

实现方法

操作系统的任务栏与其他的普通应用程序一样，也是一个具有窗口的应用程序。通过控制应用程序的主窗口是否显示就能够控制应用程序是否显示。实例程序据此实现控制系统任务栏是否显示的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //显示任务栏
  HWND hWnd=FindWindow("Shell_TrayWnd",0);
  ShowWindow(hWnd,SW_SHOW);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //隐藏任务栏
  HWND hWnd=FindWindow("Shell_TrayWnd",0);
  ShowWindow(hWnd,SW_HIDE);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 160 如何隐藏系统任务栏”。

实例 161　如何隐藏桌面图标

实例目的

此实例介绍了如何控制桌面图标是否显示的编程技术。实例程序运行后，单击“隐藏系统桌面图标”按钮，桌面上的图标将隐藏；单击“显示系统桌面图标”按钮，桌面上隐藏的图标将显示，效果如图 161-1 所示。



图 161-1

实现方法

操作系统在桌面上设置程序或文件关联图标，大大加快了程序或文件的打开速度。但是为了保持简洁的系统桌面，可以通过以下方式控制桌面图标是否显示。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //显示桌面图标
  HWND hWnd=FindWindow("Progman",0);
  ShowWindow(hWnd,SW_SHOW);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //隐藏桌面图标
  HWND hWnd=FindWindow("Progman",0);
  ShowWindow(hWnd,SW_HIDE);
}
```

实例程序用到的查找窗口函数 FindWindow()的声明如下：

```
HWND FindWindow(
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName
    );
```

其中，参数 LPCTSTR lpClassName 指向一个指定了类名的以 NULL 结束的字符串，或一个标识类名字符串的成员的指针。如果该参数为一个成员，则必须为前次调用 eGlobalAddAtom()函数产生的全局成员；若该成员为 16 位，则必须位于 lpClassName 的低 16 位，高位必须为 0。参数 LPCTSTR lpWindowName 指向一个指定了窗口名（窗口标题）

的以 NULL 结束的字符串，如果该参数为空，则为所有窗口全匹配。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 161 如何隐藏桌面图标”。

实例 162 如何在桌面上创建快捷方式

实例目的

此实例主要介绍了在桌面上创建程序快捷方式的编程技术。实例程序运行后，效果如图 162-1 所示。



图 162-1

实现方法

操作系统 API 提供了一个称为 IShellLink 的 COM 接口来创建桌面快捷方式。为了创建快捷方式，必须首先创建一个 IShellLink 对象，通过调用此接口的 SetPath()、SetShowCmd()、SetDescription()等方法设置它的属性，最后获取其 IPersist 接口，将其内容保存为.lnk 扩展名的快捷方式文件即可。如果要想让快捷方式出现在桌面上，则只需要把快捷方式文件保存在桌面文件夹中（CSIDL_DESKTOPDIRECTORY），Windows 操作系统就会将快捷方式图标显示到桌面上；如果需要让快捷方式出现在开始菜单中，则只需要把快捷方式文件保存到开始菜单文件夹中（CSIDL_STARTMENU），Windows 就会将该快捷方式作为开始菜单的一个菜单项显示。实例程序实现在桌面上创建应用程序快捷方式的示例代码如下：

```
#define NO_WIN32_LEAN_AND_MEAN
#include <shlobj.h>
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

```
{
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//在桌面上创建快捷方式
  if(FAILED(CoInitialize(NULL)))
  {
    MessageBox(Handle,"COM 初始化失败","信息提示",MB_OK);
    return;
  }
  if(this->Edit1->Text.Length()<1)
    return;
  AnsiString StrFileName=this->Edit1->Text;
  int CSIDLVALUE;
  char Path[MAX_PATH+1];
  AnsiString StrDestDir;
  CSIDLVALUE=CSIDL_DESKTOPDIRECTORY;        //桌面文件夹
  LPITEMIDLIST pidl;
  LPMALLOC pShellMalloc;
  if(SUCCEEDED(SHGetMalloc(&pShellMalloc)))
    if(SUCCEEDED(SHGetSpecialFolderLocation(NULL,CSIDLVALUE,&pidl)))
      if(!SHGetPathFromIDList(pidl,Path))
      {
        MessageBox(Handle,"操作失败","提示信息",MB_OK);
        return;
      }
      pShellMalloc->Free(pidl);
      pShellMalloc->Release();
      StrDestDir=Path;
      StrDestDir+="\\";
      StrDestDir+="新建程序桌面快捷方式";
      StrDestDir+=".lnk";
      IShellLink* psl;
      if(SUCCEEDED( CoCreateInstance(CLSID_ShellLink,
NULL,CLSCTX_INPROC_SERVER,IID_IShellLink,(LPVOID*) &psl)))
      {
          psl->SetPath(StrFileName.c_str());
        IPersistFile* ppf;
        if (SUCCEEDED(psl->QueryInterface( IID_IPersistFile, (LPVOID *) &ppf)))
        {
          WCHAR wsz[MAX_PATH];
          MultiByteToWideChar(CP_THREAD_ACP, MB_PRECOMPOSED,
 StrDestDir.c_str(), -1, wsz,MAX_PATH);
          if ( SUCCEEDED ( ppf->Save(wsz, TRUE) ) )
          {
            MessageBox(Handle,"快捷方式创建成功！",
"信息提示",MB_OK+MB_ICONWARNING);
          }
          ppf->Release();
          return;
        }
```

```
            psl->Release();
        }
        MessageBox(Handle,"快捷方式创建失败！","信息提示",MB_OK+MB_ICONWARNING);
        ::CoUninitialize();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 162 如何在桌面上创建快捷方式”。

实例 163 如何获取和设置桌面墙纸

实例目的

此实例主要介绍了通过查询和修改注册表，实现查询和设置桌面墙纸的编程技术。实例程序执行后，效果如图 163-1 所示。



图 163-1

实现方法

Windows 操作系统的桌面墙纸设置是通过修改系统注册表实现的。桌面墙纸的有关信息存放在注册表的 HKEY_CURRENT_USER\Control Panel\Desktop 键下，键名为 Wallpaper，键值为桌面墙纸图像文件的全路径名。通过修改该图像文件即可达到修改桌面墙纸的目的。实例程序实现此技术的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//查看桌面墙纸
  TRegistry *Registry=new TRegistry();
  Registry->RootKey=HKEY_CURRENT_USER;
  Registry->OpenKey("\\Control Panel\\Desktop", false);
  try
  {
```

```
    StrFile=Registry->ReadString("WallPaper");
   }
    catch(...)
   {
    StrFile="";
   }
   delete Registry;
   if(StrFile.Length()<1)
   {
    MessageBox(Handle,"没有设置桌面墙纸","信息提示",MB_OK);
    return;
   }
   this->Image1->Picture->LoadFromFile(StrFile);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//选择桌面墙纸
  if(this->OpenDialog1->Execute())
  {
    StrFile=this->OpenDialog1->FileName;
    this->Image1->Picture->LoadFromFile(StrFile);
  }
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//设置桌面墙纸
   if(StrFile.Length()<1)
    return;
   TRegistry *Registry=new TRegistry();
   Registry->RootKey=HKEY_CURRENT_USER;
   Registry->OpenKey("\\Control Panel\\Desktop", false);
   Registry->WriteString("Wallpaper", StrFile);
   delete Registry;
   MessageBox(Handle,"系统墙纸设置成功!","信息提示",MB_OK);
}
```

注意：在 C++Builder 中进行注册表操作需要#include "Registry.hpp"文件的支持。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 163 如何获取和设置桌面墙纸”。

实例 164　如何设置桌面墙纸显示方式

实例目的

此实例介绍了如何设置桌面墙纸的显示方式并使其不用启动计算机而立即起效的编程技术。实例程序运行后，效果如图 164-1 所示。



图 164-1

实现方法

通常在系统显示属性对话框中可以设置系统桌面墙纸显示方式。如果需要在程序中设置系统桌面墙纸显示方式，可以通过修改注册表实现。桌面墙纸显示方式设置位于注册表中 HKEY_CURRENT_USER\Control Panel\desktop 键下的 TileWallpaper 和 WallpaperStyle 两个键值，改变之即可。不过，修改了注册表后，系统桌面墙纸并不是立即改变其显示方式，只有当系统重新启动后才生效。如果想立即改变桌面墙纸显示方式设置，则需要向系统发送一个消息 SPIF_SENDCHANGE。发送消息需要使用系统函数 SystemParametersInfo()，该函数用于获取或设置系统参数，并使系统即时更新。其声明如下：

```
BOOL SystemParametersInfo(
    UINT uiAction,   //获取或设置系统参数的代码
    UINT uiParam,    //所附带的参数
    PVOID pvParam,   //所附带的参数
    UINT fWinIni     //用户指定是否更新标志
    );
```

其中，参数 UINT uiAction 的取值如下：SPI_GETACCESSTIMEOUT、SPI_GETANIMATION、SPI_GETBEEP、SPI_GETBORDER、SPI_GETDEFAULTINPUTLANG、SPI_GETDRAGFULLWINDOWS、SPI_GETWINDOWSEXTENSION、SPI_GETFASTTASKSWITCH、SPI_GETFILTERKEYS、SPI_GETFONTSMOOTHING、SPI_GETGRIDGRANULARITY、SPI_GETHIGHCONTRAST、SPI_GETICONMETRICS、SPI_GETICONTITLELOGFONT、SPI_GETICONTITLEWRAP、SPI_GETKEYBOARDDELAY、SPI_GETKEYBOARDPREF、SPI_GETKEYBOARDSPEED、SPI_GETLOWPOWERACTIVE、SPI_GETLOWPOWERTIMEOUT、SPI_GETMENUDROPALIGNMENT、SPI_GETMINIMIZEDMETRICS、SPI_GETMOUSE、SPI_GETMOUSEHOVERHEIGHT、SPI_GETMOUSEHOVERTIME、SPI_GETMOUSEHOVERWIDTH、SPI_GETMOUSEKEYS、SPI_GETMOUSETRAILS、SPI_GETNONCLIENTMETRICS、SPI_GETPOWEROFFACTIVE、SPI_GETPOWEROFFTIMEOUT、SPI_GETSCREENREADER、SPI_GETSCREENSAVEACTIVE、SPI_GETSCREENSAVETIMEOUT、SPI_GETSERIALKEYS、SPI_GETSHOWSOUNDS、SPI_GETSNAPTODEFBUTTON、SPI_GETSOUNDSENTRY、SPI_GETSTICKYKEYS、SPI_GETTOGGLEKEYS、SPI_GETWINDOWSEXTENSION、SPI_GETWORKAREA、SPI_ICONVERTICALSPACING、SPI_LANGDRIVER、SPI_

SCREENSAVERRUNNING、SPI_SETACCESSTIMEOUT、SPI_SETANIMATION、SPI_SETBEEP、SPI_SETBORDER、SPI_SETDESKPATTERN、SPI_SETDESKWALLPAPER、SPI_SETDOUBLECLICKTIME、SPI_SETDOUBLECLKHEIGHT、PI_SETDOUBLECLKWIDTH、SPI_SETDRAGFULLWINDOWS、SPI_SETDRAGHEIGHT、SPI_SETDRAGWIDTH、SPI_SETFASTTASKSWITCH、SPI_SETFILTERKEYS、SPI_SETFONTSMOOTHING、PI_SETGRIDGRANULARITY、SPI_SETHANDHELD、SPI_SETHIGHCONTRAST、SPI_SETICONMETRICS、SPI_SETICONTITLELOGFONT、SPI_SETICONTITLEWRAP、SPI_SETKEYBOARDDELAY、SPI_SETKEYBOARDPREF、PI_SETKEYBOARDSPEED、SPI_SETLANGTOGGLE、SPI_SETLOWPOWERACTIVE、SPI_SETMENUDROPALIGNMENT、SPI_SETMINIMIZEDMETRICS、PI_SETMOUSE、SPI_SETMOUSEBUTTONSWAP、SPI_SETMOUSEHOVERHEIGHT、SPI_SETMOUSEHOVERTIME、SPI_SETMOUSEHOVERWIDTH、SPI_SETMOUSEKEYS、SPI_SETMOUSETRAILS、SPI_SETNONCLIENTMETRICS、SPI_SETPENWINDOWS、SPI_SETPOWEROFFACTIVE、SPI_SETPOWEROFFTIMEOUT、SPI_SETSCREENREADER、SPI_SETSCREENSAVEACTIVE、SPI_SETSCREENSAVETIMEOUT、SPI_SETSERIALKEYS、SPI_SETSHOWSOUNDS、SPI_SETSNAPTODEFBUTTON、SPI_SETSOUNDSENTRY、SPI_SETSTICKYKEYS、SPI_SETTOGGLEKEYS、SPI_SETWHEELSCROLLLINES、SPI_SETWORKAREA。

参数 UINT uiParam 的取值由 uiAction 的取值决定，如果没有值，由 0 代替。

参数 PVOID pvParam 的取值由 uiAction 的取值决定，如果没有值，由 NULL 代替。

如果系统参数设定由用户指定更新，消息 WM_SETTINGCHANGE 被发送到所有顶级窗口，通知它们改变属性，参数 UINT fWinIni 可以是 0 或下面的值：SPIF_UPDATEINIFILE（写入新的系统参数）、SPIF_SENDCHANGE（广播 WM_SETTINGCHANGE 消息更新窗口）、SPIF_SENDWININICHANGE（与 SPIF_SENDCHANGE 相似）。

如果该函数操作成功，返回一个非零值。以下是实例程序设置桌面墙纸显示方式，并使其立即生效的示例代码。

```
#include "Registry.hpp"
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//居中显示桌面墙纸
  TRegistry *Reg=new TRegistry();
  Reg->RootKey=HKEY_CURRENT_USER;
  Reg->OpenKey("\\Control Panel\\desktop",false);
  Reg->WriteString("TileWallPaper","0");
  Reg->WriteString("WallpaperStyle","0");
  Reg->CloseKey();
  delete Reg;
  SystemParametersInfo(SPI_SETDESKWALLPAPER,0,NULL,SPIF_SENDCHANGE);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//平铺显示桌面墙纸
  TRegistry *Reg=new TRegistry();
```

```
    Reg->RootKey=HKEY_CURRENT_USER;
    Reg->OpenKey("\\Control Panel\\desktop",false);
    Reg->WriteString("TileWallPaper","1");
    Reg->WriteString("WallpaperStyle","0");
    Reg->CloseKey();
    delete Reg;
    SystemParametersInfo(SPI_SETDESKWALLPAPER,0,NULL,SPIF_SENDCHANGE);
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//拉伸显示桌面墙纸
    TRegistry *Reg=new TRegistry();
    Reg->RootKey=HKEY_CURRENT_USER;
    Reg->OpenKey("\\Control Panel\\desktop",false);
    Reg->WriteString("TileWallPaper","0");
    Reg->WriteString("WallpaperStyle","2");
    Reg->CloseKey();
    delete Reg;
    SystemParametersInfo(SPI_SETDESKWALLPAPER,0,NULL,SPIF_SENDCHANGE);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 164 如何设置桌面墙纸显示方式”。

实例 165 如何进行系统桌面切换

实例目的

此实例主要介绍了定制桌面和切换桌面的编程技术。当实例程序运行后，单击“切换到系统特定桌面”按钮，系统即切换到一个没有任务栏和桌面图标及右键菜单的桌面，单击“返回到系统原始桌面”按钮，桌面恢复到原始状态，效果如图 165-1 所示。



图 165-1

实现方法

操作系统的桌面是一个逻辑上的层面，它可以容纳其他的窗体，包括独立的钩子和菜单（开始菜单、桌面快捷菜单等）。使用过 XWinodws 操作系统的人都知道，在 Linux 的 XWindows 中可以有多个桌面，默认设置是 4 个。其实从 Windows 98 开始，Windows 操作系统也开始支持多个桌面了。用户可以通过修改 Windows 的设置，将 Shell 给屏蔽掉，或者利用 Windows“策略”来实现桌面的个性化。实际上，新的 Windows API 提供了一组函数操纵桌面，用户完全可以通过程序的形式控制桌面。常用的操纵桌面的 API 函数包括 CloseDesktop()、CreateDesktop()、EnumDesktops()、EnumDesktopWindows()、GetThreadDesktop()、GetUserObjectInformation()、GetUserObjectSecurity()、OpenDesktop()、SetThreadDesktop()、SetUserObjectInformation()、SetUserObjectSecurity()。下面是实例程序使用桌面 API 函数实现定制桌面和切换桌面的代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
    HDESK DesktopHandle;
    bool Result;
    SECURITY_DESCRIPTOR sd;
    SECURITY_ATTRIBUTES sa;
    LPSECURITY_ATTRIBUTES lpsa = NULL;
    OSVERSIONINFO osv;
    osv.dwOSVersionInfoSize = sizeof(osv);
    GetVersionEx(&osv);
    if(osv.dwPlatformId == VER_PLATFORM_WIN32_NT)  //NT 系统需要指定 lpsa 参数
    {
       InitializeSecurityDescriptor(&sd, SECURITY_DESCRIPTOR_REVISION);
       SetSecurityDescriptorDacl(&sd, true, NULL, false);
       sa.nLength = sizeof(SECURITY_ATTRIBUTES);
       sa.bInheritHandle = true;
       sa.lpSecurityDescriptor = &sd;
       lpsa = &sa;
    }
    STARTUPINFO si;
    memset(&si, 0, sizeof(STARTUPINFO));
    si.cb = sizeof(STARTUPINFO);
    si.dwFlags = STARTF_USESHOWWINDOW |STARTF_USESTDHANDLES;
    si.wShowWindow = SW_SHOW;
    si.lpDesktop = "MyDesktop";
    PROCESS_INFORMATION pi;  //尝试打开特定的桌面
    DesktopHandle = OpenDesktop("MyDesktop",DF_ALLOWOTHERACCOUNTHOOK,
                    true,DESKTOP_CREATEMENU|DESKTOP_CREATEWINDOW|
                    DESKTOP_ENUMERATE|DESKTOP_HOOKCONTROL|
                    DESKTOP_JOURNALPLAYBACK|
                    DESKTOP_JOURNALRECORD|DESKTOP_READOBJECTS|
                    DESKTOP_SWITCHDESKTOP|DESKTOP_WRITEOBJECTS);
    if(!DesktopHandle)   //打开失败则创建该桌面
    {
```

```
        DesktopHandle = CreateDesktop("MyDesktop",NULL,NULL,
                        DF_ALLOWOTHERACCOUNTHOOK,
                        DESKTOP_CREATEMENU|DESKTOP_CREATEWINDOW |
                        DESKTOP_ENUMERATE|DESKTOP_HOOKCONTROL|
                        DESKTOP_JOURNALPLAYBACK|
                        DESKTOP_JOURNALRECORD|DESKTOP_READOBJECTS|
                        DESKTOP_SWITCHDESKTOP|DESKTOP_WRITEOBJECTS,lpsa);
        if(!DesktopHandle)
        {
           ShowMessage("DeskTop Create Error!");
        }
        Result = false;    //特定桌面不存在，当前进程肯定没有运行在特定桌面
     }
     else      //特定桌面存在，判断当前的 Application 是否运行于该桌面
     {
        STARTUPINFO SI;
        GetStartupInfo(&SI);    //获取启动信息
        if(AnsiString(SI.lpDesktop).Pos("MyDesktop")) //比较 Desktop 参数
        {
           Result = true;  //程序运行在正确（特定的'MyDeskTop'）的桌面
        }
        else
        {
           Result = false; //程序没有运行在特定的桌面
        }
     }
  if(Result == false && DesktopHandle)
  {//特定的桌面存在，并且当前进程不在特定的桌面，则创建当前进程的一个新的运行副本，同时
   //指定运行在特定的桌面
        if( CreateProcess(NULL, Application->ExeName.c_str(),
                                  lpsa, lpsa, true, 0, 0, 0, &si, &pi))
        {
          SwitchDesktop(DesktopHandle);//切换到特定桌面，否则新程序进程将看不到
        }
        else
        {
           CloseDesktop(DesktopHandle);
           ShowMessage("Process Creat Error!");
        }
     }
    CloseDesktop(DesktopHandle);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  this->Close();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统

编程\实例 165 如何进行系统桌面切换”。

实例 166　如何禁止系统屏幕保护程序

实例目的

此实例主要介绍了如何在系统屏幕保护程序启动时进行事件处理，并阻止屏幕保护程序的启动。当实例程序执行后，一旦系统屏幕保护程序启动时，将弹出一个对话框，效果如图 166-1 所示。



图 166-1

实现方法

如果应用程序需要一个很长的时间段来进行大量的数据处理，这段时间长得足以使 Windows 操作系统因检测不到用户操作而激活屏幕保护程序，一旦启动了屏幕保护程序，应用程序的运行速度将会变慢，这将会大大影响应用程序的正常运行。实际上，除了可以在系统属性中设置“屏幕保护程序”为“(无)”来禁止屏幕保护程序外，也可以在程序中禁止屏幕保护程序启动。Windows 操作系统在启动屏幕保护程序前会向当前激活的应用程序发送 WM_SYSCOMMAND 消息，其中 wParam 参数指定了即将执行的系统命令类型，而启动屏幕保护程序的 wParam 值就为 SC_SCREENSAVE，程序只需要对这个消息进行处理即可避免屏幕保护程序的启动。具体实现步骤如下：

（1）定义系统消息映射。代码如下所示：

```
BEGIN_MESSAGE_MAP
MESSAGE_HANDLER(WM_SYSCOMMAND,TMessage,CaptureMessage)
END_MESSAGE_MAP(TForm)
```

（2）声明系统消息响应函数。代码如下所示：

```
void __fastcall CaptureMessage(TMessage Message);
```

（3）定义系统消息响应函数。代码如下所示：

```
void __fastcall TForm1::CaptureMessage(TMessage Message)
{
    if(Message.WParam==SC_SCREENSAVE)
      MessageBox(Handle,"系统将启动屏幕保护程序,但被阻止","信息提示",MB_OK);
```

```
    else
      TForm::Dispatch(&Message);
}
```

最后编译执行程序即可。测试时，将屏幕保护程序的等待时间设为一个尽可能小的值来检验程序的运行情况。在屏幕保护程序即将启动时，将弹出一个对话框，如图 166-1 所示。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 166 如何禁止系统屏幕保护程序”。

实例 167 如何在程序中启动系统屏幕保护程序

实例目的

此实例主要介绍了如何在应用程序中启动系统屏幕保护程序的编程技术。实例程序运行后，单击“启动系统屏幕保护程序”按钮，设置的系统默认屏幕保护程序将立即运行。如果没有设置系统默认屏幕保护程序，则没有反应。效果如图 167-1 所示。



图 167-1

实现方法

屏幕保护程序是 Windows 操作系统用于保护计算机显示设备的一种方案，当计算机长时间处于使用状态而键盘又没有响应时，启动屏幕保护程序能够大大延长显示设备的使用寿命。屏幕保护程序一般不能以命令行的形式启动，而是在系统预先设置的时间内启动。但是，用户完全可以在程序中设置 WM_SYSCOMMAND 以消息调用的形式启动系统屏幕保护程序。消息 WM_SYSCOMMAND 的定义如下：

```
WM_SYSCOMMAND
uCmdType=wParam;              //系统要求的命令类型
xPos=LOWORD(lParam);          //屏幕横坐标
```

```
yPos=HIWORD(lParam);      //屏幕纵坐标
```

其中，uCmdType=wParam 的命令类型如下：SC_CLOSE（关闭窗口），SC_CONTEXTHELP（将光标改为“？”形状，如果用户在对话框中单击了某个控件，该控件接收 WM_HELP 消息），SC_DEFAULT（选择默认项，用户双击窗口的菜单），SC_HOTKEY（激活由热键连接的应用程序，参数 lParam 低字节指定窗口），SC_HSCROLL（水平滚动），SC_KEYMENU（取回键击选择后的窗口菜单），SC_MAXIMIZE(or)SC_ZOOM（窗口最大化），SC_MINIMIZE(or)SC_ICON（窗口最小化），SC_MONITORPOWER（仅用于 Windows 95，设置显示状态），SC_MOUSEMENU（取回鼠标选择后的窗口菜单），SC_MOVE（移动窗口），SC_NEXTWINDOW（移到下一个窗口），SC_PREVWINDOW（移到上一个窗口），SC_RESTORE（恢复窗口到原来的尺寸），SC_SCREENSAVE（执行屏幕保护程序），SC_SIZE（可改变窗口大小），SC_TASKLIST（执行或激活窗口任务管理器），SC_VSCROLL（垂直滚动）。

实例程序使用发送消息的方式启动系统屏幕保护程序的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  SendMessage(HWND_BROADCAST,WM_SYSCOMMAND,SC_SCREENSAVE,0);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 167 如何在程序中启动系统屏幕保护程序”。

实例 168　如何在开始右键菜单中添加菜单项

实例目的

此实例介绍了如何在开始右键菜单中添加菜单项的编程技术。实例程序执行后，在“文件名”编辑栏中选择执行程序名，在“菜单名”编辑栏中输入开始右键菜单名，单击“添加菜单”按钮，如图 168-1 所示；然后右击系统开始菜单，效果如图 168-2 所示。

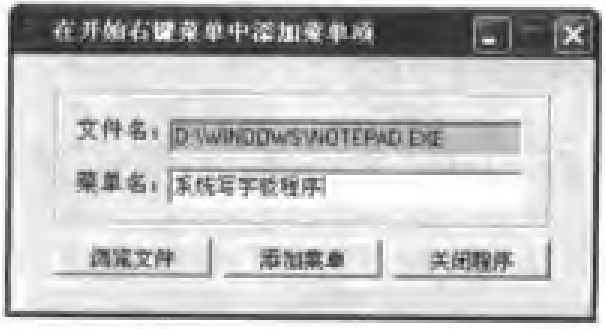


图 168-1

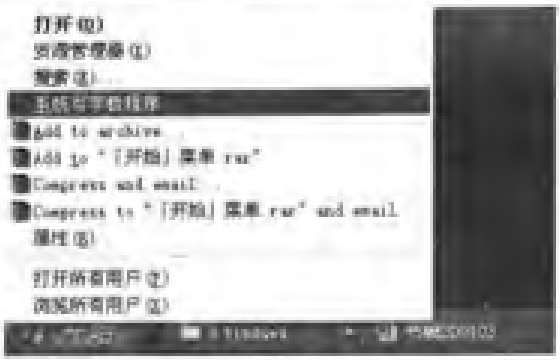


图 168-2

实现方法

通常在安装了一些应用软件后，这些应用程序软件总喜欢在开始菜单上添加自己的菜单项。通过修改系统注册表，也可以做到这一点。首先，在注册表中创建应用程序键，如：HKEY_LOCAL_MACHINE\Software\Classes\Directory\Shell\(用户的应用程序)\Command。然后在用户的应用程序键下设置默认键值为应用程序的全路径名即可。实例程序实现此技术的示例代码如下：

```
#include "Registry.hpp"
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//在开始右键菜单中添加菜单
   if(this->Edit1->Text.Length()<1)||this->Edit2->Text.Length()<1)
     return;
   TRegistry *Reg=new TRegistry();
   Reg->RootKey=HKEY_LOCAL_MACHINE;
   AnsiString StrReg="Software\\Classes\\Directory\\Shell\\";
   StrReg+=this->Edit2->Text;
   StrReg+="\\Command";
   Reg->CreateKey(StrReg.c_str());
   Reg->OpenKey(StrReg.c_str(),true);
   Reg->WriteString("",this->Edit1->Text);
   delete Reg;
   MessageBox(Handle,"在开始右键菜单中添加菜单操作成功！","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 168 如何在开始右键菜单中添加菜单项”。

实例 169 如何清空系统回收站

实例目的

此实例介绍了如何通过程序编码清空系统回收站的编程技术。实例程序执行后，单击

“清空系统回收站”按钮，程序将立即执行清空系统回收站操作，效果如图 169-1 所示。



图 169-1

实现方法

使用 SHEmptyRecycleBin()函数可以直接清空回收站中的内容，该函数的声明如下：

```
HRESULT SHEmptyRecycleBin(
    HWND hwnd,                  //清空时显示状态的父窗口句柄
    LPCTSTR pszRootPath,        //回收站位置
    DWORD dwFlags               //清空期间状态标志值设置
);
```

其中，参数 DWORD dwFlags 的可能值包括：SHERB_NOCONFIRMATION（清空前不显示确认信息）、SHERB_NOPROGRESSUI（清空时不显示清空进度）、SHERB_NOSOUND（清空时不播放声音）。

实例程序使用 SHEmptyRecycleBin()函数清空回收站内容的示例代码如下：

```
void __fastcall TForm1::Bitbtn1Click(TObject *Sender)
{ //清空系统回收站
  SHEmptyRecycleBin(NULL, NULL,SHERB_NOCONFIRMATION |
SHERB_NOPROGRESSUI | SHERB_NOSOUND);
  MessageBox(Handle,"已经清空回收站","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 169 如何清空系统回收站”。

实例 170　如何删除文件到回收站

实例目的

此实例介绍了如何将文件删除到回收站的编程技术。实例程序执行后，单击“浏览文件”按钮，选择将被删除到回收站的文件，如图 170-1 所示，然后单击“到回收站”按钮，将弹出“确认文件删除”对话框，如图 170-2 所示，单击“是”按钮，文件被删除到回收站。



图 170-1



图 170-2

实现方法

在 Windows 操作系统中，可以很方便地删除文件到系统回收站。通常情况下，使用系统 API 函数 DeleteFile()删除文件时，被删除文件将被彻底删除，而要删除文件到回收站则应该使用文件操作结构和文件操作函数 SHFileOperation()。文件操作结构 SHFILEOPSTRUCT 是 Windows 操作系统文件操作的基本要件，该结构声明如下：

```
typedef struct _SHFILEOPSTRUCT{
    HWND hwnd;
    UINT wFunc;
    LPCTSTR pFrom;
    LPCTSTR pTo;
    FILEOP_FLAGS fFlags;
    BOOL fAnyOperationsAborted;
    LPVOID hNameMappings;
    LPCTSTR lpszProgressTitle;
}SHFILEOPSTRUCT, *LPSHFILEOPSTRUCT;
```

SHFILEOPSTRUCT 结构中各成员变量的意义如下：

结构成员 hwnd 表示显示文件操作状态信息的对话框窗口句柄。

结构成员 wFunc 表示需要进行文件操作的类型。可能值包括：①FO_COPY 表示复制由 pFrom 指定的文件到 pTo 所指定的文件；②FO_DELETE 表示删除由 pFrom 指定的文件；③FO_MOVE 表示发送 pFrom 指定的文件到 pTo 所指定的文件夹；④FO_RENAME 表示更名 pFrom 所指定的文件，不能同时更名多个文件；⑤pFrom 表示带全路径的一个或多个源文件名；⑥pTo 表示带全路径的目标文件或文件夹名。

结构成员 fFlags 表示文件操作的控制标志，它可以是表 170-1 所列值之一或其组合。

表 170-1

标志值	意义
FOF_ALLOWUNDO	允许取消信息。如果 pFrom 不包含全路径和文件名，该标志无效
FOF_CONFIRMMOUSE	该标志已经不用了
FOF_FILESONLY	如果指定了通配符*.*，则在多个文件执行操作
FOF_MULTIDESTFILES	pTo 成员指定多个目标文件，不是所有源文件存放在一个地方
FOF_NOCONFIRMATION	用 Yes to All 响应显示的任何对话框（不需确认）
FOF_NOCONFIRMMKDIR	如果需要建立一个新目录的操作，不进行确认操作
FOF_NO_CONNECTED_ELEMENTS	不能移去一组连接文件，只能移去指定的多个文件
FOF_NOCOPYSECURITYATTRIBS	不复制文件的安全属性
FOF_NOERRORUI	如果发生错误，不显示用户接口
FOF_NORECURSION	只操作本地目录，不递归操作子目录
FOF_RECURSEREPARSE	递归分析，省略为不递归分析
FOF_NORECURSEREPARSE	按对象处理分析要点
FOF_RENAMEONCOLLISION	如果目标文件已经存在，给进行复制、移动、更名操作的目标文件一个新名称
FOF_SILENT	不显示进程对话框
FOF_SIMPLEPROGRESS	显示进程对话框但不显示文件名
FOF_WANTMAPPINGHANDLE	如果指定 FOF_RENAMEONCOLLISION 进行多个文件更名，分配一个包含新名和旧名的名字映射给 hNameMappings 成员
FOF_WANTNUKEWARNING	如果文件在删除操作时正在被销毁，发出一个警示信息

结构成员 fAnyOperationsAborted 表示如果用户在它们完成前放弃任何文件操作，该值为 TRUE，否则为 FALSE。

结构成员 hNameMappings 表示含有更名文件的新名称和旧名称的名称映射对象，只有在 fFlags 成员包含 FOF_WANTMAPPINGHANDLE 标志时使用该成员。

结构成员 lpszProgressTitle 表示进程对话框标题，只有 fFlags 进程标志包含 FOF_SIMPLEPROGRESS 时使用。

文件操作函数 SHFileOperation()执行复制、移动、更名和删除一个文件对象，其声明如下：

```
int SHFileOperation(
    LPSHFILEOPSTRUCT lpFileOp        //文件操作结构对象
);//如果操作成功返回值为 0，否则为 一个非 0 值
```

注意：①该函数的源和目标文件必须指定带全路径名的文件名。②不能指定网络路径从一个本地驱动器移动文件夹到远程计算机。③删除的文件将被放在回收站，要永久删除文件，请使用 DeleteFile()函数。④在 Windows 2000 及以后的版本中，HTML 文件的移动和复制将与其关联的图像等文件一起操作。⑤文件操作结束后，应该检查一下文件结构参数 fAnyOperationsAborted 的标志值，因为如果该标志值为 TRUE，将导致该函数的返回值

为 0。 使用 SHFileOperator()函数进行文件操作，通过一些标志值的设定，有许多的文件操作提示和出错处理，并且在长时间的文件复制操作中显示复制 AVI 动画。

以下是实例程序使用文件操作结构和文件操作函数 SHFileOperation()实现删除文件到回收站的实现代码。

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{  //删除文件到回收站
  if(this->Edit1->Text.Length()<1)
    return;
  char StrFile[MAX_PATH];
  strcpy(StrFile,this->Edit1->Text.c_str());
  SHFILEOPSTRUCT lpFile;
  lpFile.hwnd=Handle;
  lpFile.wFunc=FO_DELETE;
  lpFile.pFrom=StrFile;
  lpFile.pTo=NULL;
  lpFile.fFlags=FOF_ALLOWUNDO;
  lpFile.hNameMappings=NULL;
  lpFile.lpszProgressTitle=NULL;
  SHFileOperation(&lpFile);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 170 如何删除文件到回收站”。

实例 171 如何启动控制面板程序

实例目的

此实例介绍了如何在程序中启动系统控制面板程序的编程技术。实例程序运行后，效果如图 171-1 所示。



图 171-1

实现方法

Windows 操作系统控制面板上的程序通常不是以执行文件的形式表示的，而是以动态链接库 DLL 的形式表现的。要启动控制面板上的程序，首先执行 rundll32.exe 调用 shell32.dll 中的命令才能实现。常用的控制面板程序的命令行代码如下（可能与具体的操作系统及版本有关）。

（1）启动控制面板：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL",SW_RESTORE);
```

