**关于MDI模式下的“对话栏”的介绍以及在对话栏上增加控件**

**[基于MFC]**

在使用Visual Studio进行基于MFC进行MDI编程时，不可避免地会遇到“对话栏（CDialogBar）”这个概念。本文介绍了对话栏的作用以及如何在对话栏上增加控件。

**一、什么是对话栏及其作用**

根据微软MSDN上的解释，对话栏是工具栏，即一种可包含任何类型的控件的控制条。因具有无模式对话框的特征，CDialogBar 对象提供了功能更强大的工具栏。

工具栏和 CDialogBar 对象之间存在几个主要差异。CDialogBar 对象从对话框模板资源创建，而对话框模板资源可用 Visual C++ 对话框编辑器创建，并且可包含任何类型的 Windows 控件。用户可在控件之间切换，并且可以指定对齐样式，将对话栏同父框架窗口的任何部分对齐，甚至可以在调整父窗口大小时将对话栏留在原位。下图显示了具有不同控件的对话栏。

对话栏：



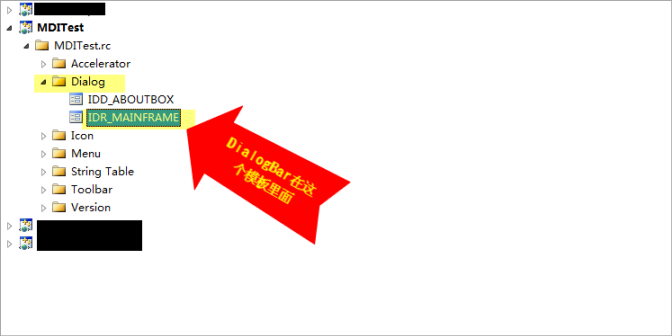
在其他方面，使用 *CDialogBar* 对象与使用无模式对话框一样。使用对话框编辑器可设计并创建对话框资源。对话栏的一个优点是可包含按钮以外的控件。

虽然从 *CDialog* 派生出您自己的对话框类很正常，但一般不派生您自己的对话栏类。对话栏是主窗口的扩展；任何对话栏控件通知消息（如 BN\_CLICKED 或EN\_CHANGE）都发送到对话栏的父级，即主窗口。

（此处摘自MSDN网站）

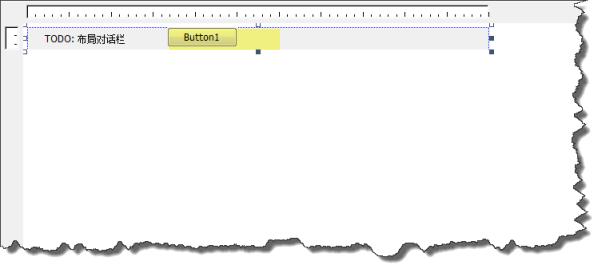
**二、对话栏在项目源代码中的什么地方？如何增加控件？**

如果使用Visual Studio 2010创建MDI项目的话，在VisualStudio工具中打开资源管理器，会看到一个ID为“IDR\_MAINFRAME”的对话框模板。如下图：

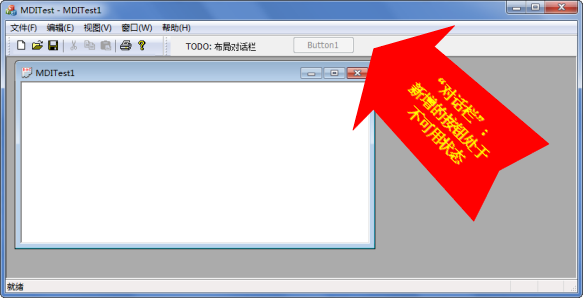


点开该模板，会看到如下图所示的对话栏模板的样式。您会看到该对话栏上已经有了一个ID为“IDC\_STATIC”的静态文本控件，标题为“TODO: 布局对话栏”。（Button1是后加上去的）。

然后您可以通过Visual Studio的工具栏，增加一个按钮控件。（相关属性如果保持默认值得话，按钮文字就是“Button1”，ID为“IDC\_BUTTON1”，如下图所示的按钮）。



编译运行后，可以看到画面如下图所示。其中右上角有一个对话栏，对话栏上有一个按钮叫做“Button1”。



很奇怪，为什么Button1是灰色的，用鼠标点击也无任何响应呢？

而且，对话栏的背景色也与旁边的ToolBar、Menu的背景色不一致！

**三、如何为对话栏上的控件增加响应事件**

通过VisualStudio在对话栏上增加的控件，默认是非激活状态的，所以运行程序后，看到的控件无法响应鼠标事件。

会不会是因为没有为该对话栏（相当于对话框）创建C++类造成的呢？（您可以在该对话栏上点击右键，然后选择“类向导”，然后会发现该对话栏没有对应的C++类）。

实际上您为该对话栏创建了新的C++类，也是无法解决这个问题的。

而且MSDN不建议您为该对话栏创建对应的C++类，因为对话栏是主窗口的扩展，消息由主窗口类（CMainFrame）处理即可：

——摘自：虽然从 *CDialog* 派生出您自己的对话框类很正常，但一般不派生您自己的对话栏类。对话栏是主窗口的扩展；任何对话栏控件通知消息（如 BN\_CLICKED 或EN\_CHANGE）都发送到对话栏的父级，即主窗口。

