wxWindows 2

用 C++编写跨平台程序

中文版说明

本教程由郭晓龙翻译并将继续维护，这是本教程的第一稿，如果发现错误请与我(Email:gxl117 [at] yahoo.com.cn)联系让我能及时修正它。之后还会对这个教程进行更多的扩充，欢迎广大网友提供意见。

本教程遵循 GPL 协议发布。

本许可授权你制作和发布本教程的拷贝，但在所有拷贝上要保留本版权声明和许可声明。

如果你准备出版本文档，请告之译者，以确保你获得本文档的最新版本。

对本文档的适用范围不作担保，它仅仅是作为一个免费的资源提供。因此，这里提供的这些信息的作者和维护者无法做出这些信息一定正确的保证。

Franky Braem

Copyright © 2001-2002 Franky Braem

You are allowed to copy and to print this book as long as you don't exploit the information commercially. The author of this book makes no warranty of any kind, expressed or implied, with regard to the programs or the documentation contained in this book.

目录

List of Figures

List of Tables

List of Examples

Chapter 1. Introduction

Table of Contents

为什么使用 wxWindows?

你希望使用 C++编写的同一个程序能够运行在 Windows, Linux 或者 Unix 上吗? 当每一个平台都有它们自己的 framework，外观、行为或者 SDK 时，这几乎是不可能的。当然你也肯定不想为每一个平台都重写你的程序，这将是极难维护的。

wxWindows 是一个解决方案。wxWindows 为你隐藏了全部平台相关的代码。它是一个与平台无关的 framework，它有如下特点：

- 它是非常全面的，拥有很多实用的类。It is very complete. There are many utility classes.
- 它仍然在快速的发展中。It is still heavily developed.
- 支持很多的编译器与平台： Windows, Linux, Mac, Unix.
- 拥有大量的文档。There's a lot of documentation.
- 个人与商业机构都可以自由的使用它。It's free for personal and commercial use.
- 只要可能 wxWindows 就使用本地 SDK。这表示如果一个程序是在 Windows 下编译的将有典型的 windows 程序的外观与行为， 当它在 Linux 下编译时它将拥有 linux 程序的外观与行为。

wxWindows 的历史

Julian Smart 1992 年在 Edinburgh 大学的人工智能程序学院开始开发 wxWindows。在 1995 年 Markus Holzem 完成了 Xt 版本的移植。在 1997 年 Windows 和 GTK+ 的移植版整合并放入了 CVS 档案库.

Hello World

这第一个例子是众所周知的 Hello World 程序。这个程序是一个在状态栏里显示"Hello World"的窗口。

每个wxWindow程序都要有一个继承自wxApp的对象。每个程序都要用OnInit()方法来实例化，你可以在这创建主窗口。 例 1.4 是 HelloWorldApp的定义：

Example 1.4. HelloWorldApp.h - The HelloWorldApp definition

```
#ifndef INCLUDED_HELLOWORLDAPP_H
#define INCLUDED_HELLOWORLDAPP_H
/** * HelloWorldApp 类  * 这个类显示一个状态栏中包含文本"Hello World"的窗口  */
class HelloWorldApp : public wxApp
{
public: virtual bool OnInit();
};
DECLARE_APP(HelloWorldApp)
#endif // INCLUDED_HELLOWORLDAPP_H
```

对于主窗口我们使用wxFrame类。这个类提供了一个可以调整大小与位置的窗口。它有粗的边框与一个标题栏。另外你可以让它有一个菜单栏、工具栏、状态栏。例 1.5 是 HelloWorldApp的实现.

例 **1.5. HelloWorldApp.cpp - HelloWorldApp 的实现**

```
//对支持预编译编译器要包含"wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
        #include "wx/wx.h"
#endif
#include "HelloWorldApp.h"
IMPLEMENT_APP(HelloWorldApp)
/* 程序从这里开始执行，类似非 wxWindows 程序中的 main() */
bool HelloWorldApp::OnInit()
{
        wxFrame *frame = new wxFrame((wxFrame*) NULL, -1, _T("Hello World"));
        frame->CreateStatusBar();
        frame->SetStatusText(_T("Hello World"));
        frame->Show(TRUE);
        SetTopWindow(frame);
        return true;
}
```

当你的编译器支持预处理器时，你可以使用 wxprec 头文件。当它不支持时，你应该包含 wx.h，它包含了所有必须的 wxWindows 头文件。你同样也可以为每一个控件分别包含相对应的头文件。(译者注：在 Linux 之下使用 wxFrame 的构造函数对其中的字符串要使用 wxChar 进行显式转换，对于多字节字符的支持可以使用宏 wxT(string)来解决)

宏 DECLARE_APP 和 IMPLEMENT_APP 为我们作下列操作：

当平台需要时，它创建一个 main() 或者 WinMain() 方法。
它建立一个全局方法 wxGetApp(). 你能够使用这个函数去得到一个程序对象的引用：

- **Example 1.6. wxGetApp() 使用方法**
- HelloWorldApp &app = (HelloWorldApp&) wxGetApp();

你可能会奇怪为什么在程序的任何地方都没有删除 frame 变量的代码。因为 frame 被设置为程序的顶层窗口所以程序将在退出时自动的删除 frame。

有一些糟糕的编译器不允许 NULL 隐式的转换到 wxFrame*，所以这就是我们为什么要显示的作它，仅仅是出于稳妥的考虑（不过如果你真的在使用这样的编译器的话，我建议你还是更新它吧）。 同时这样作也更有利于程序的移植。

在 frame 建立之后，使用 CreateStatusBar 来创建一个状态栏。状态栏的文字内容设置为 "Hello World".调用 show()来显示 frame。Show()是一个 wxFrame 从 wxWindow 类继承的方法。
当 OnInit 返回 false 时，程序将立即停止。如果在程序初始化期间一些事情发生了错误，你可以利用它来停止程序。

第二章使用 wxFrame

目录

wxFrame类提供给我们一个框架窗口。框架窗口是一个可以改变大小的窗口，它有粗的边框与一个标题栏。另外你可以让它有一个菜单栏、工具栏、状态栏。框架可以作为其它控件的窗口器，但不能包含另一个窗口或对话框。wxFrame是从wxWindow类派生来的。

在这一章里我们将创建一个小小的文本编辑器。

创建 frame

通常你能够通过继承wxFrame来创建你自己的类。这样你可以向你自己的frame类中添加功能。 例 2.1 就是这样作的，它的实现在例 2.2中。

例 2.1 -定义 TextFrame

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H
class TextFrame : public wxFrame
{
```

