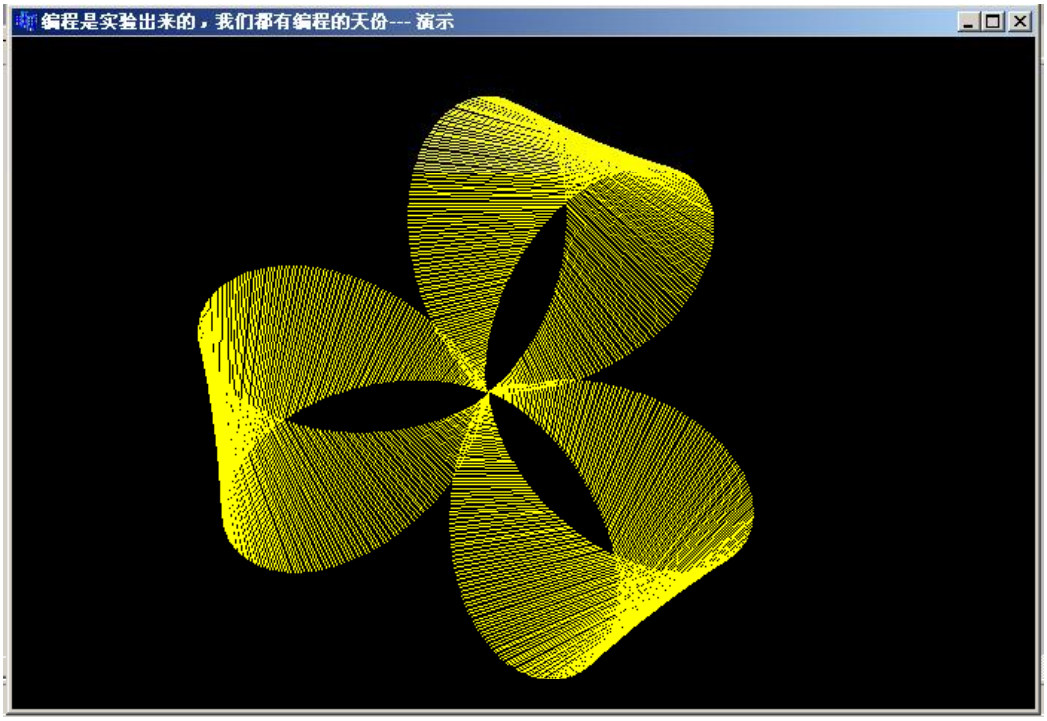
C++ Builder 6.0 界面实例开发

蔡军生 2002-11-25 深圳
caimouse1976@sina.com

实例 1　　界面图案演示



实例目标

本实例作来本书的第一个实例，主要让大家轻松地学习本书，让大家先有一个感觉，编程其实是一个很美的事情。

实现技术

主要用两函数 MoveTo 和 LineTo 实现。设置窗口的颜色为 clNone。

代码如下：

```
void __fastcall TfrmMain::FormPaint(TObject *Sender)
{
    double x1,y1,x2,y2;
    const int nScale = 100;

    Canvas->Pen->Color = clYellow;//设置 FORM 界面的画笔颜色。

    for (int i=0; i<720; i++)
    {
        double dAngle = i*M_PI/360;//M_PI 在 math.h 里。
        double E = nScale*(1+sin(3*dAngle));
        x1 = 320+E*cos(dAngle);
        x2 = 320+E*cos(dAngle+M_PI/5);
        y1 = 240+E*sin(dAngle);
        y2 = 240+E*sin(dAngle+M_PI/5);
```

```
        Canvas->MoveTo(x1,y1);//移到 x1,y1 位置。
        Canvas->LineTo(x2,y2);//从当前位置画直线到 x2,y2.

    }

}
```

实现步骤

实现步骤很简单，用 CB6.0 创建一个应用程序，把它保存为 exp1。然后设置窗口的背影颜色为 clNone，设置 FormPaint 事件函数，并加入以上代码就行了。

实例 2　　创建标题栏在左边的窗口界面

实例目标

实现左边标题，能拖动的窗口。如下图所示：

实现技术

第一步先指明窗口 BorderStyle 为 bsNone，这样就不能拉伸窗，没有标题和边框。第二步就是在 FormPaint(TObject *Sender)函数里面画上标题和边框。这个函数响应消息 OnPaint，当出现重画窗口时调用。第三步就是响应 WM_NCHITTEST 消息，以便拖动窗口。当返回这个消息的结果为 HTCAPTION 时，WINDOWS 就认为鼠标在标题框内，所以能拖动窗口移动。

实现步骤

第一步创建程序，在 Object Inspector 中设置 BorderStyle 为 bsNone。

第二步添加 DrawTitle()，代码如下：

```
void __fastcall TfrmTitle::DrawTitle(void)
{
    RECT rc;
    //左边标题栏。
    ::SetRect(&rc,0,0,nTitleWidth,ClientHeight);
    Canvas->Brush->Color = clBlue;
    Canvas->FillRect(rc);
    //右边边界。
    ::SetRect(&rc,ClientWidth-2,0,ClientWidth,ClientHeight);
    Canvas->Brush->Color = clBlue;
    Canvas->FillRect(rc);
    //上面边界。
    ::SetRect(&rc,0,0,ClientWidth,2);
    Canvas->Brush->Color = clBlue;
    Canvas->FillRect(rc);
    //下面边界。
    ::SetRect(&rc,0,ClientHeight-2,ClientWidth,ClientHeight);
    Canvas->Brush->Color = clBlue;
```

```
    Canvas->FillRect(rc);
    //设置 ICON 位置。
    ImageIcon->Left = 0;
    ImageIcon->Top =    ClientHeight - ImageIcon->Height;
    //输出标题。
    char* msg=Caption.c_str();

    LOGFONT fontRec;
    memset(&fontRec,0,sizeof(LOGFONT));
    fontRec.lfHeight = -13;
    fontRec.lfWeight = FW_NORMAL;
    fontRec.lfEscapement = 900; //字体旋转 90 度。
    lstrcpy(fontRec.lfFaceName,"宋体");

    HFONT hFont=CreateFontIndirect(&fontRec);//创建字体。
    HFONT hOld=::SelectObject(Canvas->Handle,hFont);//选中字体。

    ::SetRect(&rc,0,0,nTitleWidth,ClientHeight);
    ::SetTextColor(Canvas->Handle,RGB(255,255,255));//设置字体的颜色。
    //输出标题。
    ::TextOut(Canvas->Handle,3,ClientHeight - ImageIcon->Height,msg,lstrlen(msg));

    ::SelectObject(Canvas->Handle,hOld);//恢复。
    ::DeleteObject(hFont);

}
```

第三步添加 OnNcHitTest(TMessage& tMsg)函数，处理 WM_NCHITTEST 消息。

```
void TfrmTitle::OnNcHitTest(TMessage& tMsg)
{
    TPoint pt;
    //取得鼠标位置。
    pt.x=LOWORD(tMsg.LParam);
    pt.y=HIWORD(tMsg.LParam);
    pt =ScreenToClient(pt);//转换为客户坐标。

    RECT rc;
    ::SetRect(&rc,0,0,nTitleWidth,ClientHeight);

    if(PtInRect(&rc,pt))//是否在自画的标题框内。
        tMsg.Result = HTCAPTION;//返回在标题栏内结果。
    else
        DefaultHandler(&tMsg);//让窗口缺省消息处理函数处理。
}
```

