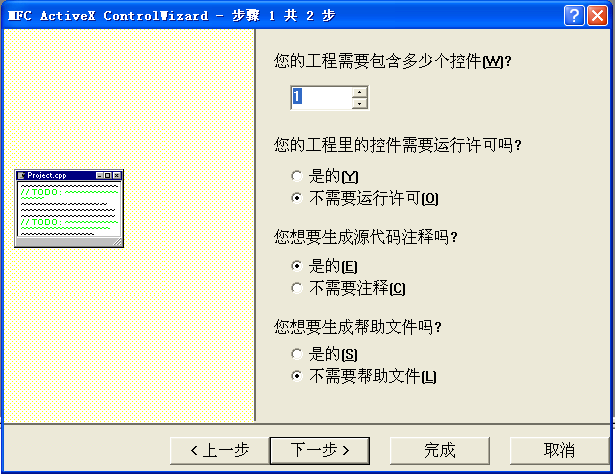
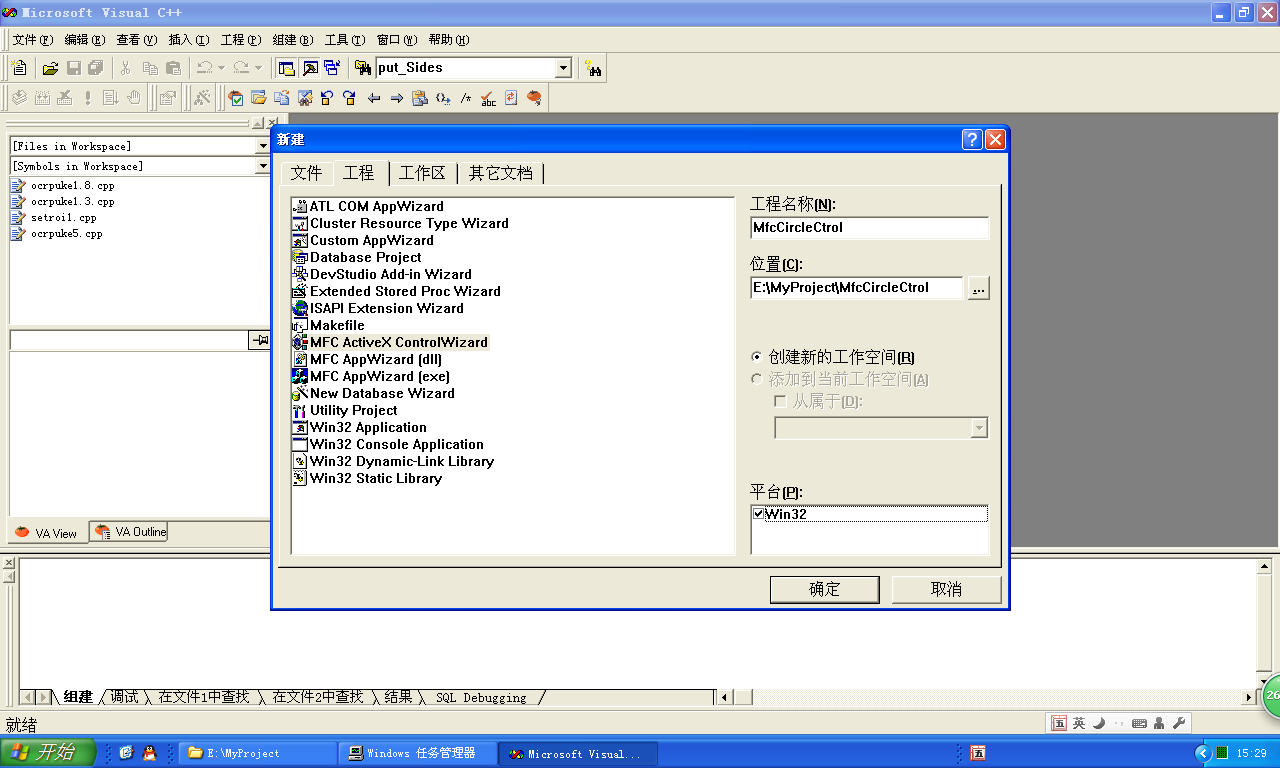
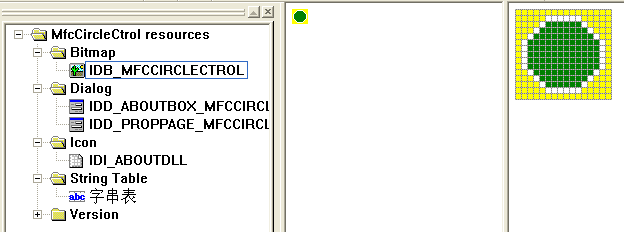
# 利用MFC创建ActiveX控件