```
    public:
    /**
     * 构造函数. 用来创建新的 TextFrame
     */
     TextFrame(const wxChar *title, int xpos, int ypos, int width, int height);
     /**
      * 析构函数
      */
     ~TextFrame();
};
#endif //_TEXTFRAME_H
```

TextFrame 的构造函数继承自 wxFrame 的构造函数。

例 2.2. TextFrame 的实现

```
// 对支持预编译编译器要包含"wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
      #include "wx/wx.h"
#endif
#include "TextFrame.h"
TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
      : wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width,
height))
{
}
TextFrame::~TextFrame()
{
}
```

构造函数

```
wxFrame(wxWindow* parent,
        wxWindowId id,
        const wxString& title,
        const wxPoint& pos= wxDefaultPosition,
        const wxSize& size= wxDefaultSize,
        long style= wxDEFAULT_FRAME_STYLE,
        const wxString& name= "frame");
```

- *parent* 是一个指向父窗口的指针，当框架没有父窗口时使用NULL。
- *id* 是唯一的窗口标识号，当你不需要它时使用-1.
- *title* 是框架的标题，它将显示在框架的标题栏中。
- *pos* 是框架的位置。WxDefaultPosition表示默认值。
- *size* 是框架的大小。 WxDefaultSize表示使用默认值。
- *style* 是框架的风格。

表 **2.1. wxFram** 风格

wxDEFAULT_FRAME_STYLE	wxMINIMIZE_BOX, wxMAXIMIZE_BOX, wxRESIZE_BOX, wxSYSTEM_MENU 和 wxCAPTION 的组合。
wxICONIZE	以最小化方式显示窗口(仅用于 windows)
wxCAPTION	显示标题
wxMINIMIZE	与 wxICONIZE 相同
wxMINIMIZE_BOX	框架将有一个最小化按钮
wxMAXIMIZE	框架以最大化方式显示(仅用于 windows)
wxMAXIMIZE_BOX	框架将有一个最大化按钮
wxSTAY_ON_TOP	框架将位于其它窗口的上层(仅用于 windows)
wxSYSTEM_MENU	框架拥有一个系统菜单。
wxSIMPLE_BORDER	显示一个没有边框的框架(仅用于 GTK 与 Windows)
wxRESIZE_BORDER	显示一个可调整大小的边框(仅用与 Unix)
wxFRAME_FLOAT_ON_PARENT	The frame will be above the parent window in the z-order and is not shown in the task bar. (Windows and GTK only)
wxFRAME_TOOL_WINDOW	显示一个小的标题栏(仅用于 Windows)

name 是窗口的名字。这个参数是用来将一个条目与一个名称相关联的。这允许用户去为每个窗口设置Motif资源值。

例 2.3 和例 2.4 是程序的实现

2.3. TextEditorApp.h - TextEditorApp 的定义

```
#ifndef TEXTEDITORAPP_H
#define TEXTEDITORAPP_H
class TextEditorApp : public wxApp
{
public:
    /**
     * 程序初始化
     */
    virtual bool OnInit();
};
DECLARE_APP(TextEditorApp)
#endif // TEXTEDITORAPP_H
```

例 **2.4. TextEditorApp.cpp - TextEditorApp** 的实现

```
// 对支持预编译编译器要包含"wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
	#include "wx/wx.h"
#endif
#include "TextEditorApp.h"
#include "TextFrame.h"
IMPLEMENT_APP(TextEditorApp)
bool TextEditorApp::OnInit()
{
	TextFrame *frame = new TextFrame("Simple Text Editor", 100, 100, 400, 300);
	frame->Show(TRUE);
	SetTopWindow(frame);
	return true;
}
```

添加控件

现在已经创建了一个窗口框架，你需要添加一些控件来处理文本。类wxTextCtrl将是我们所需要的。框架类已经包含了这个控件。

例 2.5 是一个新的TextFrame类的定义。只有一件事是改变的，它添加了一个wxTextCtrl类型的成员。这个成员是在构造函数中初始化的。

例 **2.5. TextFrame** 的重新定义

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H
class TextFrame : public wxFrame
{
   public:
    /**
     * 构造函数. 用来创建新的 TextFrame
     */
     TextFrame(const wxChar *title, int xpos, int ypos, int width, int height);
     /**
      * 析构函数
      */
     ~TextFrame();
private:
     wxTextCtrl *m_pTextCtrl;
};

#endif //_TEXTFRAME_H
```

例 2.6 是更新后的TextFraem类的构造函数。TextFrame是wxTextCtrl成员的父类。所以你应该传递this指针。字符串"Type some text..."将作为默认的文本。注意wxTextCtrl的构造函数看上去与wxFrame的相仿。这是因为每一个类都是从wxWindow继承来的所以有相同的构造函数模式。

另外，这里没有删除 wxTextCtrl 指针的代码。这里不需要它（实际上是不允许），因为当父类 TextFrame 撤消时它的所有子类都会自动的删除。

wxTE_MULTILINE 是文本控件专用的窗口风格。这种风格表示 wxTextCtrl 允许多行。

例 **2.6.TextFrame** 的实现

```
// 对支持预编译编译器要包含"wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
	#include "wx/wx.h"
#endif
#include "TextFrame.h"
TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
	: wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width,
height))
{
	m_pTextCtrl = new wxTextCtrl(this, -1, wxString("Type some text..."),
				wxDefaultPosition, wxDefaultSize,
wxTE_MULTILINE);

}
TextFrame::~TextFrame()
{
}
```

当你 build 这个项目时，你将拥有一个可以输入一些文本的窗口，你可以尝试在这个窗口中剪切、粘贴文本，你将会出现这些短短的代码已经为你作很多事了。

添加菜单栏

通常，我们总向自己的程序添加菜单栏来帮助用户操作程序， wxWindows 为菜单栏提供了下面的类：wxMenuBar 和 wxMenu.