实例 3　　创建超级连接界面

实例目标

本实例介绍如何创建的热点连接（hotlink），该连接指向一定的 Internet 资源。如果用户单击超级连接，就可以打开连接的文本。该热点连接初始为蓝色，当鼠标移动到热点连接上时，就会变成绿色，并且鼠标变为小手形状，当点击这些热点连接时，就能可以访问该连接的 Internet 资源。如果热点为粉红色，表示已经访问过，程序的界面如下：

实现技术

主要有几个要点，第一个就是当鼠标移到上面时，要设置字体不同的颜色。第二个就是要设置鼠标形状。第三个就是当按下时，打开浏览器连接网页。第四个就是当已经连接了就要改为粉红色。

实现步骤

第一步就创建一个 Label 按件，然后设置字体属性为有下划线，并设置字体颜色为蓝色。第二步就设置鼠标移动的消息响应。第三步就是实现按下连接后开始连接网站。

详细代码如下：

```
//---------------------------------------------------------------------------
__fastcall TfrmHyperLink::TfrmHyperLink(TComponent* Owner)
    : TForm(Owner)
{
    m_bHyperLinked = false;//设置为没有按下连接。
    lbHyperLink->Cursor = crHandPoint;//设置光标为手掌形状。
}
//---------------------------------------------------------------------------
//  鼠标进入响应。
//
void __fastcall TfrmHyperLink::lbHyperLinkMouseEnter(TObject *Sender)
{
    if(!m_bHyperLinked)//如果没有按下设置颜色和手
    {
        lbHyperLink->Font->Color = clGreen;//设置字体颜色为绿色。
    }
}
//---------------------------------------------------------------------------
//  鼠标离开响应。
```

```
//
void __fastcall TfrmHyperLink::lbHyperLinkMouseLeave(TObject *Sender)
{
      if(!m_bHyperLinked)//如果没有按下设置颜色和手
      {
            lbHyperLink->Font->Color = clBlue;//设置字体颜色为蓝色
      }

}
//---------------------------------------------------------------------------
//  鼠标弹起。
//
void __fastcall TfrmHyperLink::lbHyperLinkMouseUp(TObject *Sender,
         TMouseButton Button, TShiftState Shift, int X, int Y)
{
      lbHyperLink->Font->Color = clFuchsia;//设置字体颜色为粉红色
}
//---------------------------------------------------------------------------
//  鼠标按下。
//
void __fastcall TfrmHyperLink::lbHyperLinkMouseDown(TObject *Sender,
         TMouseButton Button, TShiftState Shift, int X, int Y)
{
      String strUrl = lbHyperLink->Caption; //("www.yahoo.com");
      m_bHyperLinked = true;
      ShowMessage("你已经按下连接");
      //打开网站连接。
      ShellExecute(NULL,"open",strUrl.c_str(),NULL,NULL,SW_SHOWNORMAL);
}
//---------------------------------------------------------------------------
```

实例 4　　创建不规则窗口界面

实例目标

本例实现的目标是实现如右边界面。这个是一个不规则窗口界面，跟普通那种窗口相比，具有尝心悦目感觉。

实现技术

在这个例子中关键是区域 API 使用。CreatePolygonRgn　API 函数是用来创建多边形的区域。SetWindowRgn API 函数是用来设置窗口的区域。

实现步骤

第一步，先创建一组图像区域要用的数组。

第二步，加载图像，读入区域组，然后根据数组设置图像的区域，然后显示出来。第三步，就响应鼠标消息。

主要源程序如下：

```
//---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
```

```
        : TForm(Owner)
{
    m_nMouseOnMe = -1;
}
//---------------------------------------------------------------------------
//
//创建窗口时，创建图像对象。
//
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    pBitmap =    new Graphics::TBitmap;//用来加载背景图像。
    text      =  new Graphics::TBitmap;//用来加载文字图像。
    minimize=    new Graphics::TBitmap;//用来加载最小化图像。
    close    =   new Graphics::TBitmap;//用来加载关闭图像
    help     =   new Graphics::TBitmap;//用来加载帮助图像

    pBitmap->LoadFromFile("sat_dish.bmp");//加载背景图像。
    text->LoadFromFile("text.bmp");//加载文字图像
    minimize->LoadFromFile("minimize.bmp");//加载最小化图像。
    close->LoadFromFile("close.bmp");//加载关闭图像
    help->LoadFromFile("help.bmp");//加载帮助图像

    pBitmap->Height =    520;//设置背景显示高度。
    pBitmap->Width   =   520;//设置背景显示宽度。

    Count    =   -1;//区域有多少个点。

    oldX     =   0;
    oldY     =   0;

    Moving   =   false;//是否能移动。
    tempRgn =    SKIN;//设置为多边形区域显示方式。

    LoadPath("Contour.txt");//读入多边形数组。
    SetRegion(SKIN);//设置区域。
}
//---------------------------------------------------------------------------
//   显示背景图像。
//
//
/* BOOL BitBlt(
    HDC hdcDest, //  指目标 DC
    int nXDest,    // x-coord of destination upper-left corner
    int nYDest,    // y-coord of destination upper-left corner
```

```
        int nWidth,    // width of destination rectangle
        int nHeight, // height of destination rectangle
        HDC hdcSrc,    // handle to source DC
        int nXSrc,     // x-coordinate of source upper-left corner
        int nYSrc,     // y-coordinate of source upper-left corner
        DWORD dwRop    // raster operation code
        );
*/
void __fastcall TMainForm::FormPaint(TObject *Sender)
{
        Source = pBitmap->Canvas->Handle;//取得图像句柄。
        Destination = MainForm->Canvas->Handle;//取得窗口句柄。
        BitBlt(Destination,0,0,520,520,Source,0,0,SRCCOPY);//显示图像在窗口里。
}
//---------------------------------------------------------------------------
// 读入区域多边形数组。
//
//
void __fastcall TMainForm::LoadPath(AnsiString FileName)
{
    FILE *fp = NULL;
    TPoint Point;
    AnsiString Error;
    if((fp = fopen(FileName.c_str(),"r"))==NULL)//打开文件。
    {
            Error="Error opening file:"+FileName+".";//出错提示信息。
            MessageBox(MainForm->Handle,Error.c_str()," 区 域 API 界 面 程 序
",MB_TASKMODAL);
            Application->Terminate();
    }

    while( !( feof( fp ) ) )//读入数据。
    {
            fscanf(fp,"%d",&Point.x);//以地十进制读入区域坐标。
            fscanf(fp,"%d",&Point.y);
            Path[++Count] = Point;//保存到数组。
    }
    fclose(fp);//关闭文件。
}
//---------------------------------------------------------------------------
// 调用 CreatePolygonRgn API 函数来创建区域。
// 如果 Mode==SKIN 就是创建多边形的区域。
// 如果 Mode==FORM 就是创建矩形的区域。
//
```

```
//---------------------------------------------------------------------------
void __fastcall TMainForm::SetRegion(int Mode)
{
    HRGN Rgn = NULL;
    if(Mode == SKIN)//创建多边形的区域
    {
        Rgn = CreatePolygonRgn(&Path[0],Count,ALTERNATE);
        SetWindowRgn(MainForm->Handle,Rgn,true);
    }
    else if(Mode == FORM)//创建矩形的区域
    {
        Rgn = CreateRectRgn(0,0,MainForm->Width,MainForm->Height);
        SetWindowRgn(MainForm->Handle,Rgn,true);
    }
    DeleteObject(Rgn);
    MainForm->OnPaint(NULL);
}
//---------------------------------------------------------------------------
//响应窗口鼠标按下事件。
//
void __fastcall TMainForm::FormMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    if(Button == mbRight)//如果为右键，就是移动窗口。
    {
        //记录当前位置。
        oldX = X;
        oldY = Y;
        //设置移动为 true.
        Moving = true;
        //设置鼠标为响四周可以移动图标。
        MainForm->Cursor = crSizeAll;
    };

    if(Button == mbLeft)//如果为左键，检查是否各种功能。
    {
        if(Hover(X,Y)==_TEXT)//区域切换。
        {
            if(tempRgn==SKIN)
            {
                SetRegion(FORM);
                tempRgn=FORM;
            }
            else
```

```
                {
                    SetRegion(SKIN);
                    tempRgn=SKIN;
                }
            }
            else if(Hover(X,Y) == CLOSE)//关闭
                Application->Terminate();
            else if(Hover(X,Y) == MINIMIZE)//最小化。
                MainForm->FormClick(NULL);
            else if(Hover(X,Y) == HELP)//打开帮助
                Help();
    };//if(Button==mbLeft)
}
//---------------------------------------------------------------------------
// 当鼠标移动时响应。
//
void __fastcall TMainForm::FormMouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
    curentX=X;//保存当前坐标。
    curentY=Y;

    if(Moving)//是移动
    {
        MainForm->Left += (X-oldX);//取得按右键时鼠标移动大小。
        MainForm->Top += (Y-oldY);
    }
    else
    {
        MouseOnMe(X,Y);
     };//else

}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FormMouseUp(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    if(Button==mbRight)//当右键弹起时，移动结束。
    {
        Moving = false;
        MainForm->Cursor = crDefault;
    }
    MainForm->OnPaint(NULL);
}
```