## 一、新建工程

新建一个工程，选择MFC ActiveX ControlWizard 向导，为工程取名MfcCircleCtrol：



## 二、绘制资源位图



## 三、添加绘制代码

在控件的OnDraw函数中添加代码，以实现想要的显示效果，这里绘制了一个矩形的背景，然后绘制一个椭圆区域，再中心绘制标题文字，实现代码如下：

void CMfcCircleCtrolCtrl::OnDraw(

CDC\* pdc, const CRect& rcBounds, const CRect& rcInvalid)

{

CBrush \*pOldBrush;

CBrush bkBrush(TranslateColor(GetBackColor()));

CBrush ShapeBrush(m\_crShapeColor);

CPen \*pOldPen;

CRect rc;

if (m\_circleShape)

{

int d=0.5\*(rcBounds.Width()>rcBounds.Height()?rcBounds.Height():rcBounds.Width());

int ox=(rcBounds.right+rcBounds.left)/2;

int oy=(rcBounds.bottom+rcBounds.top)/2;

rc.left=ox-d;

rc.right=ox+d;

rc.top=oy-d;

rc.bottom=oy+d;

}

else

{

rc=rcBounds;

}

CFont \*pOldFont;

//背景

pOldBrush=pdc->SelectObject(&bkBrush);

pdc->FillRect(rcBounds,&bkBrush);

//图形

pdc->SelectObject(&ShapeBrush);

pOldPen=(CPen\*)pdc->SelectStockObject(BLACK\_PEN);

pdc->Ellipse(rc);

//文字

TEXTMETRIC tm;

const CString& strCaption=InternalGetText();

pdc->SetTextColor(TranslateColor(GetForeColor()));

pdc->SetBkMode(TRANSPARENT);

pOldFont=SelectStockFont(pdc);

GetStockTextMetrics(&tm);

pdc->SetTextAlign(TA\_CENTER|TA\_TOP);

pdc->ExtTextOut((rc.left+rc.right)/2,(rc.top+rc.bottom-tm.tmHeight)/2,ETO\_CLIPPED, rc,strCaption,strCaption.GetLength(),NULL);

pdc->SelectObject(pOldFont);

pdc->SelectObject(pOldPen);

pdc->SelectObject(pOldBrush);

}

## 四、为控件添加属性

### 4.1 默认内部属性：

为控件添加以下属性：

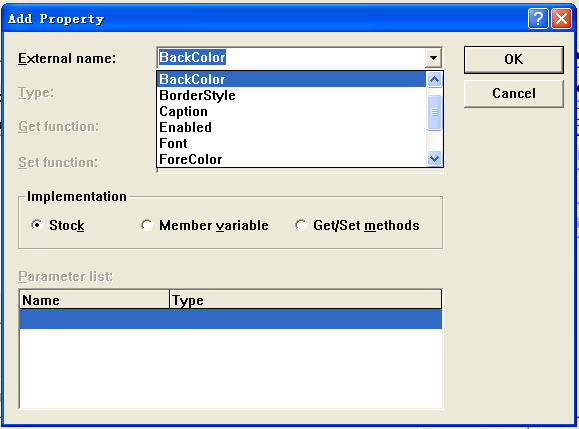
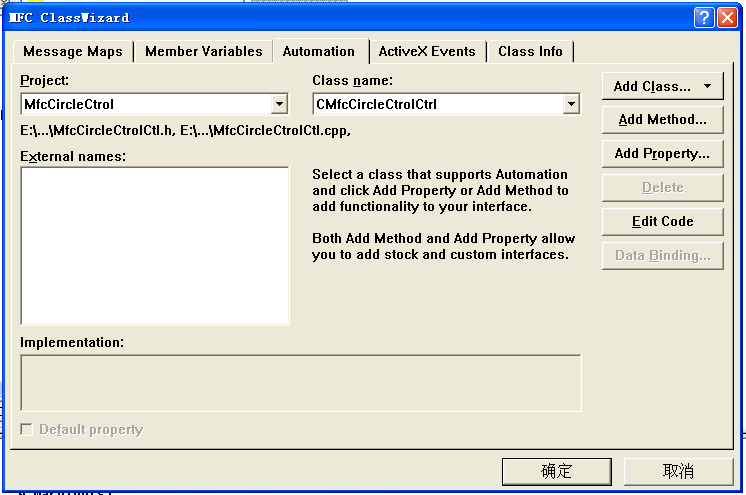
BackColor:背景颜色

