# Custom Control自定义控件的实现

在设计自定义控件时，你有很大的自由度，你可以在它的窗口中绘制任何的东西，而且，你可以定义任何通知消息及内部消息。你可以用属性窗口来建立常规的Windows消息，如WM\_LBUTTONDOWN，但你必须手工映射用户自定义的消息，并在父窗口中手工映射通知消息。

自定义控件必须用一个窗口类来指定，可以是系统已注册的MFC类，如Button，Static等，也可以是自定义的窗口类，不要将这里的窗口类与MFC类，C++类相混淆，窗口类是一个结构体WNDCLASS来定义的。在自定义控件的属性窗口的Class属性中填入这个窗口类的名字。而控件的处理代码则保存在一个MFC类中，这个类由CWnd派生。

由于窗口类必须注册才可以使用，因此必须为其定义一个注册窗口函数。

控件与父窗口的通信可用GetParent()->SendMessage(…)来完成，该代码发送消息到父窗口，在父窗口中用ON\_CONTROL来建立消息与响应函数的映射关系。

父窗口与控件的通信可以用GetDlgItem(…)->SendMessage(…)来完成，该代码发送消息到控件，在控件中用ON\_MESSAGE来建立消息与响应函数的映射关系。

如果把自定义控件封装到MFC常规DLL中，可以利用载入DLL时会运行DLL的InitInstance函数，把注册窗口类的函数在InitInstance函数中调用，为确保DLL被载入，将DLL的导出函数在父窗口的构造函数中调用，这个导出函数可以什么也不做，只是为了确保载入DLL，因为调用它必须载入DLL。

## 一、在本地实现

1、定义一个MFC类，派生于CWnd类，这里将类名取为CMyControl。

2、定义一个静态窗口过程函数，这里取名为staticWndProc:

static LRESULT CALLBACK staticWndProc(HWND hWnd, UINT nMessage, WPARAM wParam, LPARAM lParam);

3、为CMyControl定义一个注册窗口函数，该函数声明为静态类型，以便于调用。

static void RegisterWndClass(HINSTANCE hInstance);

4、实现RegisterWndClass函数：

void CMyControl::RegisterWndClass(HINSTANCE hInstance)

{

WNDCLASS wc;

wc.lpszClassName = "MyCustomControl";

wc.hInstance = hInstance;

wc.lpfnWndProc = staticWndProc;

wc.hbrBackground = (HBRUSH)::GetStockObject(LTGRAY\_BRUSH);

wc.hCursor = ::LoadCursor(NULL,IDC\_ARROW);

wc.hIcon = 0;

wc.lpszMenuName = 0;

wc.style = CS\_GLOBALCLASS;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

::RegisterClass(&wc);

}

这是的MyCustomControl是窗口类名。

5、实现staticWndProc函数：

LRESULT CALLBACK CMyControl::staticWndProc(HWND hWnd, UINT nMessage, WPARAM wParam, LPARAM lParam)

{

AFX\_MANAGE\_STATE(::AfxGetAppModuleState());

CWnd \*pControl = CWnd::FromHandlePermanent(hWnd);

if(pControl == NULL)

{

pControl = new CMyControl();

pControl->Attach(hWnd);

}

return ::AfxCallWndProc(pControl,hWnd,nMessage,wParam,lParam);

}

6、为CMyControl重写PostNcDestroy函数，否则会有内存泄漏：

void CMyControl::PostNcDestroy()

{

delete this;

CWnd::PostNcDestroy();

}

7、在主框口中加入一个自定义控件，在自定义控件的属性窗口的Class中填入该名称MyCustomControl，运行程序，可见程序运行成功。但此时控件没有添加任何消息响应函数，所以是没有任何功能的。

图1、设计界面 图2、运行结果

8、实现控件与父窗口的通信。

为控件添加一个鼠标左键点击函数，在类视图中选择CMyControl类，右击选择属性，在属性对话框中选择WM\_LBUTTON消息，添加响应函数：

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

{

GetParent()->SendMessage(WM\_COMMAND,this->GetDlgCtrlID(),

(LPARAM)this->GetSafeHwnd());

}

函数中，调用父类的SendMessage函数，给父窗口发送一个WM\_COMMAND消息，查看MSDN，WM\_COMMAND消息的消息源为控件时，wParam(32位)的高位为Notification Code(前16位)，wParam的低位为控件ID(后16位)，lParam（32位）参数为控件窗口句柄。而SendMessage的函数原型为：

LRESULT SendMessage(UINT message,WPARAM wParam=0,LPARAM lParam=0)throw()

这里我们将消息码设为0，因此wParam的值为0x0000 0000+this->GetDlgCtrlID()= this->GetDlgCtrlID()。

在父窗口的消息映射中，使用ON\_CONTROL宏，查MSDN，ON\_CONTORL的原型为：

ON\_CONTROL(wNotifyCode,id,memberFxn)，由于之前的NotifyCode设置为0，所以应该加入如下代码：

ON\_CONTROL(0,IDC\_CUSTOM1,OnClickCustomControl)