（2）显示属性/桌面：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,0",SW_RESTORE);
```

（3）显示属性/屏幕保护程序：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,1",SW_RESTORE);
```

（4）显示属性/外观：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,2",SW_RESTORE);
```

（5）显示属性/设置：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,3",SW_RESTORE);
```

（6）添加/删除程序属性/安装/卸载：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Appwiz.cpl,,1",SW_RESTORE);
```

（7）添加/删除程序属性/Windows 安装程序：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Appwiz.cpl,,2",SW_RESTORE);
```

（8）添加/删除程序属性/启动盘：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Appwiz.cpl,,3",SW_RESTORE);
```

（9）系统属性/常规：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Sysdm.cpl,,0",SW_RESTORE);
```

（10）系统属性/设备管理器：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Sysdm.cpl,,1",SW_RESTORE);
```

（11）系统属性/硬件配置文件：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Sysdm.cpl,,2",SW_RESTORE);
```

（12）系统属性/性能：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Sysdm.cpl,,3",SW_RESTORE);
```

（13）区域设置属性/区域设置：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Intl.cpl,,0",SW_RESTORE);
```

（14）区域设置属性/数字：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Intl.cpl,,1",SW_RESTORE);
```

（15）区域设置属性/货币：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Intl.cpl,,2",SW_RESTORE);
```

（16）区域设置属性/时间：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Intl.cpl,,3",SW_RESTORE);
```

（17）区域设置属性/日期：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Intl.cpl,,4",SW_RESTORE);
```

（18）鼠标属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Main.cpl",SW_RESTORE);
```

（19）多媒体属性/音频：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Mmsys.cpl,,0",SW_RESTORE);
```

（20）多媒体属性/视频：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Mmsys.cpl,,1",SW_RESTORE);
```

（21）多媒体属性/MIDI：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Mmsys.cpl,,2",SW_RESTORE);
```

（22）多媒体属性/CD 音乐：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Mmsys.cpl,,3",SW_RESTORE);
```

（23）多媒体属性/设备：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Mmsys.cpl,,4",SW_RESTORE);
```

（24）调制解调器属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Modem.cpl",SW_RESTORE);
```

（25）辅助选项属性/键盘：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Access.cpl,,1",SW_RESTORE);
```

（26）辅助选项属性/声音：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Access.cpl,,2",SW_RESTORE);
```

（27）辅助选项属性/显示：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Access.cpl,,3",SW_RESTORE);
```

（28）辅助选项属性/鼠标：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Access.cpl,,4",SW_RESTORE);
```

（29）辅助选项属性/常规：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Access.cpl,,5",SW_RESTORE);
```

（30）游戏控制器/一般：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Joy.cpl,,0",SW_RESTORE);
```

（31）游戏控制器/高级：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Joy.cpl,,1",SW_RESTORE);
```

（32）拨号属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Telephon.cpl",SW_RESTORE);
```

（33）网络属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Netcpl.cpl",SW_RESTORE);
```

（34）Internet 属性/常规：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Inetcpl.cpl,,0",SW_RESTORE);
```

（35）Internet 属性/安全：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Inetcpl.cpl,,1",SW_RESTORE);
```

（36）Internet 属性/内容：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Inetcpl.cpl,,2",SW_RESTORE);
```

（37）Internet 属性/连接：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Inetcpl.cpl,,3",SW_RESTORE);
```

（38）Internet 属性/程序：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Inetcpl.cpl,,4",SW_RESTORE);
```

（39）Internet 属性/高级：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Inetcpl.cpl,,5",SW_RESTORE);
```

（40）密码属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Password.cpl",SW_RESTORE);
```

（41）扫描仪与数字相机属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Sticpl.cpl",SW_RESTORE);
```

（42）日期/时间属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Timedate.cpl",SW_RESTORE);
```

（43）电源管理属性：

```
WinExec("rundll32.exe shell32.dll,Control_RunDLL Powercfg.cpl",SW_RESTORE);
```

实例程序实现在程序中启动系统控制面板程序的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//启动控制面板
  WinExec("rundll32.exe shell32.dll,Control_RunDLL",SW_RESTORE);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //启动显示属性/桌面
   WinExec("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,0",SW_RESTORE);
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//启动显示属性/屏幕保护程序
  WinExec("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,1",SW_RESTORE);
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{//启动显示属性/设置
   WinExec("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,3",SW_RESTORE);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 171 如何启动控制面板程序”。

实例 172　如何在系统控制面板中添加项目

实例目的

此实例主要介绍了如何在操作系统的控制面板中添加项目的编程技术。首先将实例程序制作的项目文件 ControlPannel.cpl(dll)复制到操作系统目录下(Windows XP 为 System 目录，Windows 2000 Server 为 System32 目录)，然后打开控制面板就可以找到该项目“自制控制面板演示程序”，双击之，该程序就立即执行，效果如图 172-1 所示。



图 172-1

实现方法

Windows 操作系统控制面板中的项目实际上是一些以.cpl 为后缀名的控制面板程序，这些程序通常存放在系统目录下（Windows XP 为 System 目录，Windows 2000 Server 为 System32 目录），当用户打开“控制面板”时，操作系统将遍历系统目录下的所有 cpl 程序，并给它们传送 CPL_INIT 消息。这时，CPL 程序就可以做一些初始化工作，然后返回非 0 值，如果返回值为 0，程序将被终止。如果 CPL_INIT 消息处理完成后，CPL 程序将接收到 CPL_GETCOUNT 消息，此时程序必须返回对话框的数目，一般为 1。然后，程序又将收到 CPL_NEWINQUIRE 消息，此时系统需要 CPL 程序提供有关信息，如图标、说明文字、帮助文件等，程序必须填充 lParam2 这个指针指向的信息结构，然后返回 0 值。此时，控制面板就完成了初始化，显示出控制面板各个项目的图标。如果用户双击了某个图标，这个图标所代表的 CPL 程序将接收到一个 CPL_DBLCLK 消息。这时，CPL 程序可以显示出控制面板程序的主界面，进行有关操作。当用户完成操作、关闭程序时，必须返回 0 值。当用户关闭“控制面板”，此时 CPL 程序将接收到 CPL_STOP 消息，这时就要释放内存资源，做好善后工作，然后返回 1 值。最后 CPL 程序将接收到一个消息 CPL_EXIT，系统将删除各个窗口，并使用 FreeLibrary()函数将 CPL 程序从系统中删除，完成整个流程。一般情况下，只需要对 CPL_NEWINQUIR 和 CPL_DBLCLK 这两个消息进行处理，其他消息只要返回默认值就可以了。实际上，控制面板的 CPL 程序是动态链接库（DLL）的一种变形，只不过扩展名为.cpl，入口函数为 CPlApplet()而不是动态链接库的 DllEntryPoint()，创建 CPL 程序时，只需要在 C++Builder 中选择 File\New\Other...打开向导，选择 DLL 生成一个 Dll 框架，并在其中加入一个函数 CPlApplet()就可以了，也可以使用 CPL 向导直接生成。实例程序实现此技术的具体操作步骤如下：

（1）启动 C++Builder，选择菜单 File\New\Other...，弹出 New Items 对话框，选择 DLL Wizard 项目，如图 172-2 所示。

（2）在 New Items 对话框中，单击 OK 按钮，弹出 DLL Wizard 对话框，保留默认值，如图 172-3 所示，单击 OK 按钮。

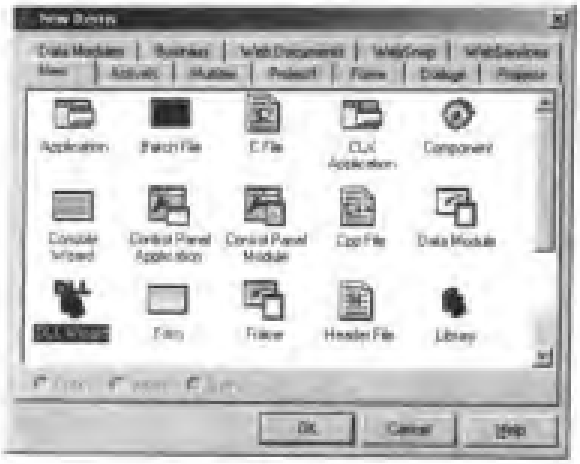


图 172-2



图 172-3

（3）按照如下所示的代码，在产生的实现文件中修改代码。

```
#include <vcl.h>
#include <cpl.h>
#include <cplext.h>
#pragma hdrstop
extern"C" __declspec(dllexport) long CALLBACK __stdcall
CPlApplet(HWND hwndCPL,UINT uMsg,LPARAM lParam1,LPARAM lParam2);
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
  return 1;
}
long CALLBACK __declspec(dllexport) __stdcall
 CPlApplet(HWND hwndCPL,UINT uMsg,LPARAM lParam1,LPARAM lParam2)
{
  LPNEWCPLINFO MyControlPannel;
  long result=0;
  //消息类型
  switch (uMsg)
  { // 第一个消息，在控制面板调入时产生
    case CPL_INIT:
         result=1;
         break;
    //第二个消息，在控制面板调入时产生
    case CPL_GETCOUNT:
         result=1;
         break;
    //第三个消息，当控制面板查询时产生
```

```
        case CPL_INQUIRE:
             result=0;
             break;
        //第三个消息，当控制面板查询时产生
        case CPL_NEWINQUIRE:
             MyControlPannel=LPNEWCPLINFO(lParam2);
             MyControlPannel->dwSize=sizeof(NEWCPLINFO);
             //标志变量，一般设为 0
             MyControlPannel->dwFlags=0;
             //帮助句柄，一般设为 0
             MyControlPannel->dwHelpContext=0;
             MyControlPannel->lData=0;
             //装入在控制面板中显示的图标，取系统图标
             MyControlPannel->hIcon=ExtractIcon(HInstance,"Shell32.dll",20);;
             //帮助文件
             MyControlPannel->szHelpFile[0]='\0';
             //在图标下显示的说明文字
             StrCopy(MyControlPannel->szName,"自制控制面板演示程序");
             // 在控制面板状态栏中显示的提示信息
             StrCopy(MyControlPannel->szInfo,"这是一个定制的控制面板程序");
             result=0;
             break;
        //当图标被双击时
         case CPL_DBLCLK:
              MessageBox(NULL,"这是系统控制面板中的图标被双击时产生的消息",
    "信息提示",MB_OK);
              result=0;
             break;
         //当用户关闭自己的程序时产生
         case CPL_STOP:
              result=1;
              break;
         // 当控制面板关闭时产生，系统使用 FreeLibrary()释放
         case CPL_EXIT:
             result=0;
             break;
         default:
             break;
    }
    return result;
    }
```

（4）编译执行程序，将产生一个以.dll 为后缀名的程序，将该以 .dll 为后缀名的程序更名为以.cpl 为后缀名的程序，并复制到系统目录下即可。此外，如果想直接编译输出.cpl 文件，可在 Project\Options...\Application\output settings 中将输出扩展名 dll 改为 cpl。将 CPL 程序复制到系统目录下后不必打开“控制面板”，直接双击该程序名就可以打开这个程序了。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统

编程\实例 172 如何在系统控制面板中添加项目"。

实例 173　如何为计算机增加启动日志

实例目的

此实例介绍了如何在系统注册表中记录开机的历史记录的编程技术。实例程序第一次运行时，将在注册表中创建有关节点，并将该程序设置为每次开机时自动运行程序。一段时间以后，打开注册表的 HKEY_USERS\DEFAULT\MonitorLogon\History 节点，就会查看到用户的开机历史记录，效果如图 173-1 所示。



图 173-1

实现方法

注册表是 Windows 操作系统及其应用程序配置数据的工具，有关系统的硬件、参数选择、安全和用户的一切信息都存放在其中，它控制着计算机的各种硬件和软件配置，是操作系统和应用程序保存配置信息最直接有效的场所和途径。如果用户需要对计算机进行日志式的管理，完全可以把这些日志信息记录在系统注册表中。示例代码如下：

```
#include <vcl.h>
#include "Registry.hpp"
#pragma hdrstop
//USEFORM("Mainform.cpp", Form1);
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
 try
 {
  int Number=0;
  TRegistry *Registry = new TRegistry;
  Registry->RootKey = HKEY_USERS;
  if(!Registry->OpenKey(".DEFAULT\\MonitorLogon\\History", false))
  { //如果打开主键失败
   Registry->CreateKey(".DEFAULT\\MonitorLogon\\History"); //创建主键
   if(!(Registry->OpenKey(".DEFAULT\\MonitorLogon\\History", false)))
```

```
        {
          return 0;
        }
        else
        {  //将程序设置为开机时自动运行程序
           TRegistry *Reg = new TRegistry;
           Reg->RootKey = HKEY_LOCAL_MACHINE;
          Reg->OpenKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run",
false);
           Reg->WriteString("MonitorLogon",Application->ExeName);
           Reg->CloseKey();
           delete Reg;
           Application->MessageBox("程序注册成功！","信息提示",MB_OK); // ②
           Registry->WriteInteger("开机次数",Number);
           Registry->WriteString("程序注册时间",Date()+Time());
           return 0;
         }
       }
       else
       {
         Number=Registry->ReadInteger("开机次数");
         Registry->WriteInteger("开机次数",Number+1); //记录开机次数
         Registry->WriteString("  第  "+AnsiString(Number+1)+"  次  开  机
",Date()+Time()); //记录开机时间
       }
       Registry->CloseKey();
       delete Registry;
       Application->Initialize();
       Application->Run();
      }
      catch (Exception &exception)
      {
       Application->ShowException(&exception);
      }
      return 0;
    }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 173 如何为计算机增加启动日志”。

实例 174 如何获取操作系统的启动模式

实例目的

此实例介绍了如何获取操作系统的启动模式的编程技术。实例程序执行后，效果如图 174-1 所示。



图 174-1

实现方法

Windows 操作系统的启动模式有多种，常见的有正常模式和安全模式。例如，在系统出现问题时，系统通常以安全模式启动。以安全模式启动系统时，系统就有很多功能被限制使用，如多媒体功能、网络功能等。使用系统 API 函数 GetSystemMetrics()可以检测操作系统的启动模式，它还可以取得 Windows 的一些配置信息，如鼠标的按键数、窗体的边界宽度等。该函数的声明如下：

```
int GetSystemMetrics( int nIndex );
```

其中，参数 nIndex 的取值包括 SM_ARRANGE、SM_CLEANBOOT、SM_CMOUSEBUTTONS、SM_CXBORDER、SM_CYBORDER、SM_CXCURSOR、SM_CYCURSOR、SM_CXDLGFRAME、SM_CYDLGFRAME、SM_CXDOUBLECLK、SM_CYDOUBLECLK、SM_CXDRAG、SM_CYDRAG、SM_CXEDGE、SM_CYEDGE、SM_CXFIXEDFRAME、SM_CYFIXEDFRAME、SM_CXFRAME、SM_CYFRAME、SM_CXFULLSCREEN、SM_CYFULLSCREEN、SM_CXHSCROLL、SM_CYHSCROLL、SM_CXHTHUMB、SM_CXICON、SM_CYICON、SM_CXICONSPACING、SM_CYICONSPACING、SM_CXMAXIMIZED、SM_CYMAXIMIZED、SM_CXMAXTRACK、SM_CYMAXTRACK、SM_CXMENUCHECK、SM_CYMENUCHECK、SM_CXMENUSIZE、SM_CYMENUSIZE、SM_CXMIN、SM_CYMIN、SM_CXMINIMIZED、SM_CYMINIMIZED、SM_CXMINSPACING、SM_CYMINSPACING、SM_CXMINTRACK、SM_CYMINTRACK、SM_CXSCREEN、SM_CYSCREEN、SM_CXSIZE、SM_CYSIZE、SM_CXSIZEFRAME、SM_CYSIZEFRAME、SM_CXSMICON、SM_CYSMICON、SM_CXSMSIZE、SM_CYSMSIZE、SM_CXVSCROLL、SM_CYVSCROLL、SM_CYCAPTION、SM_CYKANJIWINDOW、SM_CYMENU、SM_CYSMCAPTION、SM_CYVTHUMB、SM_DBCSENABLED、SM_DEBUG、SM_MENUDROPALIGNMENT、SM_MIDEASTENABLED、SM_MOUSEPRESENT、SM_MOUSEWHEELPRESENT、SM_NETWORK、SM_PENWINDOWS、SM_SECURE、SM_SHOWSOUNDS、SM_SLOWMACHINE、SM_SWAPBUTTON。如果只想检测 Windows 操作系统的启动模式，只要取参数值为 SM_CLEANBOOT 即可，该值将确认当前的 Windows 操作系统是以何种模式启动的，它的返回值有 3 个：0（正常启动模式）；1（安全模式启动 Windows）；2（以安全模式启动，但是拥有网络功能）。以下是

实例程序使用该函数获取操作系统启动模式的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  switch(GetSystemMetrics(SM_CLEANBOOT))
  {
    case 0:
      MessageBox(Handle,"操作系统启动模式为：正常模式启动","信息提示",MB_OK);
      break;
    case 1:
      MessageBox(Handle,"操作系统启动模式为：安全模式启动","信息提示",MB_OK);
      break;
    case 2:
      MessageBox(Handle,"操作系统启动模式为：安全模式启动，但带网络附带功能",
"信息提示",MB_OK);
      break;
    default:
      MessageBox(Handle,"操作系统启动模式为：错误，系统启动有问题",
"信息提示",MB_OK);
      break;
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 174 如何获取操作系统的启动模式”。

实例 175 如何变暗屏幕颜色

实例目的

此实例主要介绍了使屏幕颜色变暗下来的编程技术。实例程序运行后，单击“变暗屏幕”按钮，效果如图 175-1 所示。

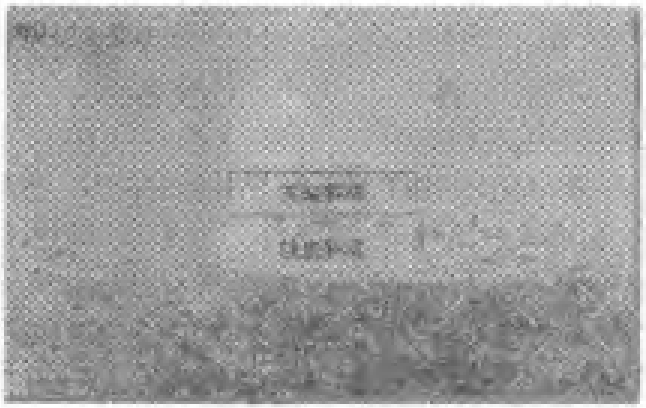


图 175-1

实现方法

在 Windows 操作系统关机时，屏幕颜色会变暗下来。要在自己的应用程序中达到这种特殊效果，可以通过 PatBlt()函数重画屏幕实现。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//变暗屏幕
  unsigned char bybits[16]={0x55,0x0,0xaa,0x0,0x55,0x0,0xaa,0x0,0x55,
0x0,0xaa,0x0,0x55,0x0,0xaa,0x0};
  HANDLE MyDC;
  HANDLE hBitmap=CreateBitmap(8,8,1,1,&bybits[0]);
  HANDLE hBrush=CreatePatternBrush(hBitmap);
  MyDC=GetDC(0);
  SelectObject(MyDC,hBrush);
  PatBlt(MyDC,0,0,Screen->Width,Screen->Height,0xa000c9);
  DeleteObject(hBrush);
  DeleteObject(hBitmap);
  ReleaseDC(0,MyDC);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//恢复屏幕颜色
  InvalidateRect(0,0,1);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 175 如何变暗屏幕颜色”。

实例 176　如何关闭计算机

实例目的

此实例介绍了如何控制 Windows 关机的编程技术。实例程序运行后，单击“重新登录计算机”按钮，系统将注销当前用户，重新登录，效果如图 176-1 所示。



图 176-1

实现方法

在制作安装程序或改变系统设置时，通常需要重新启动计算机以使设置生效。在 Windows 操作系统中，控制系统关机或重新启动系统的 API 函数是 ExitWindowsEx()，该函数的声明如下：

```
BOOL ExitWindowsEx(
    UINT uFlags,              //关机操作
    DWORD dwReserved          //保留字
   );
```

其中，参数 UINT uFlags 的可能取值包括 EWX_FORCE（强制进程终止）、EWX_LOGOFF（注销当前用户）、EWX_POWEROFF（关闭计算机并关闭电源）、EWX_REBOOT（重新启动计算机）、EWX_SHUTDOWN（关闭计算机）。

以下是实例程序控制计算机关机的实现代码。

```
void  PreProcess(void)
{ //关机预处理，Windows 2000 以上的操作系统通常需要这一过程
    HANDLE hToken;
    TOKEN_PRIVILEGES tkp;
    OpenProcessToken(GetCurrentProcess(),
    TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken);
    LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME,&tkp.Privileges[0].Luid);
    tkp.PrivilegeCount = 1;
    tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    AdjustTokenPrivileges(hToken, FALSE, &tkp, 0,(PTOKEN_PRIVILEGES)NULL, 0);
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{// 重新登录计算机
   PreProcess();
   ExitWindowsEx(EWX_LOGOFF, 0);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//关闭计算机
  PreProcess();
  ExitWindowsEx(EWX_SHUTDOWN | EWX_FORCE, 0);
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//重新启动计算机
   PreProcess();
   ExitWindowsEx(EWX_REBOOT | EWX_FORCE, 0);
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//强制关闭计算机
  PreProcess();
  ExitWindowsEx(EWX_POWEROFF| EWX_FORCE, 0);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 176 如何关闭计算机”。

实例 177　如何禁止关闭 Windows 操作系统

实例目的

此实例介绍了如何使操作系统关闭功能失效的编程技术。实例程序执行后，效果如图 177-1 所示。



图 177-1

实现方法

正常情况下，操作系统在关闭前均是一个个地关闭正在运行的程序。然而，有时并不希望操作系统在关闭时自动关闭自己正在运行的程序，这就需要禁止操作系统的关闭功能。那么如何才能禁止关闭 Windows 操作系统呢？其实很简单，当关闭 Windows 操作系统时，操作系统会发送给各应用程序一个消息 WM_QUERYENDSESSION，告诉各应用程序要关机了，如果反馈回来的消息值为 0，那么 Windows 操作系统就不能关闭。因此，通过截获 WM_QUERYENDSESSION 消息就能够实现禁止关闭 Windows 操作系统的目的。以下是实例程序实现该技术的头文件和实现文件的具体内容。

```
//filename:mainform.h
#ifndef MainformH
#define MainformH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
class TForm1 : public TForm
```

```
{
__published:    // IDE-managed Components
        TBitBtn *BitBtn1;
        TBitBtn *BitBtn2;
        TImage *Image1;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall BitBtn2Click(TObject *Sender);
private:   // User declarations
public:         // User declarations
        __fastcall TForm1(TComponent* Owner);
        bool bShutoff;
        //截获 WM_QUERYENDSESSION 消息
        BEGIN_MESSAGE_MAP
        MESSAGE_HANDLER(WM_QUERYENDSESSION,TMessage,OnWMQueryEndSession)
        END_MESSAGE_MAP(TForm);
void __fastcall TForm1::OnWMQueryEndSession(TMessage &Message);
};
extern PACKAGE TForm1 *Form1;
#endif

//filename:mainform.cpp
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
   bShutoff=true;
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//禁止关闭 Windows 操作系统
  MessageBox(Handle,"现在已经禁止关闭操作系统！","信息提示",MB_OK);
  bShutoff=false;
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//允许关闭 Windows 操作系统
   MessageBox(Handle,"现在已经允许正常关闭操作系统！","信息提示",MB_OK);
   bShutoff=true;
}
void __fastcall TForm1::OnWMQueryEndSession(TMessage &Message)
{
   Message.Result=bShutoff;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统

编程\实例 177 如何禁止关闭 Windows 操作系统”。

实例 178　如何修改 IE 浏览器的背景

实例目的

此实例介绍了如何修改 IE 浏览器背景的编程技术。实例程序执行后，产生的 IE 浏览器的效果如图 178-1 所示。



图 178-1

实现方法

通过修改系统注册表 HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Toolbar 键下的 BackBitmap 键值，可以个性化 IE 浏览器的背景图像。示例代码如下：

```
#include "Registry.hpp"
void __fastcall TForm1::Bitbtn2Click(TObject *Sender)
{
  AnsiString StrFileName=this->OpenPictureDialog1->FileName;
  if(StrFileName.Length()<1)
  {
    MessageBox(Handle,"请先选择图像文件后进行设置！","信息提示",MB_OK);
    return;
  }
  TRegistry *Reg=new TRegistry();
  Reg->OpenKey("Software\\Microsoft\\Internet Explorer\\Toolbar",true);
  Reg->WriteString("BackBitmap",StrFileName.c_str());
  Reg->WriteString("BackBitmapIE5",StrFileName.c_str());
  Reg->CloseKey();
  delete &Reg;
  MessageBox(Handle,"修改 IE 浏览器背景操作成功，请启动 IE 查看",
"信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 178 如何修改 IE 浏览器的背景”。

实例 179 如何在程序中控制 IE 窗口

实例目的

此实例介绍了如何使用 DDE 组件在程序中控制 IE 浏览器的 URL 窗口的编程技术。实例程序运行后，在 IE 浏览器中打开一个 URL（如：http://m138.mail.163.com/coremail/fcg/ldapapp?funcid=main&sid=eAsRfnBhJmFAagLZ），再单击“获取当前 IE 的 URL”按钮，效果如图 179-1 所示；如果在 URL 编辑栏中输入新的 URL（如：www.google.com），再单击“打开当前 IE 的 URL”按钮，IE 浏览器的 URL 将被切换到 www.google.com。



图 179-1

实现方法

在程序中启动 IE 浏览器可以使用 WinExec()等函数，也可以使用类似超级链接的技术设置 URL。在 C++Builder 中，使用 DDE 数据通信组件，比使用其他的技术来控制 IE 效果要好得多。DDE 组件的 DdeClientConv 组件控制 IE 浏览器时是将 IE 作为 DDE 服务器，而用户程序是当作 DDE 数据通信的客户端进行的，当 IE 浏览器运行后就可以使用这个客户端组件与 IE 浏览器进行数据通信了。使用 DDE 数据通信组件实现对 IE 浏览器的控制需要使用下面几个函数。

（1）DDE 对话组件的服务类型和主题。

```
bool __fastcall SetLink(const System:: AnsiString Service, const System::AnsiString Topic);
```

当指定 DDE 数据通信的服务类型和主题时，参数 const System:: AnsiString Service 为 DDE 服务器的应用程序名 ApplicationName（如：IExplore），参数 const System::AnsiString Topic 是 DDE 对话的 TopicName（如窗口信息）。

（2）对一个新建的 DDE 数据通信进行初始化。

```
bool __fastcall OpenLink(void);
```

（3）从一个 DDE 服务器获取所要的数据。

```
char* __fastcall RequestData(const System::AnsiString Item);
```

（4）终止一个 DDE 数据通信的进行。

```
void __fastcall CloseLink(void);
```

以下是实例程序使用 DDE 的 DdeClientConv 组件（该组件在组件模板的 System 标签页下）实现在程序中控制 IE 窗口的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//获取打开的 IE 窗口 URL
  if(this->DdeClientConv1->SetLink("IExplore","WWW_GetWindowInfo"))
  {
    this->DdeClientConv1->OpenLink();
    this->Edit1->Text=this->DdeClientConv1->RequestData("-1");
    this->DdeClientConv1->CloseLink();
  }
  else
  {
    this->Edit1->Text="浏览器没有运行！";
  }
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//打开 URL 指定的页面
  AnsiString StrURL=this->Edit1->Text;
  if(StrURL.Length()<1)
     return;
  if(this->DdeClientConv1->SetLink("IExplore","WWW_OpenURL"))
  {
    this->DdeClientConv1->OpenLink();
    this->DdeClientConv1->RequestData(StrURL);
    this->DdeClientConv1->CloseLink();
  }
  else
  {
    this->Edit1->Text="浏览器无法运行！";
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 179 如何在程序中控制 IE 窗口”。

实例 180　如何使用默认浏览器打开指定网页

实例目的

此实例介绍了如何使用默认浏览器打开指定网页的编程技术。实例程序执行后，在“网址”编辑栏中输入网址，单击“打开网页”按钮，系统将启动默认浏览器打开网页，效果

如图 180-1 所示。



图 180-1

实现方法

通过读取系统注册表，取得系统默认的浏览器，然后执行带 URL 地址参数的 WinExec() 函数，浏览器即可打开指定 URL 地址所表示的网页。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //使用默认浏览器打开指定网页
  if(this->Edit1->Text.Length()<1)
    return;
  HKEY hKeyRoot,hKeySub;
  char ValueName[MAX_PATH];
  unsigned char DataValue[MAX_PATH];
  unsigned long cbValueName=MAX_PATH;
  unsigned long cbDataValue=MAX_PATH;
  char IE[MAX_PATH];
  DWORD dwType;
  //打开根键
  if(RegOpenKey(HKEY_CLASSES_ROOT,NULL,&hKeyRoot)==ERROR_SUCCESS)
  {//打开子键
    if(RegOpenKeyEx(hKeyRoot,"htmlfile\\shell\\open\\command",
0,KEY_ALL_ACCESS,&hKeySub)==ERROR_SUCCESS)
    {//获取默认浏览器命令行
      RegEnumValue(hKeySub, 0, ValueName, &cbValueName, NULL,&dwType,DataValue,
&cbDataValue);
      strcpy(IE,(char *)DataValue);
      strcat(IE,this->Edit1->Text.c_str());
      //启动浏览器
      WinExec(IE,SW_SHOW);
    }
    else
    MessageBox(Handle,"浏览器打开错误！","信息提示",MB_OK);
  }
  else
    MessageBox(Handle,"浏览器打开错误！","信息提示",MB_OK);
  //关闭注册表
  RegCloseKey(hKeySub);
  RegCloseKey(hKeyRoot);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 180 如何使用默认浏览器打开指定网页”。

实例 181　如何获取当前正在运行的程序

实例目的

此实例介绍了如何获取当前正在运行的程序的编程技术。实例程序运行后，效果如图 181-1 所示。



图 181-1

实现方法

在 C++Builder 中，使用 GetWindow()函数能够获取指定程序的窗口，该函数的声明如下：

```
HWND GetWindow(
    HWND hWnd,  // 原始窗口的句柄
    UINT uCmd   // 查找标志
   );
```

其中，UINT uCmd 参数的可能取值包括 GW_CHILD、GW_HWNDFIRST、GW_HWNDLAST、GW_HWNDNEXT、GW_HWNDPREV、GW_OWNER。如果指定的窗口存在，则返回指定窗口的窗口句柄，否则返回 NULL 值。因此，通过 GW_HWNDFIRST 和 GW_HWNDNEXT 参数，使用枚举窗口的方式就能够获取当前正在运行的程序。示例代码如下：

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  this->ListBox1->Clear();
  HWND hWnd=GetWindow(Handle,GW_HWNDFIRST);
  while(hWnd)
  {
    char StrName[255];
```

```
        if(GetWindowText(hWnd,StrName,255)>0)
        {
          this->ListBox1->Items->Add(StrName);
        }
        hWnd=GetWindow(hWnd,GW_HWNDNEXT);
      }
    }
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 181 如何获取当前正在运行的程序”。

实例 182 如何监视系统剪贴板变化

实例目的

此实例主要介绍了如何判断系统剪贴板是否发生变化。实例程序执行后，当系统发生剪切、粘贴变化时，将弹出一个对话框，效果如图 182-1 所示。



图 182-1

实现方法

操作系统的剪贴板是系统内进行数据交换最快捷的方式，它能够从一个应用程序中将剪贴板标准格式的数据（例如文本、位图等）复制粘贴到另一个应用程序。通过系统消息 WM_DRAWCLIPBOARD 可以随时监视系统剪贴是否发生变化，以决定是否进行相应的复制、粘贴操作。实例程序实现监视系统剪贴板是否发生变化的具体步骤如下。

（1）在程序头文件中声明消息映射和消息响应函数，代码如下所示：

```
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TBitBtn *BitBtn1;
```

```
        TImage *Image1;
        void __fastcall BitBtn1Click(TObject *Sender);
private:        // User declarations
public:         // User declarations
//剪贴板改变消息映射
BEGIN_MESSAGE_MAP
  MESSAGE_HANDLER(WM_DRAWCLIPBOARD,TMessage,ClipboardChanged)
END_MESSAGE_MAP(TForm)
 __fastcall TForm1(TComponent* Owner);
//剪贴板发生改变消息响应函数
void __fastcall ClipboardChanged(TMessage& Msg);
};
```

（2）在实现文件中定义剪贴板发生改变时的动作，具体代码如下：

```
void __fastcall TForm1::ClipboardChanged(TMessage& Msg)
{
  MessageBox(this->Handle,"系统剪贴板有变化","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  static HWND Handle;
  static bool clip=false;
  if(clip==true)
  {
    ChangeClipboardChain(Form1->Handle,Handle);
    this->BitBtn1->Caption="启动系统剪贴板监视";
  }
  else
  {
    Clipboard()->Clear();
    Handle=SetClipboardViewer(Form1->Handle);
    this->BitBtn1->Caption="关闭系统剪贴板监视";
  }
  clip=!clip;
}
```

最后编译执行程序即可。注意，程序在编译时需要包括#include <vcl\Clipbrd.hpp>头文件。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 182 如何监视系统剪贴板变化”。

实例 183　如何获取系统剪贴板的数据格式

实例目的

此实例介绍了如何获取系统剪贴板的数据格式的编程技术。实例程序执行后，效果如

图 183-1 所示。



图 183-1

实现方法

系统剪贴板是同一程序或不同程序间进行数据交换的主要媒介。除了系统默认提供的数据格式外，用户程序也可以向系统剪贴板注册自己的数据格式。以下是实例程序获取系统剪贴板的数据格式的示例代码。

```
char *StrClipboardFormats[17]=
{"cf_text","cf_bitmap","cf_metafilepict","cf_sylk",
          "cf_dif","cf_tiff","cf_oemtext","cf_dib","cf_palette",
          "cf_pendata","cf_riff","cf_wave","cf_unicodetext",
          "cf_enhmetafile","cf_hdrop","cf_locale","cf_max"};
String GetUnknownFormatName(long lFormatID)
{
  char StrBuffer[255];
  long lIndex;
  lIndex = GetClipboardFormatName(lFormatID,StrBuffer,255);
  if(lIndex!=0)
    return String(StrBuffer);
  else
    return "-----None------";
}
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
 int lIndex;
 int iCount;
 ListBox1->Items->Clear();
 if(OpenClipboard(Handle))
 {
   try
   {
     lIndex=EnumClipboardFormats(0);
     if(lIndex!=0)
     {
       do
       {
         iCount++;
```

```
            if(lIndex<18)
              ListBox1->Items->Add(StrClipboardFormats[lIndex]);
            else
              ListBox1->Items->Add(GetUnknownFormatName(lIndex));
            lIndex = EnumClipboardFormats(lIndex);
          }while(lIndex!=0);
        }
      }
      catch(...)
      {
        CloseClipboard();
      }
    }
    CloseClipboard();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 183 如何获取系统剪贴板的数据格式”。