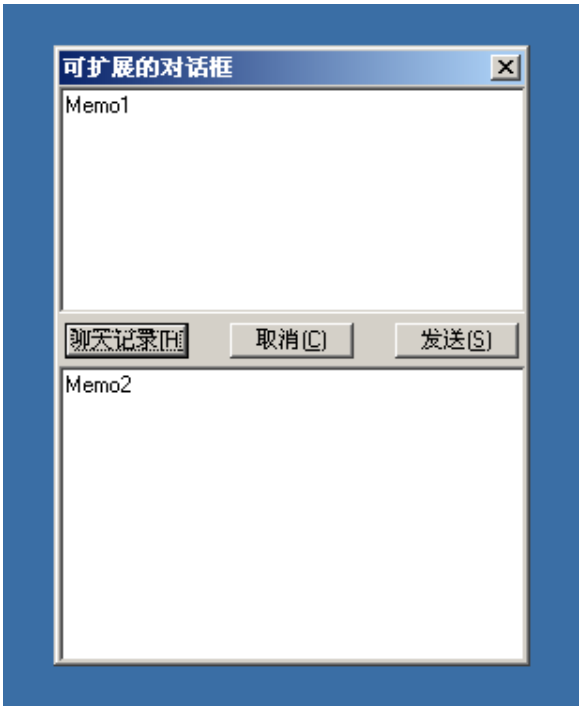
```
//---------------------------------------------------------------------------
// 判断鼠标是否在按钮上面.
//---------------------------------------------------------------------------
#pragma warn -rvl
int __fastcall TMainForm::Hover(int X,int Y)
{
    if((X>170) && (X<350) && (Y>240) && (Y<270)) return _TEXT;
    if((X>340) && (X<375) && (Y>105) && (Y<115)) return MINIMIZE;
    if((X>305) && (X<350) && (Y>60) && (Y<105)) return CLOSE;
    if((X>315) && (X<345) && (Y>365) && (Y<395)) return HELP;
}
//---------------------------------------------------------------------------
//当单击窗体时响应.
//
void __fastcall TMainForm::FormClick(TObject *Sender)
{
      if(Hover(curentX,curentY) == MINIMIZE)//如果是最小化.
            Application->Minimize();
}
//---------------------------------------------------------------------------
// 打开帮助.
//---------------------------------------------------------------------------
void __fastcall TMainForm::Help(void)
{
      ShowMessage("打开帮助文档！");
}
//---------------------------------------------------------------------------
//
//
void __fastcall TMainForm::MouseOnMe(int X, int Y)
{
            Destination = MainForm->Canvas->Handle;//取得窗口句柄。
            if(Hover(X,Y) == _TEXT)//当在 Skin/Form 上面
            {
                  Source = text->Canvas->Handle;
                  BitBlt(Destination,170,240,2*180,2*30,Source,0,0,SRCCOPY);
                  MainForm->Cursor = crHandPoint;
            }
            else if(Hover(X,Y) == MINIMIZE)//当在最小化按钮上面
            {
                  Source=minimize->Canvas->Handle;
                  BitBlt(Destination,340,105,35,10,Source,0,0,SRCCOPY);
                  MainForm->Cursor = crHandPoint;
            }
```

```
        else if(Hover(X,Y)==CLOSE)//当在关闭按钮上面
        {
            Source=close->Canvas->Handle;
            BitBlt(Destination,305,60,45,45,Source,0,0,SRCCOPY);
            MainForm->Cursor = crHandPoint;
        }
        else if(Hover(X,Y)==HELP)//当在帮助按钮上面。
        {
            Source=help->Canvas->Handle;
            BitBlt(Destination,315,365,30,30,Source,0,0,SRCCOPY);
            MainForm->Cursor = crHandPoint;
        }
        else//不在任何按钮上面。
        {
            MainForm->Cursor = crDefault;
            MainForm->OnPaint(NULL);
        }
}
```

实例 5　　创建可扩展对话框界面

实例目标

实现目标就是可以扩展的对话框，跟 QQ 的对话一样。对于一个复杂的对话框而言，大家都想能够简洁明快的解决方法。有时候一些控件被频繁地使用，而另外一些很少使用。这时，如果提供一种可收缩的解决方案，来扩展或者收缩对话框，使一些常用的控件处于可见状态，而另外一些处于不可见状态，一旦需要时，就可点击按钮打开这些控件。如下图所标：



实现技术

实现关键技术主要用到 SetWindowPos API 函数，设置了两次显示窗口的大小不一样。SetWindowPos API 函数参数说明如下：

```
BOOL SetWindowPos(
    HWND hWnd,      // 指向窗口的句柄
    HWND hWndInsertAfter,    // 窗口显示顺序
    int X,      // 水平方向坐标
    int Y,      // 垂直方向坐标
    int cx,     // 宽度
    int cy,     // 高度
    UINT uFlags  //指明那些功能不用
   );
```

实现步骤

第一步建窗口和控件。

第二步在下面的函数添加代码。

```
//---------------------------------------------------------------------------
__fastcall TfrmExpendDialog::TfrmExpendDialog(TComponent* Owner)
    : TForm(Owner)
{
    m_bExpend = true;//初始化为最小对话框。
}
//---------------------------------------------------------------------------
//
//
void __fastcall TfrmExpendDialog::FormShow(TObject *Sender)
{
    m_nWidth = frmExpendDialog->Width;//保存窗口的宽度
    m_nHeigh = frmExpendDialog->Height;//保存窗口的高度
    btnExpendClick(this);//设置为最小的对话框。
}
```

```
//---------------------------------------------------------------------------
//
//
void __fastcall TfrmExpendDialog::btnExpendClick(TObject *Sender)
{
	m_bExpend = !m_bExpend;//交替变化。
	if( m_bExpend )
	{	//扩展对话框。
		SetWindowPos(	frmExpendDialog->Handle,NULL,
				0,0,m_nWidth,m_nHeigh,
				SWP_NOMOVE | SWP_NOZORDER);
	}
	else
	{
		int nHeigh = btnExpend->Top + btnExpend->Height + 25;//
		//缩小对话框。
		SetWindowPos(	frmExpendDialog->Handle,NULL,
					0,0,m_nWidth,nHeigh,
					SWP_NOMOVE | SWP_NOZORDER);
	}
}
//---------------------------------------------------------------------------
//
//
void __fastcall TfrmExpendDialog::btnCancleClick(TObject *Sender)
{
	Close();//关闭窗口。
}
//---------------------------------------------------------------------------
```