其中OnClickCustomControl为对应的响应函数。

如果我们想将消息码设置为1，以对应别一个消息和消息响应函数，比如，我们想要响应自定义控件的鼠标右击事件，我们可以在控件的右击消息响应函数中添加以下代码：

GetParent()->SendMessage(WM\_COMMAND,0x10000+this->GetDlgCtrlID(),

(LPARAM)this->GetSafeHwnd());

而在父窗口的消息映射宏中加入：

ON\_CONTROL(1,IDC\_CUSTOM1,OnClickCustomControl2)

然后在OnClickCustomControl2函数中添加相应的处理代码即可。

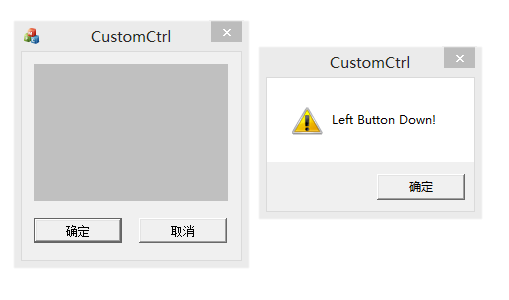


图3、控件与父窗口的通信

9、实现父窗口与控件的通信.这里实现一个改变控件上文字显示的功能。

为CMyControl类添加一个CString类对象m\_strShow，用来保存显示的文字，将其初始化为"Hello World！"。

在CMyControl的OnPaint函数中加入以下代码：

CPaintDC dc(this);

dc.TextOut(0,0,m\_strShow);

在stdafx.h中加入以下消息定义：

#define WM\_SETCONTROLTEXT (WM\_USER + 0)

在父窗口的确定按钮下添加如下代码：

GetDlgItem(IDC\_CUSTOM1)->SendMessage(WM\_SETCONTROLTEXT,0,

LPARAM("New Text"));

该代码向控件发送自定义的WM\_SETCONTROLTEXT消息。

在MyControl.cpp中加入以下消息映射代码：

ON\_MESSAGE(WM\_SETCONTROLTEXT,OnSetControlText)，并实现OnSetControlText函数：

LRESULT CMyControl::OnSetControlText(WPARAM wParam, LPARAM lParam)

{

m\_strShow = (LPCTSTR)lParam;

Invalidate();

return 0;

}

运行效果如下：

四4、父窗口与控件的通信

## 二、在DLL中实现

1、新建一个MFC扩展DLL工程CustomDLLDemo，将上面的CMyControl.h和CMyControl.cpp文件拷贝到该工程中，并添加到工程中。

2、将#define WM\_SETCONTROLTEXT (WM\_USER+0)拷贝到CMyControl.h文件中，并对CMyControl.cpp的#inlude做必要修改，删除去#include "CustomCtrl.h"。

3、在CCustomDLLDemoApp.InitInstance()中加入注册窗口类语句：

CMyControl::RegisterWndClass(AfxGetInstanceHandle())；

4、编译工程，生成CustomDLLDemo.dll，CustomDLLDemo.lib。

5、新建一个基于对话框的MFC工程TestControlDemo，将CustomDLLDemo.dll拷贝到工程中。

6、在CTestControlDemoApp::Instance()中CWinApp::InitInstance函数前加入:

LoadLibrary("CustomDLLDemo.dll")

7、在对话框中加入自定义控件，Class设置为“MyCustomControl”.

8、运行程序结果如下，这是一个没有添加任何消息响应函数的控件。



图5 未添加任何响应函数的运行结果

9、由于CMyControl类被封装在DLL中了，因此，为控件添加消息响应函数只能在父窗口中，也应该在父窗口中。为父窗口添加一个OnClicked函数，其实现如下：

void CTestControlDemoDlg::OnClicked()

{

AfxMessageBox("Clicked!");

}

再添加一个ON\_CONTROL映射宏：

ON\_CONTROL(0,IDC\_CUSTOM1,OnClicked)

这样就建立好了消息映射，运行程序如下：

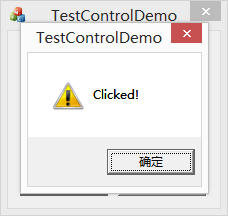


图6、添加了响应控件的Notify Code 为0 的消息函数的结果

10、在stdafx.h中添加消息定义：

#define WM\_SETCONTROLTEXT (WM\_USER +0 )

11、在主对话框中为“确定”按钮添加响应函数，加入以下代码：

GetDlgItem(IDC\_CUSTOM1)->SendMessage(WM\_SETCONTROLTEXT,0,0,LPARAM("New Text!");

运行程序，结果如下：

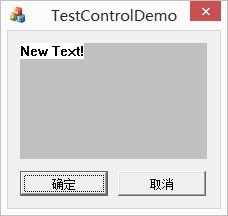


图7、为控件发送WM\_SETCONTROLTEXT消息的结果

由以上实例可以看出，用DLL来实现自定义控件，需要清楚的知道内部与外部的接口，例如能接受哪些消息，可以向父窗口发送什么消息，还有窗口类名。