每一个菜单都需要一个独立的 ID。这是通过一个枚举类型来完成的。而不能使用#define 定义的常量（比如：#define MENU_FILE_MENU 1)因为这不能保证你有独一无二的 ID。它十分容易漏掉一些值并且当你想插入新的 ID 时这会变的十分难以维护。

请参考 wxWindows 手册, wxWin 预定义了一些你在使用文档/视图 framework 时会用到的 ID。

Example 2.7. TextFrame 的定义

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H
class TextFrame : public wxFrame
{
   public:
   /**
    * Constructor. Creates a new TextFrame
```

```
    */
    TextFrame(const wxChar *title, int xpos, int ypos, int width, int height);
    /**
     * Destructor
     */
    ~TextFrame();
private:
    wxTextCtrl *m_pTextCtrl;
    wxMenuBar *m_pMenuBar;
    wxMenu *m_pFileMenu;
    wxMenu *m_pInfoMenu;
    enum
    {
       MENU_FILE_OPEN,
       MENU_FILE_SAVE,
       MENU_FILE_QUIT,
       MENU_INFO_ABOUT
    };
};
#endif //_TEXTFRAME_H
```

菜单栏是在 TextFrame 的构造函数中创建的。一个菜单项是使用 wxMenu 的 Append 方法添加到菜单中的。要注意&号是如何指出哪个字符是快捷键的。当菜单创建后你还要使用 wxMenuBar 的 Append 方法把它添加到菜单栏。

Example 2.8. TextFrame 的实现

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
      #include "wx/wx.h"
#endif
#include "TextFrame.h"
TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
      : wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width,
height))
{
      m_pTextCtrl = new wxTextCtrl(this, -1, wxString("Type some text..."),
                                   wxDefaultPosition, wxDefaultSize,
wxTE_MULTILINE);
      m_pMenuBar = new wxMenuBar();
      // File Menu
      m_pFileMenu = new wxMenu();
      m_pFileMenu->Append(MENU_FILE_OPEN, "&Open");
      m_pFileMenu->Append(MENU_FILE_SAVE, "&Save");
      m_pFileMenu->AppendSeparator();
      m_pFileMenu->Append(MENU_FILE_QUIT, "&Quit");
      m_pMenuBar->Append(m_pFileMenu, "&File");
      // About menu
      m_pInfoMenu = new wxMenu();
```

```
    m_pInfoMenu->Append(MENU_INFO_ABOUT, "&About");
    m_pMenuBar->Append(m_pInfoMenu, "&Info");
    SetMenuBar(m_pMenuBar);
}
TextFrame::~TextFrame()
{
}
```

添加状态栏

为框架添加一个状态栏是很容易的。使用 wxFrame 类中的 CreateStatusBar 方法来创建状态栏，它接受一个整形参数作为状态栏中的区域个数，使用 SetStatusText 方法来改变各域中的文本内容。

Example 2.9. 状态栏的创建

```
CreateStatusBar(3);
SetStatusText("Ready", 0);
```

Example 2.9 创建一个有三个域的状态栏。第一个域包含文本"Ready"。

状态栏可用来显示对一个菜单项的描述。例 2.10 对上一节中的向菜单栏中添加菜单项的例 2.8 作了更改，以使菜单项与状态栏关联。当一个菜单项被选择时，相关的描述就会显示在状态栏的第一个域中。

例 2.10 包含功能描述的菜单项

```
// 文件菜单

m_pFileMenu = new wxMenu();

m_pFileMenu->Append(MENU_FILE_OPEN, "&Open", "Opens an existing file");

m_pFileMenu->Append(MENU_FILE_SAVE, "&Save", "Save the content");

m_pFileMenu->AppendSeparator();

m_pFileMenu->Append(MENU_FILE_QUIT, "&Quit", "Quit the application");

m_pMenuBar->Append(m_pFileMenu, "&File");



// 关于菜单

m_pInfoMenu = new wxMenu();

m_pInfoMenu->Append(MENU_INFO_ABOUT, "&About", "Shows information about the
application");
```

m_pMenuBar->Append(m_pInfoMenu, "&Info");

处理菜单事件

现在 frame 上已经有了菜单，当用户选择一个菜单时我们要用一种方法来实现它的相应动作。每选择一个菜单都会产生一个事件。如何让一个动作与这个事件相关联呢？一个技巧就是将一个类的方法与事件相关联。在 wxWindows 的早期版本中这是通过使用回调函数或者通过重载虚拟函数来实现的。从 wxWindows2.0 开始改为使用事件表。在这一节里我们仅解释通过菜单产生的事件。关于事件处理的详细过程我们将在第三章解释。

要处理事件的每个类都需要声明一个事件表。宏 DECLARE_EVENT_TABLE 用来完成这项工作。每一个事件都必须有一个已经实现了的方法。每个方法都有一个参数用来包含事件的信息。从菜单获得的事件是一个 wxCommandEvent 类型的数据。例 2.11 是类 TextFrame 的完整定义。

例 2.11. TextFrame.h 的完整定义

```
#ifndef _TEXTFRAME_H
#define _TEXTFRAME_H
class TextFrame : public wxFrame
{
public:
    /**
     * Constructor. Creates a new TextFrame
     */
    TextFrame(const wxChar title, int xpos, int ypos, int width, int height);
    /**
      * Destructor
      */
    ~TextFrame();
    /**
      * Processes menu File|Open
      */
    void OnMenuFileOpen(wxCommandEvent &event);
    /**
      * Processes menu File|Save
      */
    void OnMenuFileSave(wxCommandEvent &event);
    /**
      * Processes menu File|Quit
      */
    void OnMenuFileQuit(wxCommandEvent &event);
    /**
      * Processes menu About|Info
      */
    void OnMenuInfoAbout(wxCommandEvent &event);
protected:
    DECLARE_EVENT_TABLE()
```

```
private:
    wxTextCtrl *m_pTextCtrl;
    wxMenuBar *m_pMenuBar;
    wxMenu *m_pFileMenu;
    wxMenu *m_pInfoMenu;
    enum
    {
       MENU_FILE_OPEN,
       MENU_FILE_SAVE,
       MENU_FILE_QUIT,
       MENU_INFO_ABOUT
    };
};
#endif //_TEXTFRAME_H
```

事件表是放在实现文件中的，wxWindows 通过一些宏来帮助的完成事件表的声明。宏 BEGIN_EVENT 在事件表声明的开始处使用，因为在一个程序中可能不止一个事件表，要将事件处理过程相关的类名传递给宏。将一个方法与事件关联要使用 EVT_MENU 宏。这个宏需要菜单 ID 与事件名。在事件表的最后用宏 END_EVENT_TABLE 作结束标记。例 2.12 显示了完整的实现文件。