实例 6　　创建 NEO Skin 窗口界面

实例目标

本实例创建如 NEO 界面的程序，但它也是一个可换皮肤的程序。当你设置不同的图片时，就可以变成不同的界面。界面如下所示：

实现技术

本实例的实现中，用到了很多图像控件。灵活地运用图像控件，可以写出很漂亮的界面程序。也用到消息处理函数，处理有焦点和无焦点时，图像的更换。最关键的就是处理好图像控件的坐标和大小的变化。

实现步骤

第一步，重载 CreateParams 函数改变窗口的类型。第二步，设置各种 Panel 和图像控件。第三步就是处理当窗口变化时，改变图像控件的位置和大小。

具体实现，关键源程序如下：

```
//---------------------------------------------------------------------------
__fastcall TfrmNeo::TfrmNeo(TComponent* Owner)
    : TForm(Owner)
{
    Color = (TColor)0x00CFCFCF;
    FCaptionMouseDown = false;
    lbCaption->Caption = TForm::Caption;
    //设置右下角的图片位置跟右边的图片对齐。
```

```
    ImageBottomRight->Left = PanelBottom->Width - ImageBottomRight->Width;
    //设置下面图片宽度。
    ImageBottom->Width = ImageBottomRight->Left - ImageBottom->Left;
    //设置是否有最小化，最大化按钮。
    ImageMin->Visible = (BorderIcons.Contains(biMinimize));
    ImageMax->Visible = (BorderIcons.Contains(biMaximize));
    //设置最小化，最大化，关闭和恢复按钮的位置。
    ImageMin->Left = PanelTop->Width - 58;
    ImageMax->Left = PanelTop->Width - 40;
    ImageRestore->Left = PanelTop->Width - 40;
    ImageClose->Left = PanelTop->Width - 22;
}
//---------------------------------------------------------------------------
// 当窗口得到焦点消息 WM_ACTIVATE 时，就调用函数，改变标题图片。
//
void TfrmNeo::OnActivateMessage(TMessage& tMsg)
{
    switch(tMsg.WParamLo)
    {
        case WA_ACTIVE:
        case WA_CLICKACTIVE://激活。
        {
            ImageTopLeft->Picture = ImageFTopLeft->Picture;
            ImageTopCenter->Picture = ImageFTopCenter->Picture;
            ImageTopRight->Picture = ImageFTopRight->Picture;
            PanelTop->Refresh();
            break;
        }

        case WA_INACTIVE://失去焦点时。
        {
            ImageTopLeft->Picture = ImageLTopLeft->Picture;
            ImageTopCenter->Picture = ImageLTopCenter->Picture;
            ImageTopRight->Picture = ImageLTopRight->Picture;
            break;
        }
    }//end of switch
}
//
//  当顶 Panel 改变大小时，就是窗口改变了大小。
//
void __fastcall TfrmNeo::PanelTopResize(TObject *Sender)
{
    //设置最顶右边的图像位置。
```

```
    ImageTopRight->Left = PanelTop->Width - ImageTopRight->Width;
    //设置最顶中间的图像位置。
    ImageTopCenter->SendToBack();//设置为显示顺序。
    //位置。
    ImageTopCenter->Left = ImageTopLeft->Left + ImageTopLeft->Width;
    //宽度。
    ImageTopCenter->Width = ImageTopRight->Left - ImageTopCenter->Left;

    //设置标题栏宽度。
    lbCaption->Width = ImageTopCenter->Width;
    //设置最小化，最大化，关闭按钮的位置。
    ImageMin->Left = PanelTop->Width - 58;
    ImageMax->Left = PanelTop->Width - 40;
    ImageRestore->Left = PanelTop->Width - 40;
    ImageClose->Left = PanelTop->Width - 22;
    PanelTop->Update();
}
//
// 改变窗口的类型。
//
void __fastcall TfrmNeo::CreateParams(TCreateParams& Params)
{
    TForm::CreateParams(Params);//设置缺省的参数。

    Params.Style |= WS_POPUP;//修改为弹出式窗口。
    Params.Style ^= WS_DLGFRAME;//没有边框。
}
//
// 当按下标题栏时，准备移动窗口。
//
void __fastcall TfrmNeo::lbCaptionMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    FCaptionMouseDown = true;
    FCaptionMouseX = lbCaption->Left + X;
    FCaptionMouseY = lbCaption->Top + Y;
}
//---------------------------------------------------------------------------
//  取消息移动窗口。
//
void __fastcall TfrmNeo::lbCaptionMouseUp(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    FCaptionMouseDown = false;
```

```
}
//---------------------------------------------------------------------------
// 当鼠标移动时，开始移动窗口。
//
void __fastcall TfrmNeo::lbCaptionMouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{
    TPoint MousePos;
    if (FCaptionMouseDown && WindowState != wsMaximized)
    {
        GetCursorPos(&MousePos);
        Top = MousePos.y - FCaptionMouseY;
        Left = MousePos.x - FCaptionMouseX;
    }
}
//---------------------------------------------------------------------------
//
//
void __fastcall TfrmNeo::PanelBottomResize(TObject *Sender)
{
    //设置右下角的图片位置跟右边的图片对齐。
    ImageBottomRight->Left = PanelBottom->Width - ImageBottomRight->Width;
    //设置下面图片宽度。
    ImageBottom->Width = ImageBottomRight->Left - ImageBottom->Left;
}
//---------------------------------------------------------------------------

void __fastcall TfrmNeo::SpeedButton1Click(TObject *Sender)
{
    ShowMessage("按下文件菜单");
}
//---------------------------------------------------------------------------

void __fastcall TfrmNeo::SpeedButton2Click(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------------
// 响应最小化
//
void __fastcall TfrmNeo::ImageMinClick(TObject *Sender)
{
    if (Application->MainForm == this)
        Application->Minimize();
```