实例 184　如何设置系统日期

实例目的

此实例介绍了如何设置系统日期的编程技术。实例程序运行后，效果如图 184-1 所示。

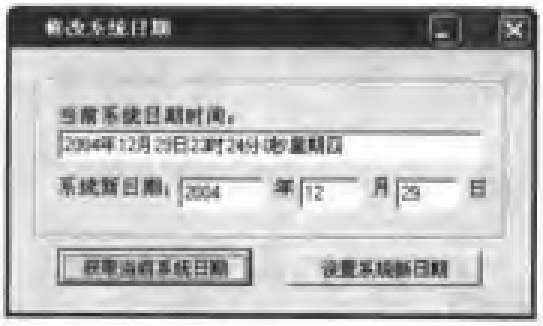


图 184-1

实现方法

在 C++Builder 中，设置系统日期可以使用系统 API 函数 SetSystemTime()实现，该函数的声明如下：

```
BOOL SetSystemTime(
    CONST SYSTEMTIME *lpSystemTime  //系统日期设置地址
  );
```

其中，参数 CONST SYSTEMTIME *lpSystemTime 的结构定义如下：

```
typedef struct _SYSTEMTIME {
  WORD wYear;                    //年份
  WORD wMonth;                   //月份
  WORD wDayOfWeek;               //星期
  WORD wDay;                     //日
  WORD wHour;                    //小时
  WORD wMinute;                  //分
  WORD wSecond;                  //秒
  WORD wMilliseconds;            //毫秒
} SYSTEMTIME;
```

在使用系统 API 函数 SetSystemTime()设置系统日期时间时，通常应该使用系统 API 函数 GetSystemTime()获取系统当前的日期时间，通过对已经取得的当前日期时间进行修改，就可以减少对时间日期结构赋值时出现的错误。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{  //获取系统当前日期时间
  TSystemTime SDate;
  GetSystemTime(&SDate);
  AnsiString StrDate;
  StrDate+=IntToStr(SDate.wYear)+"年";
  StrDate+=IntToStr(SDate.wMonth)+"月";
  StrDate+=IntToStr(SDate.wDay)+"日";
  StrDate+=IntToStr(SDate.wHour)+"时";
  StrDate+=IntToStr(SDate.wMinute)+"分";
  StrDate+=IntToStr(SDate.wSecond)+"秒";
  switch(SDate.wDayOfWeek)
  {
   case 0:
      StrDate+="星期一";
      break;
   case 1:
      StrDate+="星期二";
      break;
   case 2:
      StrDate+="星期三";
      break;
   case 3:
      StrDate+="星期四";
      break;
   case 4:
      StrDate+="星期五";
      break;
   case 5:
      StrDate+="星期六";
      break;
   case 6:
      StrDate+="星期天";
      break;
  }
  this->Edit1->Text=StrDate;
```

```
    this->Edit2->Text=IntToStr(SDate.wYear);
    this->Edit3->Text=IntToStr(SDate.wMonth);
    this->Edit4->Text=IntToStr(SDate.wDay);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//设置系统日期
  int iYear,iMonth,iDay;
  try
  {
    iYear=StrToInt(this->Edit2->Text);
    iMonth=StrToInt(this->Edit3->Text);
    iDay=StrToInt(this->Edit4->Text);
  }
  catch(...)
  {
    MessageBox(Handle,"日期数字输入错误","信息提示",MB_OK);
    return;
  }
  TSystemTime SDate;
  GetSystemTime(&SDate);
  SDate.wYear=iYear;
  SDate.wMonth=iMonth;
  SDate.wDay=iDay;
  SetSystemTime(&SDate);
  MessageBox(Handle,"设置系统日期操作成功","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 184 如何设置系统日期”。

实例 185　如何获取操作系统的产品序列号和产品名称

实例目的

此实例介绍了如何获取当前操作系统的产品序列号和产品名称的编程技术。实例程序执行后，效果如图 185-1 所示。



图 185-1

实现方法

查询操作系统的注册表子键 HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\ CurrentVersion 的 ProductId 键值项可以获取操作系统的安装序列号；查询 ProductName 键值项可以获取操作系统的产品名称。以下是实例程序据此获取操作系统安装序列号和产品名称的实现代码。

```
#include 'Registry.hpp'
void __fastcall TForm1::Bit8tn1Click(TObject *Sender)
{ //获取操作系统的序列号和产品名称
  TRegistry* Reg=new TRegistry();
  Reg->RootKey=HKEY_LOCAL_MACHINE;
  Reg->OpenKey('Software\\Microsoft\\Windows NT\\CurrentVersion',true);
  this->Edit1->Text=Reg->ReadString('ProductId');
  this->Edit2->Text=Reg->ReadString('ProductName');
  Reg->CloseKey();
  delete Reg;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 185 如何获取操作系统的产品序列号和产品名称”。

实例 186 如何获取和设置鼠标双击响应速度

实例目的

此实例介绍了如何获取和设置鼠标双击响应速度的编程技术。实例程序执行后，单击“获取鼠标双击响应速度”按钮，效果如图 186-1 所示。



图 186-1

实现方法

使用系统 API 函数 GetDoubleClickTime()可以获取鼠标双击响应速度。实例程序使用

API 函数 GetDoubleClickTime()获取鼠标双击响应速度的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //获取鼠标双击响应速度
  this->Edit1->Text=IntToStr(GetDoubleClickTime());
}
```

通过对操作系统注册表子键 HKEY_CURRENT_USER\Control Panel\Mouse 的 DoubleClick Speed 键值项的设置可以调整鼠标的双击响应速度。DoubleClickSpeed 键值对应的数据单位为毫秒，范围在 100~900 之间。也可以通过 API 函数 SetDoubleClickTime()设置鼠标双击响应速度。实例程序使用 API 函数 SetDoubleClickTime()设置鼠标双击响应速度的示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//设置鼠标双击响应速度
  if(this->Edit1->Text.Length()<1)
    return;
  if(SetDoubleClickTime(StrToInt(this->Edit1->Text)))
    MessageBox(Handle,"设置鼠标双击响应速度为操作成功","信息提示",MB_OK);
  else
    MessageBox(Handle,"设置鼠标双击响应速度为操作失败","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分　系统编程\实例 186 如何获取和设置鼠标双击响应速度”。

实例 187　如何弹出和关闭光驱

实例目的

此实例主要介绍了如何使用代码控制弹出和关闭光驱的编程技术。实例程序执行后，效果如图 187-1 所示。



图 187-1

实现方法

弹出和关闭光驱可以使用函数 mciSendString()实现。函数 mciSendString()的功能就是

向 MCI 设备发送命令，该函数定义在 MMSystem.hpp 文件中，使用时必须包括此头文件。其函数声明如下：

```
MCIERROR mciSendString(
    LPCTSTR lpszCommand,          //指令字符串
    LPTSTR lpszReturnString,      //返回信息字符串
    UINT cchReturn,               //返回信息字符串长度
    HANDLE hwndCallback           //窗口句柄
   );
```

实例程序使用此函数实现使用代码控制弹出和关闭光驱的示例代码如下：

```
#include"MMSystem.hpp"
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//弹出光驱
  mciSendString('Set cdaudio door open wait',NULL,0,Handle);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//关闭光驱
  mciSendString('Set cdaudio door Closed wait',NULL,0,Handle);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 187 如何弹出和关闭光驱”。

实例 188 如何检测网络连接

实例目的

此实例介绍了如何使用 C++Builder 的 NMEcho 组件实现网络连接检测的编程技术。实例程序运行前，需要在远程主机上启动一个服务器程序，如“实例 066 如何制作聊天服务器程序”中的实例程序，然后单击“连接远程主机”按钮即可，效果如图 188-1 所示。

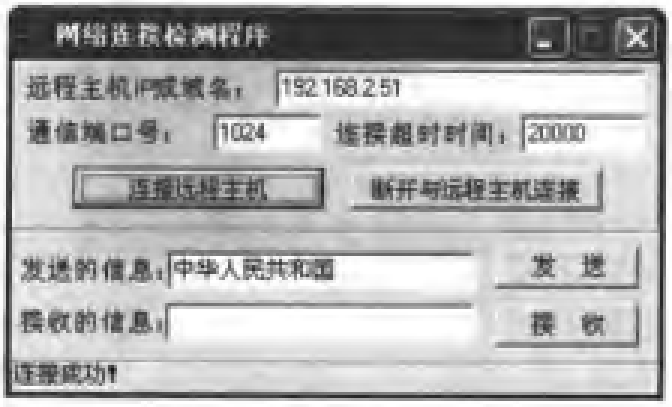


图 188-1

实现方法

在 Windows 操作系统中，经常用 Ping.exe 程序来检测网络的连接情况。Ping 的实现过程很简单：在执行该命令时，将引发 IP 层向远程主机发送一个简单的 IP 包，一般是 32 字节；远程主机收到这个包后，将源地址和目的地址交换一下，重新发送回这个包即可。C++Builder 组件面板的 FastNet 标签页中的 NMEcho 控件也同样能够实现网络连接检测功能。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  NMEcho1->ReportLevel = Status_Basic;
  NMEcho1->TimeOut = StrToInt(Edit5->Text);
  NMEcho1->Host = Edit2->Text;
  NMEcho1->Port = StrToInt(Edit3->Text);
  NMEcho1->Connect();
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  NMEcho1->Disconnect();
}
void __fastcall TForm1::NMEcho1Connect(TObject *Sender)
{
   StatusBar1->SimpleText = "连接成功！";
}
void __fastcall TForm1::NMEcho1ConnectionFailed(TObject *Sender)
{
   StatusBar1->SimpleText = "连接失败！";
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit4->Text = NMEcho1->Echo(Edit1->Text);
}
void __fastcall TForm1::Button4Click(TObject *Sender)
{
  NMEcho1->Abort();
}
void __fastcall TForm1::NMEcho1Disconnect(TObject *Sender)
{
  if (StatusBar1 != 0)
    StatusBar1->SimpleText = "连接已经被断开！";
}
void __fastcall TForm1::NMEcho1HostResolved(TComponent *Sender)
{
  StatusBar1->SimpleText = "成功解析远程主机域名！";
}
void __fastcall TForm1::NMEcho1InvalidHost(bool &Handled)
{
  AnsiString NewHost;
  if (InputQuery("远程主机域名无效！", "请重新选择远程主机", NewHost))
```

```
  {
    NMEchol->Host = NewHost;
    Handled = true;
  }
}
void __fastcall TForm1::NMEcholStatus(TComponent *Sender,
     AnsiString Status)
{
 if (StatusBar1 != 0)
   StatusBar1->SimpleText = Status;

}
void __fastcall TForm1::NMEcholConnectionRequired(bool &Handled)
{
 if (MessageBox(0, "是否需要连接远程主机?", "信息提示",
MB_YESNO + MB_ICONEXCLAMATION) == IDYES)
  {
    Handled = true;
    Button1Click(NULL);
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 188 如何检测网络连接”。

实例 189 如何创建拨号网络

实例目的

此实例介绍了如何在程序中创建拨号网络的编程技术。实例程序执行后，在“服务电话号码”编辑栏中输入 ISP 的电话号码，在“登录用户名”编辑栏中输入 ISP 提供给用户的用户名，在“登录密码”编辑栏中输入 ISP 提供给用户的密码，单击“创建拨号网络”按钮，系统将立即拨号连接互联网，如果不再使用互联网，则单击“断开拨号网络”按钮，程序运行效果如图 189-1 所示。

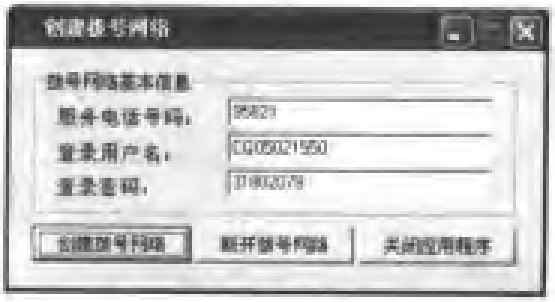


图 189-1

实现方法

操作系统提供了 RasDial()函数调用系统中已经创建好的拨号连接完成 RAS 拨号，同时提供了 RasHangUp()函数用于断开连接。RasDial()函数的声明如下：

```
DWORD RasDial(LPRASDIALEXTENSIONS lpRasDialExtensions,
    LPCTSTR lpszPhonebook,LPRASDIALPARAMS lpRasDialParams,
DWORD dwNotifierType,LPVOID lpvNotifier, LPHRASCONN lphRasConn );
```

其中，参数 LPRASDIALPARAMS lpRasDialParams 为指向 RASDIALPARAMS 结构的指针，它用于配置 RAS 拨号参数，参数 LPVOID lpvNotifier 为完成后执行的回调函数，参数 LPHRASCONN lphRasConn 是 LPHRASCONN 结构的指针，拨号连接建立后将返回此连接的句柄。

以下是实例程序使用 RasDial()函数创建拨号连接的示例代码。

```
#include "ras.h"
#include "raserror.h"
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//创建拨号连接
    RASDIALPARAMS rdParams;
    rdParams.dwSize = sizeof(RASDIALPARAMS);
    rdParams.szEntryName[0] = '\0';
    lstrcpy( rdParams.szPhoneNumber,this->Edit1->Text.c_str());
    rdParams.szCallbackNumber[0] = '\0';
    lstrcpy( rdParams.szUserName, this->Edit2->Text.c_str() );
    lstrcpy( rdParams.szPassword, this->Edit3->Text.c_str());
    rdParams.szDomain[0] = '\0';
    HRASCONN hRasConn = NULL;
    DWORD dwRet = RasDial( NULL, NULL, &rdParams, 0L, NULL, &hRasConn );
    if ( dwRet == 0 )
      return ;
    char  szBuf[256];
    if ( RasGetErrorString( (UINT)dwRet, (LPSTR)szBuf, 256 ) != 0 )
          wsprintf( (LPSTR)szBuf, "未知的 RAS 拨号错误: (%ld).", dwRet );
    RasHangUp( hRasConn );
    MessageBox( Handle,(LPSTR)szBuf, "信息提示", MB_OK | MB_ICONSTOP );
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //断开拨号连接
  RASCONN ras[20];
  DWORD  dSize, dNumber;
  char  szBuf[256];
  ras[0].dwSize = sizeof( RASCONN );
  dSize = sizeof( ras );
  DWORD  dwRet = RasEnumConnections( ras, &dSize, &dNumber );
  if ( dwRet != 0 )
  {
    if ( RasGetErrorString( (UINT)dwRet, (LPSTR)szBuf, 256 ) != 0 )
       wsprintf( (LPSTR)szBuf, "未定义 RAS 连接错误 (%ld).", dwRet );
    MessageBox(Handle, (LPSTR)szBuf, "信息提示", MB_OK | MB_ICONSTOP );
```

```
        return ;
      }
      for( DWORD dCount = 0;  dCount < dNumber;  dCount++ )
      {
       HRASCONN hRasConn = ras[dCount].hrasconn;
       DWORD dwRet = RasHangUp( hRasConn );
       if ( dwRet != 0 )
       {
            char  szBuf[256];
         if ( RasGetErrorString( (UINT)dwRet, (LPSTR)szBuf, 256 ) != 0 )
           wsprintf( (LPSTR)szBuf, "未知的 RAS 错误(%ld).", dwRet );
         MessageBox(Handle, (LPSTR)szBuf, "信息提示", MB_OK | MB_ICONSTOP );
        }
      }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第五部分 系统编程\实例 189 如何创建拨号网络”。

第六部分　数据库应用

实例 190　如何创建不写代码的数据库程序

实例目的

此实例主要介绍了 C++Builder 最基本的数据库编程过程。当实例程序启动后，在主界面中已经有一个表格显示了系统示例数据库中的内容，单击导航条上的上一记录、下一记录、首记录、末记录、增加记录、删除记录按钮时，程序能够实现相应的数据库功能。效果如图 190-1 所示。



图 190-1

实现方法

与 Visual C++开发工具相比，C++Builder 的最大特点就是在数据库应用程序方面大大简化了操作流程，在 C++Builder 中，可以不用编写一行代码而实现一个具有基本数据库操作功能的应用程序。其实现关键在于，数据库的一切操作均能够通过组件实现。通常 C++Builder 创建的数据库应用程序包含 3 个基本的数据库组件：数据集组件、数据源组件、数据控制组件。数据集组件负责与 BDE 联系，用于直接访问数据库中的数据库表，在程序运行时是不可见的；数据源组件负责连接数据集组件和用户界面；数据控制组件则为用户界面提供浏览和编辑数据。常用的数据集及数据源组件的功能如表 190-1 所示。

表 190-1

组件名称	主要功能
TDataSource	作为数据集组件 TTable 和 TQuery 与数据浏览组件 TDBGrid 和 TDBEdit 之间传送数据的通道
TTable	存取磁盘上数据库的媒介，它通过 BDE 存取数据库表中的数据，数据浏览组件通过 TDataSource 从 TTable 中访问数据并显示和编辑其中的数据
TQuery	利用 SQL 语句访问数据库表中的数据，通过 TDataSource 组件实现数据浏览组件对数据库的访问
TSourceProc	用来访问远程服务器中的存储过程
TDatabase	用来建立应用程序与要登录的远程服务器上的数据库的永久性连接
TSession	用来控制数据库应用程序和数据库的连接
TBatchMove	用来复制数据库表的结构或表中的记录
TUpdateSQL	根据预定义的 SQL 语句来修正数据集
TNestedTable	通过 TDataSource 组件存取嵌套数据集字段中的数据并提供给数据浏览组件

常用的数据控制组件的功能如表 190-2 所示。

表 190-2

组件名称	主要功能
TDBGrid	浏览数据库中数据的网格，它以网格的方式显示数据库中的数据，在网格中可以对数据进行编辑。利用 FieldsEditor 可以对数据库表中字段的显示格式、显示顺序及是否显示等进行控制
TDBNavigator	用来向前或向后移动记录指针，可以对单条记录进行编辑，还可以用来插入、删除记录以及刷新显示和取消上一次操作
TDBText	用来显示数据库中数据的文本框，但它只能显示数据库表中当前记录的字段值，用户不能对其进行修改
TDBEdit	用来显示和编辑数据库表中数据的编辑框，可以显示和编辑数据库表中当前记录的字段值
TDBCheckBox	浏览数据库中数据的检查框，可以用来显示和编辑数据库中布尔型字段的字段值
TDBListBox	浏览数据库中数据的列表框，可以用一个列表框来显示数据库表中一个字段的值
TDBComboBox	浏览数据库中数据的组合框，可以用一个组合框来显示数据库表中一个字段的值
TDBRadioGroup	浏览数据库中数据的单选框，用一组单选按钮来确定显示数据库表中哪一个字段
TDBMemo	用来浏览数据库中备注型的字段，可以显示数据库表中当前记录中的 BLOB 型字段
TDBImage	用来浏览数据库中数据的图像框，可以显示、拷贝、粘贴数据库表中数据图像类型的字段
TDBLookUpList	浏览数据库表中数据的列表框，在基于一个数据库表的应用中，用它可以显示另一个数据库表中一个指定的字段值

续表

组件名称	主要功能
TDBLookUpCombo	浏览数据库表中数据的组合框，在基于一个数据库表的应用中，用它可以显示另一个数据库表中一个指定的字段值
TDBCtrlGrid	用来显示和编辑数据库表中查询的网格，与 TDBGrid 组件不同，它不是在一行上显示一条记录，而是允许数据集中数据自由分布，每一条记录都显示在各自的面板上
TDBRichEdit	用来显示和编辑数据库表中数据的编辑框，可以多行文本显示和编辑 RTF 格式的备注类型字段的内容
TDBChart	以图表的形式显示数据库中的数据，用法与 TChart 组件相似

实例程序通过使用数据库组件，而不使用代码编写数据库应用程序的具体步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。注意，这时的应用程序还没有编译执行，就可以看到应用程序执行后的结果，这就是 C++Builder 的所谓 Live Data 技术。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 190 如何创建不写代码的数据库程序”。

实例 191　如何添加 TDBNavigator 组件的按钮功能

实例目的

此实例主要介绍了如何在 TDBNavigator 组件已有功能的基础上添加新功能。实例程序执行后，效果如图 191-1 所示。



图 191-1

实现方法

TDBNavigator 组件为在数据集中进行记录导航和为用户操作数据集中的记录提供了一组简单明了的控制按钮，这些按钮功能包括对数据库记录的增加、修改、删除和浏览等，其中每个按钮分别代表其不同的功能。TDBNavigator 组件中的按钮功能说明如表 191-1 所示。

表 191-1

按钮名称	功能说明
First	将当前记录指针移到数据表中第一条记录处
Prior	将记录指针移到当前记录的前一条记录处
Next	将记录指针移到当前记录的后一条记录处
Last	将当前记录指针移到数据表中最后一条记录处
Insert	调用数据集组件的 Insert()方法，在当前记录的前面插入一条新记录，并将数据集组件置为插入状态
Delete	删除当前记录，如果 TDBNavigator 组件的 ConfirmDelete 属性设置为 TRUE，会弹出删除确认对话框
Edit	将数据集组件置为编辑状态，以便用户修改当前记录
Post	提交对当前记录的更新
Cancel	取消对当前记录的更新，并将数据集组件置为浏览状态
Refresh	清除数据访问组件的显示缓冲区，并用与其相连的数据集组件中的记录刷新显示缓冲区

如果想在 TDBNavigator 组件每个按钮现有功能的基础上再加上其他功能，可以使用 TDBNavigator 组件内的一个 OnClick()事件函数来实现。示例代码如下：

```
void __fastcall TForm1::OnClickDBNavigator(TObject *Sender, TNavigateBtn Button)
{
  switch(Button)
  {
    case nbFirst:
      MessageBox(NULL,"浏览第一行","信息提示",MB_OK);
```

```
        break;
        case nbPrior:
          MessageBox(NULL,"浏览前一行","信息提示",MB_OK);
        break;
        case nbNext:
          MessageBox(NULL,"浏览下一行","信息提示",MB_OK);
        break;
        case nbLast:
          MessageBox(NULL,"浏览最后一行","信息提示",MB_OK);
        break;
        case nbInsert:
          MessageBox(NULL,"进入新增记录状态","信息提示",MB_OK);
        break;
        case nbDelete:
          MessageBox(NULL,"删除一条记录","信息提示",MB_OK);
        break;
        case nbEdit:
          MessageBox(NULL,"进入编辑记录状态","信息提示",MB_OK);
        break;
        case nbPost:
          MessageBox(NULL,"提交记录改变","信息提示",MB_OK);
        break;
        case nbCancel:
          MessageBox(NULL,"撤消记录改变","信息提示",MB_OK);
        break;
        case nbRefresh:
          MessageBox(NULL,"更新数据","信息提示",MB_OK);
        break;
      }
    }
```

最后编译执行程序即可。需要说明的是：应用程序执行时，会先去执行 TDBNavigator 组件内已有的功能，再执行在 OnClick()函数内新添加的功能。例如，当单击“+”新增记录按钮后，程序在执行内定的新增记录功能后，再弹出消息框。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 191 如何添加 TDBNavigator 组件的按钮功能”。

实例 192　如何对数据库记录求和

实例目的

此实例主要介绍了如何对数据库中的记录进行求和并显示的编程技术。实例程序执行后，效果如图 192-1 所示。



图 192-1

实现方法

在标准 SQL 语句中，可以使用 SUM()函数进行数据库记录的求和运算，但是在 C++Builder 中，如何在调用这种 SQL 语句后，将求和结果显示出来呢（因为求和字段并没有在数据库表中定义）？以下是实例程序实现对数据库记录进行求和运算并显示的具体步骤：

（1）在组件模板的 BDE 标签页下拖动 Query1:TQuery 组件到 Form1 上，在 Object Inspector 中设置 Query1 组件的 DatabaseName 属性为 BCDEMOS，SQL 属性为 SELECT * FROM Country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Query1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（5）编写如下所示的代码实现数据库记录的求和运算。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//求面积字段总和
  TQuery* pQ=this->Query1;
  pQ->Close();
  pQ->SQL->Clear();
  pQ->SQL->Text="SELECT SUM(Area) FROM Country.db";
  pQ->Open();
  int iField=pQ->FieldCount;
  AnsiString StrFieldName;
  if(iField>0)
    StrFieldName=pQ->Fields->Fields[0]->FieldName;
  double TotalArea=pQ->FieldByName(StrFieldName)->AsFloat;
  this->Edit1->Text=FloatToStr(TotalArea);
  pQ->Close();
  pQ->SQL->Clear();
  pQ->SQL->Text="SELECT * FROM Country.db";
  pQ->Open();
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 192 如何对数据库记录求和”。

实例 193　如何对数据库记录求平均值

实例目的

此实例主要介绍了如何对数据库中的记录求平均值并显示的编程技术。实例程序执行后，效果如图 193-1 所示。



图 193-1

实现方法

在标准 SQL 语句中，可以使用 AVG()函数求数据库记录的平均值，但是在 C++Builder 中，如何在调用这种 SQL 语句后，将其结果显示出来呢（因为平均值字段并没有在数据库表中定义）？以下是实例程序实现对数据库记录进行平均值运算并显示结果的具体步骤：

（1）在组件模板的 BDE 标签页下拖动 Query1:TQuery 组件到 Form1 上，在 Object Inspector 中设置 Query1 组件的 DatabaseName 属性为 BCDEMOS，SQL 属性为 SELECT * FROM Country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Query1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（5）编写如下所示的代码实现数据库记录的平均值运算。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
```

```
{//求面积字段平均值
 TQuery* pQ=this->Query1;
 pQ->Close();
 pQ->SQL->Clear();
 pQ->SQL->Text="SELECT AVG(Area) FROM Country.db";
 pQ->Open();
 int iField=pQ->FieldCount;
 AnsiString StrFieldName;
 if(iField>0)
   StrFieldName=pQ->Fields->Fields[0]->FieldName;
 double TotalArea=pQ->FieldByName(StrFieldName)->AsFloat;
 this->Edit1->Text=FloatToStr(TotalArea);
 pQ->Close();
 pQ->SQL->Clear();
 pQ->SQL->Text="SELECT * FROM Country.db";
 pQ->Open();
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 193 如何对数据库记录求平均值”。

实例 194 如何使用 Locate 方法查询数据库记录

实例目的

此实例主要介绍了如何使用 Locate()方法查询数据库记录的编程技术。实例程序执行后，效果如图 194-1 所示。



图 194-1

实现方法

在 C++Builder 中，查询数据库记录有多种方式，但是如果待查询的数据库字段没有建

立索引，就不能使用一般的查询方法。然而，Locate()方法能够解决此问题，它几乎适用于所有类型的查询，如果待查询的字段有索引，则 Locate()方法将自动使用索引，否则就使用一般的方法查询数据库记录。以下是实例程序使用 Locate()方法查询数据库记录的具体步骤：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 Additionl 标签页下拖动一个按钮组件到 Form1 上（其他组件包括 TLabel 和 TEdit），然后为该按钮添加如下所示的实现代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //使用 Locate()数据库查询记录
  if(this->Edit1->Text.Length()<1)
    return;
  TLocateOptions Opts;
  Opts.Clear();
  Opts<<loCaseInsensitive;
  Variant LocateValues;
  LocateValues=Variant(this->Edit1->Text);
  if(this->Table1->Locate("Capital",LocateValues,Opts))
    MessageBox(Handle,"已经找到该国","信息提示",MB_OK);
  else
    MessageBox(Handle,"没有找到该国","信息提示",MB_OK);
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 194 如何使用 Locate 方法查询数据库记录”。

实例 195　如何查询固定范围的数据库记录

实例目的

此实例主要介绍了使用 SQL 语句查询固定范围的数据库记录的编程技术。实例程序执行后，效果如图 195-1 所示。



图 195-1

实现方法

在 C++Builder 中，查询固定范围的数据库记录可以使用 SetRange()方法实现，但是如果使用 SQL 语句，则灵活得多。以下是实例程序使用 SQL 语句查询固定范围的数据库记录的具体步骤：

（1）在组件模板的 BDE 标签页下拖动 Query1:TQuery 组件到 Form1 上，在 Object Inspector 中设置 Query1 的 DatabaseName 属性为 BCDEMOS，SQL 属性为 SELECT * FROM country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Query1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 Additionl 标签页下拖动一个按钮组件到 Form1 上（其他组件包括两个 TLabel 和两个 TEdit 组件），然后为该按钮添加如下所示的实现代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //查询固定范围的数据库记录
  try
  {
    this->Query1->Close();
    this->Query1->SQL->Clear();
    AnsiString StrSQL="SELECT * FROM country.db WHERE Area>";
    StrSQL+=this->Edit1->Text+" AND Area<";
    StrSQL+=this->Edit2->Text;
    this->Query1->SQL->Text=StrSQL;
    this->Query1->Open();
    MessageBox(Handle,"查询固定范围的数据库记录操作成功！","信息提示",MB_OK);
  }
  catch(...)
  {
    MessageBox(Handle,"查询固定范围的数据库记录操作失败！","信息提示",MB_OK);
  }
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 195 如何查询固定范围的数据库记录”。

实例 196　如何提高数据库应用程序查询效果

实例目的

此实例主要介绍了如何在相同的条件下提高数据库应用程序的查询效率。实例程序执行后，效果如图 196-1 所示。



图 196-1

实现方法

使用 C++Builder 编写的数据库应用程序可以通过查询组件 TQuery 直接调用 SQL 语句实现对数据库的查询甚至更新操作。当使用 SQL 语句查询数据库时，可以通过两种方法来提高程序的查询效率。一是在执行 SQL 语句前，预先准备查询 Prepare()方法。所谓预先准备查询就是先将 SQL 语句传送给 BDE 和远程数据库系统，预先进行 SQL 的语法分析、最佳化处理、判断用户权限等操作，这样将会增加 SQL 程序的查询效率，尤其是在相同的 SQL 查询程序必须重复执行多次时，效率改善更为显著。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  Query1->Close();
  Query1->SQL->Clear();
  Query1->SQL->Text="SELECT * FROM COUNTRY WHERE Continent
 LIKE '%'+this->Edit1->Text+'%'";
  Query1->Prepare();
  Query1->Open();
}
```

第二种改变查询效率的方法是将 TQuery 组件的 UniDirectional 属性设置为 true。TQuery

组件的 UniDirectional 属性用来决定是否可以往后查看结果集的内容，虽然大多数的数据库系统都不允许用户向后翻阅查询出来的结果集，可是 C++Builder 的 BDE 却支持双向记录指针，允许前后移动结果集上的记录指针。由于双向记录指针的功能会增加一些额外的处理负担，使得查询效率变得比较差，所以如果确定不需要向后查看数据时，可以将 TQuery 组件的 UniDirectional 属性设置为 true，从而提高查询效率。需要特别注意的是，必须在准备及执行 SQL 查询之前设置 TQuery 组件的 UniDirectional 属性。

此外，TQuery 组件在一定的条件下支持数据库更新操作。即 SQL 语句只涉及到一个数据库表，而且 SQL 语句不包含集合性函数，如 SUM、AVG 等。在这种情况下，只要将 TQuery 组件的 RequestLive 属性设置成 true，就可以直接修改查询的结果集。也就是说，TQuery 组件除了可以执行 SELECT 查询命令之外，还可以在 TQuery 组件内执行 Insert、Update、Delete 等 SQL 命令。如果 TQuery 组件内的 SQL 命令包含超过一个以上的数据库表时，所产生的 DataSet 将会是一个只读的数据集，此时如果强制设置 TQuery 组件的 RequestLive 属性为 true，则将会得到一个错误信息。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 196 如何提高数据库应用程序查询效果”。

实例 197 如何使用数据集的书签

实例目的

此实例主要介绍了在遍历数据集的所有记录时，如何在找到目标记录后设定书签，供以后一次定位到该记录位置的编程技术。实例程序执行后，效果如图 197-1 所示。



图 197-1

实现方法

书签就是在数据集的某个指定位置做一个标记，以便以后能够快速地找到那个位置。

要在数据集中创建一个书签，首先要声明一个 TBookmark 类型的变量，然后调用数据集的 GetBookmark()函数创建一个标记当前记录的书签。TBookmark 类型的变量实际上是一个 void 类型的指针，在调用 GotoBookmark()函数重新回到书签之前，最好先调用 BookmarkValid()函数判断该书签是否合法，因为书签标记的记录有可能已经被删除，如果 BookmarkValid()函数的返回值为 TRUE，说明书签是合法的，可以调用 GotoBookmark()函数重新返回到书签位置，否则需要另做处理。实例程序使用书签的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //确定书签
   this->Table1->DisableControls();
   this->Table1->First();
   while(!this->Table1->Eof)
   {
     AnsiString StrName=this->Table1->FieldValues["Name"];
     if(StrName=="Brazil")
     {
       AnsiString StrInfo="Brazil's Capital is ";
       StrInfo+=this->Table1->FieldValues["Capital"];
       MessageBox(NULL,StrInfo.c_str(),"信息提示",MB_OK);
       this->MyBookmark=this->Table1->GetBookmark();
       this->Table1->Last();
     }
     this->Table1->Next();
   }
   this->Table1->First();
   this->Table1->EnableControls();
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //查找书签
   try
   {
     this->Table1->GotoBookmark(this->MyBookmark);
     this->Table1->FreeBookmark(this->MyBookmark);
   }
   catch(...)
   {
     MessageBox(NULL,"书签已不存在","信息提示",MB_OK);
   }
}
```

注意：书签也是一种资源，当书签不再需要时，应及时调用 FreeBookmark()函数删除之。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 197 如何使用数据集的书签”。

实例 198 如何浏览数据库中的图像

实例目的

此实例主要介绍了如何在C++Builder中直接使用TDBImage组件实现浏览数据库中的图像字段的编程技术。实例程序执行后，效果如图 198-1 所示。

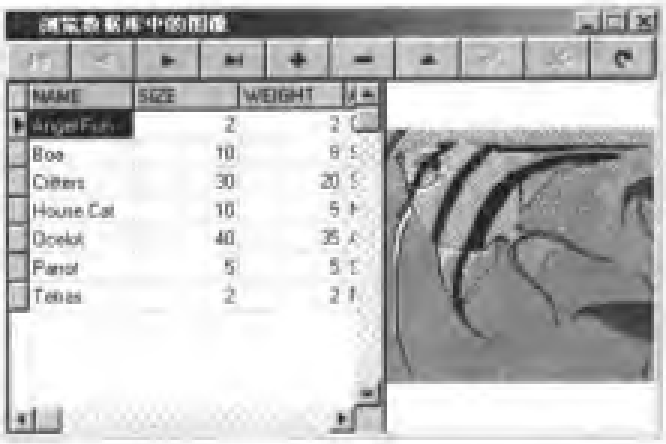


图 198-1

实现方法

C++Builder 的 TDBImage 组件主要用于显示和编辑数据库表中 BLOB 类型的图像字段，它能够随着数据库记录当前位置的变化而自动改变显示的图像。默认情况下，该组件允许用户编辑图像，如将图像剪切或复制到剪贴板上或从剪贴板上将图像粘贴到 TDBImage 组件中等操作；它也可以使用 CutToClipboard()、CopyToClipboard()和 PasteFromClipboard()方法实现图像的剪切、复制和粘贴操作，当然要进行上述方法调用，必须设置 TDBImage 组件的 ReadOnly 属性为 false。实例程序实现使用 TDBImage 组件显示数据库图像字段的具体步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 Animals.dbf，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（5）在组件模板的 DataControls 标签页下拖动 DBImage1:TDBImage 组件到 Form1