Example 2.12. TextFrame.cpp - The full implementation

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
      #include "wx/wx.h"
#endif
#include "TextFrame.h"
TextFrame::TextFrame(const wxChar *title, int xpos, int ypos, int width, int height)
      : wxFrame((wxFrame *) NULL, -1, title, wxPoint(xpos, ypos), wxSize(width,
height))
{
      m_pTextCtrl = new wxTextCtrl(this, -1, wxString("Type some text..."),
                                   wxDefaultPosition, wxDefaultSize,
wxTE_MULTILINE);
      CreateStatusBar();
      m_pMenuBar = new wxMenuBar();
      // File Menu
      m_pFileMenu = new wxMenu();
      m_pFileMenu->Append(MENU_FILE_OPEN, "&Open", "Opens an existing
file");
      m_pFileMenu->Append(MENU_FILE_SAVE, "&Save", "Saves the file");
      m_pFileMenu->AppendSeparator();
      m_pFileMenu->Append(MENU_FILE_QUIT, "&Quit, "Close the application");
      m_pMenuBar->Append(m_pFileMenu, "&File");
      // About menu
      m_pInfoMenu = new wxMenu();
      m_pInfoMenu->Append(MENU_INFO_ABOUT, "&About", "Shows info about
the application");
```

```
        m_pMenuBar->Append(m_pInfoMenu, "&Info");
        SetMenuBar(m_pMenuBar);
}
TextFrame::~TextFrame()
{
}
BEGIN_EVENT_TABLE(TextFrame, wxFrame)
     EVT_MENU(MENU_FILE_OPEN, TextFrame::OnMenuFileOpen)
     EVT_MENU(MENU_FILE_SAVE, TextFrame::OnMenuFileSave)
     EVT_MENU(MENU_FILE_QUIT, TextFrame::OnMenuFileQuit)
     EVT_MENU(MENU_INFO_ABOUT, TextFrame::OnMenuInfoAbout)
END_EVENT_TABLE
void TextFrame::OnMenuFileOpen(wxCommandEvent &event)
{
   wxLogMessage("File Open Menu Selected");
}
void TextFrame::OnMenuFileSave(wxCommandEvent &event)
{
   wxLogMessage("File Save Menu Selected");
}
void TextFrame::OnMenuFileQuit(wxCommandEvent &event)
{
   Close(FALSE);
}
void TextFrame::OnMenuInfoAbout(wxCommandEvent &event)
{
   wxLogMessage("File About Menu Selected");
}
```

第三章事件处理

目录

在上一章里已经学习了如何处理菜单事件。这章将解释事件处理是如何工作的，同时解释了如何去处理其它事件。

介绍 Introduction

事件是出现在程序内部或外部的一些事情。一个事件可能是通过用户或是其它程序、操作系统等来触发的。这时需要一个机制来让程序对期望的事件产生反应。

在 wxWindows 的早期版本中，程序的事件处理是完全通过回调函数或者重载虚拟函数来实现的。从 wxWindows2.0 开始转而使用事件表了。如例 3，事件表告诉 wxWindows 将事件映射到成员函数。事件表是在实现文件(cpp)中声明的。

例 3.1.事件表

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(wxID_EXIT, MyFrame::OnExit)
    EVT_SIZE(MyFrame::OnSize)
    EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
END_EVENT_TABLE()
```

上面的事件表告诉wxWindows当它接收到一个WM_SIZE事件时调用MyFrame的成员函数OnSize。 宏 *BEGIN_EVENT_TABLE* 声明wxFrame和它的子类MyFrame拥有这个事件表。

处理事件的成员函数不能是虚拟的。实际上事件处理器将忽略虚拟声明。事件处理函数有类似的形式：返回类型为 void 并且接受一个事件参数。这个参数的类型与具体的事件相关。对于 size 事件，使用的类型是 wxCommandEvent。当控件变的更复杂时它们使用自己的事件类。

在类定义中，*必须有一个DECLARE_EVENT_TABLE宏。参见*例 2.11。

所有这些宏隐藏了复杂的事件处理系统。

它是如何工作的？

当接收到一个事件时，wxWindows 首先调用窗口上产生事件的对象上的 wxEventHandler 的 ProcssEvent 处理器。wxWindow（和其它的窗口类）继承自 wxEventHander。ProcessEvent 查找事件表里的每一个事件并且调用零或多个事件处理器函数。下面是处理一个事件的步骤：

1. 当对象关闭时（包括 wxEvtHandler 的 SetEvtHandle)函数跳转到第六步。
2. 如果对象是一个 wxWindow 对象，ProcessEvent 在窗口的 wxValidator 上递归调用。如果返回真函数退出。
3. SearchEventTable是事件处理器调用的函数。当它失败时，开始在基类上尝试直到没有更多的事件表或者发现了一个合适的函数时这个函数退出。被发现的函数开始执行。
4. 查找过程应用于整个事件处理器链，当应用成功时（表示事件处理完毕）函数退出。
5. 当对象是一个 wxWindow 对象时，并且事件为 wxCommandEvent 类型时，ProcessEvent 向上递归应用于父窗口的事件处理器。当它返回真，函数退出。这可以让一个按钮的父亲来处理按钮的点击而不是让按钮自己来处理。
6. 最后ProcessEvent 在wxApp对象上调用。

事件跳转 Event skipping

ProcessEvent在发现一个可以处理事件的函数后退出。这表示当你的类对一个事件起反应时，下层的类不会得到这个事件。有时我们不希望这样。这个问题

可以根据基类的事件类型用wxEvent类的Skip方法来解决，使事件处理器的查找继续进行。

下面的例子展示了事件跳转的用处。例 3.2 是一个文本控件的定义，它只接受数字字符。

例 3.2. NumTextCtrl.h

```
class NumTextCtrl : public wxTextCtrl
{
public:
   NumTextCtrl(wxWindow *parent);
   void OnChar(wxKeyEvent& event);
protected:
   DECLARE_EVENT_TABLE()
};
```

当NumericTextCtrl接收到一个键盘事件时，就进行keycode检查。如果输入的是数字，基类wxTextCtrl就可以处理这个事件。这就是我们要对这个事件使用跳转的原因。你必须在这调用Skip方法，否则基类不会处理任何键。

例 3.3. NumTextCtrl.cpp

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
      #include "wx/wx.h"
#endif
#include "NumTextCtrl.h"
#include <ctype.h>
NumTextCtrl::NumTextCtrl(wxWindow *parent) : wxTextCtrl(parent, -1)
{
}
void NumTextCtrl::OnChar(wxKeyEvent& event)
{
    if ( isdigit(event.GetKeyCode()) )
    {
       // Numeric characters are allowed, so the base class wxTextCtrl
       // is allowed to process the event. This is done using the Skip() method.
       event.Skip();
    }
    else
    {
       // Character is not allowed.
       wxBell();
    }
}
BEGIN_EVENT_TABLE(NumTextCtrl, wxTextCtrl)
    EVT_CHAR(NumTextCtrl::OnChar)
END_EVENT_TABLE()
```

禁止事件

一些事件是可以禁止的。当你禁止一个事件时，这个事件不会被进一步处理。例 3.4 演示了，当一个文本控件内的文本改变后如何禁止这个简单文本编辑器的关闭事件。这表示当在用户还没有保存改变后的文本内容时这个窗口不能被关闭。

例 3.4. 禁止事件