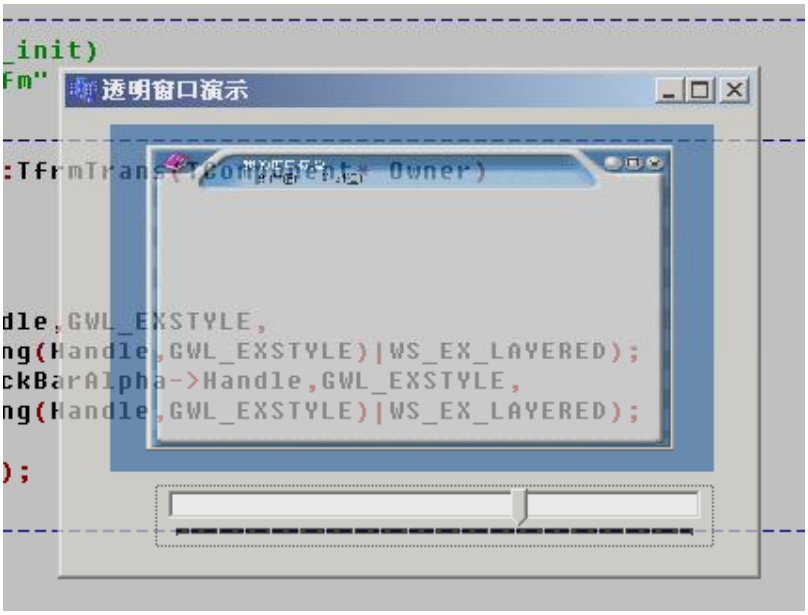
```
        else
            WindowState = wsMinimized;
}
//---------------------------------------------------------------------------
// 响应关闭。
//
void __fastcall TfrmNeo::ImageCloseClick(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
// 响应最大化
//
void __fastcall TfrmNeo::ImageMaxClick(TObject *Sender)
{
        WindowState = wsMaximized;
        ImageRestore->Enabled = true;
        ImageRestore->Visible = true;
        ImageMax->Enabled = false;
        ImageMax->Visible = false;
}
// 响应恢复
//
void __fastcall TfrmNeo::ImageRestoreClick(TObject *Sender)
{
        WindowState = wsNormal;
        ImageRestore->Enabled = false;
        ImageRestore->Visible = false;
        ImageMax->Enabled = true;
        ImageMax->Visible = true;
}
//---------------------------------------------------------------------------
// 双击标题栏响应
//
void __fastcall TfrmNeo::lbCaptionDblClick(TObject *Sender)
{
        if(BorderIcons.Contains(biMaximize))
        {
            if( WindowState == wsMaximized )
            {
                ImageRestoreClick(this);
            }
            else if( WindowState == wsNormal )
            {
```

```
                ImageMaxClick(this);
            }
        }

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

实例 7　　创建 Windows2000 透明窗口界面

实例目标

本例目标就是实现 win2000 透明窗口，其实很简单。界面如下所示：



实现技术

主要调用 API 函数 SetWindowLong 和 SetLayeredWindowAttributes。其中用 SetWindowLong 函数来设置窗口的透明类型参数，用 SetLayeredWindowAttributes 设置窗口透明度。

实现步骤

第一步添加 TTrackBar 控件，添加位置变化时响应函数。

第二步添加如下代码：

```
//---------------------------------------------------------------------------
__fastcall TfrmTrans::TfrmTrans(TComponent* Owner)
    : TForm(Owner)
{
    m_nAlpha = 255;
    //设置 WINDOWS2000 透明窗口参数 WS_EX_LAYERED。
    SetWindowLong(Handle,GWL_EXSTYLE,
        GetWindowLong(Handle,GWL_EXSTYLE)|WS_EX_LAYERED);
    SetWindowLong(TrackBarAlpha->Handle,GWL_EXSTYLE,
        GetWindowLong(Handle,GWL_EXSTYLE)|WS_EX_LAYERED);
    //设置为不透明。
    SetAlpha(m_nAlpha);
}
//---------------------------------------------------------------------------
// 设置透明度。
```

```
//
void inline __fastcall TfrmTrans::SetAlpha(int nAlpha)
{
    //设置窗口透明度。LWA_ALPHA 指明用混合值 nAlpha 设置窗口透明度。
    SetLayeredWindowAttributes(Handle,NULL,nAlpha,LWA_ALPHA);
    SetLayeredWindowAttributes(TrackBarAlpha->Handle,NULL,255,LWA_ALPHA);
}
//
//  当改变 TrackBar 时调用此函数来改变透明度。
//
void __fastcall TfrmTrans::TrackBarAlphaChange(TObject *Sender)
{
    m_nAlpha = TrackBarAlpha->Position;
    SetAlpha(m_nAlpha);
}
//---------------------------------------------------------------------------
```

实例 8　　创建自画弹出式菜单界面

实例目标

本实例实现自画弹出式菜单，可以用自画菜单做出很多漂亮的界面，让人耳目一新。界面如下：

实现技术

本实例主要用设置弹出式菜单中自画属性的函数 ExpandMenuItemWidth 和 DrawNewItem。

实现步骤

第一步添加弹出式菜单和图像列表，设置每项菜单的名称和 ICON。第二步添加自画函数。主要代码如下：

```
//---------------------------------------------------------------------------
__fastcall TfrmPopupMenu::TfrmPopupMenu(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
//
// 设置每个菜单项的宽度和高度。
//
void __fastcall TfrmPopupMenu::ExpandMenuItemWidth(TObject *Sender,
     TCanvas *ACanvas, int &Width, int &Height)
{
    Width += TEXT_SPACE;
}
//
// 画每个菜单项。
//
```

```
void __fastcall TfrmPopupMenu::DrawNewItem(TObject *Sender, TCanvas *ACanvas,
    const TRect &ARect, bool Selected)
{
    TMenuItem *MenuItem = ((TMenuItem*)Sender);//得到当前菜单项。

    //得到复选框的大小。
    CheckmarkSize = GetSystemMetrics(SM_CXMENUCHECK);
    //整个菜单的高度。
    MenuHeight = ARect.Height() * MenuItem->Parent->Count;
    VerticalBarLength = MenuHeight / 4;

    if(!VerticalBarDrawn)//判断是否要重画。
    {
        //设置左边显示的字体。
        OldFont = (TFont*)SelectObject(ACanvas->Handle,
                                       CreateFontIndirect(&VerticalFont));
        //设置字体显示颜色。
        OldForegroundColor = SetTextColor(ACanvas->Handle, clRed);
        //设置字体背景显示的颜色。
        OldBackgroundColor = SetBkColor(ACanvas->Handle, TColor(RGB(50,150,220)));
        //设置显示窗口大小。
        VerticalDrawingRect = Rect(0, 0, CheckmarkSize+TEXT_SPACE, MenuHeight);
        //设置左边显示字符串为空字符串。
        VerticalText = VerticalText.StringOfChar(' ',VerticalBarLength);
        //设置左边显示字符串。
        VerticalText.Insert("  我的自画菜单  ",1);
        //显示左边字符串,并且进行剪裁字符串。
        ExtTextOut(ACanvas->Handle, 2, MenuHeight, ETO_CLIPPED,
          &VerticalDrawingRect, VerticalText.c_str(), VerticalBarLength, NULL);
        //恢复原来的字体。
        SelectObject(ACanvas->Handle,OldFont);
        //恢复字体背景的颜色。
        SetBkColor(ACanvas->Handle, OldBackgroundColor);
        //设置已经画了左边字符串。
        VerticalBarDrawn = true;
    }
    //
    TempRect = ARect;

    if(Selected)//选中的菜单。
    {
        //设置菜单项字符串显示区域。
        TempRect.Left += LOWORD(CheckmarkSize)+ICON_SPACE;
        //设置菜单项画笔类型。
```

```
ACanvas->Pen->Style = psSolid;
//设置画笔颜色。
ACanvas->Pen->Color = clWhite;
//设置背景填充颜色。
ACanvas->Brush->Color = clLtGray;
//画矩形。
ACanvas->Rectangle(
    TempRect.Left-MENU_TEXT_LEFT +1,
    MenuItem->MenuIndex*MENU_ITEM_OFFSET-SPACE_BETWEEN_MENUS
        +1 ,
    ARect.Width() -2,
    MenuItem->MenuIndex*MENU_ITEM_OFFSET+MENU_TEXT_HEIGHT -2);
//设置画笔颜色。
ACanvas->Pen->Color = clGray;
//画矩形。
ACanvas->Rectangle(
    TempRect.Left-MENU_TEXT_LEFT + 2,
    MenuItem->MenuIndex*MENU_ITEM_OFFSET-SPACE_BETWEEN_MENUS
        +2 ,
    ARect.Width() -1,
    MenuItem->MenuIndex*MENU_ITEM_OFFSET+MENU_TEXT_HEIGHT -1);
//设置菜单字体的颜色。
SetTextColor(ACanvas->Handle, clBlue    );
//在矩形内显示菜单字符串。
TempRect.left += 2;
TempRect.top += 2;