上，在 Object Inspector 中设置 DBImage1 的 DataSource 属性为 DataSource1，DataField 属性为 BMP。

最后编译执行程序即可。注意：TDBImage 组件也一个 AutoDisplay 属性，如果该属性设置为 true，当 TDBImage 组件显示的图像字段内容改变时，TDBImage 组件会自动刷新图像；如果该属性设置为 false，则 TDBImage 组件上只显示图像字段名，必须双击 TDBImage 组件才能显示图像。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 198 如何浏览数据库中的图像”。

实例 199　如何在删除数据库记录前给予提示

实例目的

此实例主要解决了在执行 Delete()函数后，如何取消 Delete()函数所要执行的删除数据库记录操作。实例程序执行后，效果如图 199-1 所示。



图 199-1

实现方法

在 C++Builder 中使用 TTable 组件维护数据库与应用程序之间的关系时，在执行增加、修改、插入数据库记录等操作后，必须调用 Post()函数才能使以前的操作生效，惟独删除数据库记录 Delete()函数不需要调用 Post()函数就立即生效。为了让用户在使用 TTable 组件时再一次确认是否真正要删除数据，TTable 组件提供了 BeforeDelete 事件函数，程序员可以在该事件函数内加入 Abort()函数来取消删除操作。当执行 Delete()函数之后，程序会先执行 BeforeDelete()事件函数，如果在 BeforeDelete()事件函数内执行到 Abort()函数时，则取消删除操作，否则将实际地删除数据库记录。实例程序实现此技术的示例步骤如下（以

“实例 190　如何创建不写代码的数据库程序”的实例为基础）：

（1）用鼠标选择 Table1::TTable 组件，在左下角的 Object Inspector 中单击 Event 标签页，并为事件 BeforeDelete 添加处理函数 OnBeforeDelete()。

（2）在函数 OnBeforeDelete()中添加如下所示的处理代码。

```
void __fastcall TForm1::OnBeforeDelete(TDataSet *DataSet)
{
  if(MessageBox(NULL,"是否需要真正删除数据库记录？","信息提示",MB_YESNO)==IDNO)
    Abort();
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 199 如何在删除数据库记录前给予提示”。

实例 200　如何取得数据集的当前状态

实例目的

此实例主要介绍了如何判断数据集的当前操作状态的编程技术。实例程序运行后，单击“+”按钮增加数据库记录，则会弹出一个对话框，提示当前数据集处于插入状态（当数据集处于不同的状态时会显示不同的信息），效果如图 200-1 所示。



图 200-1

实现方法

数据集的当前状态对用户对数据库的操作有着决定性的作用，对数据库的每一步操作都必须根据当前数据集所处的状态而定，因此必须随时知道数据集的状态。TDataSet 类的 State 属性能够描述数据集的当前操作状态，它是一个 TDataState 枚举，该枚举值的意义如表 200-1 所示。

表 200-1

状态标志	中文名称	功能
dsInactive	非活动状态	数据集是关闭的，不能访问数据
dsBrowse	浏览状态	数据集是打开的，只能浏览表中的数据，不能修改或删除记录，默认情况下数据集打开时即处于该状态
dsEdit	编辑状态	可以浏览表中的记录并能修改当前的记录或删除记录
dsInsert	插入状态	可以在表中插入一条记录
dsSetKey	查找状态	在该状态下，可以调用 FindKey、GotoKey、GotoNearest 和 FindNearest 方法来查找表中的记录，此状态只适用于 TTable 组件
dsCalcFields	处理计算字段状态	在该状态下，当数据集组件的 OnCalcFields 事件被执行时，C++Builder 会自动将数据集组件设置为该状态，以防止其他非计算字段的值被修改
dsFilter	过滤状态	正在进行过滤操作，可以查看受限的数据集，但不能编辑和插入记录
dsCurValue	当前值状态	只能内部使用
dsNewValue	新值状态	只能内部使用
dsOldValue	旧值状态	只能内部使用

以下是实例程序获取数据库当前所处状态的具体步骤：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。并为 OnStateChange 事件添加如下所示的实现代码。

```
void __fastcall TForm1::DataSource1StateChange(TObject *Sender)
{ //获取数据集的操作状态
  switch(this->Table1->State)
  {
   case dsInactive:
       MessageBox(Handle,"数据集已经关闭","信息提示",MB_OK);
       break;
   case dsBrowse:
       MessageBox(Handle,"数据集处于浏览状态","信息提示",MB_OK);
       break;
   case dsEdit:
       MessageBox(Handle,"数据集处于编辑状态","信息提示",MB_OK);
       break;
   case dsInsert:
       MessageBox(Handle,"数据集插入一条新记录","信息提示",MB_OK);
       break;
   case dsSetKey:
```

```
            MessageBox(Handle,"数据集被搜索或由 SetRange 限制了范围，只能浏览","信息提
示",MB_OK);
            break;
        case dsCalcFields:
            MessageBox(Handle,"数据集计算字段的OnCalcFields事件正在进行","信息提示",MB_OK);
            break;
        case dsFilter:
            MessageBox(Handle,"数据集 OnFilterRecord 事件正在进行","信息提示",MB_OK);
            break;
        case dsNewValue:
            MessageBox(Handle,"数据集正在访问 TField::NewValue 属性","信息提示",MB_OK);
            break;
        case dsOldValue:
            MessageBox(Handle,"数据集正在访问 TField::OldValue 属性","信息提示",MB_OK);
            break;
        case dsCurValue:
            MessageBox(Handle,"数据集正在访问 TField::CurValue 属性","信息提示",MB_OK);
            break;
        case dsBlockRead:
            MessageBox(Handle,"当指针移动时，数据库没有更新数据控制控件和没有引发事件
","信息提示",MB_OK);
            break;
        case dsInternalCalc:
            MessageBox(Handle,"数据集正处于计算状态","信息提示",MB_OK);
            break;
        case dsOpening:
            MessageBox(Handle,"数据集正处于打开状态（还没有完全打开）","信息提示",MB_OK);
            break;
      }
    }
```

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 200 如何取得数据集的当前状态”。

实例 201 如何创建主从关系的数据集

实例目的

此实例主要介绍了在数据集组件中使用 MasterSource 和 MasterFields 属性创建一对多

关系的数据集的编程技术。实例程序执行后，当选择一个客户时，将显示该客户的所有定单，效果如图 201-1 所示。



图 201-1

实现方法

一对多的数据集就是通常所说的 Master/Detail 关系，C++Builder 的 TTable 组件就自动具有这种一对多的特性，它的 MasterSource 属性用于指定一个 TDataSource 组件，该组件的 DataSet 属性指定了 Master 表，而 MasterFields 属性则指定一个或几个字段，连接 Master 和 Detail 表，如果有多个字段，彼此之间用分号分隔。实例程序创建一对多数据集关系的具体步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table2:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table2 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 orders.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource2:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource2 的 DataSet 属性为 Table2。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid2:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid2 的 DataSource 属性为 DataSource2。

（4）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 customer.db，设置 MasterSource 属性为 DataSource2，MasterFields 属性为 CustNo，Active 属性为 true。

（5）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（6）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（7）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到

Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 201 如何创建主从关系的数据集”。

实例 202 如何在一个数据集中浏览另一个数据集的数据

实例目的

此实例主要介绍了在一个数据集中浏览另一个数据集中的数据的编程技术。当实例程序启动后，在列表中选择一项时，程序就在 Customer 表中查找与 Orders 表的列表框中匹配的记录，如果找到，就立即将数据集游标定位到 Customer 表的该位置。效果如图 202-1 所示。



图 202-1

实现方法

C++Builder 的 TDBLookupListBox 组件能够实现从一个数据集中显示另一个数据集中的数据，而不使用代码编写数据库应用程序。具体步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 custom.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（5）在组件模板的 BDE 标签页下拖动 Table2:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table2 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 orders.db，Active 属性为 true。

（6）在组件模板的 DataAccess 标签页下拖动 DataSource2:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource2 的 DataSet 属性为 Table2。

（7）在组件模板的 DataControls 标签页下拖动 TDBLookupListBox1: TDBLookupListBox 组件到 Form1 上，在 Object Inspector 中设置 TDBLookupListBox1 组件的 DataSource 属性为 DataSource2，设置 TDBLookupListBox1 组件的 DataField 属性为 CustNo，设置 TDBLookupListBox1 组件的 ListSource 属性为 DataSource1，设置 TDBLookupListBox1 组件的 KeyField 属性为 CustNo，该属性主要用来指定 Lookup 表中哪个字段与 DataField 属性指定的字段相匹配，设置 TDBLookupListBox1 组件的 ListField 属性为 Company。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 202 如何在一个数据集中浏览另一个数据集的数据”。

实例 203　如何在数据库更新过程中增加事务操作

实例目的

此实例主要解决数据库更新操作过程中如何处理具有关联性质的事务。实例程序运行后，效果如图 203-1 所示。



图 203-1

实现方法

在对数据库进行更新操作时，常常有这样的情况：一个事件必须要经过若干个过程才

能完成，如果其中一个过程出现问题，那么这个事件就是不完整的，以前完成的部分过程中产生的数据将会成为一堆垃圾。例如：一个排车数据总共会涉及到客户资料、货物清单和车辆情况三部分，假设用户在将客户资料和货物清单增加完成时，突然发生停电或网络中断，导致车辆情况尚未写入数据库；当恢复供电和网络连接正常重新查询排车数据时，用户却发现刚才停电前输入的客户资料和货物清单已经写入数据库，因此也可能错误地认为车辆情况的数据也写入了数据库，而实际上恰恰相反。这就产生了数据的不一致问题。为了克服这一问题，C++Builder 提供了 Database 组件对这种问题进行事务性处理，将所有具有关联数据表的更新操作设计成一个 Transaction 事务，如果在更新数据库操作过程中有不完整的情况发生，数据库系统将会把数据库恢复到操作执行前的状态，这样一来，用户也就不会在停电后重新开机时发现写入数据库的客户资料和货物清单。默认情况下，BDE 提供隐含的事务处理能力，即当用户要把某条记录写到数据库中时，BDE 可以保证不会出现部分字段被更新而另一部分字段没有被更新的情况发生。

基于 BDE 的应用程序要显式地使用事务有两种方式：一是通过 TDatabase 组件连接数据库，然后通过 TDatabase 组件的 StartTransaction、Commit、RollBack、InTransaction 和 TransIsolation 等属性和方法来控制事务。StartTransaction()用来开始一个 Transaction 控制，Commit()将修改过的数据确定写入数据库中，并且结束此 Transaction 控制，Rollback()将修改过的数据恢复到原来未更新前的数据，并且结束此 Transaction 控制。二是通过 SQL 方式，即通过 TQuery、TStoredProc、TUpdateSQL 组件把 SQL 语句直接或间接地传递给远程的 SQL 服务器或 ODBC 服务器。要使用这种方式，必须正确安装 SQL Links 驱动程序配置有关的网络协议，并具有访问远程服务器的权限。

TDatabase 组件控制事务的步骤如下：①首先调用 StartTransaction()方法启动一个事务；②启动一个事务后，如果不是显式地终止事务，那么以后所有的读写操作都和那个事务有关，可以通过 TDatabase 组件的 InTransaction 属性来判断是否已经到了一个事务，如果开始，则对数据的查看取决于 TransIsolation，即事务隔离级别；③调用 TDatabase 组件的 Commit()方法使修改后的数据永久地保存到数据库中，Commit()方法通常在 try...catch 块中进行调用，这样能够保证即使 Commit()方法没有执行成功，也还有机会使用 Rollback()方法使操作回滚。示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  Database1->StartTransaction();
  try
  {
    Table1->Post();
    Database1->Commit();
  }
  catch(...)
  {
    Database1->Rollback();
    MessageBox(NULL,"更新记录出现错误!","信息提示",MB_OK);
  }
```

```
    Table1->CommitUpdates();
}
```

注意：如果希望在程序中自行控制 Transaction 的操作，则需先利用 BDE Administrator 工具，将 Aliases 的 SQLPASSTHRU MODE 参数设置成 SHARED NOAUTOCOMMIT，而不是设置成 SHARED AUTOCOMMIT，如图 203-2 所示。否则当执行 POST()方法时，数据会马上提交并更新数据库，而设定的 Transaction 事务功能就会失去作用。



图 203-2

此外，如果有多个事务同时处理并访问同一张表，这些事务之间的相互影响要由 TransIsolation 属性设置的事务隔离级别决定。常用的 TransIsolation 属性值如表 203-1 所示。

表 203-1

TransIsolation 属性值	功能
tiDirtyRead	允许读取其他事务尚未提交的修改
tiReadCommitted	允许读取其他事务对数据库已经提交的修改
tiRepeatableRead	不允许读取其他事务对数据库的修改，可以保证一旦读取了一个记录，就不会受其他事务的影响

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 203 如何在数据库更新过程中增加事务操作”。

实例 204 如何在程序中实现查找字段

实例目的

此实例主要解决在使用 C++Builder 编写数据库应用程序时，不使用关联查询，而在两个数据库表间实现数据共享。实例程序执行后，效果如图 204-1 所示。



图 204-1

实现方法

在 C++Builder 中开发数据库应用程序时，在两个数据库表之间的数据需要共享时，除了可以通过关联查询的方式实现外，也可以通过查找字段的方式实现。例如：在表 A 中有国名、首都等；在表 B 中有地址、国名等，但没有首都；如果要在显示 B 表时根据国名显示 A 表中对应的首都，这时就可以使用查找字段。在实例程序中添加查找字段的具体步骤如下：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 BDE 标签页下拖动 Table2:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table2 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 customer.db，Active 属性为 true。

（4）在组件模板的 DataAccess 标签页下拖动 DataSource2:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource2 的 DataSet 属性为 Table2。

（5）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource2。

（6）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource2。

（7）鼠标右击 Table2 组件，在弹出的右键菜单中选择 Fields Editor...，弹出 Form1->Table2 对话框。

（8）鼠标右击 Form1->Table2 对话框中的空白区域，在弹出的右键菜单中选择 Add Fields...，弹出 Add Fields 对话框，默认情况下，所有字段处于选择状态，可以选择部分字段。在此保留默认值，单击 OK 按钮。这些字段自动传送到 Form1->Table2 对话框。

（9）鼠标右击 Form1->Table2 对话框中的空白区域，在弹出的右键菜单中选择 New Field...，弹出 New Field 对话框。在 New Field 对话框的 Name 编辑栏中输入字段名，如 Capital；Component 编辑栏保留默认值；在 Type 列表框中选择字段类型，如 String；在 Field type 单选组中选择查找字段 Lookup；在 Key Fields 列表框中选择当前表中的关联字段，如 Country；在 Dataset 列表框中选择目标表，如 Table1；在 Lookup Keys 列表框中选择关联字段名，如 Name(Country)；在 Result Field 列表框中选择查找字段 Capital，其余保留默认值，如图 204-2 所示，单击 OK 按钮即可。如果系统提示在一个打开数据集中不能执行此项操作时，请先设置 Table2 组件的 Active 属性为 false，再执行以上操作；以上操作执行完毕后，再设置 Table2 组件的 Active 属性为 true。

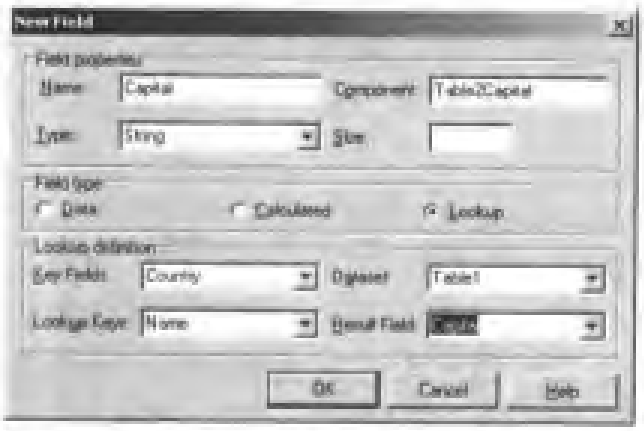


图 204-2

最后编译执行程序即可。注意：一旦选择 Lookup，则 Lookup definition 区将被激活。其中，Dataset 用于指定要查找的数据集，但它不能是 Lookup 字段所在的数据集；Key Fields 用于指定关键字段，如果有多个关键字段，彼此之间要用分号隔开；Lookup Keys 用于指定 Dataset 所指定的数据集中与关键字段匹配的字段，如果有多个字段，彼此之间也要用分号隔开；Result Field 用于指定从 Dataset 所指定的数据集中的哪个字段获取值。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 204 如何在程序中实现查找字段”。

实例 205 如何在程序中实现自定义字段

实例目的

此实例主要介绍了在 C++Builder 中，添加在数据库表字段集中没有的字段，在程序使用时该自定义字段与已经在数据库表字段集中存在的字段的执行效果相当。实例程序执行后，效果如图 205-1 所示。

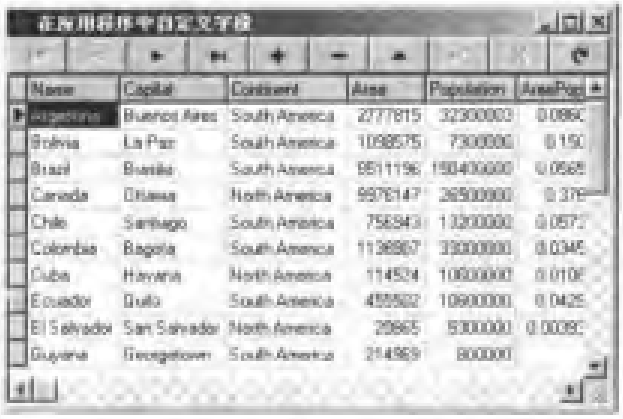


图 205-1

实现方法

在 C++Builder 中开发数据库应用程序时，字段编辑器中的字段类型除了可以是数据库表的字段以外，也可以是程序员在程序中自行定义的字段，因而它极大地增强了数据库设计和程序设计的灵活性。例如：一个国家具有人口和面积两个属性，知道了这两个属性就可以知道人均面积是多少，在数据库设计中完全可以不添加人均面积属性，而把它放在程序设计时添加，这样就减少了数据库的冗余量，同时又增加了程序的灵活性，这就是所谓的计算字段。计算字段的原理是 BDE 每读取一行数据后，它会先执行事件函数 OnCalcFields，然后才将计算结果显示在有关组件上。在程序中添加计算字段的具体步骤如下（"实例 190 如何创建不写代码的数据库应用程序"的实例为基础）：

（1）鼠标右击 Table1 组件，在弹出的右键菜单中选择 Fields Editor...，弹出 Form1->Table1 对话框。

（2）鼠标右击 Form1->Table1 对话框中的空白区域，在弹出的右键菜单中选择 Add Fields...，弹出 Add Fields 对话框，默认情况下，所有字段处于选择状态，如图 205-2 所示，可以选择部分字段。在此保留默认值，单击 OK 按钮，这些字段自动传送到 Form1->Table1 对话框。

（3）鼠标右击 Form1->Table1 对话框中的空白区域，在弹出的右键菜单中选择 New Field...，弹出 New Field 对话框。在 New Field 对话框的 Name 编辑栏中输入字段名，如

AreaPopulation；Component 编辑栏保留默认值，在 Type 列表框中选择字段类型，如 Float，在 Field type 单选组中选择计算字段 Calculated，其余保留默认值，如图 205-3 所示，单击 OK 按钮即可。如果系统提示在一个打开数据集中不能执行此项操作时，请先设置 Table1 组件的 Active 属性为 false，再执行以上操作；以上操作执行完毕后，再设置 Table1 组件的 Active 属性为 true。



图 205-2

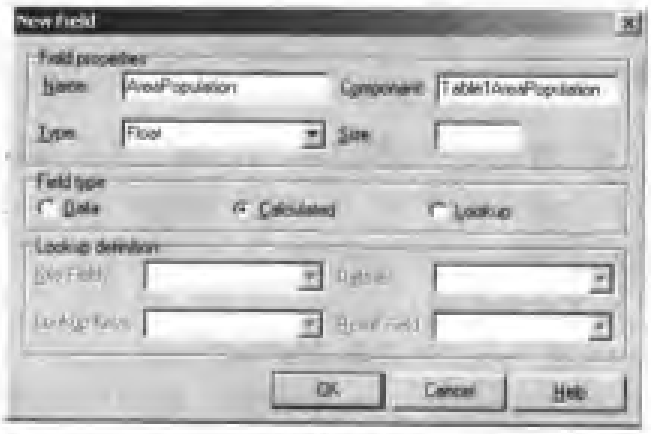


图 205-3

（4）单击 Table1 组件，在左下角的 Object Inspector\Properties 中设置 Table1 组件的 AutoCalcFields 属性为 true。

（5）单击 Table1 组件，在左下角的 Object Inspector\Events 中添加 Table1 组件的 OnCalcFields 事件函数，如下面的代码所示。

```
void __fastcall TForm1::CalcAreaPopulation(TDataSet *DataSet)
{
  this->Table1AreaPopulation->Value=
this->Table1Area->Value/this->Table1Population->Value;
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 205 如何在程序中实现自定义字段”。

实例 206　如何使用字段拖放功能

实例目的

此实例主要介绍了在 C++Builder 中，一次性实现数据库表多个字段在表单中同时创建编辑栏及编辑栏对应标签的技术，在数据库表拥有大量字段的情况下，该编程技巧具有非常明显的效率优势。实例程序执行后，效果如图 206-1 所示。



图 206-1

实现方法

在开发数据库应用程序时，数据录入界面通常以编辑栏的形式出现。C++Builder 为此提供了一个非常优秀的功能，即字段拖放。它可以一次性实现多个字段在 Form 中同时创建编辑栏及编辑栏对应的标签。具体步骤如下（以“实例 190 如何创建不写代码的数据库程序”的实例为基础）：

（1）在 Form1 上删除 DBGrid1 组件。

（2）鼠标右击 Table1 组件，在弹出的右键菜单中选择 Fields Editor...，弹出 Form1->Table1 对话框。

（3）鼠标右击 Form1->Table1 对话框中的空白区域，在弹出的右键菜单中选择 Add Fields...，弹出 Add Fields 对话框，默认情况下，所有字段处于选择状态，如图 206-2 所示，可以选择部分字段。在此保留默认值，单击 OK 按钮。这些字段自动传送到 Form1->Table1 对话框。



图 206-2

（4）在 Form1->Table1 对话框中选择所有字段，拖动所有的字段到 Form1。可以发现，Form1 上自动添加了这些字段的编辑栏及标签，标签即字段名，编辑栏的长度就是字段的长度，如图 206-3 所示。默认情况下，这些编辑栏与数据库表字段已经建立了一一对应关系，剩下的工作就是把这些编辑栏和标签排列整齐。因此，它使得程序员不必一个个地拖动 DBEdit 和 Label 组件，再一个个设定这些组件与数据库表字段的连接关系，大大降低了程序设计时的工作量。

最后编译执行程序即可。

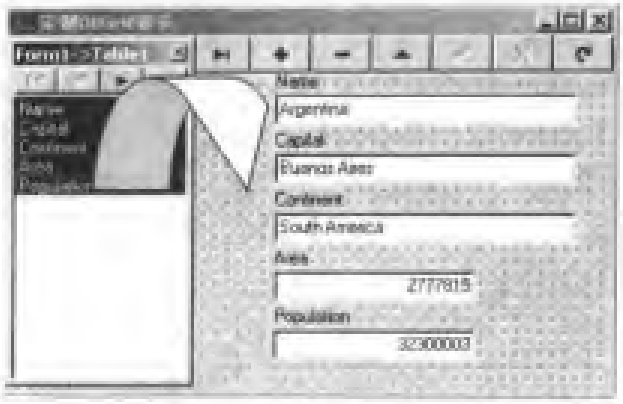


图 206-3

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 206 如何使用字段拖放功能”。

实例 207　如何为程序添加报表打印功能

实例目的

此实例主要介绍了在 C++Builder 中制作打印报表的基本技术。实例程序执行后，效果如图 207-1 所示。



图 207-1

实现方法

C++Builder 的 VCL 里面提供了一组处理报表打印的组件 QuickReport，这组组件可以

以可视化的方式来设计所需要的打印报表，它主要是根据所谓的“带状”设计原理实现。首先，拖动一个名为 QuickReport 的组件到 Form 上，此时的 Form 就会变成报表模式，然后利用 QRBand 组件将 Form 切成一条条的带（Band）状区域，并且分别设定每条 QRBand 组件的状态（BandType 属性），如报表的标题、表头、表体、表尾等，最后将各种类型的组件放在不同状态的 Band 上，即完成一份报表的设计。

常用数据报表组件的功能如表 207-1 所示。

表 207-1

组件名称	主要功能
QuickRep	报表控制器，是产生报表最基本的组件
QRSubDetail	用来增加数据库到数据报表中
QRStringBand	用来使单元格的内容由字符串填充，而不是通常的数据库中的数据填充
QRBand	用来建立可打印区域
QRChildBand	用来建立可打印的子区域
QRGroup	用来建立分组可打印区域
QRLabel	用来给报表建立标签
QRDBText	用来在报表上打印数据库中字段的内容，包括 MEMO 字段
QRExpr	用来打印以计算表达式为主的数据信息
QRSysData	用来显示一些系统数据，如日期、时间和页码等
QRMemo	用来显示多行文本
QRExprMemo	综合了 QRExpr 和 QRMemo 组件的功能
QRRichText	用来在报表上显示 Rich 格式的文字信息
QRDBRichText	用来在报表上显示数据库字符中的 Rich 格式的文字信息
QRShape	用来在报表上打印一些简单的几何图形
QRImage	用来在报表上显示图像
QRDBImage	用来在报表上显示数据库字段中的图像
QRCompositeReport	用来综合多个报表
QRPreview	用来在程序运行时预览报表的打印效果
QRTextFilter	用来把报表按文本方式输出到文件
QRCSVFilter	用来把报表按文本方式输出到文件，数据与数据之间用逗号分隔
QRHTMLFilter	用来把报表按 HTML 文件格式输出到文件
QRChart	用来在报表上绘制统计图形

实例程序制作报表的具体操作步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 customer.db，Active 属性为 true。

（2）在组件模板的 QReport 标签页下拖动 QuickRep1 组件到 Form1 上，在 Object Inspector 中设置 DataSet 属性为 Table1。

（3）在组件模板的 QReport 标签页下拖动 4 个 QRBand 组件到 QuickRep1 上，在 Object Inspector 中分别设置 BandType 属性为 rbTitle、rbColumnHeading、rbDetail、rbPageFooter。

（4）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand1 上，在 Object Inspector 中设置 Caption 属性为“客户基本数据表”。

（5）在组件模板的 QReport 标签页下拖动 5 个 QRLabel 组件到 QRBand2 上，在 Object Inspector 中分别设置 Caption 属性为“客户编号”、“公司名称”、“公司地址”、“联系电话”、“联系人”。

（6）在组件模板的 QReport 标签页下拖动 5 个 QRDBText 组件到 QRBand3 上，在 Object Inspector 中设置 5 个组件的 DataSet 属性为 Tabel1，分别设置 5 个组件的 DataField 属性为 CustNo、Company、Add1、Phone、Contact。

（7）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand4 上，在 Object Inspector 中分别设置 Caption 属性为“页面：”。

（8）在组件模板的 QReport 标签页下拖动 QRSysData 组件到 QRBand4 上，在 Object Inspector 中分别设置 Data 属性为 qrsPageNumber。

（9）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand4 上，在 Object Inspector 中分别设置 Caption 属性为“打印日期：”。

（10）在组件模板的 QReport 标签页下拖动 QRSysData 组件到 QRBand4 上，在 Object Inspector 中分别设置 Data 属性为 qrsDate，如图 207-2 所示。



图 207-2

（11）在主表单的构造函数中加入如下所示的代码。

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
   this->QuickRep1->Preview();
//   this->QuickRep1->Print();
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据

库应用\实例 207 如何为程序添加报表打印功能”。

实例 208 如何使用向导创建打印报表

实例目的

此实例主要介绍了如何在 C++Builder 中使用向导创建打印报表的编程技术。实例程序执行后，效果如图 208-1 所示。

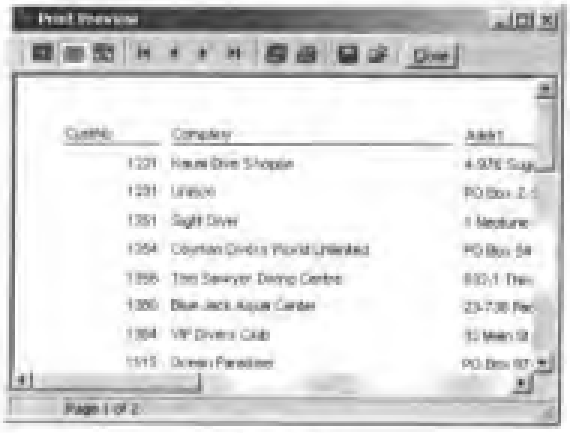


图 208-1

实现方法

C++Builder 的 QuickReport 源自 QuSoft AS 公司，具有强大的报表设计功能，主要表现在：①可视化方式的报表设计，能高精度地定位表格，并能在设计时预览报表的设计效果；②利用事件驱动报表，可以形成支持任何数据的报表；③可以在报表中输出数据值、图像、图标和图形；④兼容 BDE 支持的数据库，包括 ODBC；⑤可以产生超长度数据报表，支持数据库中不定长的备注字段和 Richtext 格式的备注字段；⑥可以输出 ASCII 文本格式和 HTML 格式的文件。除了手工创建打印报表以外，还可以使用 C++Builder 自带的报表向导 Quick Report Wizard 来创建报表。具体步骤如下：

（1）选择菜单 File\New\Other...，弹出 New Items 对话框中，单击 Business 标签页，选择 QuickReportWizard 对象，选择 Copy 单选按钮，如图 208-2 所示，单击 OK 按钮，弹出 New Report Wizard 对话框。

（2）在 New Report Wizard 对话框中，单击 Start Wizard 按钮，如图 208-3 所示，进入数据库及数据字段选择对话框。

（3）在数据库及数据字段选择对话框中，Alias or directory 用于指定数据库的别名或所在的子目录；Table name 用于指定数据库表；Selected fields 用于指定需要显示的字段。实例选择情况如图 208-4 所示。单击 Finish 按钮完成报表的创建。



图 208-2

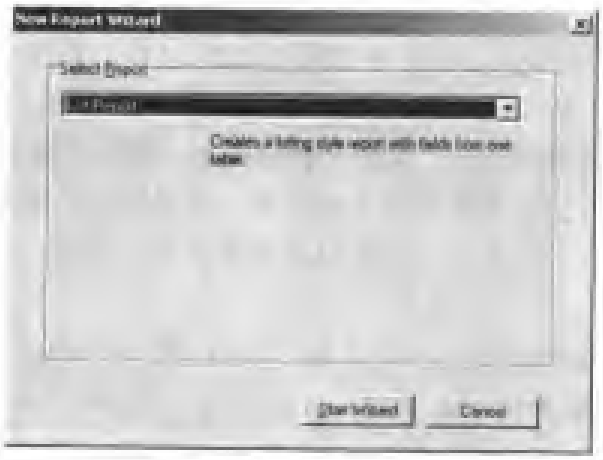


图 208-3

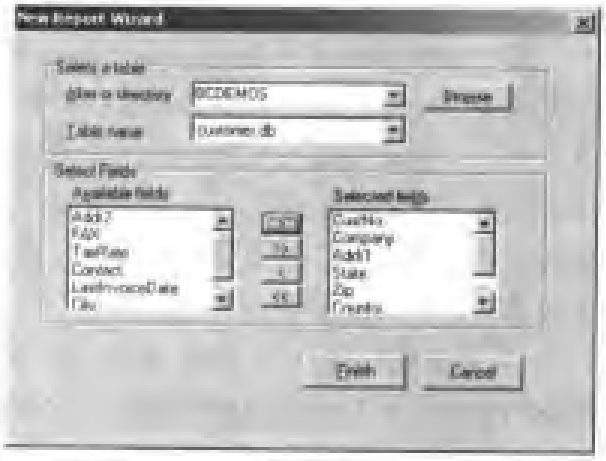


图 208-4

（4）选中 QuickRep 组件，单击鼠标右键，从弹出的右键菜单中选择 Preview，就可以得到如图 208-1 所示的预览效果。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 208 如何使用向导创建打印报表”。

实例 209　如何设置一对多数据库表

实例目的

此实例主要解决在数据库表间关系处于一对多的情况下，如何将这种关系关联起来，并通过程序反映出来。实例程序执行后，当在客户表中单击任一记录时，订单表立即显示

该客户的所有订单记录，效果如图 209-1 所示。



图 209-1

实现方法

所谓数据库表关系的一对多就是指某个数据库表内的某一记录可能包含另一个数据库表的多行记录。最典型的就是客户数据和订单数据之间的关系：一个客户可能有多个订单，而一个订单则只有一个客户。如果在数据库表间建立了这种一对多关系，则非常容易保持数据库中各种数据的一致性，例如，如果要删除某个客户，则与该客户有关的所有订单就应该删除，这样才不至于在数据库内造成许多垃圾数据。实例程序设置数据库表一对多关系的具体步骤如下：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 customer.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 BDE 标签页下拖动 Table2:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table2 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 orders.db，Active 属性为 true。

（4）在组件模板的 DataAccess 标签页下拖动 DataSource2:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource2 的 DataSet 属性为 Table2。

（5）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（6）在组件模板的 DataControls 标签页下拖动 DBGrid2:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid2 的 DataSource 属性为 DataSource2。

（7）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（8）选择 Table2 组件，在左下角的 Object Inspector 中设置 Table2 的 MasterSource 属性为 DataSource1，再设定 MasterFields 属性为 CustNo（因为 CustNo 是两个表的关联字段），设

置时可以参考图 209-2，即 Table1 是主表，Table2 是从表。

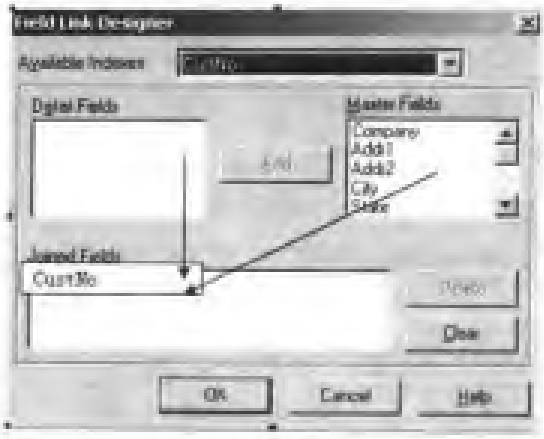


图 209-2

（9）参考步骤（1）、（3），将 Table1 和 Table2 两个组件的 Active 属性设置成 true。最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 209 如何设置一对多数据库表”。

实例 210　如何设置一对多报表打印功能

实例目的

此实例主要介绍了在 C++Builder 中打印一对多报表的基本技术。实例程序执行后，效果如图 210-1 所示。

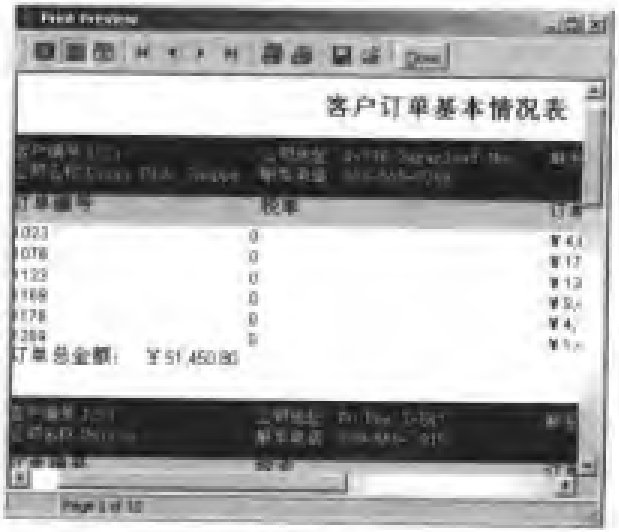


图 210-1

实现方法

在日常生活中，电信局的话费清单是一个典型的一对多的报表样例。C++Builder 的 QuickReport 中提供了一个 QRSubDetail 组件，完全支持具有主/从关系的一对多报表的打印。具体操作步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 customer.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 BDE 标签页下拖动 Table2:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table2 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 orders.db，MasterSource 属性为 DataSource，MasterField 属性为 CustNo，Active 属性为 true。

（4）在组件模板的 DataAccess 标签页下拖动 DataSource2:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource2 的 DataSet 属性为 Table2。

（5）在组件模板的 QReport 标签页下拖动 QuickRep1 组件到 Form1 上，在 Object Inspector 中设置 DataSet 属性为 Table1。因为 MasterBand 用于放置客户数据，所以必须把 QuickRep 指向客户数据表。

（6）在组件模板的 QReport 标签页下拖动 5 个 QRBand 组件到 QuickRep1 上，在 Object Inspector 中分别设置 BandType 属性为 rbPageHeader、rbDetail、rbSubDetail、rbGroupFooter、rbPageFooter。

（7）在组件模板的 QReport 标签页下拖动 QRSubDetail 组件到 QuickRep1 上，在 Object Inspector 中设置 HeaderBand 属性为 QRBand3，设置 FootBand 属性为 QRBand4，设置 DataSet 属性为 Table2，设置 Master 属性为 QuickRep1。

（8）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand1 上，在 Object Inspector 中设置 Caption 属性为“客户订单基本情况表”。

（9）在组件模板的 QReport 标签页下拖动五个 QRLabel 组件到 QRBand2 上，在 Object Inspector 中分别设置 Caption 属性为“客户编号”、“公司名称”、“公司地址”、“联系电话”、“联系人”。

（10）在组件模板的 QReport 标签页下拖动 5 个 QRDBText 组件到 QRBand2 上，在 Object Inspector 中设置所有 5 个组件的 DataSet 属性为 Tabel1，分别设置 5 个组件的 DataField 属性为 CustNo、Company、Add1、Phone、Contact。

（11）在组件模板的 QReport 标签页下拖动 3 个 QRLabel 组件到 QRBand3 上，在 Object Inspector 中分别设置 Caption 属性为“订单编号”、“税率”、“订单金额”。

（12）在组件模板的 QReport 标签页下拖动 3 个 QRDBText 组件到 QRSubDetail 上，在 Object Inspector 中设置所有 3 个组件的 DataSet 属性为 Tabel2，分别设置五个组件的

DataField 属性为 OrderNo、TaxRate、AmountPaid。

（13）在组件模板的 QReport 标签页下拖动 3 个 QRLabel 组件到 QRBand4 上，在 Object Inspector 中设置 Caption 属性为“订单总金额”。

（14）在组件模板的 BDE 标签页下拖动 Query1:TQuery 组件到 Form1 上，在 Object Inspector 中设置 Query1 的 DatabaseName 属性为 BCDEMOS，DataSource 属性为 DataSource1，SQL 属性为 Select Sum(AmountPaid) From Orders Where CustNo=:CustNo，Active 属性为 true。

（15）在组件模板的 QReport 标签页下拖动 QRDBText 组件到 QRBand4 上，在 Object Inspector 中设置 DataSet 属性为 Query1，设置 DataField 属性为 SUM OF AmountPaid。

（16）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand5 上，在 Object Inspector 中分别设置 Caption 属性为“页面：”。

（17）在组件模板的 QReport 标签页下拖动 QRSysData 组件到 QRBand5 上，在 Object Inspector 中分别设置 Data 属性为 qrsPageNumber。

（18）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand5 上，在 Object Inspector 中分别设置 Caption 属性为“打印日期：”。

（19）在组件模板的 QReport 标签页下拖动 QRSysData 组件到 QRBand5 上，在 Object Inspector 中分别设置 Data 属性为 qrsDate，如图 210-2 所示。



图 210-2

（20）在主表单的构造函数中加入如下所示的代码。

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  this->QuickRep1->Preview();
//   this->QuickRep1->Print();
}
```