```
void TextFrame::OnClose(wxCloseEvent& event)
{
   bool destroy = true;
   if ( event.CanVeto() )
   {
      if ( m_pTextCtrl->IsModified() )
      {
         wxMessageDialog *dlg =
                     new wxMessageDialog(this, "Text is changed!\nAre you sure you
want to exit?",
                                                 "Text changed!!!", wxYES_NO |
wxNO_DEFAULT);
         int result = dlg->ShowModal();
         if ( result == wxID_NO )
         {
            event.Veto();
            destroy = false;
         }
      }
   }
   if ( destroy )
   {
      Destroy();
   }
```

当 CanVeto 返回 false 时程序作什么呢？你将不能禁止这个事件你的程序将会退出。

阻塞事件处理器 Plug an event handler

考虑下面的问题：每个菜单命令都必须被记录。一个解决方案是创建一个在每个命令事件处理函数中调用的函数。这种方法带来的问题是使维护变得十分困难。当添加一个新的菜单并且没有调用这个函数时，这个菜单命令将不被记录。

解决这个问题是去创建一个新的事件处理器并添加到一个 wxWindow 类。要完成它需要创建一个从 wxEvtHandler 派生的新类。在新类中处理事件与一个正常的窗口是相同的。

例 3.5. LogEventHandler.h -LogEventHandler 的定义

```
#ifndef _LogEventHandler_H
#define _LogEventHandler_H
class LogEventHandler : public wxEvtHandler
```

```
{
public:
   LogEventHandler() : wxEvtHandler()
   {
   }
   virtual ~LogEventHandler()
   {
   }
protected:
   DECLARE_EVENT_TABLE()
   void OnMenu(wxMenuEvent &event);
private:
};
#endif // _LogEventHandler_H
```

例 3.6. LogEventHandler.cpp - LogEventHandler 的实现

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
      #include "wx/wx.h"
#endif
#include "LogEventHandler.h"
void LogEventHandler::OnMenu(wxMenuEvent &event)
{
   wxLogMessage("Someone selected a menu item");
   event.Skip();
}
BEGIN_EVENT_TABLE(LogEventHandler, wxEvtHandler)
      EVT_MENU_RANGE(-1, -1, LogEventHandler::OnMenu)
END_EVENT_TABLE()
```

在宏EVT_MENU_RANGE里，所有的菜单事件都可以被处理。前两个参数用来指定要处理的菜单的ID范围。-1 表示处理所有的菜单项。不要忘记在事件上调用Skip，否则不会有任何事件传递到类wxWindow的事件处理器。

下一步是把新的事件处理器压入处理器栈。这是使用 wxWindow 的 PushEventHandler 来完成的。要从栈上移除事件处理器使用 PopEventHandler。

```
PushEventHandler(new LogEventHandler());
```

PopEventHandler 有一个布尔类型的参数，当这个参数为真时，wxWindows删除事件处理器。注意在没有事件处理器被取出时,wxWndows将在窗口撤消时尝试删除事件处理器。这在访问在栈上创建的事件处理器时会发生一个访问错误。可以在一个类撤消之前调用使用false参数的PopeventHandler来避免这个问题。

Chapter 4. 通用对话框

Table of Contents

在第二章的例子中有打开与存储一个文本文件的菜单命令。这时要使用一个通用对话框 wxFileDialog 来让用户选择或者输入一个文件名。在使用通用对话框时，用户得到一个统一的界面。这可以让用户不管在什么样的程序中都可以看到类似的对话框。

wxWindows 提供了下列通用对话框。

- wxFileDialog 一个存储或打开文件的对话框。
- wxColourDialog 一个选择颜色的对话框。
- wxFontDialog 选择字体的对话框。
- wxPrintDialog 用于打印与设置打印机的对话框。
- wxDirDialog 选择目录的对话框。

- wxTextEntryDialog 请求用户输入一行文本的对话框。
- wxMessageDialog 用于显示一行或多行信息，可包括OK，Yes，No，与Cancel按钮。
- wxSingleChoiceDialog 显示一个包含字符串列表的对话框，并且允许用户选择其中一个。

wxWindows 是交叉平台的 FrameWork。当一个通用对话框在某个特别的平台上不能使用时，wxWindow 将显示一个一般对话框。

注意

在下面的每一个对话框的例子中，将使用 Destroy 方法来代替用删除指针的方法来撤消窗口。这是为了稳定的目的。WxWindows 控件应该总是在堆上创建并且使用 Destroy 方法来安全的移除。因为 wxWindows 有时会延迟删除，直到所有事件都被处理完毕。这样 wxWindows 可以避免将事件发送到一个不存在的窗口上。

wxFileDialog

wxFileDialog 为用户提供一个当用户在打开或者储存一个文件时可以选择或者输入文件名的对话框。一个全局方法wxFileSelector同样可以用来让用户选择一个文件。详情请看章节 “wxFileSelector”.

构造函数

```
#include <wx/filedlg.h>

wxFileDialog(wxWindow* parent,
             const wxString& message= ″Choose a file″,
             const wxString& defaultDir= ″″,
             const wxString& defaultFile= ″″,
             const wxString& wildcard= ″″,
             long style= 0,
             const wxPoint& pos= wxDefaultPosition);
```

- *parent* 拥有这个对话框的窗口。
- *message* 对话框的标题。
- *defaultDir* 默认目录。
- *defaultFile* 默认文件名。
- *wildcard* 是一些类似 "*.*" or "*.txt"之类的字符串。这是用来过滤文件的。它的语法为 描述|后缀。例 "All files(*.*)|*.*|Text files(*.txt)|*.txt|Bitmap files(*.bmp)|*.bmp".
- *style* 对话框的类型。

表 **4.1. wxFileDialog** 风格

wxCHANGE_DIR	改变当前目录
wxFILE_MUST_EXIST	输入的文件必须存在

wxHIDE_READONLY	隐藏只读文件
wxMULTIPLE	只是一个打开对话框，允许选择多个文件。
wxOPEN	只是一个打开对话框。
wxOVERWRITE_PROMPT	当一个文件将被覆写时询问已确定。
wxSAVE	一个保存对话框。

- *pos* 没有使用。

方法

```
public void SetDirectory(const wxString& dir);
public wxString GetDirectory();
```

设置与得到当前目录

```
public void SetFilename(const wxString& name);
public wxString GetFilename();
```