DrawText(ACanvas->Handle,MenuItem->Caption.c_str(),MenuItem->Caption.Length()
,
    &TempRect, 0);

//下面开始画 ICON。
ACanvas->Pen->Style = psSolid;
ACanvas->Pen->Color = clWhite;

ACanvas->Rectangle(
    24,
    MenuItem->MenuIndex * MENU_ITEM_OFFSET,
    25+MENU_ITEM_OFFSET+1,
    MenuItem->MenuIndex * MENU_ITEM_OFFSET+MENU_TEXT_HEIGHT);

ACanvas->Pen->Color = clGray;
ACanvas->Rectangle(
    25,
```

```
            MenuItem->MenuIndex * MENU_ITEM_OFFSET+1,
            25+MENU_ITEM_OFFSET+1,
            MenuItem->MenuIndex * MENU_ITEM_OFFSET+MENU_TEXT_HEIGHT+1);
        //从 IMAGELIST 得到 ICON。
        ImageListMenu->GetIcon(MenuItem->ImageIndex,Icon);
        //显示 ICON。
        ACanvas->Draw(26+2,MenuItem->MenuIndex * MENU_ITEM_OFFSET+2,Icon);

    }
    else//没有选中。
    {
        //设置菜单项字符串显示区域。
        TempRect.Left += LOWORD(CheckmarkSize)+ICON_SPACE;
        //设置菜单项画笔类型。
        ACanvas->Pen->Style = psClear;
        //画矩形。
        ACanvas->Rectangle(
            TempRect.Left-MENU_TEXT_LEFT-2,
            MenuItem->MenuIndex*MENU_ITEM_OFFSET-SPACE_BETWEEN_MENUS
                -2,
            ARect.Width()+2,
            MenuItem->MenuIndex*MENU_ITEM_OFFSET+MENU_TEXT_HEIGHT+2);
        //设置菜单显示颜色。
        SetTextColor(ACanvas->Handle, clBlack);
        //在矩形内显示菜单字符串。

        DrawText(ACanvas->Handle,MenuItem->Caption.c_str(),MenuItem->Caption.Length(),
        &TempRect, 0);
        ////////////////////////////////////////////////////////////////////////
        ACanvas->Pen->Style = psClear;
        ACanvas->Rectangle(
            24,
            MenuItem->MenuIndex * MENU_ITEM_OFFSET-2,
            25+MENU_ITEM_OFFSET+2,
            MenuItem->MenuIndex * MENU_ITEM_OFFSET+MENU_TEXT_HEIGHT+2);
        //从 IMAGELIST 得到 ICON。
        ImageListMenu->GetIcon(MenuItem->ImageIndex,Icon);
        //显示 ICON。
        ACanvas->Draw(26,MenuItem->MenuIndex * MENU_ITEM_OFFSET,Icon);

    }
    //刷新
    ACanvas->Refresh();
}
```

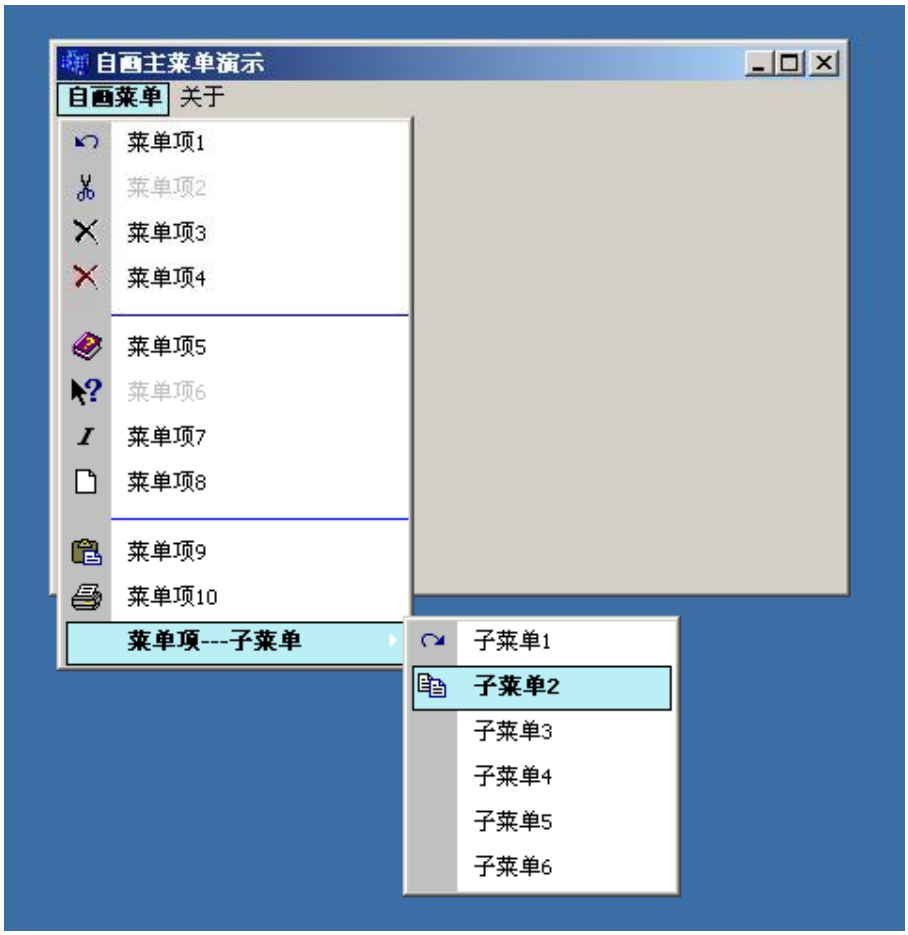
```
//
// 创建竖排菜单的字体。
//
void __fastcall TfrmPopupMenu::CreateVerticalFont(void)
{
      ZeroMemory(&VerticalFont,sizeof(VerticalFont));
      VerticalFont.lfHeight = -15;
      VerticalFont.lfEscapement = 900;
      VerticalFont.lfOrientation = 900;
      VerticalFont.lfWeight = FW_BLACK;
      StrPCopy(VerticalFont.lfFaceName, "宋体");
}
//
// 创建窗口时，设置自画菜单。
//
void __fastcall TfrmPopupMenu::FormCreate(TObject *Sender)
{
      if(PopupMenuOwerDraw->Items->Count > 0)
      {
            for(int i=0; i <= PopupMenuOwerDraw->Items->Count-1; i++)
            {
                  //设置得到每项菜单大小函数。
                  PopupMenuOwerDraw->Items->Items[i]->OnMeasureItem
                        = ExpandMenuItemWidth;
                  //设置每项菜单自画。
                  PopupMenuOwerDraw->Items->Items[i]->OnDrawItem
                        = DrawNewItem;
            }
    }
    //创建竖排的菜单字体。
    CreateVerticalFont();

    Icon = new TIcon;
}
//---------------------------------------------------------------------------

void __fastcall TfrmPopupMenu::FormClose(TObject *Sender,
        TCloseAction &Action)
{
      delete Icon;
}
//---------------------------------------------------------------------------
// 当弹出菜单时，设置没有画最左边的菜单。
//
```

```
void __fastcall TfrmPopupMenu::PopupMenuOwerDrawPopup(TObject *Sender)
{
	VerticalBarDrawn = false;
}
//---------------------------------------------------------------------------
```

实例 9　　创建自画主菜单界面

实例目标

本实现自画主菜单，现在有很多软件的界面都有自己特色的菜单。实现菜单的界面如下：



实现技术

主要设置自画菜单回调函数。当每项菜单显示时就会调用回调函数进行自画，达到每项菜单一样的风格。当选中后作不同的自画，就显示了不同的效果。

实现步骤

第一步添加主菜单和 ICON 图像列表。第二步添加所有事件响应函数。第三步添加回调函数。主要代码如下：