Caption:控件标题

Font:标题字体

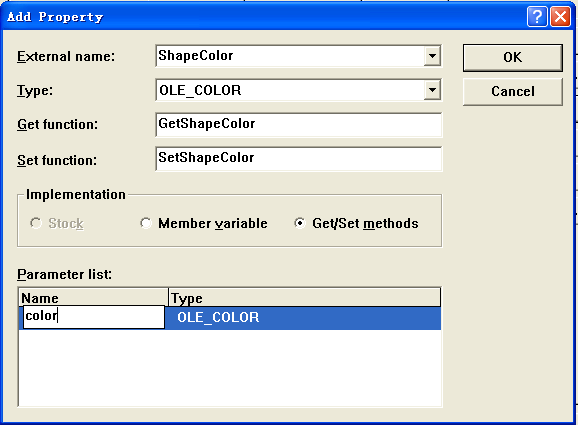
ForeColor:前景颜色（字体颜色）

添加的过程用向导完成，注意这些属性是标准属性，可以在Extenal name的下拉框中选择，Implementation默认选择Stock，不要修改。

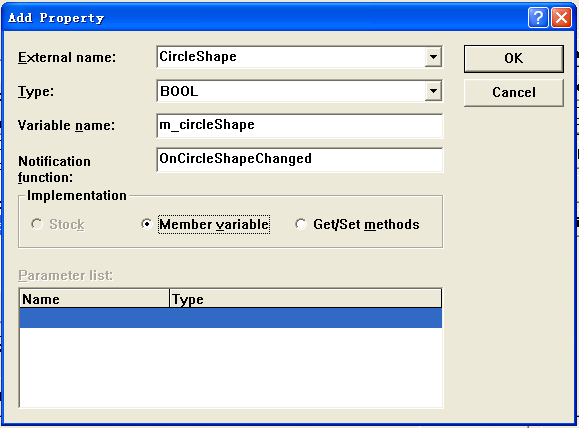


### 4.2 自定义属性

添加一个名为ShapeColor属性，类型选择OLE\_COLOR,Implementation选择Get/Set methods，该属性用来表示控件中圆形的颜色，向导会为我们生成GetShapeColor和SetShapeColor函数。



添加一个名为CircleShape的属性，类型选择BOOL，用来表示绘制椭圆还是标准圆，Implemantation选择Member variable，向导会为我们生成一个与之关联的属性变量，这里是m\_circleShape，当该变量发生变化的时候，系统会自动触发为我们生成的函数OnCircleShapeChanged。



对添加的属性在控件的构造函数中初始化。

### 4.3 完善属性相关代码

void CMfcCircleCtrolCtrl::OnCircleShapeChanged()

{

InvalidateControl();

SetModifiedFlag();

}

OLE\_COLOR CMfcCircleCtrolCtrl::GetShapeColor()

{

return (OLE\_COLOR)m\_crShapeColor;

}

void CMfcCircleCtrolCtrl::SetShapeColor(OLE\_COLOR nNewValue)

{

m\_crShapeColor=TranslateColor(nNewValue);

SetModifiedFlag();

InvalidateControl();

}

这里要注意几点是：

1、在属性发生变化后，要记得调用InvalidateControl函数以重绘控件。

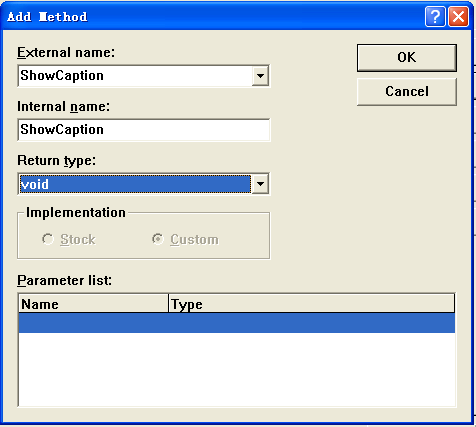
2、COLORREF与OLE\_COLORG两种颜色变量之间的转换：

COLORREF->OLE\_COLORG:直接用OLE\_COLOR强制转换；

OLE\_COLORG->COLORREF:使用TranslateColor函数

## 五、为控件添加方法

与添加属性的方法类似，在向导的Automation属性页中选择Add Method，添加一个ShowCaption的方法：



为方法添加代码：

void CMfcCircleCtrolCtrl::ShowCaption()