设置与得到当前文件名

```
public const void  GetFilenames(wxArrayString& files);
```

使用所选择的文件名来添充文件名数组。这个函数只用于 wxMULTIPLE 风格的对话框。其它的使用 GetFilename。

```
public void SetFilterIndex(int filterIndex);
public int GetFilterIndex();
```

设置与得到给出的过滤器列表的索引。在对话框显示之前，这个索引决定对话框显示时首先使用的过滤器。在显示之后这个索引由用户选择。

```
public void SetMessage(const wxString& message);
public wxString GetMessage();
```

设置与等到新的对话框标题。

```
public void SetPath(const wxString& path);
public wxString GetPath();
```

设置与得到当前路径(目录与文件名)

```
public void SetStyle(long style);
public long GetStyle();
```

设置与得到对话框的风格，参看“Constructor”

```
public void  GetPaths(wxArrayString& paths);
```

用所选择的文件绝对路径名来填充路径数组。这个函数仅仅用于 wxMULTIPLE 风格的对话框。其它的使用 GetPath。

```
public void SetWildcard(const wxString& wildCard);
public wxString GetWildcard();
```

设置与得到wildcard参见"Constructor".

```
public int ShowModal();
```

显示对话框，当用户按下 OK 时返回 wxID_OK。其它情况返回 wxID_CANCEL。

例

在第二章的文本编辑器例子中，使用ShowModal来显示一个对话框。"Modal"表示在对话框关闭之前程序的其它窗口不能得到焦点。使用ShowModal 显示对话框时，当用户按下OK时返回wxID_OK。其它情况返回wxID_CANCEL。 GetFilename 是用来得到选择的文件的文件名。GetPath 将返回所选择文件的完整路径(目录与文件名)。在使用wxTextCtrl控件时加载与保存文件是非常简单的。方法 LoadFile 和 SaveFile分别用于加载与保存文件。

例 **4.1. wxFileDialog** 的使用

```
void TextFrame::OnMenuFileOpen(wxCommandEvent &event)
{
   wxFileDialog *dlg = new wxFileDialog(this, "Open a text file", "", "",
                                        "All files(*.*)|*.*|Text
Files(*.txt)|*.txt",
                                        wxOPEN, wxDefaultPosition);
   if ( dlg->ShowModal() == wxID_OK )
   {
      m_pTextCtrl->LoadFile(dlg->GetFilename());
      SetStatusText(dlg->GetFilename(), 0);
   }
   dlg->Destroy();
}
void TextFrame::OnMenuFileSave(wxCommandEvent &event)
{
   wxFileDialog *dlg = new wxFileDialog(this, "Save a text file", "", "",
                                        "All files(*.*)|*.*|Text
Files(*.txt)|*.txt",
                                        wxSAVE, wxDefaultPosition);
   if ( dlg->ShowModal() == wxID_OK )
   {
      m_pTextCtrl->SaveFile(dlg->GetPath());
      SetStatusText(dlg->GetFilename(), 0);
   }
   dlg->Destroy();
```

}

wxFileSelector

wxFileSelector 弹出一个文件选择框。当用户忽略它时返回一个空字符串。

```
wxString wxFileSelector(const wxString& message,
                        const wxString& defaultPath= "",
                        const wxString& defaultFile= "",
                        const wxString& defaultExtension= "",
                        const wxString& wildcard= "*.*",
                        int flags= 0,
                        wxWindow* parent= NULL,
                        int x= -1,
                        int y= -1);
```

- *message* 文件选择器的标题。
- *defaultPath* 默认路径i
- *defaultFile* 默认文件
- *defaultExtension* 默认的后缀。当用户省略后缀时这个后缀将自动添加到文件后。当没有指定wildcard时，wxWindows自动附加一个过滤器。
- *wildcard* (参见 wxFileDialog 构造函数).
- *flags* 是对话框的风格。参见wxFileDialog 构造函数。不支持 *wxMULTIPLE风格*
- *parent* 父窗口
- *x* 选择器的x轴位置。
- *y* 选择器的y轴位置。

例

例 4.2 要求用户选择一个 JavaScript 文件。

例 **4.2. wxFileSelector** 的使用。

```
wxString selection = wxFileSelector("Open a Javascript file", "", "", "js",
                                    "All files(*.*)|*.*|JavaScript
Files(*.js)|*.js|Text Files(*.txt)|*.txt",
                                    wxOPEN, this);
```

wxColourDialog

wxColourDialog实现了一个供用户选择颜色的窗口。WxColourDialog是与wxColourdata共同使用的。WxColourData是用来在对话框中设置当前颜色并用来得到选择的颜色。

构造函数

```
#include <wx/colordlg.h>
```

```
wxColourDialog(wxWindow* parent,
               wxColourData* data= NULL);
```

- *parent* i是我们这个对话框的父窗口。
- *data* 包含颜色信息。它包括默认颜色，定制颜色并且在windows之下你可以告诉它显示一个包含自定义颜色选择控件的填充对话框。

方法

```
public wxColourData& GetColourData();
```

得到与这个对话框关联的 coloudata 的引用。

```
public void  SetTitle(const wxString& title);
public wxString GetTitle();
```

设置或得到对话框的标题

```
public int ShowModal();
```

显示对话框，当用户按下 OK 时返回 wxID_OK。其它情况返回 wxID_CANCEL。

wxColourData

这个类包含与 wxColourDialog 相关的信息。

构造函数

```
wxColourData();
```

默认构造函数

```
wxColourData(const wxColourData& data);
```

拷贝构造函数

方法

```
public void  SetChooseFull(bool flag);
public bool GetChooseFull();
```

在 windows 上，这个方法显示一个包含自定义颜色选择控件的对话框，默认值为 true。

```
public void  SetColour(wxColour& colour);
public wxColour& GetColour();
```

设置或者得到选择的颜色。

```
public void  SetCustomColour(int i,
                             wxColour& colour);
public wxColour GetCustomColour(int i);
```

设置或得到给定位置的自定义颜色。I 必须在 0 与 15 之间。

例

例 4.3 为程序添加一个菜单项允许用户选择其它的背景颜色。当前的背景色被赋给 colourData。如果程序是运行在windows下的话将显示一个填充对话框。

例 **4.3.wxColourDialog** 的使用

```
void TextFrame::OnMenuOptionBackgroundColor(wxCommandEvent &event)
{
   wxColourData colourData;
   wxColour colour = m_pTextCtrl->GetBackgroundColour();
   colourData.SetColour(colour);
   colourData.SetChooseFull(true);
   wxColourDialog *dlg = new wxColourDialog(this, &colourData);
   if ( dlg->ShowModal() == wxID_OK )
   {
      colourData = dlg->GetColourData();
      m_pTextCtrl->SetBackgroundColour(colourData.GetColour());
      m_pTextCtrl->Refresh();
   }
   dlg->Destroy();
}
```

wxGetColourFromUser

wxGetColourFromUser 显示颜色选择对话框并返回用户选择的颜色或者在用户取消对话框时返回一个无效的颜色。使用wxColour的OK方法来测试颜色是否有效。

```
wxColour wxGetColourFromUser(wxWindow* parent= NULL,
                             const wxColour& colInit= wxNullColour);
```