```
//---------------------------------------------------------------------------
__fastcall TfrmMainMenu::TfrmMainMenu(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
// 创建窗口时设置菜单项的参数。
//
```

```
void __fastcall TfrmMainMenu::FormCreate(TObject *Sender)
{
    Custom = TColor(RGB(185,239,245));

    MainMenuHighlightColor = Custom;
    MainMenuTextColor = clBlack;
    MainMenuTextBackground = clSilver;
    MainMenuHighlightTextColor = clBlack;

    VerticalColor = clSilver;
    MenuColor = clWhite;
    HighlightColor = Custom;
    BorderColor = clBlack;//菜单边框颜色。
    NormalTextColor = clBlack;//
    NormalTextBackground = clWhite;
    HighlightTextColor = clBlack;
    DisabledTextColor = clSilver;

    VerticalWidth = 26;
    FocusRectRightIndent = 3;
    FocusRectLeftIndent = 3;
    LeftTextPos = 35;
    SideBuffer = 1;

    if(Menu->Images == NULL)
        MenuIncreaseWidth = 100;
    else
        MenuIncreaseWidth = 50;

    Offset = 5;
}
//---------------------------------------------------------------------------
// 第二步添加。
// 主菜单第一项自画。
//
void __fastcall TfrmMainMenu::MenuOwerDrawDrawItem(TObject *Sender,
      TCanvas *ACanvas, TRect &ARect, bool Selected)
{
    TRect FocusRectBorder;
    TRect FocusRectFill;
    //得到菜单项。
    TMenuItem *MenuItem = ((TMenuItem*)Sender);
    //保存菜单字符串。
    AnsiString Text = MenuItem->Caption;
```

```
    // 填充菜单项背景颜色。
    ACanvas->Brush->Color = Color;
    ACanvas->FillRect(ARect);
    //没有菜单内容就返回。
    if(Text == "")
        return;

    if(Selected)//选中菜单。
    {
        // 画菜单外面边框。
        FocusRectBorder = ARect;
        ACanvas->Brush->Color = BorderColor;
        ACanvas->FrameRect(FocusRectBorder);
        //填充菜单内部
        FocusRectFill = ARect;
        //设置内部边框比外面要小点。
        FocusRectFill.Top += SideBuffer;
        FocusRectFill.Left += SideBuffer;
        FocusRectFill.Right -= SideBuffer;
        FocusRectFill.Bottom -= SideBuffer;
        //设置为高度显示的颜色。
        ACanvas->Brush->Color = MainMenuHighlightColor;
        ACanvas->FillRect(FocusRectFill);

        // 设置当菜单选中后字体的颜色。
        ACanvas->Font->Color = MainMenuHighlightTextColor;
        ACanvas->Font->Style = TFontStyles() << fsBold;
    }
    else//没有选中。
    {
        // 设置当菜单字体的颜色。
        ACanvas->Font->Color = MainMenuTextColor;
        ACanvas->Font->Style = TFontStyles();
    }

    int TextLength = Text.Length();
    TRect TextRect = ARect;
    //设置从什么地方开始画菜单项字符串。
    TextRect.Left += 5;
    TextRect.Top += 1;
    // 显示菜单项字符串。
    DrawText(ACanvas->Handle,Text.c_str(), TextLength, &TextRect, 0);
}
```

```
//---------------------------------------------------------------------------
//画每项菜单。
//
void __fastcall TfrmMainMenu::MenuItemDrawItem(TObject *Sender,
      TCanvas *ACanvas,const TRect &ARect, bool Selected)
{
    int TopPos, TextLength;
    AnsiString Text;
    TRect TempRect;
    TRect VerticalRect;
    TRect FocusRectBorder;
    TRect FocusRectFill;
    TRect TextRect;
    //得到当前菜单项。
    TMenuItem *MenuItem = ((TMenuItem*)Sender);
    //得到菜单项的字符串。
    Text = MenuItem->Caption;
    //刷新菜单项背景。
    ACanvas->Brush->Color = MenuColor;
    ACanvas->FillRect(ARect);
    // 画菜单项的分隔条。
    if(Text==BLANK_LINE)
    {
        // 画分隔条最左边垂直的 Bar
        VerticalRect = ARect;
        VerticalRect.Top -= SideBuffer;
        VerticalRect.Right = VerticalWidth;
        VerticalRect.Bottom += SideBuffer;
        ACanvas->Brush->Color = VerticalColor;
        ACanvas->FillRect(VerticalRect);

        // 画一条分隔直线。
        TColor oldColor = ACanvas->Pen->Color;
        ACanvas->Pen->Color = clBlue;

        ACanvas->MoveTo(VerticalWidth,ARect.Top+ARect.Height()/2);
        ACanvas->LineTo(ARect.Right,ARect.Top+ARect.Height()/2);

        ACanvas->Pen->Color = oldColor;
        return;
    }
    // 其它菜单项的自画
    TextLength = Text.Length();
    if(Selected)//选中菜单。
```

```
{
    //画最左边的垂直的 BAR。
    VerticalRect = ARect;
    VerticalRect.Top -= SideBuffer;
    VerticalRect.Right = VerticalWidth;
    VerticalRect.Bottom += SideBuffer;
    ACanvas->Brush->Color = VerticalColor;
    ACanvas->FillRect(VerticalRect);

    if(MenuItem->Enabled)//如果菜单有效。
    {
        //画一个焦点四边框。
        FocusRectBorder = ARect;
        FocusRectBorder.Left += FocusRectLeftIndent - SideBuffer;
        FocusRectBorder.Right -= FocusRectRightIndent - SideBuffer;
        ACanvas->Brush->Color = BorderColor;
        ACanvas->FrameRect(FocusRectBorder);
        //填充内部颜色。
        FocusRectFill = ARect;
        FocusRectFill.Right -= FocusRectRightIndent;
        FocusRectFill.Left += FocusRectLeftIndent;
        FocusRectFill.Bottom -= SideBuffer;
        FocusRectFill.Top += SideBuffer;
        ACanvas->Brush->Color = HighlightColor;
        ACanvas->FillRect(FocusRectFill);
        // 设置显示字符串的颜色和字体类型。
        ACanvas->Font->Color = HighlightTextColor;
        ACanvas->Font->Style = TFontStyles() << fsBold;
    }
    else //菜单没有效。
    {
        // 显示为没有效。
        ACanvas->Font->Style = TFontStyles();
        ACanvas->Brush->Color = NormalTextBackground;
        ACanvas->Font->Color = DisabledTextColor;
    }//end of if(MenuItem->Enabled)//如果菜单有效。
}
else//没有选中的菜单自画。
{
    // 填充左边垂直边框。
    VerticalRect = ARect;
    VerticalRect.Top -= SideBuffer;
    VerticalRect.Right = VerticalWidth;
    VerticalRect.Bottom += SideBuffer;
```

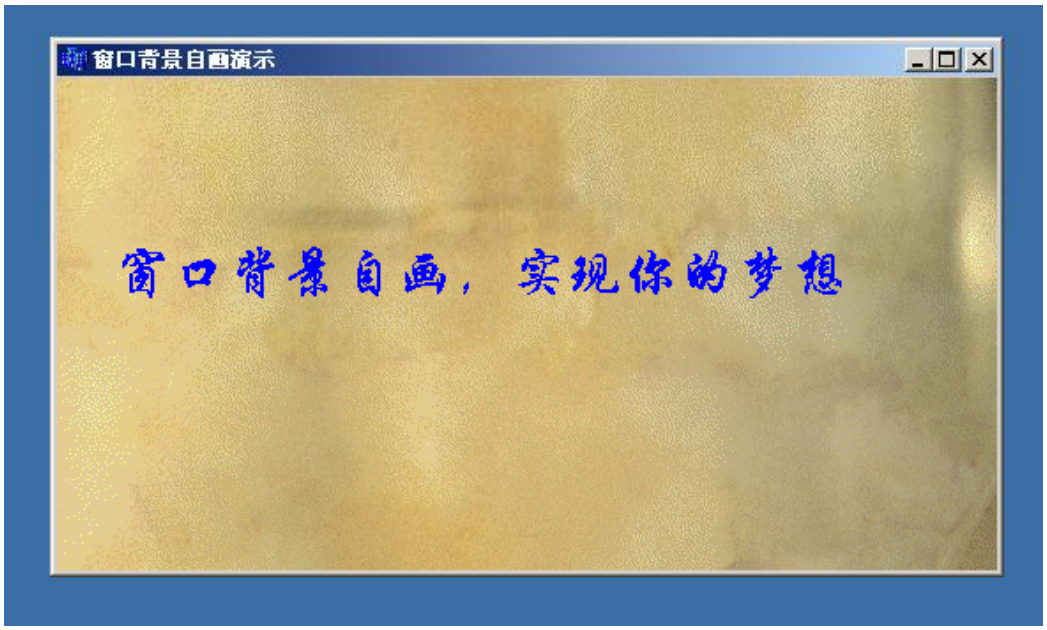
```
            ACanvas->Brush->Color = VerticalColor;
            ACanvas->FillRect(VerticalRect);