{

MessageBox(InternalGetText());

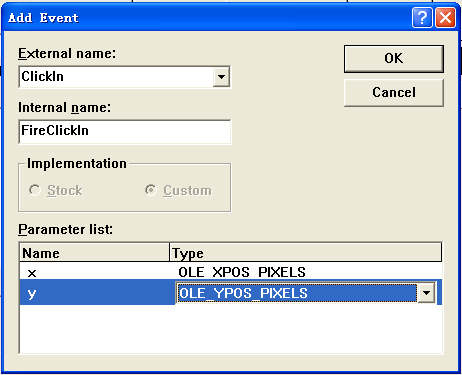
}

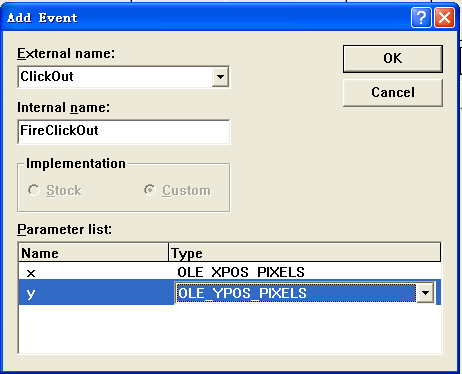
注意，这里InternalGetText函数就是得到控件的标题字符串。

## 六、为控件添加事件

### 6.1自定义事件

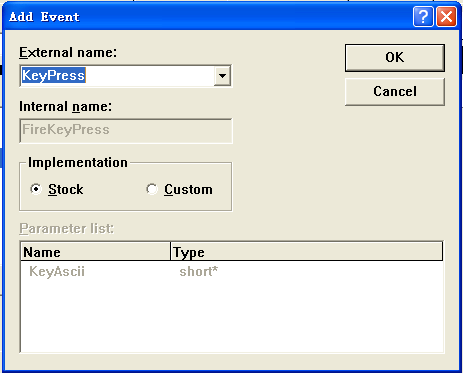
向导中切换到ActiveX Events页面，为控件添加事件，这里添加两个自定义事件ClickIn和ClickOut，当鼠标在椭圆内点击时激发ClickIn事件，在椭圆外点击时激发ClickOut事件。向导自动为我们添加两个函数FireClickIn和FireClickOut，不要编辑修改它，只要在我们需要激发事件的地方调用它们即可。由于要获得鼠标的位置，因此事件的参数我们用来接收鼠标的坐标，类型为OLE\_XPOS\_PIXELS和OLE\_YPOS\_PIXELS。





### 6.2 默认内部事件

在Extenal name的下拉框中选择默认内部事件，这里选择KeyPress，Implementation选择Stock。不需要编写代码，当键盘被按下时，会触发该事件。



### 6.3 完善事件代码

为控件添加右键单击响应函数，填写代码如下：

void CMfcCircleCtrolCtrl::OnLButtonDown(UINT nFlags, CPoint point)

{

CRect rc;

GetClientRect(rc);

double a=abs((rc.right-rc.left))/2;

double b=abs((rc.bottom-rc.top))/2;

double x=point.x-a;

double y=point.y-b;

m\_circleShape=((x\*x)/(a\*a)+(y\*y)/(b\*b)<=1);

InvalidateControl();

if (m\_circleShape)

{

FireClickIn(OLE\_XPOS\_PIXELS(x),OLE\_YPOS\_PIXELS(y));

}

else

{

FireClickOut(OLE\_XPOS\_PIXELS(x),OLE\_YPOS\_PIXELS(y));

}

COleControl::OnLButtonDown(nFlags, point);

}

此代码中，当按钮点击在椭圆内时，会将m\_circleShape置TRUE，并激发ClickIn事件，按钮点击在椭圆外时，会将m\_circleShape置FALSE，并激发ClickOut事件。

## 七、为控件添加属性页

### 7.1 修改相关宏代码

在MfcCircleCtrolCtl.cpp中找到属性页宏相关代码，修改它们，使之看起来成这样：

BEGIN\_PROPPAGEIDS(CMfcCircleCtrolCtrl, 3)