- *parent* 选择器的父窗口。
- *colInit* 初始颜色。

例

例 4.4 要示用户选择一个颜色

例 **4.4. wxGetColourFromUser** 的使用

```
wxColour colour = wxGetColourFromUser();
if ( colour.Ok() )
{
```

```
  // 用户选择的颜色
}
```

wxFontDialog

wxFontDialog是供用户选择字体的对话框。这个对话框与wxFontData,wxFont和wxColour 一起使用。WxFontData用于设置对话框内的当前字体并得到选择的字体。

构造函数

```
#include <wx/fontdlg.h>

wxColourDialog(wxWindow* parent,
               wxFontData* data= NULL);
```

- *parent* 对话框的父窗口
- *data* 包含字体信息。它包含默认字体、当前字体颜色等信息。

方法

```
public wxFontData& GetFontData();
```

得到与对话框关联的fontdata的引用。

```
public int ShowModal();
```

显示对话框，当用户按下 OK 时返回 wxID_OK。其它情况返回 wxID_CANCEL。

wxFontData

```
wxFontData();
```

默认构造函数

```
public void  EnableEffects(bool flag);
public bool GetEnableEffects();
```

在 Windows 上, 指出是否使用字体效果比如下划线、斜体等。

```
public void  SetAllowSymbols(bool flag);
public bool GetAllowSymbols();
```

在 Windows 上，指出是否可以选择符号字体。

```
public void  SetColour(const wxColour& colour);
public wxColour & GetColour();
```

设置或得到字体颜色。

```
public void  SetShowHelp(bool flag);
public bool GetShowHelp();
```

On 在 windows 上，指出是否显示”帮助“按钮。

```
public void  SetInitialFont(const wxFont& font);
public wxFont GetInitialFont();
```

得到或设置对话框使用的原始字体。

```
public wxFont GetChosenFont();
```

得到选择的字体。

```
public void  SetRange(int minRange,
                      int maxRange);
```

在 windows 上，指出字体大小最大与最小值。默认情况下为 0 到无限。

例

Example 例 4.5 为第二章的文本编辑器实现了一个新的菜单项来改变编辑器内的字体。首先fontdata用当前的字体与颜色填充，当程序在Windows上运行时，用户将看到帮助按钮。当用户选择了字体时，fontdata将被赋与新的字体与文本控件的前景色。

例 **4.5. wxFontDialog** 的使用

```
void TextFrame::OnMenuOptionFont(wxCommandEvent& event)
{
   wxFontData fontData;
   wxFont font;
   wxColour colour;
   font = m_pTextCtrl->GetFont();
   fontData.SetInitialFont(font);
   colour = m_pTextCtrl->GetForegroundColour();
   fontData.SetColour(colour);
   fontData.SetShowHelp(true);
   wxFontDialog *dlg = new wxFontDialog(this, &fontData);
   if ( dlg->ShowModal() == wxID_OK )
   {
      fontData = dlg->GetFontData();
      font = fontData.GetChosenFont();
      m_pTextCtrl->SetFont(font);
      m_pTextCtrl->SetForegroundColour(fontData.GetColour());
      m_pTextCtrl->Refresh();
   }
   dlg->Destroy();
```

}

wxPrintDialog

TODO.

Constructor

Methods

Example

wxDirDialog

wxDirDialog用来选择一个目录

构造函数

```
#include <wx/dirdlg.h>

wxDirDialog(wxWindow* parent,
            const wxString& message= ″Choose a directory″,
            const wxString& defaultPath= ″″,
            long style= 0,
            const wxPoint& pos= wxDefaultPosition);
```

- *parent* 父窗口指针
- *message* 标题
- *defaultPath* 默认路径
- *style* 不使用
- *pos* 不使用

方法

```
public void  SetMessage(const wxString& message);
public wxString GetMessage();
```

得到与设置对话框标题

```
void  SetPath(const wxString& path);
public wxString GetPath();
```

得到与设置当前路径

```
public int ShowModal();
```

显示对话框，当用户按下 OK 时返回 wxID_OK。其它情况返回 wxID_CANCEL。

例

例 4.6 显示了设置新工作目录的菜单动作的实现。WxGetCwd用来用户得到当前的工作目录。当用户选择一个目录时你可以使用GetPath方法来得到目录名。
WxSetWorkingDirectory是用来设置新的工作目录。

例 **4.6. wxDirDialog** 的使用。

```
void TextFrame::OnMenuOptionDirectory(wxCommandEvent& event)
{
   wxDirDialog *dlg = new wxDirDialog(this, "Select a new working directory",
wxGetCwd());
   if ( dlg->ShowModal() == wxID_OK )
   {
      wxSetWorkingDirectory(dlg->GetPath());
   }
   dlg->Destroy();
}
```

wxDirSelector

wxDirSelector 弹出一个目录选择框。

```
wxString wxDirSelector(const wxString& message,
                       const wxString& defaultPath= "",
                       long style= 0,
                       const wxPoint& pos= wxDefaultPosition,
                       wxWindow* parent= NULL);
```

- *message* 标题
- *defaultPath* 默认路径i
- *style* 对话框的风格，参见wxDirDialogis
- *pos* 对话框位置。
- *Parent* *父窗口。*

例

例 4.7 要求用户选择一个文件夹。

例 **4.7. Using wxDirSelector** 的使用

```
wxString selection = wxDirSelector("Select a folder");
if ( ! selection.empty() )
{
   // The user selected a folder.
}
```

wxTextEntryDialog

wxTextEntryDialog 请求用户输入一行文本的对话框。

构造函数

```
#include <wx/textdlg.h>

wxTextEntryDialog(wxWindow* parent,
                  const wxString& message,
                  const wxString& caption= "Please enter text",
                  const wxString& defaultValue= "",
                  long style= wxOK | wxCANCEL | wxCENTRE,
                  const wxPoint& pos= wxDefaultPosition);
```

- *parent* 父窗口。
- *message* 对话框中显示的信息。
- DefaultValue 默认文本。
- *caption* 标题
- *style* 对话框的风格。可以使用wxTextCtrl的风格。