            // 设置有效和没有效的菜单。
            if(MenuItem->Enabled)
            {
                ACanvas->Brush->Color = NormalTextBackground;
                ACanvas->Font->Color = NormalTextColor;
            }
            else
            {
                ACanvas->Brush->Color = NormalTextBackground;
                ACanvas->Font->Color = DisabledTextColor;
            }
        }

        // 计算输出字符串的位置。
        TextRect = ARect;
        TextRect.Left += LeftTextPos;
        if(Offset > 0)
            TextRect.Top += Offset/2 + SideBuffer;
        else
            TextRect.Top += 2 + SideBuffer;

        TextRect.Top += SideBuffer;

        // 画菜单的 ICON。
        if(Menu->Images != NULL)
        {
            Icon = new TIcon();
            Menu->Images->GetIcon(MenuItem->ImageIndex,Icon);
            ACanvas->Draw(5,ARect.Top+ItemOffset+1,Icon);
            delete Icon;
        }

        // 显示菜单字符串。
        DrawText(ACanvas->Handle,Text.c_str(), TextLength, &TextRect, 0);

}
//---------------------------------------------------------------------------
//设置每一项菜单的高度和宽度。
//
void __fastcall TfrmMainMenu::MenuItemMeasureItem(TObject *Sender,
      TCanvas *ACanvas, int &Width, int &Height)
```

```
{
    Width += MenuIncreaseWidth;
    Height += Offset;
    MenuItemHeight = Height;
    ItemOffset = Offset/2;
}
//---------------------------------------------------------------------------
//当按下第一项菜单时，设置字菜单每行自画回调函数。
//
void __fastcall TfrmMainMenu::MenuOwerDrawClick(TObject *Sender)
{
    TMenuItem *MenuItem = ((TMenuItem*)Sender);//得到菜单项。
    //设置每一项菜单。
    if(MenuItem->Count > 0)
    {
        for(int i=0; i <= MenuItem->Count-1; i++)
        {
            //设置每项菜单高度和宽度回调函数。
            MenuItem->Items[i]->OnMeasureItem = MenuItemMeasureItem;
            //设置每项菜单自画回调函数。
            MenuItem->Items[i]->OnDrawItem = MenuItemDrawItem;
            //设置下一级字菜单。
            if(MenuItem->Items[i]->Count > 0)
            {
                for(int x=0; x <= MenuItem->Items[i]->Count-1; x++)
                {
                    MenuItem->Items[i]->Items[x]->OnMeasureItem =
                            MenuItemMeasureItem;
                    MenuItem->Items[i]->Items[x]->OnDrawItem =
                            MenuItemDrawItem;
                }
            }
        }
    }
}
//---------------------------------------------------------------------------
```

实例 10　　创建自画窗口背景界面

实例目标

本实例的目的很简单，就是实现窗口背景自画。界面如下所示：



实现技术

关键技术就是响应开始画窗口背景的消息 WM_ERASEBKGND，然后就开始画背景的图片。

实现步骤

第一步创建消息处理函数 ClientProc(TMessage & Msg)，第二步设置窗口处理函数。其关键代码如下：

```
//---------------------------------------------------------------------------
__fastcall TfrmDrawBkGround::TfrmDrawBkGround(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
//
// 消息处理函数。
//
void __fastcall TfrmDrawBkGround::ClientProc(TMessage & Msg)
{
      switch (Msg.Msg)
      {
            case WM_ERASEBKGND://处理背景自画消息。
                  DrawClientWindow((HDC)Msg.WParam);
```

```
                Msg.Result = true;
                return;
            default://调用缺省消息处理。
                Msg.Result = CallWindowProc((FARPROC)OriginalProc,
                    ClientHandle,Msg.Msg,Msg.WParam,Msg.LParam);
                break;
        }

}
//
//屏蔽掉原来的背景消息处理函数。
//
void __fastcall TfrmDrawBkGround::WMEraseBkgnd(TWMEraseBkgnd & Msg)
{
        Msg.Result = false;
}
//
// 画窗口背景图像。
//
void __fastcall TfrmDrawBkGround::DrawClientWindow(HDC & Hdc)
{
        TRect rect;
        //取得窗户区大小。
        ::GetClientRect(ClientHandle,(RECT *)&rect);
        int left,top;
        // 使用 Windows API 函数 BitBlt
        for (int i=0;i<ClientHeight/BGBitmap->Height+1;i++)
            for (int j=0;j<ClientWidth/BGBitmap->Width+1;j++)
            {
                left = j*BGBitmap->Width;
                top = i*BGBitmap->Height;
                //调用 API 函数画满背景。
                ::BitBlt(Hdc,left,top,left+BGBitmap->Width,top+BGBitmap->Height,
                        BGBitmap->Canvas->Handle,0,0,SRCCOPY);
            }
}
//
//  创建窗口时调用。
//
void __fastcall TfrmDrawBkGround::FormCreate(TObject *Sender)
{
      //从函数 ClientProc 转换成指针类型。
      ObjectInstance = MakeObjectInstance(ClientProc);
      //设置新的窗口消息处理函数，同时保存原来的消息函数。
```

```
    OriginalProc = (Pointer) SetWindowLong(Handle,GWL_WNDPROC,
                                (long)ObjectInstance);
    //创建背景位图。
    BGBitmap = new Graphics::TBitmap();
    //从程序执行的路径取得背景图片文件。
    BGBitmap->LoadFromFile(ExtractFilePath(Application->ExeName)+"BG.bmp");

}
//---------------------------------------------------------------------------
// 关闭窗口时调用。
//
void __fastcall TfrmDrawBkGround::FormClose(TObject *Sender,
      TCloseAction &Action)
{
    //设置原来的窗口消息处理函数。
    SetWindowLong(Handle,GWL_WNDPROC,(long)OriginalProc);
    //释放创建的对象。
    FreeObjectInstance(ObjectInstance);
    //删除位图。
    delete BGBitmap;
}
//---------------------------------------------------------------------------
```