最后编译执行程序即可。注意：TQRBand 组件可以在 TQuickRep 组件上形成多种不同功能的表格元，表格元的功能由 TQRBand 组件的 BandType 属性来设置，常用的 BandType 属性的取值及意义如表 210-1 所示。

表 210-1

属性取值	意义
rbTitle	标题表格元，打印在表格的第一页
rbSummary	总结性表格元，打印在 rbDetail 表格之后
rbDetail	数据表格元，在报表上打印数据库中的每一条记录
rbPageHeader	页眉表格元，打印在每页的开始
rbPageFooter	页脚表格元，打印在每页的结尾
rbGroupHeader	组页眉表格元，也能作为 TQRSubDetail 的 Header 表格元
rbGroupFooter	组页脚表格元，适用于 TQRGroup 和 TQRSubDetail
rbSubDetail	子数据表格元，用户不需要手工设置该值
rbColumnHeader	表头表格元，打印在每栏开始位置
rbChild	子表格元，用户不需要手工设置该值

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 210 如何设置一对多报表打印功能”。

实例 211 如何根据一对多数据库表绘制图形

实例目的

此实例主要解决在数据库的表间关系处于一对多的情况下，如何将这种关系关联起来，并通过图形的形式根据主表的变动情况显示从表跟随的变动情况。实例程序执行后，当在客户表中单击任一记录时，订单表立即显示该客户的所有订单记录，同时图形变化同步更新，效果如图 211-1 所示。

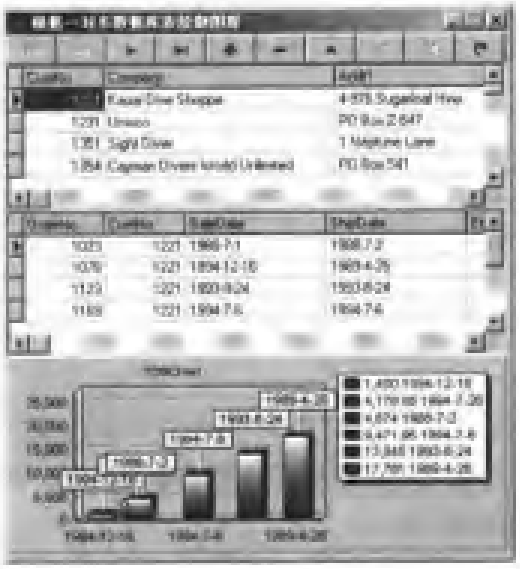


图 211-1

实现方法

C++Builder 的 TDBChart 组件能够通过 DataSource 属性获得数据并显示，如果数据源是一个 TTable、TQuery、TClientDataSet 或其他有效的数据集组件，TDBChart 组件将在符合过滤条件和范围内自动获得记录。实例程序设置数据库表一对多关系并以图形方式显示从表动态变化的具体步骤如下：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 customer.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 BDE 标签页下拖动 Table2:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table2 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 orders.db，Active 属性为 true。

（4）在组件模板的 DataAccess 标签页下拖动 DataSource2:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource2 的 DataSet 属性为 Table2。

（5）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（6）在组件模板的 DataControls 标签页下拖动 DBGrid2:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid2 的 DataSource 属性为 DataSource2。

（7）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（8）选择 Table2 组件，在左下角的 Object Inspector 中设置 Table2 的 MasterSource 属性为 DataSource1，再设定 MasterFields 属性为 CustNo（因为 CustNo 是两个表的关联字段）。设置时可以参考图 211-2，即 Table1 是主表，Table2 是从表。

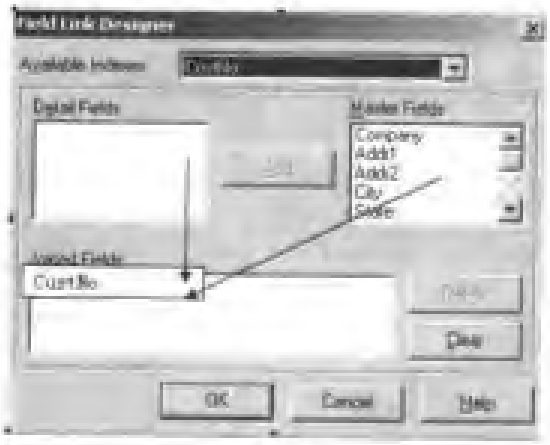


图 211-2

（9）参考步骤（1）、（3），将 Table1 和 Table2 两个组件的 Active 属性设置成 true。

（10）在组件模板的 DataControls 标签页下拖动 DBChart1:TDBChart 组件到 Form1 上，然后双击之，弹出 Editing DBChart1 对话框，单击该对话框上的按钮 Add...，弹出 TeeChart Gallery 对话框，如图 211-3 所示，双击 Bar 图形，返回 Editing Chart1 对话框。

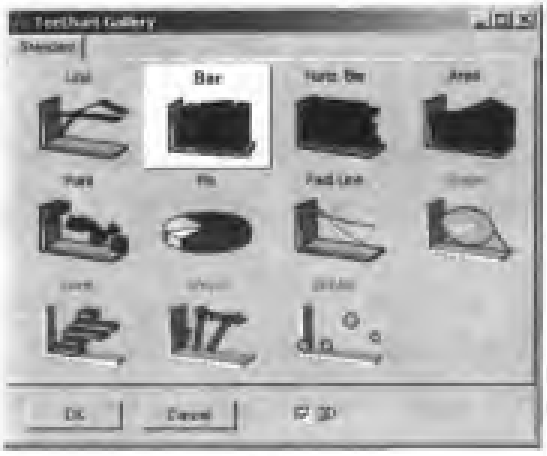


图 211-3

（11）单击 Editing Chart1 对话框的标签页 Series，再单击该标签页下的子标签页 DataSource，设置显示饼图的数据集 Dataset 为 Table2，Label 为 ShipDate，Bar 为 AmountPaid，如图 211-4 所示。单击 Close 按钮关闭组件属性对话框。

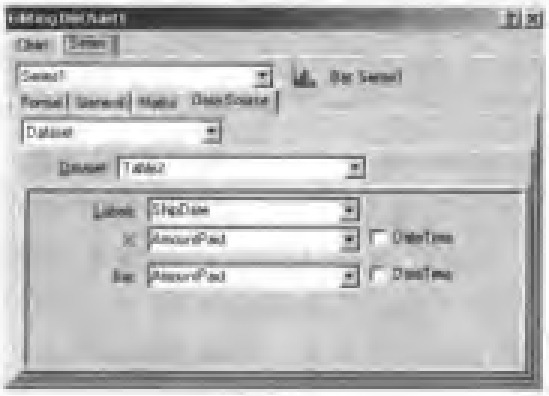


图 211-4

（12）单击 DataSource2 组件，在 Object Inspector\Events 中为 OnDataChange 事件添加事件处理函数，代码如下所示：

```
void __fastcall TForm1::RefreshData(TObject *Sender, TField *Field)
{
  this->DBChart1->RefreshData();
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 211 如何根据一对多数据库表绘制图形”。

实例 212　如何根据数据库表显示饼图

实例目的

此实例主要介绍了根据数据库表直接显示饼图的编程技术。实例程序执行后，效果如图 212-1 所示。

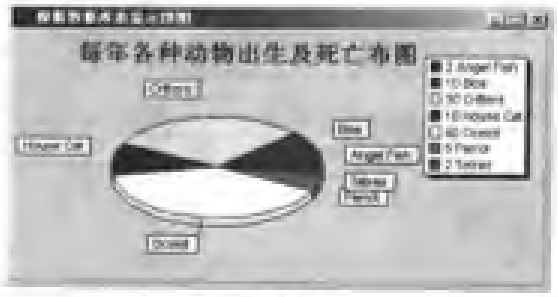


图 212-1

实现方法

C++Builder 提供的 TDBChart 组件能够直接根据数据库表绘制多种风格的图形，它可以直接在该组件的属性编辑器内指定有哪些数据库表的哪些列连接上该组件，完全省略数据读取部分的代码；并且如果直接修改数据库表内的数据时，该组件能够同步更新数据。实例程序实现此技术的示例步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 animals.dbf，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBChart1:TDBChart 组件到 Form1 上，然后双击之，弹出 Editing DBChart1 对话框，单击该对话框上的 Add...按钮，弹出 TeeChart Gallery 对话框，如图 212-2 所示，双击 Pie 图形，返回 Editing Chart1 对话框。



图 212-2

（4）单击 Editing Chart1 对话框的标签页 Series，再单击该标签页下的子标签页 DataSource，设置显示饼图的数据集 Dataset 为 Table1，Label（饼图的显示标签）为 Name，Pie（饼图切分的数据）为 Size，如图 212-3 所示。单击 Close 按钮关闭组件属性对话框。

最后编译执行程序即可。



图 212-3

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 212 如何根据数据库表显示饼图”。

实例 213 如何根据数据库表绘制条形图

实例目的

此实例主要介绍了根据数据库表绘制条形图的编程技术。实例程序执行后，效果如图 213-1 所示。



图 213-1

实现方法

C++Builder 提供了丰富的图表制作组件，这些组件可以在设计时直接设定对应到数据库表的某些数据的具体位置，动态地把数据绘制成统计图表。实例程序实现根据数据库表绘制条形图的示例步骤和代码如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 Additional 标签页下拖动 Chart1:TChart 组件到 Form1 上，然后双击之，弹出 Editing Chart1 对话框，单击该对话框上的按钮 Add...，弹出 TeeChart Gallery 对话框，该对话框下一共有 11 个不同样式的图形，分别是 TLineSeries、TAreaSeries、TPointSeries、TBarSeries、THorizBarSeries、TPieSeries、TchartShape、TFastLineSeries、ArrowSeries、GanttSeries 和 BubbleSeries，并且这些图形均有是否为 3D 图形之分，如图 213-2 所示。双击 Bar(TBarSeries)图形，返回 Editing Chart1 对话框；再单击 Close 按钮，关闭该对话框。

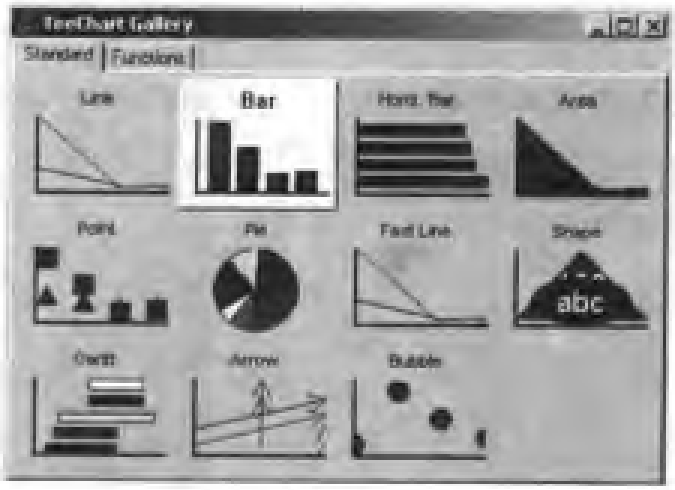


图 213-2

（4）在组件模板的 Additional 标签页下拖动 BitBtn1:TBitBtn 组件到 Form1 上，设置其为绘制图形按钮，再按照同样的方法添加打印图形按钮，然后为这两个按钮添加单击事件函数，其实现代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//绘制条形图
  this->Chart1->Title->Text->Strings[0]="国家面积大小对比图";
  this->Chart1->LeftAxis->Title->Caption="国家面积";
  this->Chart1->BottomAxis->Title->Caption="国家名称";
  for(int i=0;i<Table1->RecordCount;i++)
  {
```

```
        this->Series1->Add(this->Table1->FieldValues['Area'],
    this->Table1->FieldValues['Name'],clBlue);
        this->Table1->Next();
      }
    }
    void __fastcall TForm1::BitBtn2Click(TObject *Sender)
    {//打印条形图
      this->Chart1->Print();
    }
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 213 如何根据数据库表绘制条形图”。

实例 214 如何动态绘制数据库图表

实例目的

此实例主要介绍了动态绘制数据库数据的曲线图的编程技术。实例程序执行后，效果如图 214-1 所示。

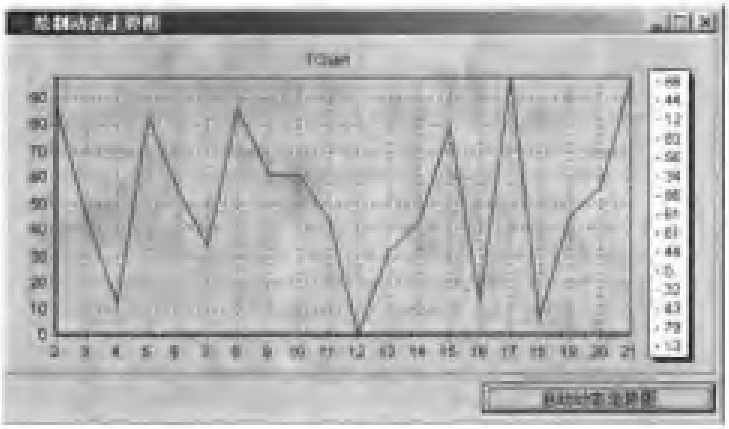


图 214-1

实现方法

利用 C++Builder 提供的 TChart 图形绘制组件，可以把动态产生的数据适时地绘制成统计曲线图表。示例步骤和代码如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 Additional 标签页下拖动 Chart1:TChart 组件到 Form1 上，然后双击之，弹出 Editing Chart1 对话框，单击该对话框上的 Add...按钮，弹出 TeeChart Gallery 对话框，如图 214-2 所示，双击 Line(TLineSeries)图形，返回 Editing Chart1 对话框；再单击 Close 按钮，关闭该对话框。

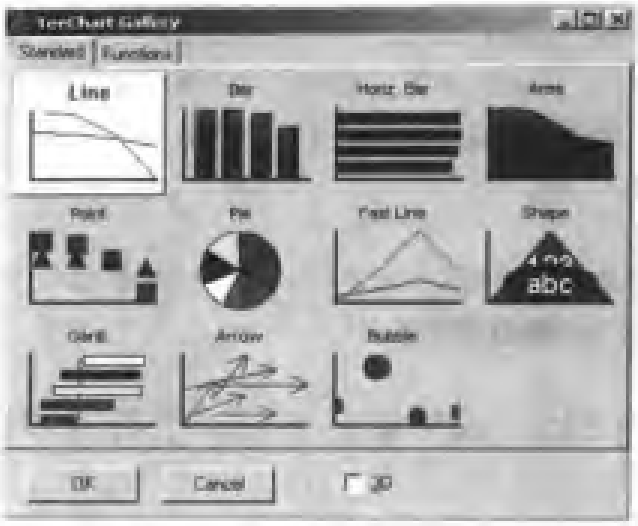


图 214-2

（2）在组件模板的 System 标签页下拖动 Timer1:TTimer 组件到 Form1 上，在 Object Inspector\Events 中设置 Timer1 的 OnTimer 事件处理函数为 PaintChart()，该函数的实现代码如下：

```
void __fastcall TForm1::PaintChart(TObject *Sender)
{
  for(int i=0;i<(this->Chart1->SeriesCount());i++)
  {
double TempD=this->Chart1->Series[i]->XValues->Value[1]-
this->Chart1->Series[i]->XValues->Value[0];
    this->Chart1->Series[i]->Delete(0);

Chart1->Series[i]->AddXY(this->Chart1->Series[i]->XValues->Last()+TempD,
    this->Chart1->Series[i]->YValues->Last()+random(100)-50,"",clBlue);
  }
}
```

（3）在组件模板的 Additional 标签页下拖动 BitBtn1:TBitBtn 组件到 Form1 上，在 Object Inspector\Events 中设置 BitBtn1 的 OnClick 事件处理函数，该函数的实现代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   this->Timer1->Enabled=true;
}
```

（4）按照如下所示的代码修改主表单的构造函数。

```
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner)
{
  this->Timer1->Interval=1000;
  this->Timer1->Enabled=false;
  for(int i=0;i<20;i++)
  {
   this->Series1->AddY(random(100)*1.0,"",clBlue);
  }
}
```

最后编译执行程序即可。可以发现，程序实际上是利用定时器定时产生的随机数而产生了动态曲线图，实际上，股价走势图也是根据这一原理制作的，它每隔一定的时间去读取数据库采集的股价，再根据数据库表进行股价曲线图绘制。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 214 如何动态绘制数据库图表”。

实例 215 如何设置多重群组的打印报表

实例目的

此实例主要介绍了在 C++Builder 中制作多重群组打印报表的编程技术。实例程序执行后，效果如图 215-1 所示。



图 215-1

实现方法

如果希望制作一份每位客户拥有的所有订单数据，其中必须包含 4 个表，分别是客户、订单表头、订单项目和订单明细 4 个数据库表，这就是所谓的多重群组报表形式。QuickReport 的 QRGroup 和 QRExpr 组件为实现多重群组报表打印提供了强有力的支持，QRGroup 组件可以用来建立分组可打印区域；QRExpr 组件可以用来打印以计算表达式为主的数据信息。实例程序制作多重群组报表的具体操作步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）。

（1）在组件模板的 BDE 标签页下拖动 Query1:TQuery 组件到 Form1 上，在 Object

Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，SQL 属性的字符串为：

```
Select * from customer a,orders b,item c,parts d
Where a.custno=b.custno
And b.orderno=c.orderno
And c.partno=d.partno
Order by a.company,orderno
```

（2）在组件模板的 QReport 标签页下拖动 QuickRep1 组件到 Form1 上，在 Object Inspector 中设置 DataSet 属性为 Query1。

（3）在组件模板的 QReport 标签页下拖动 QRBand 组件到 QuickRep1 上，在 Object Inspector 中设置 BandType 属性为 rbTitle。

（4）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand1 上，在 Object Inspector 中设置 Caption 属性为“多重群组报表制作范例”。

（5）在组件模板的 QReport 标签页下拖动 QRGroup 组件到 QuickRep1 上，在 Object Inspector 中设置 Expression 属性为 Company，设置 Master 属性为 QuickRep1。

（6）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRGroup1 上，在 Object Inspector 中设置 Caption 属性为“公司名称：”。

（7）在组件模板的 QReport 标签页下拖动 QRDBText 组件到 QRGroup1 上，在 Object Inspector 中设置 DataSet 属性为 Query1，设置 DataField 属性为 Company。

（8）在组件模板的 QReport 标签页下拖动 QRGroup 组件到 QuickRep1 上，在 Object Inspector 中设置 Expression 属性为 OrderNo，设置 Master 属性为 QuickRep1。

（9）在组件模板的 QReport 标签页下拖动两个 QRExpr 组件到 QRGroup2 上，在 Object Inspector 中分别是其设置 Expression 属性为“'订单编号：'+str(Query1.OrderNo)”和“'销售日期：'+Query1.SaleDate”。

（10）在组件模板的 QReport 标签页下拖动 3 个 QRLabel 组件到 QRGroup2 上，在 Object Inspector 中分别设置 Caption 属性为“产品代码”、“产品详细情况”、“定价”。

（11）在组件模板的 QReport 标签页下拖动 QRBand 组件到 QuickRep1 上，在 Object Inspector 中设置 BandType 属性为 rbDetail。

（12）在组件模板的 QReport 标签页下拖动 3 个 QRDBText 组件到 QRBand3 上，在 Object Inspector 中设置所有 3 个组件的 DataSet 属性为 Query1，分别设置 3 个组件的 DataField 属性为 PartNo、Description、ListPrice，如图 215-2 所示。



图 215-2

（13）在主表单的构造函数中加入如下所示的代码。

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
   this->QuickRep1->Preview();
//   this->QuickRep1->Print();
}
```

最后编译执行程序即可。注意：利用 TQRExpr 组件的 Expression 属性可以输入计算的表达式，Master 属性可以输入统计函数的表达式。TQRExpr 组件支持的函数的意义如表 215-1 所示。

表 215-1

函数	意义
Copy(<x><St><Len>)	返回字符串的子串
Average(<x>)	计算平均值
Count	返回总个数
DIV(<x>,<y>)	返回两数相除的余数
Max(<x>)	求 X 中的最大值
Min(<x>)	求 X 中的最小值
Pretty(<x>)	使字符串的第一个字母大写
Sqrt(<x>)	返回 X 的平方根
Str(<x>)	把数值转化为字符串
Sum(<x>)	计算总和
Upper(<x>)	把 X 中的字符转化为大写
Lower(<x>)	把 X 中的字符转化为小写
Int(<x>)	返回 X 的整数部分
Frac(<x>)	返回 X 的小数部分
Date	按字符串形式返回当前日期
Time	按字符串形式返回当前时间
TRUE	逻辑真
FALSE	逻辑假

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 215 如何设置多重群组的打印报表”。

实例 216 如何编写商业决策分析程序

实例目的

此实例主要介绍了如何使用 C++Builder 的商业决策分析组件，编写商业决策分析程序

的编程技术。实例程序执行后的效果如图 216-1 所示。



图 216-1

实现方法

C++Builder 共提供了 6 个组件来开发商业决策分析应用程序，分别是：DecisionCube（存放决策分析的多维数据）、DecisionQuery（存放决策分析使用的 SQL 命令）、DecisionSource（决策分析数据源）、DecisionPivot（根据数据源产生决策分析维度按钮）、DecivisionGrid（显示数据源的数据）和 DecisionGraph（根据数据源绘制图表）。实例程序使用以上 6 个组件编写商业决策应用程序的步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）。

（1）在组件模板的 Decision Cube 标签页下拖动 DecisionCube、DecisionQuery、DecisionSource、DecisionPivot、DecivisionGrid 和 DecisionGraph 组件到 Form1 上，在 Object Inspector 中设置 DecisionCube1 的 DataSet 属性为 DecisionQuery1，设置 DecisionSource1 的 DecisionCube 属性为 DecisionCube1，设置 DecisionPivot1 的 DecisionSource 属性为 DecisionSource1，设置 DecisionGrid1 的 DecisionSource 属性为 DecisionSource1，设置 DecisionGraph1 的 DecisionSource 属性为 DecisionSource1。

（2）双击 DecisionQuery1 组件，打开 Decision Query Editor 对话框，在 Database 列表框中选择 BCDEMOS，在 Table 列表框中选择 Orders.db；此时 List of Available Fields:窗口中会出现 Orders.db 表中的所有字段；再将 List of Available Fields 字段列表中的 ShipVIA、PaymentMethod、Terms 和 ShipDate 字段添加到 Dimensions 列表框中；将 List of Available Fields 字段列表中的 AmountPaid 字段以 SUM 方式添加到 Summaries 列表框中，将 List of Available Fields 字段列表中的 AmountPaid 字段以 COUNT 方式添加到 Summaries 列表框中，如图 216-2 所示。以上过程产生的 SQL 语句可以在 SQL Query 标签页查看，如图 216-3 所示。

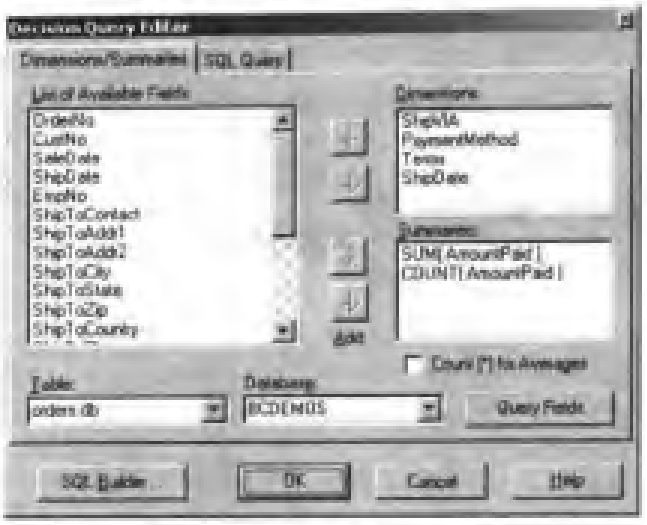


图 216-2

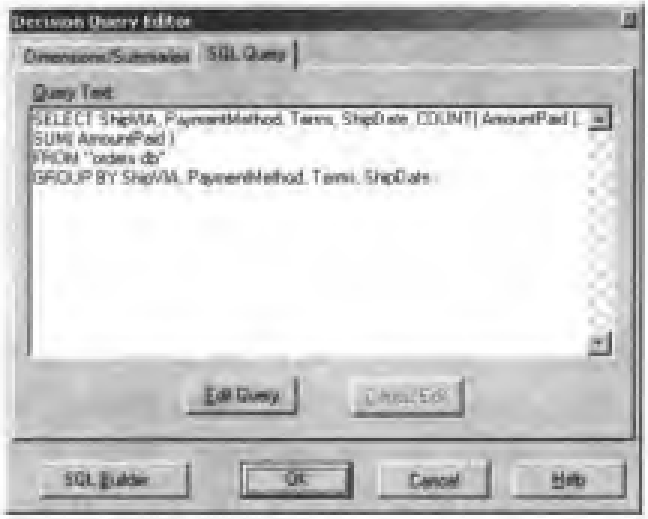


图 216-3

（3）双击 DecisionCube1 组件，弹出 Decision Cube Editor 对话框，可以在此将有关标题改写成中文，如图 216-4 所示。

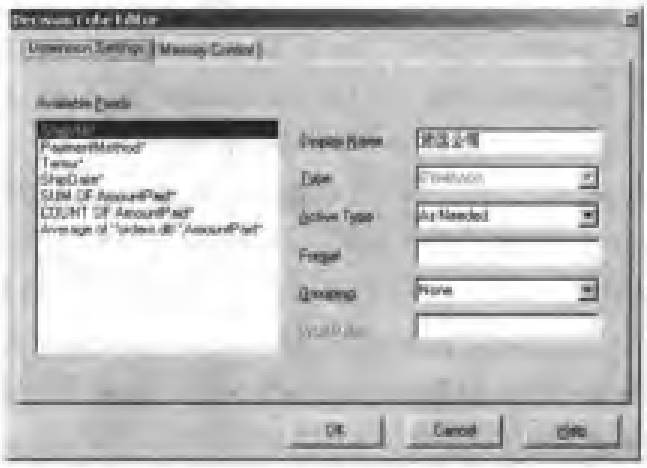


图 216-4

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 216 如何编写商业决策分析程序”。

实例 217　如何导出对象库中的商业决策分析程序

实例目的

此实例主要介绍了如何从对象库中导出商业决策分析程序样例的编程技术。实例程序执行后，效果如图 217-1 所示。该样例程序的主要功能是分析公司年度出货的货运成本，包括 Payment、Terms、ShipVIA、ShipDate、SUM OF AmountPaid 等按钮，其主要功能就是用来确定分析维度。例如：单击 ShipVIA 按钮可以观察公司分别付给货运公司多少钱；再单击 Payment 按钮时则详细列出付给每家货运公司的金额；再单击 Terms 按钮则详细列出付给每家货运公司的金额的支付方式等，以此类推。经由不同的分析维度将可以得到不同的分析报表，这种动态分析方式使得决策者能够得到更多、更详细的分析报表，有助于决策者进行决策。



图 217-1

实现方法

所谓商业决策分析，就是利用大量的旧有数据，依据一定的商业规则，通过交叉分析方法产生的结果作为商业决策的参考。传统意义上，如果以程序设计的方式来解决商业决策分析的需求，则极有可能一种分析需求必须写一个应用程序，而当分析需求发生变化时，则可能重新修改程序或另写一个程序。在 C++Builder 的对象库中，有一个进行商业决策分析的经典案例，演示了如何使用 C++Builder 商业决策分析组件的方法。下面是导出这个经典案例的具体步骤：

（1）选择菜单 File\New\Other...，弹出 New Items 对话框。

（2）单击 New Items 对话框的 Businesss 标签页，选择 Decision Cube Sample 对象，选择 Copy 使用方式，如图 217-2 所示，单击 OK 按钮。

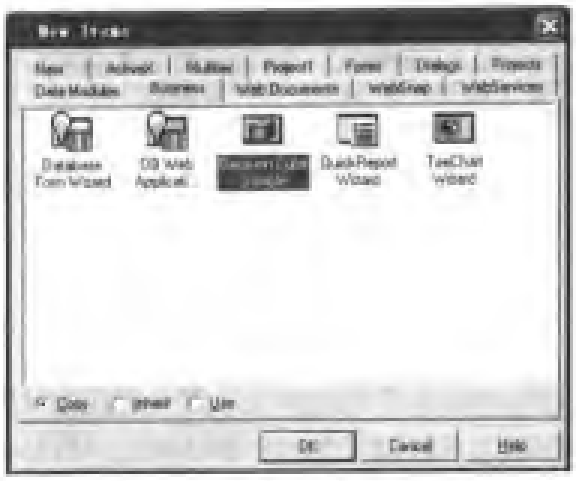


图 217-2

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 217 如何导出对象库中的商业决策分析程序”。

实例 218 如何打印含有图像字段的数据库表

实例目的

此实例主要介绍了在 C++Builder 中打印含有图像字段数据库表的编程技术。实例程序执行后，效果如图 218-1 所示。



图 218-1

实现方法

C++Builder 的 QuickReport 中，有两个打印图像的组件（QRImage 和 QRDBImage），支持图像资源和数据库图像字段在报表中的打印。使用时只要将其放置在要想打印的 Band 区域即可。实例程序实现此技术的具体操作步骤如下（以 C++Builder 启动时默认产生的 Form1 为主窗口）：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 animals.dbf，Active 属性为 true。

（2）在组件模板的 QReport 标签页下拖动 QuickRep1 组件到 Form1 上，在 Object Inspector 中设置 DataSet 属性为 Table1。

（3）在组件模板的 QReport 标签页下拖动两个 QRBand 组件到 QuickRep1 上，在 Object Inspector 中分别设置 BandType 属性为 rbPageHeader 和 rbDetail。

（4）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand1 上，在 Object Inspector 中设置 Caption 属性为“图形报表打印范例。

（5）在组件模板的 QReport 标签页下拖动 QRLabel 组件到 QRBand1 上，在 Object Inspector 中设置 Caption 属性为 C++Builder。

（6）在组件模板的 QReport 标签页下拖动 QRImage 组件到 QRBand1 上，在 Object Inspector 中设置 Picture 属性为一个具体的图像文件。

（7）在组件模板的 QReport 标签页下拖动 5 个 QRLabel 组件到 QRBand2 上，在 Object Inspector 中分别设置 Caption 属性为“动物名称：”、“生长区域：”、“动物身长：”、“动物体重：”、“动物照片：”。

（8）在组件模板的 QReport 标签页下拖动 4 个 QRDBText 组件到 QRBand2 上，在 Object Inspector 中设置所有 4 个组件的 DataSet 属性为 Tabel1，分别设置 4 个组件的 DataField 属性为 NAME、AREA、SIZE、WEIGHT。

（9）在组件模板的 QReport 标签页下拖动 QRDBImage 组件到 QRBand2 上，在 Object Inspector 中分别设置 Dataset 属性为 Table1，设置 DataField 属性为 BMP，如图 218-2 所示。



图 218-2

（10）在主表单的构造函数中加入如下所示的代码：

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  this->QuickRep1->Preview();
//   this->QuickRep1->Print();
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 218 如何打印含有图像字段的数据库表”。

实例 219 如何实现数据库表整表复制

实例目的

此实例主要介绍了在程序中实现数据库表整表复制的编程技术。实例程序执行后，效果如图 219-1 所示。



图 219-1

实现方法

C++Builder 的 TBatchMove 组件能够实现大批量的数据复制、添加、删除和更新操作，它能够在目标数据集中建立一个新表，并自动映射源表中的字段名和字段类型。在使用 TBatchMove 组件进行批量数据处理时，主要应该设置：数据源表 Source，目标表 Destination，复制方式 Mode，匹配方式 Mappings 等属性，然后调用 Execute()方法即可实现数据库表记录的批量处理。复制方式 Mode 属性值的意义如表 219-1 所示。

表 219-1

操作方式	功能
batAppend	把源数据集中的记录增加在目标数据集的最后，目标数据集必须已经存在，该方式为默认方式
batUpdate	用源数据集中的记录替换目标数据集中匹配的记录，目标数据集必须已经存在并建立了索引
batAppendUpdate	用源数据集中的记录替换目标数据集中匹配的记录，不匹配的记录就添加在目标数据集的最后，目标数据集必须已经存在并建立了索引
batCopy	创建一个与源表结构一致的目标表，如果目标表已经存在，将首先删除它
batDelete	删除目标数据集中与源数据集匹配的记录，目标数据集必须已经存在并建立了索引

Mappings 属性用于决定源表和目标表的匹配方式，默认时 TBatchMove 组件是根据源表和目标表的位置进行匹配的，即源表的第一列和目标表的第一列匹配，源表的第二列与目标表的第二列匹配，依此类推。下面是实例程序实现数据库表整表复制的代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
   this->Table2->Close();
   this->Table2->DatabaseName="C:";
   this->Table2->TableName="CopyData.db";
   this->Table2->TableType=ttParadox;
   this->Table2->FieldDefs->Clear();
   this->Table2->FieldDefs=this->Table1->FieldDefs;
   this->Table2->CreateTable();
   this->Table2->Open();
   this->BatchMove1->Source=this->Table1;
   this->BatchMove1->Destination=this->Table2;
   this->BatchMove1->Mappings->Clear();
   this->BatchMove1->Mode=batAppend;
  // this->BatchMove1->RecordCount=2;
   this->BatchMove1->Execute();
   this->Table1->Close();
   MessageBox(NULL,"现在从复制表中读取数据","信息提示",MB_OK);
   this->DataSource1->DataSet=this->Table2;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 219 如何实现数据库表整表复制”。

实例 220　如何实现数据库记录的批量删除

实例目的

此实例主要介绍了在程序中实现数据库记录批量删除的编程技术。实例程序执行后，

首先单击“增加数据表一”按钮，产生一个数据库表；再单击“增加数据表二”按钮，产生另一个完全相同的数据库表；然后单击“-”按钮，删除需要保留的记录；再单击“批量删除记录”按钮，结果所有未删除的记录成批被删除。效果如图 220-1 所示。



图 220-1

实现方法

批量删除数据库记录通常使用 SQL 语句，但是在某种情况下，使用 TBatchMove 组件也能实现数据库记录的批量删除。TBatchMove 组件允许在一组记录或整个数据库的整个表中执行特定的操作，它经常用于从服务器中下载数据到本地以供分析，还可以在目标数据库中建立对应于源数据库表内容的数据库表，并自动将字段名及数据类型进行适当的转换。TBatchMove 组件的操作主要集中于 Mode 和 Mappings 属性，当把 Mode 属性设置为 batDelete 时，即可删除目标表中与源表中匹配的记录。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //批量删除数据库记录
  this->BatchMove1->Source=this->Table2;
  this->BatchMove1->Destination=this->Table1;
  this->BatchMove1->Mappings->Clear();
  this->BatchMove1->Mode=batDelete;
  this->BatchMove1->Execute();
  this->DataSource1->DataSet=this->Table1;
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //创建数据库表一
  this->Table1->Close();
  this->Table1->DatabaseName="C:";
  this->Table1->TableName="Table1.db";
  this->Table1->TableType=ttParadox;
  this->Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "No";
  pNewDef->DataType = ftString;
  pNewDef->Required = true;
  pNewDef->Size = 15;
```

```
        Table1->IndexDefs->Clear();
        Table1->IndexDefs->Add("","No", TIndexOptions() <<ixPrimary << ixUnique);
        this->Table1->FieldDefs->Add("Name",ftString,12,true);
        this->Table1->FieldDefs->Add("Sex",ftString,2,true);
        this->Table1->FieldDefs->Add("Dept",ftString,24,true);
        this->Table1->CreateTable();
        this->Table1->Open();
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100701","王彬","男
","轻工机械")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100702","汪兰","女
","轻工机械")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100703","刘功德","
男","轻工机械")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100704","汤栏兰","
女","幼师")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100705","汤小敏","
男","国际贸易")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100706","罗成","女
","农机")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100707","汤波","男
","农业种植")));
    }
    void __fastcall TForm1::BitBtn3Click(TObject *Sender)
    {  //创建数据库表二
       this->Table2->Close();
       this->Table2->DatabaseName="C:";
       this->Table2->TableName="Table2.db";
       this->Table2->TableType=ttParadox;
       this->Table2->FieldDefs->Clear();
       TFieldDef *pNewDef = Table2->FieldDefs->AddFieldDef();
       pNewDef->Name="No";
       pNewDef->DataType = ftString;
       pNewDef->Required = true;
       pNewDef->Size = 15;
       Table2->IndexDefs->Clear();
       Table2->IndexDefs->Add("","No", TIndexOptions() <<ixPrimary << ixUnique);
       this->Table2->FieldDefs->Add("Name",ftString,12,true);
       this->Table2->FieldDefs->Add("Sex",ftString,2,true);
       this->Table2->FieldDefs->Add("Dept",ftString,24,true);
       this->Table2->CreateTable();
       this->Table2->Open();
       this->Table2->AppendRecord(OPENARRAY(TVarRec,("2004100701","王彬","男
","轻工机械")));
       this->Table2->AppendRecord(OPENARRAY(TVarRec,("2004100702","汪兰","女
","轻工机械")));
       this->Table2->AppendRecord(OPENARRAY(TVarRec,("2004100703","刘功德","
男","轻工机械")));
       this->Table2->AppendRecord(OPENARRAY(TVarRec,("2004100704","汤栏兰","
女","幼师")));
       this->Table2->AppendRecord(OPENARRAY(TVarRec,("2004100705","汤小敏","
男","国际贸易")));
```

```
        this->Table2->AppendRecord(OPENARRAY(TVarRec,("2004100706","罗成","女
","农机")));
        this->Table2->AppendRecord(OPENARRAY(TVarRec,("2004100707","汤波","男
","农业种植")));
        this->DataSource1->DataSet=this->Table2;
    }
```