**1、具体做法**

**a)在CMainFrame类中增加消息映射**：

                ON\_UPDATE\_COMMAND\_UI(IDC\_BUTTON1, OnUpdateButton1)

            注：CMainFrame类实现的加到BEGIN\_MESSAGE\_MAP与END\_MESSAGE\_MAP中间。

**b）增加消息处理函数**

                afx\_msg void OnUpdateButton1(CCmdUI\* pCmdUI);

            该函数实现如下：

            void CMainFrame::OnUpdateButton1(CCmdUI\* pCmdUI)

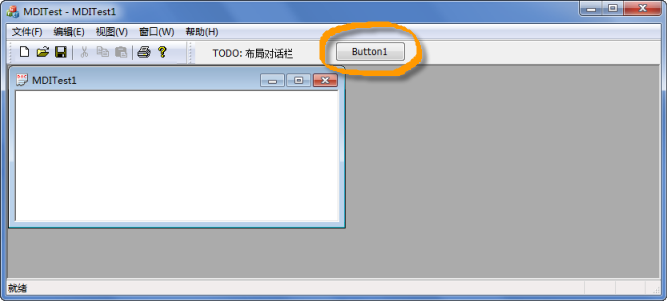
            {

                            pCmdUI->Enable( TRUE );

            }

**c）运行程序查看效果**

可以看到，Button1已经处于激活状态了。



**2、原理分析**

**a）Windows的消息循环机制**

在Windows操作系统中，消息机制是核心。任何键盘、鼠标及其他I/O设备的操作结果，均经由Windows操作系统分发至各个应用程序。分发是通过消息机制来实现的。

具体可参考MSDM：[Messages and Message Queues](http://msdn.microsoft.com/en-us/library/ms632590(v=vs.85).aspx)

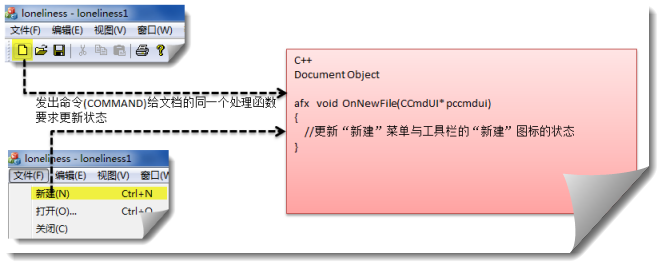
在传统的windows编程模式下，Windows消息的处理是通过在WinMain函数中放置一个巨大的switch分支处理开展的。MFC不同，MFC中是用message maps机制来将消息映射到唯一的一个类成员函数上去。这个机制允许消息被适当的C++类对象处理，比如应用程序对象、视图对象、文档对象等。你可以映射一个或一组消息（以WM\_开头的消息，除了WM\_COMMAND）、控件ID（诸如按钮/多选框等的IDs）、命令ID（Command IDs，诸如菜单/工具栏/快捷键的IDs）。

**b）拥有状态的用户接口的状态更新**

通常，菜单项、工具栏和对话栏等用户接口（UI），具有一个以上的状态。如果它们在当前上下文环境中不可用，它们会变成灰色（无效）。比如菜单可以在当前上下文环境下是无效的（灰色的），另一种上下文环境下是有效的。当程序条件改变时，谁来更新这些项目的状态？理论上，用户接口会发出“激活我吧”、“让我无效吧”等与状态匹配的命令，如果发给某文档（MFG的View-Document模式下的Document对象），那么就需要文档（Document）来更新这个菜单的状态。该文档就包含了基于此来更新状态的信息。

如果一个命令（COMMAND）有多个用户接口（UI）对象（也许是一个菜单和一个对话栏或工具栏），那么将都映射到同一个处理函数，这就将您的多个等效的用户接口（UI）更新代码封装到了一个地方。比如一个菜单项（打开文件）与一个工具栏项（打开文件），它们的作用是等效的，所以更新它们状态的处理函数可以是一个。当它们的其中一个发出“使我无效吧”的命令（COMMAND）后，处理函数会把它们俩都更新为无效的。

大概的示例如下：



**c)宏ON\_UPDATE\_COMMAND\_UI**

为了使处理状态变化命令的函数得到响应，需要使用ON\_UPDATE\_COMMAND\_UI宏，来将命令与函数映射关系建立起来。

比如，在MainFrame类中处理对话栏上的按钮的状态的话，则添加ON\_UPDATE\_COMMAND\_UI(IDC\_BUTTON1, OnUpdateButton1)

其中处理函数OnUpdateButton1的原型为：afx\_msg void OnUpdateButton1(CCmdUI \*pCmdUI);

ON\_UPDATE\_COMMAND\_UI会在窗口被初始化完毕时、菜单弹出时以及其它许多情况下得到调用。

**d)类CCmdUI**

当处理函数接收到一个update command时，框架传递一个类CCmdUI或其派生类对象的指针给这个处理函数。这个指针指向的对象，就是代表发出update command的菜单、工具栏、快捷键、对话栏等UI。

CCmdUI类提供一个函数原型为“void Enable(BOOL bON = 1)”的方法来供调用，传递true来使UI可用，传递false使UI不可用。

**三、结束语**

本文只是简单地介绍了对话栏，其具体应用，您可以根据需要扩展。

MFC框架提供的message maps机制，允许用户更加方便地进行Windows的消息传递与处理，如果要更深入地开展Windows编程，以及对话栏相关的编程的话，深入学习Windows的消息处理机制仍然是必要的。