PROPPAGEID(CMfcCircleCtrolPropPage::guid)

PROPPAGEID(CLSID\_CColorPropPage)

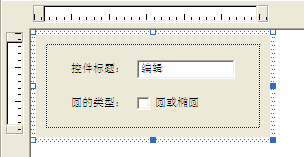
PROPPAGEID(CLSID\_CFontPropPage)

END\_PROPPAGEIDS(CMfcCircleCtrolCtrl)

注意这里的3，表明有三项，如果增加和减少属性页，要相应的改变该值。

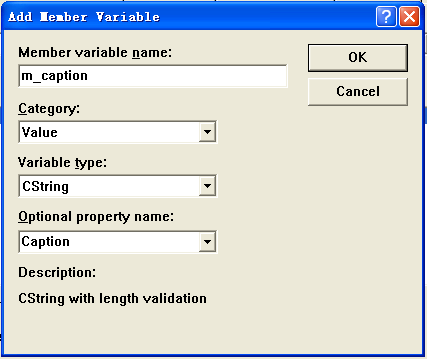
### 7.2 编辑属性页对话框

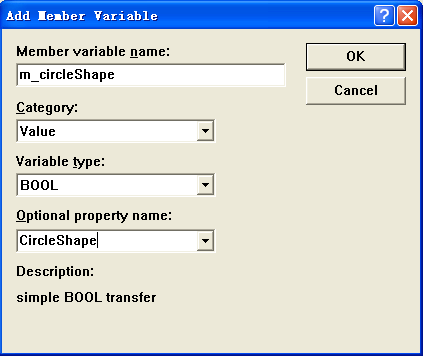
在对话框模板中增加一个文本框以表示标题，一个勾选框以代表是否要绘制标准圆：



### 7.3 为控件生成属性变量

为以上添加的两个控件生成变量，注意这里多出一个Optional property name下拉框，这是外部可以看见的属性名，由于Caption是标准的属性，可以在下拉框中选择，而CircleShape是自定义的属性，因而需要自己填入。这里要注意一点是，变量名，如这里的m\_caption和m\_circleShape要与之前在第三步中的属性名要一致。





## 八、为属性添加持久性

### 8.1 属性的持久性

在DoPropExchange函数中添加PX\_XXXX函数，其中XXXX为变量类型，默认内部属性已具有持久性，在这里只需要为自定义属性添加持久性即可，实验发现ShapeColor属性虽然也是自定义属性，但也默认具有持久性，原因不明。这里只为CircleShape添加持久性，代码如下：

void CMfcCircleCtrolCtrl::DoPropExchange(CPropExchange\* pPX)

{

ExchangeVersion(pPX, MAKELONG(\_wVerMinor, \_wVerMajor));

COleControl::DoPropExchange(pPX);

PX\_Bool(pPX,"CircleShape",m\_circleShape,TRUE);

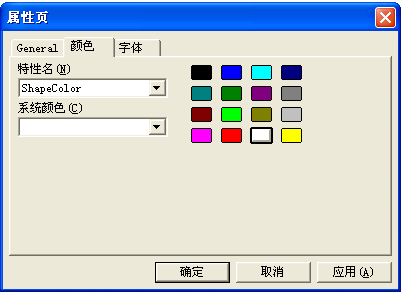
}

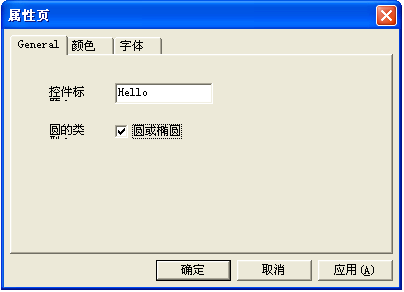
其中CircleShape为属性的外部名称，m\_circleShape为属性对应的变量名，TRUE为默认值。

### 8.2 通知容器属性已改变

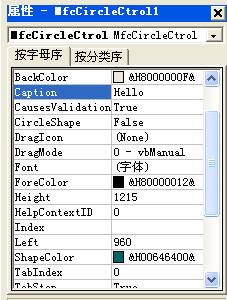
在VB下对控件进行测试时，设置属性有两种方法，第一种用VB默认的属性窗口，第二种用我们自己的属性页，右击控件，选择特性，在弹出的属性页中修改属性。