注意：默认情况下，TBatchMove 组件在进行数据库记录批量删除时，需要数据库有索引支持。不过也可以通过 TBatchMove 组件的 AbortOnKeyViol 属性和 AbortOnProblem 属性进行这方面的违例设置。TBatchMove 组件的 AbortOnKeyViol 属性用于决定在批操作期间如果出现了关键字违反完整性是否放弃执行，如果该属性设置为 TRUE 并且出现了关键字违反完整性，TBatchMove 组件的 Execute()方法将立即停止操作，如果出现了关键字违反了完整性但还想继续操作时，则应该将属性设置为 FALSE，同时还要设置用来保存错误记录的表名；TBatchMove 组件的 AbortOnProblem 属性用于决定在批操作期间如果出现了数据类型转换错误时是否放弃执行，如果该属性设置为 TRUE 并且出现了数据类型转换错误，TBatchMove 组件的 Execute()方法将放弃从源记录传递到目标记录的数据，如果出现了数据类型转换错误但用户还想继续操作时，要将该属性设置为 FALSE，同时还要设置用来保存有问题记录的表名。

TTable->IndexDefs->Add()方法用于向数据库表中增加索引，增加索引时要指定索引名、被索引的字段及索引选项。索引选项根据数据库的类型不同而不同，常用的索引选项的含义如表 220-1 所示。

表 220-1

索引选项	含义
ixPrimary	索引采用数据库主索引
ixUniqueEach	采用惟一索引方式
ixDescending	按降序进行索引
ixExpression	索引取决于 dBase 的关键字表达式
ixCaseInsensitive	索引区分大小写

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 220 如何实现数据库记录的批量删除”。

实例 221 如何实现数据库记录的批量修改

实例目的

此实例主要介绍了在两个数据库表中实现记录成批修改提交的技术。实例程序执行

后，先单击“复制表数据”按钮，创建一个目标数据库表；然后再在表一中修改数据，单击“批量修改数据”按钮，则表二中的数据同步修改。效果如图 221-1 所示。



图 221-1

实现方法

在使用 C++Builder 编写服务器类型的数据库应用程序时，常常遇到这样的问题：在一张表中修改多个数据时，修改结果尚未来得及保存，就被服务器数据库的自动同步数据功能作废。在这种情况下，可以考虑使用 TBatchMove 组件的批量数据修改功能，即先将服务器数据库的数据批量复制到本地数据库中，然后在本地数据库中进行修改，修改完成后再批量提交到服务器数据库。与数据库记录的批量增加类似，在实现批量记录修改时，只需要修改 TBatchMove 组件的 Mode 属性，将该属性设置为 batAppendUpdate 即可。它的作用是：在 TBatchMove 组件实现批量移动数据时，首先匹配原有数据库记录，若没有对应的记录匹配上，则作为新记录进行添加。实例程序实现此技术的步骤如下：首先复制一个与源表一样的目标表，然后对源表进行编辑修改，修改完成后以添加并更新的方式移动到目标表中，示例代码如下：

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{ //创建源表
  this->Table1->Close();
  this->Table1->DatabaseName='C:';
  this->Table1->TableName='Table1.db';
  this->Table1->TableType=ttParadox;
  this->Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = 'No';
  pNewDef->DataType = ftString;
  pNewDef->Required = true;
  pNewDef->Size = 15;
  Table1->IndexDefs->Clear();
  Table1->IndexDefs->Add('','No',    TIndexOptions()    <<ixPrimary    <<
ixUnique);
  this->Table1->FieldDefs->Add('Name',ftString,12,true);
```

```
        this->Table1->FieldDefs->Add("Sex",ftString,2,true);
        this->Table1->FieldDefs->Add("Dept",ftString,24,true);
        this->Table1->CreateTable();
        this->Table1->Open();
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100701","王彬","男
","轻工机械")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100702","汪兰","女
","轻工机械")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100703","刘功德","
男","轻工机械")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100704","汤栏兰","
女","幼师")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100705","汤小敏","
男","国际贸易")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100706","罗成","女
","农机")));
        this->Table1->AppendRecord(OPENARRAY(TVarRec,("2004100707","汤波","男
","农业种植")));
    }
    //---------------------------------------------------------------------------
    void __fastcall TForm1::BitBtn1Click(TObject *Sender)
    {//创建目标表
       this->Table2->Close();
       this->Table2->DatabaseName="C:";
       this->Table2->TableName="CopyData.db";
       this->Table2->TableType=ttParadox;
       this->Table2->FieldDefs->Clear();
       this->Table2->FieldDefs=this->Table1->FieldDefs;
       Table2->IndexDefs->Clear();
       Table2->IndexDefs->Add("","No", TIndexOptions() <<ixPrimary << ixUnique);
       this->Table2->CreateTable();
       this->Table2->EmptyTable();
       this->Table2->Open();
       this->BatchMove1->Source=this->Table1;
       this->BatchMove1->Destination=this->Table2;
       this->BatchMove1->Mappings->Clear();
       this->BatchMove1->Mode=batAppend;
      // this->BatchMove1->RecordCount=2;
       this->BatchMove1->Execute();
    }
    void __fastcall TForm1::BitBtn2Click(TObject *Sender)
    {//批量修改数据
       this->BatchMove1->Source=this->Table1;
       this->BatchMove1->Destination=this->Table2;
       this->BatchMove1->Mappings->Clear();
       this->BatchMove1->Mode=batAppendUpdate;
      // this->BatchMove1->RecordCount=2;
       this->BatchMove1->Execute();
    }
```

注意：此项操作也要求数据库提供索引支持。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 221 如何实现数据库记录的批量修改”。

实例 222　如何使用 SQL Builder 工具创建 SQL 语句

实例目的

此实例主要介绍了在 C++Builder 中使用 SQL Builder 工具创建 SQL 语句的方法。实例程序执行后，效果如图 222-1 所示。

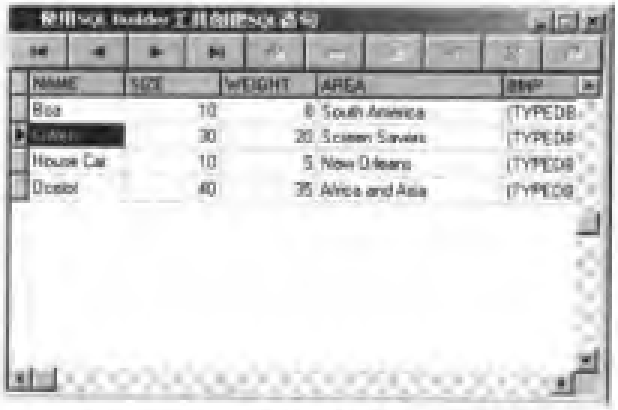


图 222-1

实现方法

SQL Builder 是 C++Builder 所附带的一个自动生成 SQL 语句的工具，它的最大特点就是在编写 SQL 语句时，不需要程序员懂得 SQL 语法，只需要作几个简单的选择和输入一些查询值即可获得一个完整的 SQL 语句。SQL Builder 工具不是一个可执行程序，而是一个动态链接库，必须在启动 C++Builder 后，在适当的环境下才能打开该工具。下面是实例程序使用 SQL Builder 工具创建 SQL 语句的具体步骤（以 C++Builder 启动时默认产生的 Form1 为主窗口）。

（1）在组件模板的 BDE 标签页下拖动 Query1:TQuery 组件到 Form1 上。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Query1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（5）右击 Query1:TQuery 组件，在弹出的右键菜单中选择 SQL Builder...，弹出 SQL

Builder 对话框，如图 222-2 所示。SQL Builder 的窗口主要分为两部分。



图 222-2

1）表窗格。该窗格显示了要查询的表中的所有字段名，每个字段的左边都有一个复选框，让用户决定是否要选择该字段。如果用户要选择表中的所有字段，则应该选中最上面的一个复选框。如果需要同时查询几个表，只要在表窗格中选择一个表，SQL Builder 会自动将该表加入到查询中。

2）查询窗格。查询窗格实际上是一个多标签页组件，产生 SQL 语句的操作主要集中在 Criteria 标签页上，实际上，它就是产生 SQL 语句 WHERE 条件子句的地方。SQL Builder 有 3 种条件表达式的类型，它们是简单表达式、SQL 表达式和 EXISTS；要选择条件表达式的类型，可以在 Criteria 标签页上单击右键，从弹出的右键菜单中选择 Simple Equation、SQL Expression 或 EXISTS。如果选择的是 Simple Equation，则首先要在最左边的栏中选择一个字段或直接输入一个常量，然后在中间栏中选择比较操作符，如=、>、<、>=、<=、<>、LIKE、NOT LIKE、IN、NOT IN、BETWEEN、NOT BETWEEN、IS NULL、IS NOT NULL，最后在最右边的栏中选择另一个字段或直接输入一个常量。注意：如果直接输入字符串或日期型常量，应该用单引号括起来。

（6）在 SQL Builder 对话框的 Database 下拉列表框中选择数据库别名 BCDEMOS，在 Table 下拉列表框中选择 Animals.dbf，在 Field or Value 栏中选择 Animals."SIZE"，在 Compare 栏中选择>=，在 Field or Value 栏中输入 10。SQL Builder 会自动把刚才在以上 3 栏输入或选择的字段名、常量和比较符组合成一个 SQL 语句，此时只要选择菜单 Query\Show SQL，就可以看到 SQL Builder 自动生成的 SQL 语句，如图 222-3 所示。要执行 SQL 语句，则应

该选择菜单 Query\Run Query 或者按 F9 键，结果如图 222-4 所示。

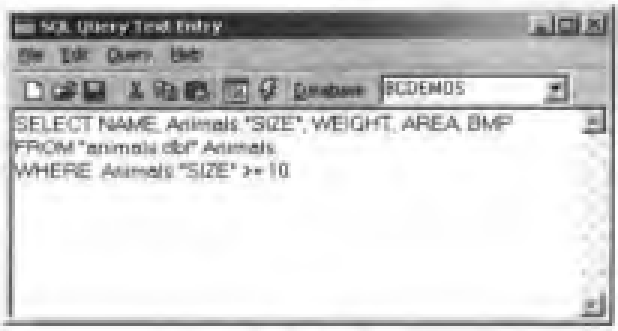


图 222-3

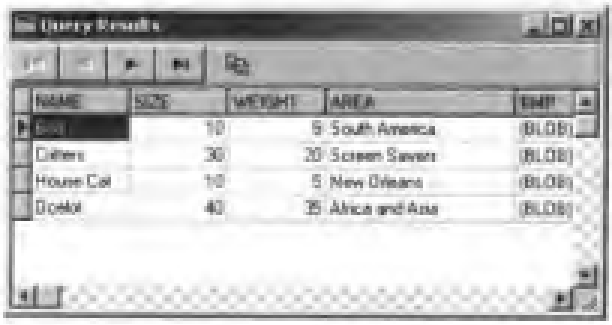


图 222-4

（7）关闭 SQL Builder 对话框，在弹出的 Confirm 对话框中单击 Yes 按钮，保存刚才的操作，则 TQuery 组件会自动将刚才创建的 SQL 语句添加到 SQL 属性中。此时设置 Query1 组件的 Active 属性为 true，即可看到程序的运行结果。

最后，编译执行程序即可。注意：如果在 SQL Builder 对话框的 of the following criteria are met:列表框中选择 ALL，则表示设置的条件必须全部满足；如果选择 ANY，表示只满足这些条件中的一个即可；如果选择 NONE，表示这些条件必须都不满足；如果选择 NOT ALL，表示至少要有一个条件不满足。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 222 如何使用 SQL Builder 工具创建 SQL 语句”。

实例 223　如何在程序运行时指定 SQL 语句

实例目的

此实例主要介绍了在 C++Builder 中使用 TQuery 组件时，如何在程序运行期间使用 SQL 语句完成对数据集的操作。实例程序执行后，效果如图 223-1 所示。

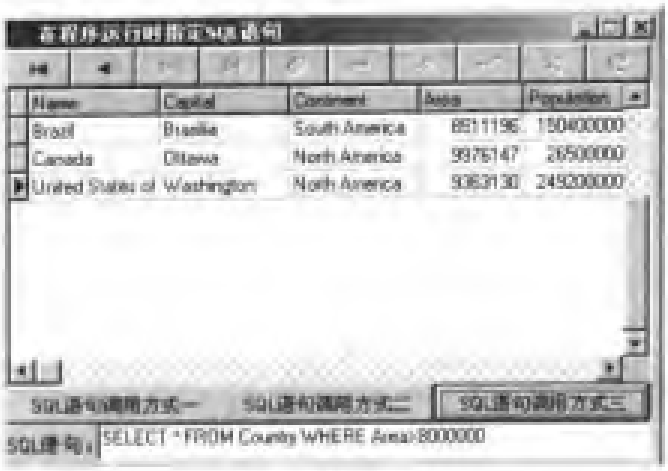


图 223-1

实现方法

C++Builder 的 TQuery 组件是一个数据集组件，在编写数据库应用程序时占有相当重要的地位，特别是编写远程数据库应用程序时，更是不可或缺。TQuery 组件实现了 C++Builder 对 SQL 语句的支持，SQL 语句通过该组件传递到要访问的数据库系统的数据库引擎中，由数据库引擎具体执行 SQL 语句以实现对数据库的操作，而不是像 TTable 组件那样传递给 C++Builder 中的 BDE，由 BDE 实施具体的 SQL 转换，再传递到数据库引擎执行。TQuery 组件的 SQL 属性用于指定要执行的 SQL 语句，在设计期间，可以在 Object Inspector 中设置 SQL 语句，如图 223-2 所示，再把 Active 属性设置为 true 即可。



图 223-2

此外，TQuery 组件的 SQL 属性也可以在程序运行期间指定。在程序运行期间指定 SQL 语句大致有 3 种方式，其操作方法如下。

（1）直接设置 SQL 属性。在直接设置 SQL 属性前，首先要调用 TQuery 组件的 Close() 方法使该组件处于不活动状态；如果 SQL 属性中本来已经有 SQL 语句，还要调用 Clear() 方法把原来的 SQL 语句清除掉。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
   this->Query1->DatabaseName="BCDEMOS";
   this->Query1->Close();
   this->Query1->SQL->Clear();
   this->Query1->SQL->Add("SELECT * FROM country.db");
   this->Query1->SQL->Add("WHERE Continent='South America'");
   this->Query1->Open();
}
```

（2）调用 LoadFromFile()函数从文件中读取 SQL 语句。因为 TStrings 对象支持文件操作，所以也可以调用 LoadFromFile()函数从文件中获取 SQL 语句。注意：LoadFromFile()函数会自动把原来的 SQL 语句清除掉，因此不必再次调用 Clear()方法把原来的 SQL 语句清除掉。示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   this->Query1->DatabaseName="BCDEMOS";
   this->Query1->Close();
   this->Query1->SQL->LoadFromFile("SQL.txt");
   this->Query1->Open();
}
```

（3）从另一个 TStrings 对象中获取 SQL 语句。从另一个 TStrings 对象中获取 SQL 语句时，需要调用 TStrings 对象的 Assign()函数，该函数会自动把原来的 SQL 语句清除掉，因此不必再次调用 Clear()方法把原来的 SQL 语句清除掉。示例代码如下：

```
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
   this->Query1->DatabaseName="BCDEMOS";
   this->Query1->Close();
   this->Query1->SQL->Assign(Memo1->Lines);
   this->Query1->Open();
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 223 如何在程序运行时指定 SQL 语句”。

实例 224　如何在程序中创建并使用数据库

实例目的

此实例介绍了直接在程序中创建并使用数据库的编程技术。实例程序执行后，效果如图 224-1 所示。

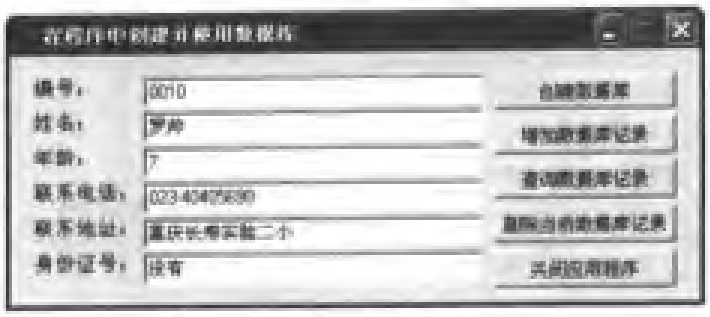


图 224-1

实现方法

在进行数据库管理系统开发时，一般是创建数据库后再在程序中使用数据库。实际上，也可以根据实际需要在程序中临时创建并使用数据库。C++Builder 组件面板的 TTable 组件有一个 CreateTable()方法，用于创建数据库表。通常创建一个数据库需要以下操作步骤。

（1）定义数据库名和数据库类型。它是通过设置 TTable 组件的 TableType 变量实现，该变量的可能取值包括 ttDefault（默认值，由表的扩展名确定类型）、 ttParadox（Paradox 数据库）、 ttDBase（dBase 数据库）、ttFoxPro（FoxPro 数据库）、ttASCII（文本文件）。数据库类型和数据库名称的定义通常应该在数据库表被关闭的状态下进行，若数据库表未被关闭，则不能进行这些初始设置，否则系统可能产生异常。

（2）定义数据库表的字段名及类型。定义数据库表的字段名及类型由 TTable 组件的 FieldDefs 成员变量实现，增加字段的 Add()方法的声明如下：

```
HIDESBASE void __fastcall Add(const AnsiString Name,
                    TFieldType DataType, int Size = 0, bool Required = false);
```

其中，参数 const AnsiString Name 为字段名，参数 TFieldType DataType 为字段类型，参数 int Size 为字段大小（根据具体字段类型而定），参数 bool Required 为是否是必填字段（根据具体的数据库类型而定）。常见的字段类型有 ftUnknown、ftString、ftSmallint、ftInteger、ftWord、ftBoolean、ftCurrency、ftBCD、ftDate、ftTime、ftDateTime、ftBytes、ftVarBytes、ftAutoInc、ftBlob、ftMemo、ftGraphic、ftFmtMemo、ftParadoxOle、ftDBaseOle、ftTypedBinary、ftCursor、ftFixedChar、ftWideString、ftLargeint、ftADT、ftArray、ftReference、ftDataSet、ftOraBlob、ftOraClob、ftVariant、ftInterface、ftIDispatch、ftGuid、ftTimeStamp、ftFMTBcd。

（3）定义数据库表关键字。该步骤特别重要。否则，即使能够创建一个数据库，但是不能使用。定义关键字由 TTable 组件的 IndexDefs 成员变量的 Add()方法实现。该函数的声明如下：

```
HIDESBASE void __fastcall Add(const AnsiString N, const AnsiString F,
                    TIndexOptions Options);
```

其中，参数 TIndexOptions Options 的可能取值包括 ixPrimary、ixUnique、ixDescending、ixExpression、ixCaseInsensitive、ixNonMaintained。

（4）调用 TTable 组件的 CreateTable()方法创建表，若创建不成功，系统将产生异常。

以下是在实例程序中创建并使用数据库的示例代码：

```
#include <vcl.h>
#pragma hdrstop
#include "Mainform.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
#include <DB.hpp>
#include <DBTables.hpp>
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //创建数据库
  try
  {
   TTable* newTable = new TTable(this);
   newTable->Active = false;
   newTable->DatabaseName =""; //数据库文件存放位置，默认为当前目录
   newTable->TableName ="MyDatabase.db"; //数据库文件名
   newTable->TableType = ttParadox;   //数据库类型
   //定义数据表字段类型
   newTable->FieldDefs->Clear();
   newTable->FieldDefs->Add("PID",ftInteger,0,true); //定义字段、类型、长度等
   newTable->FieldDefs->Add("No",ftString,50,false);
   newTable->FieldDefs->Add("Name",ftString,50,false);
   newTable->FieldDefs->Add("Age",ftString,50,false);
   newTable->FieldDefs->Add("Phone",ftString,50,false);
   newTable->FieldDefs->Add("Address",ftString,50,false);
   newTable->FieldDefs->Add("IDCode",ftString,50,false);
   newTable->IndexDefs->Add("","PID",   TIndexOptions()   <<ixPrimary   <<
ixUnique); //定义关键字
   newTable->CreateTable(); //创建数据表
   delete newTable;
  }
  catch(...)
  {
    MessageBox(Handle,"创建数据库操作失败！","信息提示",MB_OK);
    return;
  }
  MessageBox(Handle,"创建数据库操作成功！","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{ //增加数据库记录
 try
 {
  Table1->DatabaseName="";
  Table1->TableName="MyDatabase.db";
  Table1->Active=true;
```

```
    Table1->Last();
    int iPID=1;
    if(Table1->IsEmpty())
       iPID=1;
    else
      iPID=Table1->FieldValues["PID"]+1;
    Table1->Append();
    Table1->FieldValues["PID"]=iPID;
    Table1->FieldValues["No"]=this->Edit1->Text;
    Table1->FieldValues["Name"]=this->Edit2->Text;
    Table1->FieldValues["Age"]=this->Edit3->Text;
    Table1->FieldValues["Phone"]=this->Edit4->Text;
    Table1->FieldValues["Address"]=this->Edit5->Text;
    Table1->FieldValues["IDCode"]=this->Edit6->Text;
    Table1->ApplyUpdates();
    Table1->CommitUpdates();
   }
   catch(...)
   {
     MessageBox(Handle,"增加数据库记录操作失败！",'信息提示',MB_OK);
     return;
   }
   MessageBox(Handle,"增加数据库记录操作成功！","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{//查询数据库记录
   AnsiString StrName=InputBox("输入查询条件","请输入查询姓名：          ","");
   Table1->DatabaseName="";
   Table1->TableName="MyDatabase.db";
   Table1->Close();
   Table1->Filter="Name=\'";
   Table1->Filter+=StrName;
   Table1->Filter+="\'";
   Table1->Filtered=true;
   Table1->Active = true;
   if(Table1->IsEmpty())
   {
    this->Edit1->Text="";
    this->Edit2->Text="";
    this->Edit3->Text="";
    this->Edit4->Text="";
    this->Edit5->Text="";
    this->Edit6->Text="";
    MessageBox(Handle,"没有找到！","信息提示",MB_OK);
   }
   else
   {
    this->Edit1->Text=Table1->FieldValues["No"];
    this->Edit2->Text=Table1->FieldValues["Name"];
    this->Edit3->Text=Table1->FieldValues["Age"];
```

```
        this->Edit4->Text=Table1->FieldValues["Phone"];
        this->Edit5->Text=Table1->FieldValues["Address"];
        this->Edit6->Text=Table1->FieldValues["IDCode"];
      }
}
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{//删除数据库记录;
  if(MessageBox(Handle,"是否确认删除当前数据库记录? ","信息提示",MB_YESNO)==IDNO)
     return;
   try
   {
    this->Table1->Delete();
    this->Table1->CommitUpdates();
    this->Table1->ApplyUpdates();
   }
   catch(...)
   {
    MessageBox(Handle,"请首先创建数据库! ","信息提示",MB_OK);
    return;
   }
  MessageBox(Handle,"删除数据库记录操作成功! ","信息提示",MB_OK);
}
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{
   this->Close();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 224 如何在程序中创建并使用数据库”。

实例 225　如何在程序中控制与数据库的连接

实例目的

此实例主要解决如何将应用程序与数据库连接时的英文登录框改为中文登录框，并根据程序运行的需要，随时决定是否与数据库进行连接。实例程序执行后，效果如图 225-1 所示。

实现方法

当使用 C++Builder 开发的应用程序与一些大型服务器数据库（如 InterBase 、MS-SQL Server）相连接时，程序总是先弹出一个英文登录对话框，要求用户输入用户名和密码，用户必须输入正确的用户名和密码后，才能进入应用程序访问数据库。实际上，通过 Database 组件的设置，可以随时决定在何时弹出数据库连接时的登录对话框，并可以进行

定制，示例代码如下：

```
void __fastcall TForm2::BitBtn1Click(TObject *Sender)
{
  if(Form1->Database1->Connected==false)
  {
    Form1->Database1->LoginPrompt=false;
    Form1->Database1->Params->Values['UserName']=this->Edit1->Text;
    Form1->Database1->Params->Values['Password']=this->Edit2->Text;
    try
    {
      Form1->Database1->Connected=true;
      Form1->Table1->Active=true;
    }
    catch(...)
    {
      MessageBox(NULL,'与数据库连接失败','信息提示',MB_OK);
    }
  }
  this->Close();
}
```



图 225-1

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 225 如何在程序中控制与数据库的连接”。

实例 226 如何在程序中创建 Access 数据库 ODBC 数据源

实例目的

此实例介绍了直接在程序中创建 Access 数据库 ODBC 数据源的编程技术。实例程序执行后，效果如图 226-1 所示。

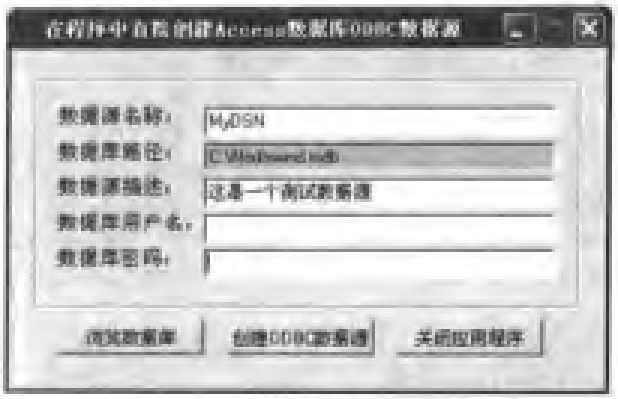


图 226-1

实现方法

ODBC 数据源通常是在操作系统的“控制面板\性能和维护\管理工具\数据源（ODBC）中创建。实际上，创建 ODBC 数据源就是一个写操作系统注册表的过程，下面就是一个通过直接写注册表产生 Access 数据库 ODBC 数据源的示例代码。

```
#include "Registry.hpp"
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{//创建 ODBC 数据源
  if(this->Edit2->Text.Length()<1)
    return;
  TRegistry *SystemReg=new TRegistry();
//设置根键值为 HKEY_LOCAL_MACHINE
  SystemReg->RootKey = HKEY_LOCAL_MACHINE;
  //找到 Software\ODBC\ODBC.INI\ODBC 数据源
  if(SystemReg->OpenKey("Software\\ODBC\\ODBC.INI\\ODBC Data Sources",
true))
  {//注册数据源名称
    SystemReg->WriteString(this->Edit1->Text, "Microsoft Access Driver
(*.mdb)" );
    SystemReg->CloseKey();
  }
  else
  { //创建键值失败
    throw Exception("创建 Access 数据库 ODBC 数据源失败！");
    return;
  }
  //写入数据源配置信息
  if(SystemReg->OpenKey("Software\\ODBC\\ODBC.INI\\" +
this->Edit1->Text, true))
  {
    //数据库文件全路径
    SystemReg->WriteString("DBQ", this->Edit2->Text);
    //数据源描述
    SystemReg->WriteString("Description", this->Edit3->Text);
    char buf[MAX_PATH];
    ::GetSystemDirectory(buf, MAX_PATH);
    //驱动程序 DLL 文件
```

```
        SystemReg->WriteString("Driver",
    AnsiString(buf) + "file://odbcjt32.dll/" );
        //驱动程序标识
        SystemReg->WriteInteger("DriverId", 25 );
        SystemReg->WriteString("FIL", "Ms Access;" );      //Filter 依据
        SystemReg->WriteInteger("SafeTransaction", 0 );    //支持的事务操作数目
        SystemReg->WriteString("UID", this->Edit4->Text);  //用户名称
        SystemReg->WriteString("PWD", this->Edit5->Text);  //用户密码
        BYTE bData = 0;
        SystemReg->WriteBinaryData("Exclusive", &bData, 1); //独占方式
        SystemReg->WriteBinaryData("ReadOnly", &bData, 1 ); //只读方式
        SystemReg->CloseKey();
      }
      else
      {
        throw Exception("创建 Access 数据库 ODBC 数据源失败！");
        return;
      };
     //找到或创建 Software\ODBC\ODBC.INI\MyAccess\Engines\Jet,
    //写入数据源数据库引擎配置信息
      if (SystemReg->OpenKey("Software\\ODBC\\ODBC.INI\\" +
    this->Edit1->Text + "file://Engines//Jet", true))
       {
        SystemReg->WriteString( "ImplicitCommitSync", "Yes");
        SystemReg->WriteInteger("MaxBufferSize", 2048 );  //缓冲区大小
        SystemReg->WriteInteger("PageTimeout", 10 );      //页超时
        SystemReg->WriteInteger("Threads", 3 );           //支持的线程数目
        SystemReg->WriteString("UserCommitSync", "Yes");
        SystemReg->CloseKey();
      }
      else
      {
        throw Exception("创建 Access 数据库 ODBC 数据源失败！");
        return;
      }
      MessageBox(Handle,"创建 Access 数据库 ODBC 数据源成功！","信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 226 如何在程序中创建 Access 数据库 ODBC 数据源”。

实例 227 如何在程序运行时创建数据库 BDE 别名

实例目的

此实例介绍了如何在程序运行时创建数据库 BDE 别名的编程技术。实例程序运行后，

在“数据库别名”编辑栏中输入新的数据库 BDE 别名后，单击“创建 BDE 别名”按钮，程序将立即给出创建数据库 BDE 别名成功或失败的提示，效果如图 227-1 所示。

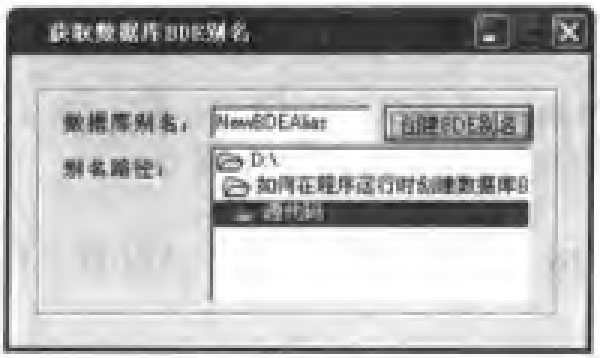


图 227-1

实现方法

通常在使用 C++Builder 编写数据库应用程序时，是在 BDE Administrator 中建立一个 BDE 别名，然而完全可以使用 BDE 函数 DbiAddAlias()创建 Paradox 类型的数据库的 BDE 别名。BDE 函数 DbiAddAlias()的声明如下：

```
DBIResult DbiAddAlias( [hCfg], pszAliasName,
                       pszDriverType, pszParams, bPersistent );
```

其中，参数[hCfg]通常设置为 NULL，表示新的数据库别名将要加入到 Session 对象的配置文件中；参数 pszAliasName 指定要建立的数据库 BDE 别名的名称；参数 pszDriverType 指定数据库驱动程序的类型；参数 pszParams 指定要建立 BDE 别名的数据库所在的路径；参数 bPersistent 指定新建立别名的存在范围。

以下是实例程序使用 BDE 函数 DbiAddAlias()在程序运行时创建 BDE 别名的实现代码（程序中的 TDirectoryListBox 组件在组件模板的 Win 3.1 标签页下）。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//创建数据库 BDE 别名
  if(this->Edit1->Text.Length()<1)
    return;
  try
  {
    Screen->Cursor=crHourGlass;
    hDBISes Session;
    DbiInit(NULL);
    DbiStartSession(NULL,Session,"");
    AnsiString StrPath="PATH:"+this->DirectoryListBox1->Directory;
    DbiAddAlias(0,this->Edit1->Text.c_str(),"PARADOX",StrPath.c_str(),true);
    DbiCloseSession(Session);
    DbiExit();
    Screen->Cursor=crDefault;
    MessageBox(Handle,"BDE 别名创建成功！","信息提示",MB_OK);
  }
  catch(...)
  {
```

```
        MessageBox(Handle,'BDE 别名创建失败！','信息提示',MB_OK);
    }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 227 如何在程序运行时创建数据库 BDE 别名”。

实例 228 如何在安装程序中设置 BDE 引擎

实例目的

此实例主要解决了曾经相当烦人的使用 C++Builder 编写的数据库应用程序 BDE 引擎的设置。实例程序执行后，在“BDE 配置文件路径”编辑栏中输入 IDAPI32.CFG 文件的路径，在“BDE 动态链接库路径”编辑栏中输入 BDE 所有 15 个动态链接库的路径，单击“设置”按钮即可，效果如图 228-1 所示。



图 228-1

实现方法

要充分发挥 C++Builder 数据库编程的优势，就离不开 BDE 引擎。要使数据库应用程序脱离 C++Builder 开发环境运行，就要与数据库应用程序一起发布 BDE。而 BDE 的动态链接库的使用不是简单的文件复制操作能够实现的，因为它在系统注册表中进行大量的设置。如图 228-2 所示的 BDE 注册表设置只是其中相当一小部分的注册表 BDE 设置。



图 228-2

实际上，在非开发环境中运行 C++Builder 编写的数据库应用程序时，应用程序通常需要在注册表中至少指定图 228-2 所示的 CONFIGFILE01 和 DLLPath 这两个参数，然后在这两个参数指定的文件夹下调用 BDE 动态链接库文件即可，这些文件包括 bantam.dll、blw32.dll、ceeurope.btl、charset.cvb、europe.btl、fareast.btl、idapi32.dll、idasci32.dll、idbat32.dll、idda3532.dll、iddao32.dll、iddbas32.dll、iddr32.dll、idodbc32.dll、idpdx32.dll、idqbe32.dll、idr20009.dll、idsql32.dll、japan.btl、other.btl、usa.btl、idapi32.cfg。同时，建议带上 Bdeadmin.exe 这个配置工具。IDAPI32.CFG 文件保存的是在 Bdeadmin.exe 这个配置工具中的配置结果，即通常所用的路径、服务器名、缓存长度、语言驱动程序、用户名、口令等。以下是实例程序在注册表中设置 BDE 配置的实现代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  AnsiString StrCfg=this->Edit1->Text;
  AnsiString StrDll=this->Edit2->Text;
  TRegistry *Reg = new TRegistry;
  try
  {
    Reg->RootKey = HKEY_LOCAL_MACHINE;
    if (Reg->OpenKey("\\Software\\Borland\\Database Engine", true))
    {
      Reg->WriteString("CONFIGFILE01",StrCfg);
      Reg->WriteString("DLLPath",StrDll);
      Reg->CloseKey();
    }
  }
  __finally
  {
    delete Reg;
  }
  MessageBox(Handle,"BDE 引擎设置成功","信息提示",MB_OK);
}
```

说明：对于非文件型的数据库管理系统，必须设置 HKEY_LOCAL_MACHINE、Software、Borland、Database Engine、Settings、Drivers 节点下的内容。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 228 如何在安装程序中设置 BDE 引擎”。

实例 229　如何消除 MS-SQL Server 数据库连接的登录框

实例目的

此实例主要解决如何除掉应用程序与数据库连接时的登录框。实例程序执行后将直接进入程序界面，而不弹出数据库登录对话框。效果如图 229-1 所示。



图 229-1

实现方法

当使用 C++Builder 开发的应用程序与一些大型服务器数据库（如 InterBase 、MS-SQL Server）相连接时，程序总是先弹出一个登录对话框，要求用户输入用户名和密码，用户必须输入正确的用户名和密码后，才能进入应用程序访问数据库。从保证数据库的安全性来说，这一过程是必需的；从操作的简便性来说，这是烦人的。这是因为，当调用数据库组件的 Open()函数或把 Connected 属性设置为 true 时，将触发 OnLogin 事件，如果程序没有处理该事件，就会弹出一个标准的登录对话框让用户登录。如果在没有连接服务器的情况下试图打开一个数据集，将首先调用 Open()连接服务器，并根据需要自动创建一个临时的 TDatabase 组件，一旦与服务器建立了连接，只要至少一个数据集是活动的，连接就会一直保持，如果所有的数据集都不处于活动状态，是否断开连接取决于 KeepConnection 属性的值，如果 KeepConnection 属性的值设置为 true，即使没有一个数据集处于活动状态，也保持连接，对于那些频繁地打开和关闭数据集的应用程序，这样可以避免每次都要登录，如果 KeepConnection 属性的值设置为 false，当所有的数据集都关闭时，将断开与服务器的连接。要断开连接，需要调用 Close()函数或把 Connected 属性设置为 false。实际上，把 Connected 属性设置为 false 就会自动调用 Close()函数，Close()函数会自动关闭所有的数据集，然后断开连接。如果只想关闭所有的数据集，但不想断开连接，则应该把 KeepConnection 属性的值设置为 true，然后调用 CloseDataSet()。

配置 BDE 时期，当试图连接一个远程服务器，BDE 会自动弹出一个标准的登录对话框，让用户输入用户名和密码。在程序运行时期，登录方式有以下 3 种：①当把 LoginPrompt 属性设置为 true 时，在连接远程服务器时，就会自动弹出一个标准的登录对话框，让用户输入用户名和密码；②当把 LoginPrompt 属性设置为 false 时，就应该设置 Params 属性，配置默认的用户名和密码，这样就不会弹出登录对话框；③在处理 OnLogin()事件的句柄中设置 LoginParams 参数，在这种情况下，当退出处理 OnLogin()事件时，LoginParams 参数的值就自动赋给了 Params 属性。实例程序按照第二种方式实现隐藏登录对话框的具体步骤如下。

（1）在组件模板的 BDE 标签页下拖动 Database1:TDatabase 组件到 Form1 上，在 Object Inspector 中设置 Database1 的 AliasName 属性为 MySQL；DatabaseName 属性为 SQLPrompt；LoginPrompt 属性为 false；单击 Params 属性右边的“...”按钮，弹出数据库用户名和密码设

置框，如图 229-2 所示，请根据实际情况进行设置（实例程序使用的是 MS、SQL Server 数据库，并且这个数据库是自定义的，因此，如果直接执行实例程序可能不能正常执行，但是可以依照在实例程序中的配置，重新设置数据库）。

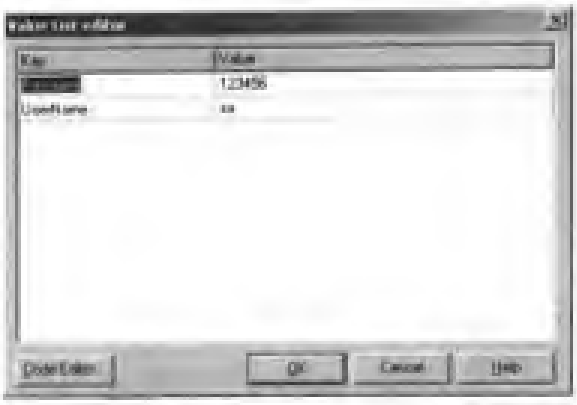


图 229-2

然后设置 Connected 属性为 true，如果数据库用户名和密码正确及网络连接正常，该属性能够正确设置，否则会弹出一个错误提示对话框，并自动重新设置 Connected 属性为 false。如果在最后设置完成后能够实现正确连接，但有些数据类型无法进行转换，请使用 SQL Server 的其他数据库进行连接。

（2）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 SQLPrompt，TableName 属性为 dbo.customers，Active 属性为 true。

（3）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（4）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（5）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 229 如何消除 MS-SQL Server 数据库连接的登录框”。

实例 230　如何连接 MS-SQL Server 数据库

实例目的

在使用 C++Builder 编写数据库应用程序时，首先必须解决应用程序如何与数据库进行

连接，MS-SQL Server 数据库是目前数据库领域中占有主导地位的数据库，通常情况下，解决 C++Builder 如何与 MS-SQL Server 数据库进行连接设置的问题，对连接其他类型的关系数据库具有重要的启示意义。此编程技巧主要介绍如何与 MS-SQL Server 数据库进行连接。此编程技巧没有实例程序。

实现方法

在 C++Builder 中，既可以通过 ODBC 方式实现与 MS-SQL Server 数据库的连接，也可以通过设置 BDE 的方式实现与 MS-SQL Server 数据库的连接，非常明显，通过设置 BDE 的方式实现与 MS-SQL Server 数据库的连接的应用程序的执行效率远远高于通过设置 ODBC 的方式实现与 MS-SQL Server 数据库的连接的应用程序。在 BDE 中设置与 MS-SQL Server 数据库连接的具体步骤如下：

（1）确认系统已经安装了 MS-SQL Server 客户端或服务端系统，以便于测试连接。

（2）启动 BDE Administrator 工具。选择 Object\New...菜单，弹出 New Database Alias 对话框，在 Database Driver Name 列表框中选择 MSSQL 选项，单击 OK 按钮，BDE 工具自动在左边列表中新增 MSSQL1 节点，更名为 MySQL。

（3）选择 MySQL 节点，在右边的 DATABASE NAME 中设置数据库名，如：Northwind，在 HOST NAME 中设置主机名，在 SERVER NAME 中设置服务器名，在 USER NAME 中设置用户名。注意：用户名必须是 MS-SQL Server 数据库中具有合法权限打开该数据库的用户名，否则可能无法打开数据库，其余保留默认值即可，如图 230-1 所示。



图 230-1

（4）右击 MySQL 节点，在弹出的右键菜单中选择 Apply 菜单项，保存 BDE 设置结果即可。

（5）双击 MySQL 节点，弹出 Database Login 对话框，在 User Name 编辑栏中输入用户名称。在 Password 编辑栏中输入用户密码，如图 230-2 所示。单击 OK 按钮，如果所有设置正确，MySQL 节点图标改变；否则弹出一个错误信息对话框。如果确认主机名、服务器名和数据库名正确无误，并且也无法连接，请在 MS-SQL Server 数据库的企业管理器的安全性的登录管理中进行检查用户名和密码设置。

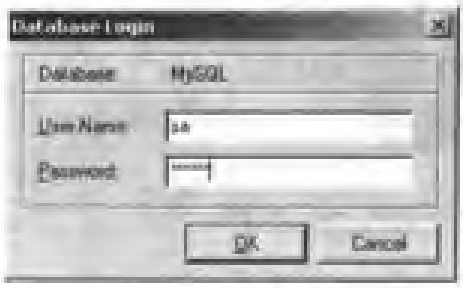


图 230-2

至此，在 BDE 中设置 C++Builder 数据库程序与 MS-SQL Server 数据库的连接设置完毕。

实例源代码

此实例没有源代码及执行程序。

实例 231　如何连接 MS-Access 数据库

实例目的

使用 C++Builder 编写数据库应用程序时，首先必须解决应用程序如何与数据库进行连接，通常情况下，各种数据库在连接设置上均有不同，甚至采用不同的数据库策略。此编程技巧主要介绍如何与 MS-Access 数据库进行连接。此编程技巧没有实例程序。

实现方法

在 C++Builder 1.0 时，应用程序连接 MS-Access 数据库的惟一方式就是通过 ODBC 数据源，此后曾经有些 C++Builder 版本支持过 MS-Access 95、97。但是，目前的 C++Builder 和 MS-Access 版本似乎又开始不支持了，C++Builder 应用程序连接 MS-Access 数据库的方式又是通过ODBC 数据源来完成。在操作系统中创建ODBC 数据源的步骤如下(以 Windows 2000 Server 操作系统为例)。

（1）单击菜单“开始\设置\控制面板\管理工具\ODBC 数据源”，弹出“ODBC 数据源管理器”对话框。单击“添加”按钮，弹出“创建新数据源”对话框，在“选择您想为其安装数据源的驱动程序”列表框中选择 Driver do Microsoft Access(*.mdb)选项，如图 231-1 所示。



图 231-1

（2）在“创建新数据源”对话框中单击“完成”按钮，弹出“ODBC Microsoft Access 安装”对话框。在“数据源名”中添加 Access 数据源，如 MyAccessODBC，如图 231-2 所示。单击“选择”按钮，弹出“选择数据库”对话框。



图 231-2

（3）在“选择数据库”对话框中选择一个 Access 数据库，如 Custom.mdb，如图 231-3 所示。单击“确定”按钮，返回“ODBC Microsoft Access 安装”对话框，再单击“确定”按钮，返回“ODBC 数据源管理器”对话框。可以发现，在“用户数据源”列表框中已经自动增加了 MyAccessODBC 数据源。单击“确定”按钮，退出系统 ODBC 数据源设置。



图 231-3

（4）启动 BDE Administrator 工具，可以发现在 Databases 中已经自动添加了 MyAccessODBC 节点，如图 231-4 所示。

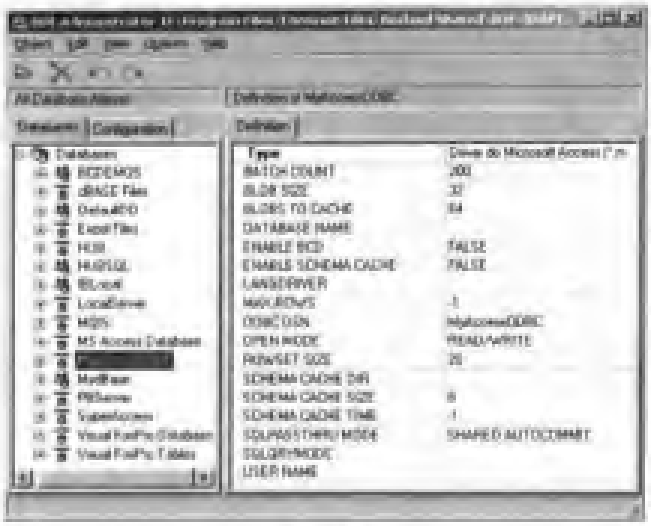


图 231-4

至此，C++Builder 与 MS-Access 数据库的连接设置完毕。

实例源代码

此实例没有源代码及执行程序。

实例 232　如何连接 dBase 数据库

实例目的

使用 C++Builder 编写数据库应用程序时，首先必须解决应用程序如何与数据库进行连接，通常情况下，各种数据库在连接设置上均有不同。此编程技巧主要介绍如何与 dBase 数据库进行连接。此编程技巧没有实例程序。

实现方法

使用 C++Builder 开发本地数据库应用程序时，通常使用 Paradox 数据库，因为 Borland 的 BDE 原本就是 Paradox 数据库的引擎核心，它和 Paradox 数据库的兼容性是最高的。如果出于兼容以前开发的 dBase 数据库的需要，则需要在 C++Builder 的 BDE Administrator 工具中按照以下步骤设定数据库连接。

（1）启动 BDE Administrator 工具，打开 Configuration\Drivers\Native\DBASE 选项，在右边的 Definition 标签页的 LANGDRIVER 中选择 dBASE CHS cp936，以使数据库支持中文数据，如图 232-1 所示。

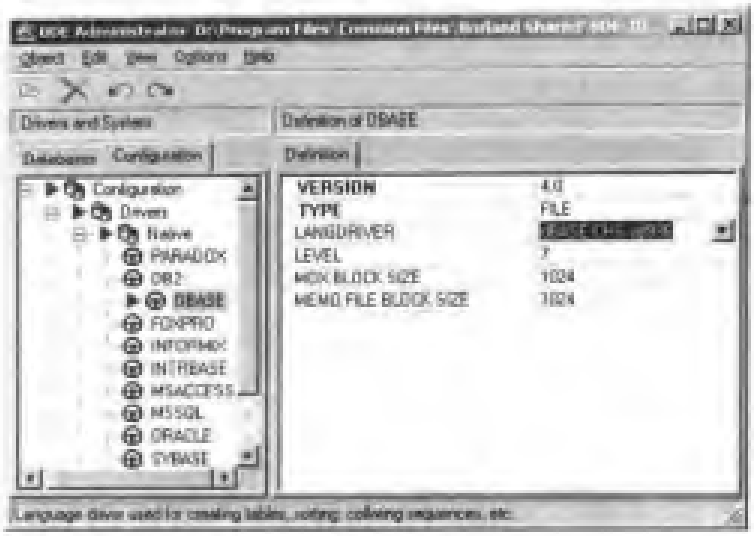


图 232-1

（2）单击左边的 Databases 标签页，选择菜单 Object\New，弹出 New Database Alias 对话框，选择 Standard 选项，单击 OK 按钮。BDE Administrator 工具自动在左边产生一个 STANDARD1 节点，更名为 MydBase。然后在右边的 DEFAULT DRIVER 中选择 DBASE，在 PATH 中指明数据库所在的路径，如图 232-2 所示。



图 232-2

（3）设定完成后，存储刚才设定的参数。右击左边的 MydBase 节点，在弹出的右键菜单中选择 Apply 菜单项即可。然后打开 Database Desktop 工具，单击刚才设定的数据库别名，任意打开一个 DBF 文件测试连接是否成功。在程序中只要指定数据库别名就可以存取数据库中的数据表内容。

实例源代码

此实例没有源代码及执行程序。

实例 233　如何使用 ODBC 连接 Excel 表

实例目的

此实例主要介绍了如何使用 ODBC 数据源操作 Excel 文件的技术。实例程序执行后，效果如图 233-1 所示。

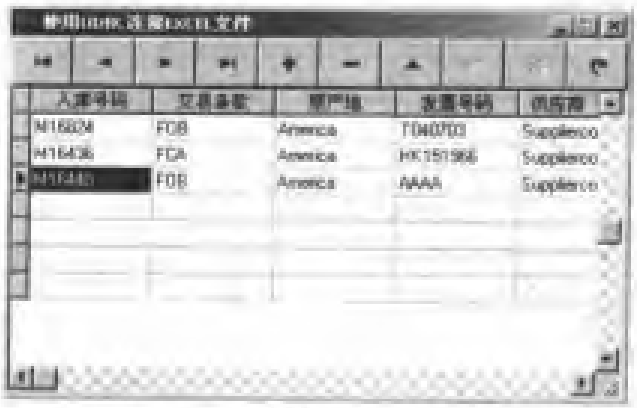


图 233-1

实现方法

在应用程序中操作 Excel 表文件的方式可谓多种多样，如 ADO、OLE、ODBC 等，最常用的是 OLE 方式，这种方式功能最强大，操作也最复杂。最简单的应是 ADO 方式，但实践中经常发现在一些 Excel 表中根本无法实现数据连接。ODBC 是开放式数据库链接标准，不同种类的数据库只需提供各自的 ODBC 驱动程序就可按相同的命令操纵，微软同样为 Excel 提供了 ODBC 驱动程序，可以在程序中像数据库一样存取 Excel 表格。但实际上 ODBC 也存在同样的问题：在定义好 ODBC 数据源后，在控件 Table 的属性 TableName 中总是检索不到表名。通过对 Excel 的分析，终于发现了问题的关键所在：ODBC 的表名并不就是 Excel 的工作表名（如 Sheet1），在 Excel 表格中必须对要求操作的行列区域定义一个“名称”作为数据库的表名，该区域的首行各列必须是字段名（否则首行数据会当成字段名），可以定义多个表名。下面是实例程序使用 ODBC 操作 Excel 表文件全过程的具体步骤。

（1）启动 Excel 创建一个 Excel 文件 SExcel.xls，内容如图 233-2 所示。

（2）在图 233-2 中选择一个范围作为数据库表的内容，选择菜单“插入\名称\定义...”，弹出“定义名称”对话框，在“在当前工作簿中的名称”列表框中增加一个项目 DBTableName 作为数据库的表名，“引用位置”栏中即标明了数据库表的范围，如图 233-3 所示。单击“确定”按钮保存设置结果。



图 233-2



图 233-3

（3）保存并关闭 Excel 文件 SExcel.xls。至此，Excel 数据库表名定义完成。

（4）创建 ODBC 数据源（以 Windows 2000 Server 操作系统为例）。选择系统菜单“开始\程序\管理工具\ODBC 数据源”，弹出“ODBC 数据源管理器”对话框，如图 233-4 所示。



图 233-4

（5）在图 233-4 中单击“添加”按钮，弹出“创建新数据源”对话框，在“选择您想为其安装数据源的驱动程序”列表框中选择 Excel 文件驱动程序 Driver do Microsoft Excel(*.xls)，如图 233-5 所示。



图 233-5

（6）在图 233-5 中单击“完成”按钮，弹出“ODBC Microsoft Excel 安装”对话框，在“数据源名”编辑栏中输入 Excel 文件 SExcel.xls 的 ODBC 数据源名 MyODBCExcel，如图 233-6 所示。



图 233-6

（7）在图 233-6 中单击“选择工作簿”按钮，弹出“选择工作簿”对话框，展开文件夹找到 Excel 文件 SExcel.xls，如图 233-7 所示，单击“确定”按钮，完成 Excel 文件 SExcel.xls 的 ODBC 数据源的创建。



图 233-7

（8）启动 C++Builder，在组件模板的 BDE 标签页下拖动 Database1:TDatabase 组件到 Form1 上，在 Object Inspector 中设置 Database1 的 AliasName 属性为 MyODBCExcel。

LoginPrompt 属性为 false，Connected 属性为 true，DatabaseName 属性为 MyExcel。

（9）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 MyExcel，TableName 属性为 DBTableName，Active 属性为 true。

（10）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（11）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（12）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

最后，编译执行程序即可。注意：程序编译前，必须将刚打开的表关闭，使 Table1 控件的 Active 属性为 false，否则程序运行时报告出错，因为 Excel 表总是被 ODBC 以“独占”方式打开。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 233 如何使用 ODBC 连接 Excel 表”。

实例 234 如何使用 ADO 操作 Access 数据库

实例目的

此实例介绍了使用 ADO 操作 Access 数据库的编程技术。实例程序执行后，效果如图 234-1 所示。



图 234-1

实现方法

ADO（Active Data Object）是微软公司基于 OLE DB 的数据库模型。它实现了一系列

的 COM 接口，通过数据提供者（Data Provider）和数据使用者（ Data Consumer）来实现数据存取。ADO 模型一方面简化了数据存取，例如使用 Jet OLE DB Provider 可以实现 Access 数据库的无 DSN 连接；另一方面，在 ASP 等 Internet 应用中也得到了广泛使用。在 C++Builder 中，使用 ADO 组件操作数据库，就不必在 BDE 管理器中设置 BDE 连接，而是直接在 ADO 连接组件中进行设置。下面是实例程序使用 ADO 组件操作 Access 数据库的示例步骤。

（1）在组件模板的 ADO 标签页中选择 TADOTABLE 组件，在 Object Inspector 面板中单击 ConnectionString 右边的“...”按钮，弹出 Form1->ADOTable1 ConnectionString 对话框，如图 234-2 所示。单击该对话框中的 Build...按钮，弹出“数据链接属性”对话框，在“提供程序”标签页中选择 OLEDB 提供程序 Microsoft Jet 4.0 OLE DB Provider，如图 234-3 所示。

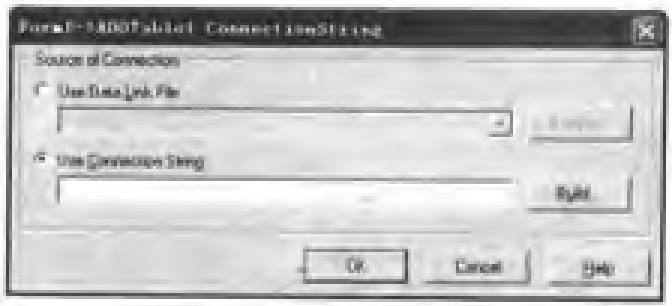


图 234-2



图 234-3

在“数据链接属性/提供程序”对话框中单击“下一步”按钮，“数据链接属性”对话框切换到“连接”标签页，在“选择或输入数据库名称”编辑栏中输入 Access 数据库文件

所在的路径，如图 234-4 所示，单击“测试连接”按钮可以检测数据库的连接情况。单击“确定”按钮返回到 Form1->ADOTable1 ConnectionString 对话框，再单击 OK 按钮保存 ConnectionString 字符串的设置。



图 234-4

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 ADOTable1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 234 如何使用 ADO 操作 Access 数据库”，如果实例程序执行有误，请检查数据库的存放位置。

实例 235 如何定制 DBGrid 组件的列表头

实例目的

此实例主要介绍了在 C++Builder 中定制 DBGrid 组件表头的技术，从而使 DBGrid 组

件显示的数据更具有个性化。实例程序执行后，效果如图 235-1 所示。



图 235-1

实现方法

DBGrid 组件是在 C++Builder 中进行数据库编程使用频率较高的组件之一，它所建立的网格是由行和列组成的，数据集的每条记录显示为一行，数据集的每个字段显示为一列。Columns 属性用于设置该组件列的显示属性，Columns 属性实际上是一个 TDBGridColumns 对象，用来管理和操作网格中的列。在设计时，可以通过列编辑器来建立列对象，并设置其属性即可实现对列进行定制的目标。常用列对象的属性如表 235-1 所示。

表 235-1

属性	功能说明
Alignment	用于设置数据的对齐方式，默认值是 TField 的 Alignment
ButtonStyle	设置为 cbsAuto，表示加上一个组合框；设置为 cbEllipsis，表示加上一个省略号按钮，当用户单击此按钮时将触发 OnEditButtonClick()事件；设置为 cbsNone，表示既没有组合框也没有省略号按钮
Color	用于设置背景颜色，默认值是 TDBGrid 的 Color
DropDownRows	下拉列表框中的项数，默认值是 7
Expanded	如果列对象所对应的字段是 ADT 或数组字段，该属性用于控制是否展开
FieldName	用于指定一个字段，可以让它空着
ReadOnly	设置为 TRUE 表示该列是只读的，不能编辑
Width	用于设置列的宽度，默认值是 TField 的 DisplayWidth
Font	用于设置该列中字符的字体，默认值是 TDBGrid 的 Font
PickList	用于建立一个静态的列表
Title\Alignment	设置列标题的对齐方式，设置为 taLeftJustify 表示左对齐，设置为 taCenter 表示居中，设置为 taRightJustify 表示右对齐
Title\Caption	用于指定列标题的文字，默认值是 TField 的 DisplayLabel
Title\Color	用于设置列标题的背景颜色，默认值是 TDBGrid 的 FixedColor
Title\Font	用于设置列标题的字体，默认值是 TDBGrid 的 TitleFont

默认情况下，DBGrid 组件在载入数据库表中的内容时，它会以数据库表的字段名为表头，以字段的长度为表的列宽。通过在 Object Inspector 中对字段属性进行设置，可以实现对 DBGrid 的表头进行定制。具体步骤如下（以“实例 190　如何创建不写代码的数据库程序”为基础）：

（1）鼠标右击 Form1 上的 DBGrid1 组件，在弹出的右键菜单中选择 Columns Editor...，弹出 Editing DBGrid1->Columns 对话框。

（2）鼠标右击 Editing DBGrid1->Columns 对话框中的空白区域，在弹出的右键菜单中选择 Add All Fields，在 Editing DBGrid1->Columns 对话框的空白区域立即显示了数据库表中的所有字段，如图 235-2 所示。



图 235-2

（3）单击字段名 0-Name，在左下角的 Object Inspector 中设置字段 0-Name 的 Title\Caption 属性为“国名”，Title\Alignment 属性为 taCenter，Title\Color 属性为 alAqua，Width 属性为 100。如果将 Visible 属性设置为 false，该列在程序执行时将会隐藏，单击 Font 右边的“...”按钮，可以对该表头的字体进行设置。

其他字段的设置与此类似，也可以同时选择多个字段，对这些字段的公共部分进行设置。最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 235 如何定制 DBGrid 组件的列表头”。

实例 236　如何统计 DBGrid 组件多选单元格

实例目的

此实例主要介绍如何在 TDBGrid 组件中统计已经选择的多个单元格，并取出单元格中的值。实例程序执行后，效果如图 236-1 所示。



图 236-1

实现方法

C++Builder 的 TDBGrid 组件有一个 Options 属性，该属性主要用于设置 TDBGrid 组件的一些显示样式和控制状态，例如，如果将 dgTitle 属性设置为 false，则将取消标题栏的显示。如果将 TDBGrid 组件的 Options\dgMultiSelect 设置为 true，则可以在 TDBGrid 组件上同时选择多行数据。TDBGrid 组件的 Options 属性中的扩展属性项的意义如表 236-1 所示。

表 236-1

属性	功能说明
dgEditing	允许用户修改网格中的数据
dgAlwaysShowEditor	网格自动处于编辑状态，否则需要按 F2 键才能进入编辑状态
dgTitles	显示字段名或字段对应的标题
dgIndicator	当前行的最左端将显示一个>符号
dgColumnResize	列的宽度可以重新设置
dgColLines	在网格中显示各列之间的垂直分隔线
dgRowLines	在网格中显示各行之间的水平分隔线
dgTabs	可以用 Tab 键和 Shift+Tab 键在列与列之间移动输入焦点
dgRowSelect	用户可以选择一整行，否则就只能选择一个单元
dgAlwaysShowSelection	即使将输入焦点移走，选择的单元仍然保持选择状态
dgConfirmDelete	按 Ctrl+Delete 键删除一行时将显示一个确认对话框让用户确认
dgCancelOnExit	在网格退出时，任何未处理的修改将被取消
dgMultiSelect	用户可以在网格中选择多行，相当于一个复选的列表框

下面是根据此原理实现统计选择多行数据后的单元格值的具体步骤（以 C++Builder 启动时默认产生的 Form1 为主窗口）。

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。设置 Options\dgRowSelect 属性为 true，设置 Options\dgMultiSelect 属性为 true。

（5）在组件模板的 Standard 标签页下拖动 Edit 组件到 Form1 上，在组件模板的 Additional 标签页下拖动 BitBtn 组件到 Form1 上，并为该按钮添加 OnClick 事件处理函数，该函数的实现代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  int i=0;
   int iArea=0;
   for(int i=0;i<this->DBGrid1->SelectedRows->Count;i++)
   {
      this->DBGrid1->DataSource->DataSet->Bookmark=
this->DBGrid1->SelectedRows->Items[i];
      int Area=this->DBGrid1->DataSource->DataSet->FieldValues["Area"];
      iArea+=Area;
   }
   this->Edit1->Text=IntToStr(iArea);
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 236 如何统计 DBGrid 组件多选单元格”。

实例 237 如何设置 DBGrid 组件指定单元格的颜色值

实例目的

此实例介绍了如何设置 DBGrid 组件指定单元格颜色的编程技术。实例程序执行后，效果如图 237-1 所示。



图 237-1

实现方法

在使用 DBGrid 组件操作数据库中的数据时，有时用户要求用特殊的颜色强调显示数据库中的一些重要数据，以达到类似 Excel 软件的单元格着色效果，这就需要处理 DBGrid 组件的 OnDrawColumnCell 事件（从道理上说，应该处理 OnDrawDataCell 事件，但是笔者无论如何都没有着色成功，有的书上也是这么说的，并且 C++Builder 的帮助文件也是这么讲的，但是将其源代码粘贴到该事件中时，始终没有反应）。以下是实例程序实现此技术的具体步骤。

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。并为该组件的 OnDrawColumnCell 事件添加如下所示的实现代码。