如果此时我们在属性页中对ShapeColor颜色进行设置，例如这里选择白色，又勾选了圆的类型，确认后回到设计界面。





此进我们发现VB的属性窗口对应的属性值并没有改变。



这样会使用户产生迷惑，因此我们要在属性改变的时候通知容器，这里是VB属性已经改变，方法如下：

在ODL文件下我们可以发现如下代码：

[id(1)] boolean CircleShape;

[id(2)] OLE\_COLOR ShapeColor;

这里表示CircleShape的调度ID为1，ShapeColor的调度ID为2，通知容器属性改变用到的函数是BoundPropertyChanged，其参数便是属性的调度ID。因此我们在属性改变的函数下调用BoundPropertyChanged函数。

修改OnCircleShapeChanged函数和SetShapeColor函数代码如下：

void CMfcCircleCtrolCtrl::OnCircleShapeChanged()

{

InvalidateControl();

SetModifiedFlag();

BoundPropertyChanged(0x1);

}

void CMfcCircleCtrolCtrl::SetShapeColor(OLE\_COLOR nNewValue)

{

m\_crShapeColor=TranslateColor(nNewValue);

SetModifiedFlag();

BoundPropertyChanged(0x2);

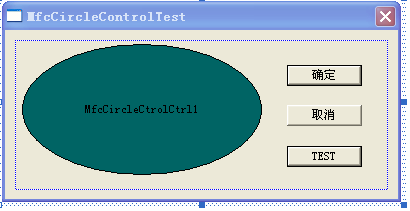
InvalidateControl();

}

## 九、测试

### 9.1 创建工程

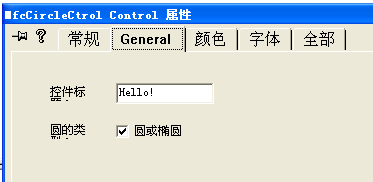
创建一个基于对话框的工程，设计界面如下：



为自定义的控件生成一个变量，名为m\_circle，这时系统会为我们生成一个CMfcCircleCtrl类。

### 9.2 属性测试

右击选择属性，效果如下：





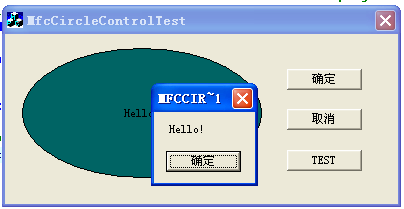
在颜色页面中，BackColor和ForeColor是默认的颜色属性，而我们自定义的颜色属性ShapeColor也同样被归类到该页面中，这为我们设置颜色提供了方便。

### 9.3 方法测试

在测试按钮的响应函数中添加如下代码：

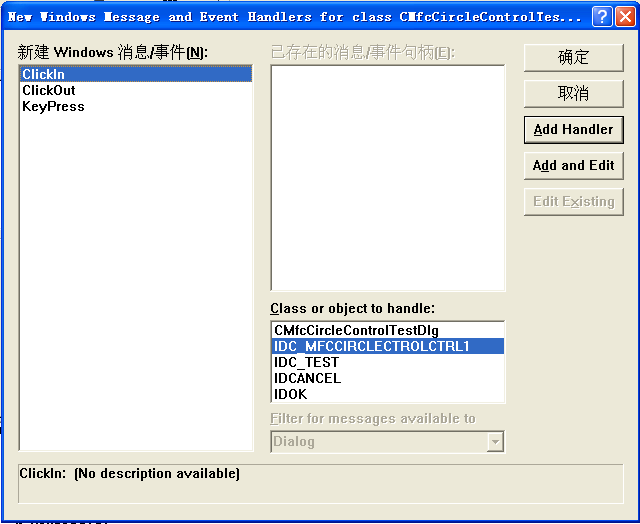
m\_circle.ShowCaption();

单击该按钮，效果如下：



### 9.4 事件测试

右击控件，选择事件，添加三个事件：



添加响应函数代码：

void CMfcCircleControlTestDlg::OnClickInMfccirclectrolctrl1(long x, long y)

{

MessageBox("Click In");

}

void CMfcCircleControlTestDlg::OnClickOutMfccirclectrolctrl1(long x, long y)

{

MessageBox("Click Out");

}

void CMfcCircleControlTestDlg::OnKeyPressMfccirclectrolctrl1(short FAR\* KeyAscii)

{

MessageBox("Key Press");

}

测试结果如下：