```
void __fastcall TForm1::DBGrid1DrawColumnCell(TObject *Sender,
      const TRect &Rect, int DataCol, TColumn *Column,
      TGridDrawState State)
{ //将国家面积大于1000000的数据着色显示
  if(Column->Field->FieldName=='Area')
   if(Column->Field->AsInteger>1000000)
   {
    DBGrid1->Canvas->Brush->Color=clBlue;
    DBGrid1->Canvas->FillRect(Rect);
    SetTextColor(this->DBGrid1->Canvas->Handle,clWhite);
    DrawText(this->DBGrid1->Canvas->Handle,Column->Field->Text.c_str(),
```

```
-1,(RECT*)&Rect,DT_SINGLELINE|DT_VCENTER|DT_CENTER);
    SetTextColor(this->DBGrid1->Canvas->Handle,clBlack);
  }
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 237 如何设置 DBGrid 组件指定单元格的颜色值”。

实例 238 如何在 DBGrid 组件单元格中增加下拉菜单

实例目的

此实例主要介绍了如何在 C++Builder 的 DBGrid 组件单元格中增加下拉菜单的编程技术。实例程序执行后，单击 DBGrid 组件中的单元格，弹出一个下拉菜单供选择，效果如图 238-1 所示。

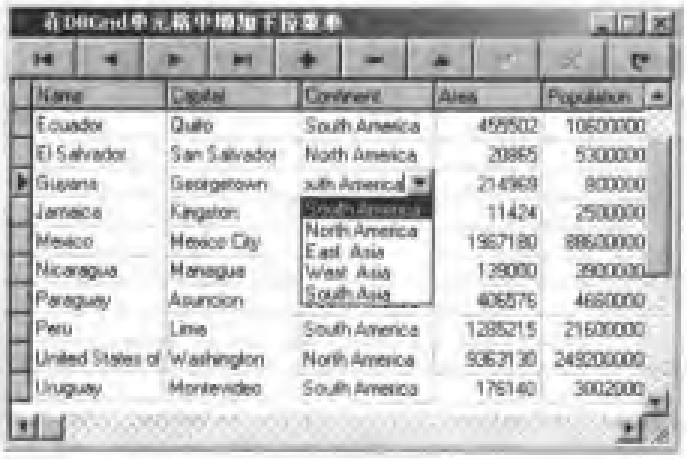


图 238-1

实现方法

DBGrid 组件是 C++Builder 中进行数据库编程使用频率较高的组件之一，默认情况下，DBGrid 组件的单元格自动支持下拉菜单的添加，用户只需要在该组件的列的 PickList 属性中直接增加下拉菜单的菜单项即可。具体步骤如下（以“实例 190 如何创建不写代码的数据库程序”为基础）：

（1）鼠标右击 Form1 上的 DBGrid1 组件，在弹出的右键菜单中选择 Columns Editor...，弹出 Editing DBGrid1->Columns 对话框。

（2）鼠标右击 Editing DBGrid1->Columns 对话框中空白区域，在弹出的右键菜单中选

择 Add All Fields，在 Editing DBGrid1->Columns 对话框的空白区域立即显示了数据库表中的所有字段，如图 238-2 所示。



图 238-2

（3）单击字段名 2-Continent，再单击左下角的 Object Inspector 中的 PickList 属性右边的 TString 按钮，弹出 String List Editor 对话框，在其中的编辑栏中直接添加将在 DBGrid 单元格中增加的下拉菜单项即可，如图 238-3 所示。

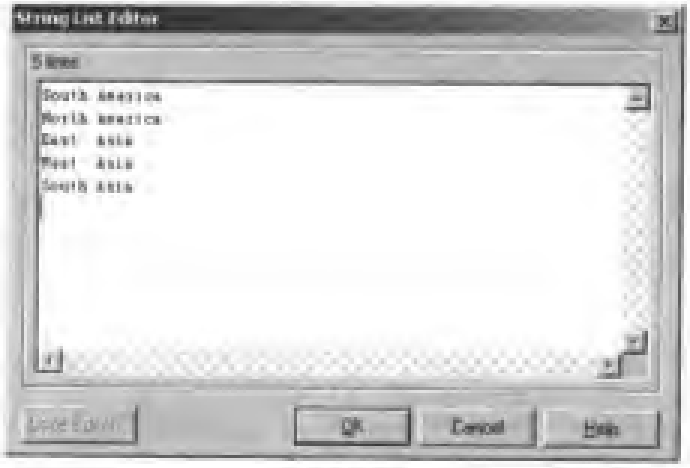


图 238-3

其他字段的设置与此类似，最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 238 如何在 DBGrid 组件单元格中增加下拉菜单”。

实例 239　如何定制 StringGrid 组件指定单元格的颜色

实例目的

此实例主要介绍了如何在 StringGrid 组件中设置指定单元格的文字颜色和背景颜色的编程技术。实例程序执行后，效果如图 239-1 所示。

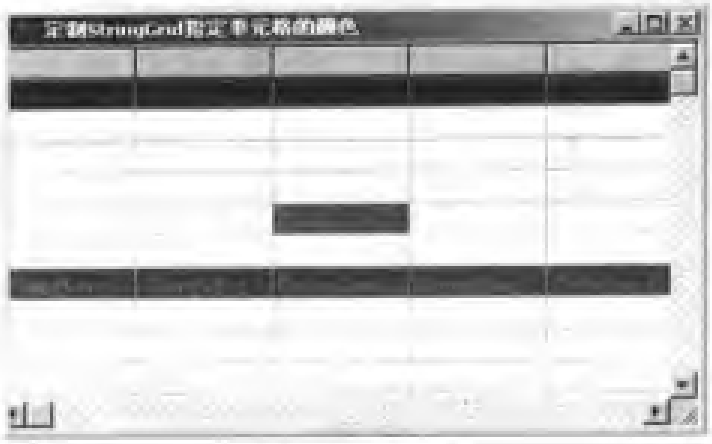


图 239-1

实现方法

C++Builder 的 StringGrid 组件可以设置单元格的背景颜色、文字颜色及字体等。但是美中不足的是，它通常只能直接设置所有单元格的背景颜色、文字颜色及字体，而不能设置指定的行或列及某个单元格的背景颜色、文字颜色及字体。如果要设置指定的行或列及某个单元格的背景颜色、文字颜色及字体，可以在 StringGrid 组件的 OnDrawCell 事件中加入如下所示的代码。

```
void __fastcall TForm1::OnDrawCellStringGrid(TObject *Sender, int ACol,
     int ARow, TRect &Rect, TGridDrawState State)
{
  if(ARow==5&&ACol==2)
  {
    this->StringGrid1->Canvas->Brush->Color =RGB(134,120,100);
    this->StringGrid1->Canvas->FillRect(Rect);
    this->StringGrid1->Canvas->Font->Color=RGB(12,189,244);
    this->StringGrid1->Canvas->TextOutA(Rect.left+3,Rect.top+3,'China");
  }
  if(ARow==7)
  {
    this->StringGrid1->Canvas->Brush->Color =RGB(34,120,100);
    this->StringGrid1->Canvas->FillRect(Rect);
    this->StringGrid1->Canvas->Font->Color=RGB(12,189,244);
    this->StringGrid1->Canvas->TextOutA(Rect.left+3,Rect.top+3,'ChongQing");
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 239 如何定制 StringGrid 组件指定单元格的颜色”。

实例 240　如何获取 StringGrid 组件鼠标指定单元格的值

实例目的

此实例介绍了如何获取 StringGrid 组件中鼠标指定单元格的值的编程技术。实例程序执行后，效果如图 240-1 所示。



图 240-1

实现方法

在 C++Builder 中，获取 StringGrid 组件中鼠标指定单元格的值通常要经过以下 3 个步骤：①使用 StringGrid 组件的 MouseCoord()事件获取单元格在屏幕中的坐标；②使用 TGridCoord 结构装载该坐标；③根据 TGridCoord 坐标在指定的 StringGrid 组件 Cells 中获取对应的值。示例代码如下：

```
void __fastcall TForm1::StringGrid1MouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{//StringGrid 组件的 OnMouseDown 事件
 struct TGridCoord MyGridCoord;
 MyGridCoord=StringGrid1->MouseCoord(X,Y);
 AnsiString StrValue="当前单元格的值为：";
 StrValue+=StringGrid1->Cells[MyGridCoord.X][MyGridCoord.Y];
 MessageBox(Handle,StrValue.c_str(),"信息提示",MB_OK);
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 240 如何获取 StringGrid 组件鼠标指定单元格的值”。

实例 241 如何使用 StringGrid 组件显示数据库查询结果

实例目的

此实例主要演示了从数据库表中读取记录在 StringGrid 组件中显示的编程技巧。实例程序执行后，效果如图 241-1 所示。



图 241-1

实现方法

在使用 C++Builder 编写数据库应用程序时，大多数人通常喜欢使用 TDBGrid 组件来显示对数据库的查询结果。实际上，完全可以使用 StringGrid 组件来显示对数据库的查询结果。因为不论 TQuery 组件包括的 SQL 语句有多复杂，最后产生的结果集一定是一个数据表，根据数据表的 RecordCount、Fields[]及 FieldCount，就可以读取整个结果集数据表。其中，RecordCount 代表数据集有多少条记录，FieldCount 表示数据集有多少个字段，Fields[]存放每个列的值。实例程序根据以上特性编写的对数据库的查询结果在 StringGrid 组件中显示的示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
 int i,j;
 Currency amount;
 Query1->Close();
 Query1->DatabaseName="BCDEMOS";
 Query1->SQL->Text="SELECT * FROM Orders";
 Query1->Prepare();
 Query1->Open();
 StringGrid1->RowCount=Query1->RecordCount;
 StringGrid1->ColCount=Query1->FieldCount;
 for(i=0;i<Query1->RecordCount;i++)
 {
```

```
    for(j=0;j<Query1->FieldCount;j++)
    {
      if(i==0)
      {
        StringGrid1->Cells[j+1][0]=Query1->Fields->Fields[j]->FieldName;
      }
      StringGrid1->Cells[j+1][i+1]=Query1->Fields->Fields[j]->AsString;
      amount+=Query1->FieldByName("AmountPaid")->AsCurrency;
    }
    StringGrid1->Cells[0][i+1]=IntToStr(i+1);
    Query1->Next();
  }
  this->Edit1->Text=CurrToStr(amount);
}
```

以上只演示了为 StringGrid 组件的某一个单元格直接赋值的技术。实际上，为 StringGrid 组件的单元格赋值还可以使用以下两种方式成组赋值。

（1）为某一行单元格赋值，示例代码如下：

```
TStringList *Str=new TStringList();//声明一个字符串列表
Str->Add("12345");  //在字符串列表中增加新值
Str->Add("12346");
Str->Add("12347");
StringGrid1->Rows[2]=Str;//将字符串列表中的值赋给 StringGrid 组件的第三行
delete Str;
```

（2）为某一列单元格赋值，示例代码如下：

```
TStringList *Str=new TStringList();//声明一个字符串列表
Str->Add("12345");  //在字符串列表中增加新值
Str->Add("12346");
Str->Add("12347");
StringGrid1->Cols[2]=Str;//将字符串列表中的值赋给 StringGrid 组件的第三列
delete Str;
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 241 如何使用 StringGrid 组件显示数据库查询结果”。

实例 242　如何禁止数据控制组件与数据集组件同步刷新

实例目的

此实例主要介绍了在遍历数据集的所有记录时，如何禁止数据控制组件同步刷新。实例程序执行后，效果如图 242-1 所示。

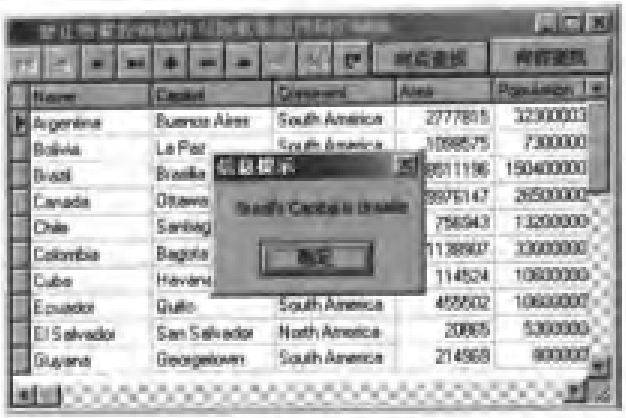


图 242-1

实现方法

C++Builder 的数据集组件 TDataSet 是一个不可见组件，它是 TQuery 、TTable 等所有数据集组件的抽象基类，其大部分属性和方法都是抽象或虚拟的。即数据集组件的方法只有声明没有定义。用户不能直接创建 TDataSet 组件的实例，否则会引起运行时错误。TDataSet 有两个只读的属性 Bof 和 Eof，分别用于判断数据集的当前位置是否达到数据集的开始和末尾。在遍历数据集的所有记录时经常要用到这两个属性：如果 Eof 属性为 true，表示现在已经处于数据集的末尾位置，如果 Bof 属性为 true，表示现在已经处于数据集的开始位置；在数据集进行循环时，每一次操作都应该判断这两个属性的属性值，以避免对不存在的记录进行操作。由于数据集组件通常与数据控制组件通过数据源组件联系着，每当数据集组件的当前位置发生变化时，数据控制组件也会作出相应的改变，并引起数据控制组件的不断闪烁。为了克服这一问题，可以在遍历数据集的所有记录前调用派生数据集的 DisableControls()方法禁止刷新，遍历结束后，再调用派生数据集的 EnableControls()方法恢复刷新。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  this->Table1->DisableControls();
  this->Table1->First();
  while(!this->Table1->Eof)
  {
    AnsiString StrName=this->Table1->FieldValues['Name'];
    if(StrName=='Brazil')
    {
      AnsiString StrInfo='Brazil's Capital is ';
      StrInfo+=this->Table1->FieldValues['Capital'];
      MessageBox(NULL,StrInfo.c_str(),'信息提示',MB_OK);
    }
    this->Table1->Next();
  }
  this->Table1->First();
  this->Table1->EnableControls();
```

```
}
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
   this->Table1->DisableControls();
   this->Table1->Last();
   while(!this->Table1->Bof)
   {
     AnsiString StrName=this->Table1->FieldValues['Name'];
     if(StrName=='Brazil')
     {
       AnsiString StrInfo='Brazil's Capital is ';
       StrInfo+=this->Table1->FieldValues['Capital'];
       MessageBox(NULL,StrInfo.c_str(),'信息提示',MB_OK);
     }
     this->Table1->Prior();
   }
   this->Table1->EnableControls();
}
```

最后编译执行程序即可。可以发现，在使用 DisableControls()方法禁止数据控制组件刷新后，遍历速度明显加快，因为刷新也相当耗费时间。数据集指针移动方法如表 242-1 所示。

表 242-1

方法名称	功能描述
First	移动数据集记录指针到表中的第一条记录位置
Last	移动数据集记录指针到表中的最后一条记录位置
Prior	移动数据集记录指针到当前记录的前一条记录位置
Next	移动数据集记录指针到当前记录的下一条记录位置
MoveBy	向前或向后移动数据集记录指针到用户指定的位置

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 242 如何禁止数据控制组件与数据集组件同步刷新”。

实例 243　如何把数据库表转换成文本文件

实例目的

此实例主要介绍了如何将数据库文件保存为文本文件以及将文本文件读入数据库的编程技术。实例程序执行后，效果如图 243-1 所示。

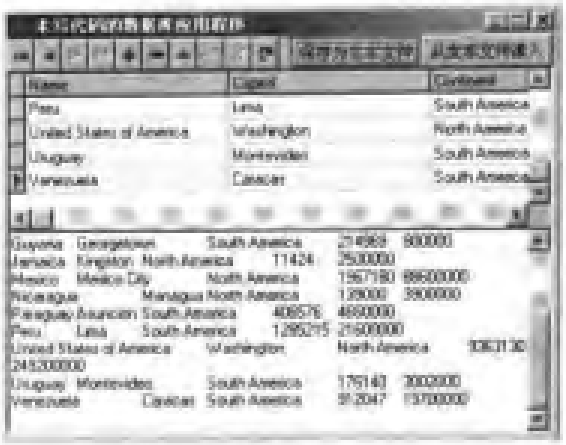


图 243-1

实现方法

文本文件格式是很多不同平台的应用程序互相交换数据的最主要方式，在使用C++Builder编写数据库应用程序时，如果需要将 TTable、 TQuery 数据库存取组件的结果集保存为一个文本文件，可以使用数据结果集的 RecordCount、FieldCount 和 Fields 属性实现。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  this->Memo1->Text="";
  int i=0,j=0;
  Table1->First();
  for(i=0;i<Table1->RecordCount;i++)
  {
    AnsiString StrRecord="";
    for(j=0;j<Table1->FieldCount;j++)
    {
      StrRecord+=Table1->Fields->Fields[j]->AsString+'\t';
    }
    Memo1->Lines->Add(StrRecord);
    Table1->Next();
  }
  try
  {
  this->Memo1->Lines->SaveToFile("C:\\BCText.txt");
  }
  catch(...)
  {
    MessageBox(NULL,"文件已经存在!","信息提示",MB_OK);
    return;
  }
}
```

C++Builder 的 BDE 也提供了 ASCII 数据表的驱动程序，允许将文本文件的内容读入数据库，用户可以在 BDE Administrator 中进行文本文件驱动设置，如图 243-2 所示。

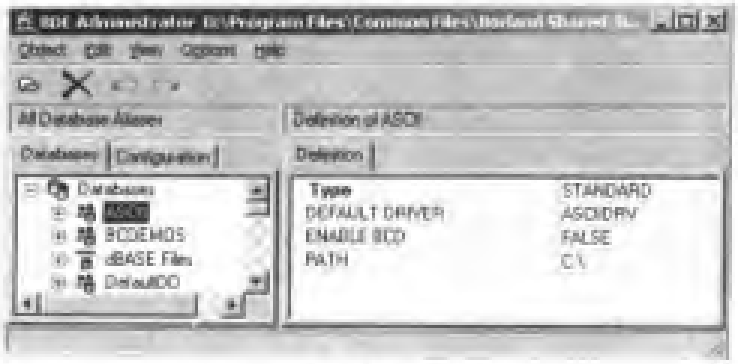


图 243-2

用户也可以直接使用 TTable 组件来直接存取文本文件的内容。示例代码如下：

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
    this->Table1->Close();
    this->Table1->DatabaseName="C:\\";
    this->Table1->TableType=ttASCII;
    this->Table1->TableName="C:\\BCText.txt";
    this->Table1->Open();
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 243 如何把数据库表转换成文本文件”。

实例 244　如何把数据库中的数据写入 Word 文档

实例目的

此实例介绍了直接将数据库中的数据导入 Word 文档的编程技术。实例程序执行后，单击“保存为 Word 文件”按钮，数据库数据将导入新建的 Word 文档，效果如图 244-1 所示。



图 244-1

实现方法

一般情况下，数据库中的数据都是导入 Excel 文件里进行处理，但是 OLE 技术完全支持将数据库中的数据导入到 Word 文档中进行处理。实例程序将数据库中的数据导入 Word 文件的示例步骤如下：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 country.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。

（5）为按钮“保存为 Word 文件”添加如下所示的实现代码：

```
#include"ComObj.hpp"
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  if(this->Edit1->Text.Length()<1)
  {
    MessageBox(Handle,"请输入WORD文件名！",'信息提示',MB_OK);
    return;
  }
  Variant WordObject;
  AnsiString WordDocData;
  String StrCurrentPath=GetCurrentDir()+"\\"+this->Edit1->Text+".Doc";
  WordObject=CreateOleObject("Word.Basic");
  WordObject.Exec(Procedure("FileNew"));
  WordObject.Exec(Procedure("EndOfDocument"));
  this->DBGrid1->DataSource->DataSet->First();
  while(!this->DBGrid1->DataSource->DataSet->Eof)
  {
    WordDocData=this->DBGrid1->DataSource->DataSet->FieldValues["Name"]+
        this->DBGrid1->DataSource->DataSet->FieldValues["Capital"]+
        this->DBGrid1->DataSource->DataSet->FieldValues["Continent"];
    WordDocData+="\n";
    WordObject.Exec(Procedure("Insert")<<WordDocData);
    this->DBGrid1->DataSource->DataSet->Next();
  }
  WordObject.Exec(Procedure("StartOfDocument"));
  WordObject.Exec(Procedure("FileSaveAs")<<StrCurrentPath);
  WordObject.Exec(Procedure("FileClose")<<1);
```

```
    MessageBox(Handle,"WORD 文件保存成功！","信息提示",MB_OK);
}
```

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分　数据库应用\实例 244 如何把数据库中的数据写入 Word 文档”。

实例 245　如何设置数据库字段的输入显示格式

实例目的

此实例主要介绍了在 C++Builder 中，如何定制数据库字段的输入和显示格式的编程技巧。实例程序执行后，效果如图 245-1 所示。



图 245-1

实现方法

在 C++Builder 中，对于一般的文字输入，可以使用组件面板 Additional 标签页中的 MaskEdit 组件实现；而对于数据库字段中的数据录入，由于不能直接使用该组件，对于日期、电话号码等需要控制其录入格式的字段，则往往难以处理。实际上，在 C++Builder 中，对于数据库字段的数据录入格式，也可以通过对字段属性的设置，控制其数据录入格式。示例步骤如下：

（1）在组件模板的 BDE 标签页下拖动 Table1:TTable 组件到 Form1 上，在 Object Inspector 中设置 Table1 的 DatabaseName 属性为 BCDEMOS，TableName 属性为 employee.db，Active 属性为 true。

（2）在组件模板的 DataAccess 标签页下拖动 DataSource1:TDataSource 组件到 Form1 上，在 Object Inspector 中设置 DataSource1 的 DataSet 属性为 Table1。

（3）在组件模板的 DataControls 标签页下拖动 DBGrid1:TDBGrid 组件到 Form1 上，在 Object Inspector 中设置 DBGrid1 的 DataSource 属性为 DataSource1。

（4）在组件模板的 DataControls 标签页下拖动 DBNavigator1:TDBNavigator 组件到 Form1 上，在 Object Inspector 中设置 DBNavigator1 的 DataSource 属性为 DataSource1。以下为设置数据库字段数据录入格式的关键步骤。

（5）右击 Table1:TTable 组件，在弹出的浮动菜单中选择 Fields Editor...菜单，弹出 Form1->Table1 对话框；在其中的空白处右击，在弹出的浮动菜单中选择 Add all fields 项，employee.db 数据库表的所有字段将显示在该对话框中，如图 245-2 所示。



图 245-2

（6）在 Form1->Table1 对话框中，拖动所有字段到 Form1 表单上，Form1 表单上将出现 employee.db 数据库表的所有字段的提示文字和输入编辑框。

（7）在图 245-2 中选择 PhoneExt 字段，按 F11 键，该字段的 Object Inspector 对话框出现，在该对话框中找到 EditMask 属性，单击右边的“...”按钮，弹出 Input Mask Editor 对话框，如图 245-3 所示，在这里就可以完全按照 MaskEdit 组件的操作过程设置数据库字段的数据录入格式。其余字段的数据录入格式与此类似。

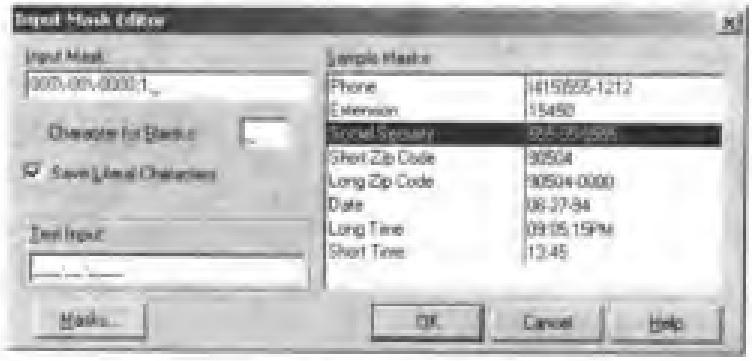


图 245-3

最后编译执行程序即可。

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 245 如何设置数据库字段的输入显示格式”。

实例 246　如何管理数据库程序的错误信息

实例目的

此实例主要介绍了应用程序如何捕捉和处理数据库操作过程中产生的错误的编程技术。实例程序执行后，效果如图 246-1 所示。



图 246-1

实现方法

传统的数据库应用程序错误处理都是由程序员在程序设计时自行处理，然而，在 C/S 模式下，这类程序可以借助数据库管理系统的错误判断功能实现。例如，在数据库写入一条记录时，数据库管理系统会自行检查数据的主键是否有重复的情况发生，如果存在，系统本身会产生一个错误信息，而且不会将数据写入数据库内。当然，这类数据库产生的错误信息最终也会传递到应用程序本身。在使用 C++Builder 编写的数据库应用程序中，当 BDE 收到数据库传出的错误信息时，会将其统一地转换成 BDE 专属的错误码，再传给应用程序，常用的 BDE 错误码如表 246-1 所示。

表 246-1

错误值	错误码	说明
DBIERR_KEYVIOL	9729	主键重复
DBIERR_MINVALERR	9730	小于列的最小值限制
DBIERR_MAXVALERR	9731	大于列的最大值限制
DBIERR_REQDERR	9732	必须输入值的列，不能是空的
DBIERR_FORIEGNKEYERR	9733	找不到一对多关联的主数据表
DBIERR_DETAILRECORDSEXIST	9734	不能修改或删除一对多关联的主数据表记录
DBIERR_MASTERTBLLEVEL	9735	一对多关联的主数据表等级错误
DBIERR_LOOKUPTABLEERR	9736	列值超出 LOOKUP 的范围

续表

错误值	错误码	说明
DBIERR_LOOKUPTBLOPENERR	9737	LOOKUP 数据表打开失败
DBIERR_DETAILTBLOPENERR	9738	一对多关联的子数据表打开失败
DBIERR_MASTERTBLOPENERR	9739	一对多关联的主数据库打开失败
DBIERR_FIELDISBLANK	9740	列是空的

而 C++Builder 数据存取组件 TTable 和 TQuery 本身也提供了 4 个事件函数，专门用来处理这些数据库系统产生的错误信息，分别是 OnDeleteError()、OnEditError()、OnPostError() 和 OnUpdateError()。并且数据库存取组件又都集中放置在数据模块里面，所以这些数据库的错误处理程序也会被分离出来，集中在数据模块内。这意味着可以集中管理数据库的错误处理。下面是实例程序演示数据库应用程序错误处理的示例代码。

```
void __fastcall TForm1::OnPostError(TDataSet *DataSet, EDatabaseError *E,
     TDataAction &Action)
{
   if(dynamic_cast<EDBEngineError*>(E)->Errors[0]->ErrorCode==9729)
   {
     MessageBox(NULL,"主键重复,不能执行!","信息提示",MB_OK);
     return;
   }
}
void __fastcall TForm1::OnDeleteError(TDataSet *DataSet, EDatabaseError *E,
     TDataAction &Action)
{
   if(dynamic_cast<EDBEngineError*>(E)->Errors[0]->ErrorCode==9734)
   {
     MessageBox(NULL,"不能修改或删除一对多关联的主数据表记录!","信息提示",MB_OK);
     return;
   }
}
void __fastcall TForm1::OnUpdateError(TDataSet *DataSet, EDatabaseError *E,
     TUpdateKind UpdateKind, TUpdateAction &UpdateAction)
{
   if(dynamic_cast<EDBEngineError*>(E)->Errors[0]->ErrorCode==9734)
   {
     MessageBox(NULL,"不能修改或删除一对多关联的主数据表记录!","信息提示",MB_OK);
     return;
   }
}
void __fastcall TForm1::OnEditError(TDataSet *DataSet, EDatabaseError *E,
     TDataAction &Action)
{
  int iDBIError=dynamic_cast<EDBEngineError*>(E)->Errors[0]->ErrorCode;
  switch(iDBIError)
  {
    case 9731:
     MessageBox(NULL,"大于列的最大值限制!","信息提示",MB_OK);
```

```
        return;
      case 9732:
        MessageBox(NULL,"主键一定要输入!","信息提示",MB_OK);
        return;
    }
}
```

此外，也可以使用 Try/Catch()块捕捉数据库程序产生的异常。示例如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  try
  {
    //Query1->Close();
    Query1->SQL->Clear();
    Query1->SQL->Text="SELECT * FROM Country.db WHERE Area>1000000";
    Query1->Open();
  }
  catch(EDBEngineError&E)
  {
    AnsiString StrInfo="\n";
    for(int Count=0;Count<E.ErrorCount-1;Count++)
      StrInfo+=E.Errors[Count]->Message+"\n";
    MessageBox(Handle,StrInfo.c_str(),"信息提示",MB_OK);
  }
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 246 如何管理数据库程序的错误信息”。

实例 247　如何获取数据库 BDE 别名

实例目的

此实例介绍了如何使用 Session 组件获取数据库 BDE 别名的编程技术。实例程序运行后，单击“获取数据库别名”按钮，程序立即检索出系统 BDE 已经具有的别名，效果如图 247-1 所示。

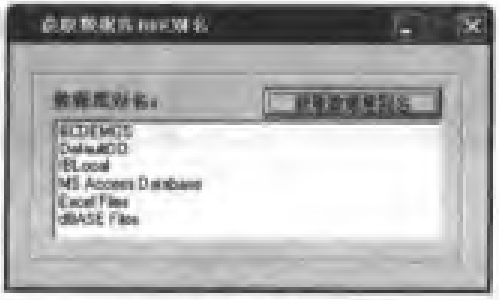


图 247-1

实现方法

使用 C++Builder 编写数据库应用程序时，常常需要知道数据库的 BDE 别名。使用组件面板的 Session 组件（位于组件面板的 BDE 标签页下）的 GetAliasNames()方法可以获取一个已经存在的 BDE 数据库别名的列表。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//获取数据库 BDE 别名
  this->Session1->SessionName='MySession';
  this->Session1->Active=true;
  this->ListBox1->Clear();
  TStringList *MyList=new TStringList();
  this->Session1->GetAliasNames(MyList);
  for(int i=0;i<MyList->Count;i++)
  {
    this->ListBox1->Items->Add(MyList->Strings[i]);
  }
  delete MyList;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 247 如何获取数据库 BDE 别名”。

实例 248　如何获取数据库 BDE 别名参数信息

实例目的

此实例介绍了如何使用 Session 组件获取数据库 BDE 别名参数信息的编程技术。实例程序运行后，单击“获取”按钮，程序立即检索出系统数据库示例程序 BCDEMOS 别名的参数信息，效果如图 248-1 所示。



图 248-1

实现方法

使用 C++Builder 编写数据库应用程序时，可以使用组件面板的 Session 组件（位于组

件面板的 BDE 标签页下)的 GetAliasParams()方法获取指定的 BDE 数据库别名的参数信息。GetAliasParams()方法的声明如下：

```
void __fastcall GetAliasParams(const AnsiString AliasName,
                               Classes::TStrings* List);
```

其中，参数 const AnsiString AliasName 表示数据库 BDE 别名，参数 Classes:: TStrings* List 表示 BDE 别名的参数信息。

以下是实例程序使用 Session 组件的 GetAliasParams()方法以获取 BCDEMOS 别名参数信息的示例代码。

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{//获取数据库 BCDEMOS 别名参数信息
  this->Session1->SessionName='MySession';
  this->Session1->Active=true;
  this->ListBox1->Clear();
  TStringList *MyList=new TStringList();
  this->Session1->GetAliasParams('BCDEMOS',MyList);
  for(int i=0;i<MyList->Count;i++)
  {
    this->ListBox1->Items->Add(MyList->Strings[i]);
  }
  delete MyList;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 248 如何获取数据库 BDE 别名参数信息”。

实例 249　如何获取数据库的工作目录

实例目的

此实例主要介绍了获取当前程序所使用的数据库的工作目录和缺省目录。实例程序执行后，效果如图 249-1 所示。



图 249-1

实现方法

C++Builder 的数据集组件的 Database 属性用于指定要访问的数据库。如果要访问的是本地 dBase 或 Paradox 数据库，则该属性可以设置为 BDE 别名；如果要访问的是本地 Interbase Server 上的表，则该属性可以设置为数据库文件的全路径名。通过 BDE 的 API 函数 DbiGetDirectory()，可以获取数据库的当前目录和缺省目录。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
    char DefaultDir[MAX_PATH+1],WorkDir[MAX_PATH+1];
    if(DbiGetDirectory(this->Table1->DBHandle,true,DefaultDir)!=DBIERR_NONE)
     MessageBox(NULL,"不能取得缺省目录","信息提示",MB_OK);
    else
     this->Edit1->Text=DefaultDir;
    if(DbiGetDirectory(this->Table1->DBHandle,false,WorkDir)!=DBIERR_NONE)
     MessageBox(NULL,"不能取得工作目录","信息提示",MB_OK);
    else
     this->Edit2->Text=WorkDir;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 249 如何获取数据库的工作目录”。

实例 250 如何获取数据库驱动程序名称

实例目的

此实例介绍了如何使用 Session 组件获取数据库驱动程序名称的编程技术。实例程序运行后，单击“获取数据库驱动程序名称”按钮，程序立即检索出系统 BDE 支持的数据库类型，效果如图 250-1 所示。



图 250-1

实现方法

使用 C++Builder 编写数据库应用程序时，常常需要知道系统支持的数据库类型（即数据库驱动程序）。使用组件面板的 Session 组件（位于组件面板的 BDE 标签页下）的 GetDriverNames()方法可以获取系统已经存在的数据库驱动类型的列表。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //获取数据库驱动程序名称
   this->Session1->SessionName="MySession";
   this->Session1->Active=true;
   this->ListBox1->Clear();
   TStringList *MyList=new TStringList();
   this->Session1->GetDriverNames(MyList);
   for(int i=0;i<MyList->Count;i++)
   {
      this->ListBox1->Items->Add(MyList->Strings[i]);
   }
   delete MyList;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 250 如何获取数据库驱动程序名称”。

实现方法

使用 C++Builder 编写数据库应用程序时，常常需要知道系统支持的数据库类型（即数据库驱动程序）。使用组件面板的 Session 组件（位于组件面板的 BDE 标签页下）的 GetDriverNames()方法可以获取系统已经存在的数据库驱动类型的列表。示例代码如下：

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ //获取数据库驱动程序名称
   this->Session1->SessionName="MySession";
   this->Session1->Active=true;
   this->ListBox1->Clear();
   TStringList *MyList=new TStringList();
   this->Session1->GetDriverNames(MyList);
   for(int i=0;i<MyList->Count;i++)
   {
      this->ListBox1->Items->Add(MyList->Strings[i]);
   }
   delete MyList;
}
```

实例源代码

有关程序项目文件全部源代码及执行程序，请参考配套光盘“源代码\第六部分 数据库应用\实例 250 如何获取数据库驱动程序名称”。