  Table 表 **4.2. wxTextEntryDialog** 的风格 **styles**

wxCANCEL	显示 cancel 按钮
wxCENTRE	中央对话框
wxOK	显示 OK 按钮

- *pos* 对话框的位置

方法

```
void  SetValue(const wxString& val);
public wxString GetValue();
```

得到与设置文本区域的值。Get/Set the value of the text field.

```
public int ShowModal();
```

显示对话框，当用户按下 OK 时返回 wxID_OK。其它情况返回 wxID_CANCEL。

例

例 4.8 显示一个对话框提示用户输入一个密码。

例 **4.8. wxTextEntryDialog** 的使用

```
wxTextEntryDialog *dlg = new wxTextEntryDialog(this, "Enter your password",
"Enter your password", "", wxOK | wxCANCEL | wxCENTRE |
wxTE_PASSWORD);
```

```
if ( dlg->ShowModal() == wxID_OK )
{
   // Check the password
}
else
{
   // Stop the application
}
dlg->Destroy();
```

wxGetTextFromUser

wxGetTextFromUser 弹出一个允许用户输入一些文本的对话框。

```
wxString wxGetTextFromUser(const wxString& message,
                           const wxString& caption= "Input text",
                           const wxString& defaultValue= "",
                           wxWindow* parent= NULL,
                           int x= -1,
                           int y= -1,
                           bool centre= true);
```

- *message* 对话框中显示的信息。信息中可以包含换行符
- *caption* 标题
- *defaultValue* 默认文本。
- *parent* 父窗口。
- *x* 对话框的X轴位置
- *y* 主轴位置。
- *centre*: 当为真时对话框位于中央，其它情况为左对齐。

例

例 4.9 要求用户输入一些文本。a

例 **4.9. wxGetTextFromUser** 的使用

```
wxString text = wxGetTextFromUser("Please, enter some text");
```

wxGetPasswordFromUser

wxGetPasswordFromUser 弹出一个输入密码的对话框。

```
wxString wxGetPasswordFromUser(const wxString& message,
                               const wxString& caption= "Input text",
                               const wxString& defaultValue= "",
                               wxWindow* parent= NULL);
```

- *message* 对话框中显示的信息。信息中可以包含换行符
- *caption* 标题
- *defaultValue* 默认文本
- *parent* 父窗口

例

Example例 4.10 要求用户输入密码

例 **4.10. wxGetPasswordFromUser** 的使用

```
wxString pwd = wxGetPasswordFromUser("Please, enter your password");
```

wxMessageDialog

wxMessageDialog 可以用来显示一行或多行信息，可以包含 OK、Yes、No 或者 Cancel 按钮。

Constructor 构造函数

```
#include <wx/msgdlg.h>

wxMessageDialog(wxWindow* parent,
                const wxString& message,
                const wxString& caption= "Message box",
                long style= wxOK | wxCANCEL | wxCENTRE,
                const wxPoint& pos= wxDefaultPosition);
```

- *parent父窗口。*
- *message* 对话框中显示的信息，可以是多行文本。
- *caption* 标题。
- *style* 对话框的风格。

表 **4.3. wxMessageDialog** 的风格。 **styles**

wxOK	显示 OK button.
wxCANCEL	显示 Cancel button.
wxYES_NO	显示 Yes 与 No 按钮.
wxYES_DEFAULT	与 wxYES_NO 共同使用，可以让 Yes 按钮作为默认按钮。这是默认的行为。
wxNO_DEFAULT	Used with 同 wxYES_NO 共同使用，让 No 按钮为默认。
wxCENTRE	让信息位于中央，不是窗口位于中央。
wxICON_EXCLAMATION	显示一个惊叹号图标。
wxICON_HAND	显示一个错误图标。

wxICON_ERROR	与上相同。
wxICON_QUESTION	显示问号图标。
wxICON_INFORMATION	显示信息(i)图标。

- *pos* 对话框的位置。

Methods 方法

```
public int ShowModal();
```

显示对话框，当用户按下 OK 时返回 wxID_OK。其它情况返回 wxID_CANCEL。

例

Example 3.4 显示 wxMessageDialog 询问用户是否真的退出程序。

wxMessageBox

wxMessageBox 显示一个信息框。返回值为 *wxYES*, *wxNO*, *wxCANCEL*, *wxOK* 中的一个。

```
wxMessageBox(const wxString& message,
             const wxString& caption= "Message box",
             int style= wxOK | wxCENTRE,
             wxWindow* parent,
             int x= -1,
             int y= -1);
```

- *message* 显示的信息。
- *caption* 标题
- *style* 风格。参见wxMessageDialog
- *parent* 父窗口。
- *x* x轴位置。
- *y* y轴位置。

Note 注意

在windows上本地的MessageBox函数是不使用*wxCENTRE*风格的。*而在通用函数中是使用的。这是因为本地函数不能将文本居中。当通用函数使用时不显示符号。*

wxSingleChoiceDialog

Constructor 构造函数

```
#include <wx/choicdlg.h>
```

```
wxSingleChoiceDialog(wxWindow* parent,
                     const wxString& message,
                     const wxString& caption,
                     int n,
                     const wxString* choices,
                     char ** clientData= (char**) NULL,
                     long style= wxCHOICEDLG_STYLE,
                     const wxPoint& pos= wxDefaultPosition);
```

- *parent* 父窗口。
- *message* 对话框中显示的信息。
- *caption* 标题。
- *n* 条目编号。
- *choices* 是一个字符串数组。
- *clientData*
- *style* 对话框风格。

表 **4.4. wxSingleChoiceDialog** 风格

wxOK	显示 OK button.
wxCANCEL	显示 Cancel button.
wxCENTRE	在非 Windows 上将信息居中.
wxCHOICEDLG_STYLE	wxOK | wxCANCEL | wxCENTRE

- *pos* 对话框的位置。

Methods 方法

```
public void SetSelection(int sel);
```

设置已选项

```
public int GetSelection();
```

返回已选项的索。

```
public wxString GetStringSelection();
```

返回已选项。

```
public int ShowModal();
```

显示对话框，当用户按下 OK 时返回 wxID_OK。其它情况返回 wxID_CANCEL。

例

例子中显示一个对话框让用户选择自己所在的国家。

例 **4.11. wxSingleChoiceDialog** 的使用。

```
wxString countries[] = { "Belgium", "United Kingdom", "U.S.A.", "France" };
wxSingleChoiceDialog *dlg = new wxSingleChoiceDialog(NULL, "Where do you
live?",
                                                     "Select a country",
                                                     4, countries);
if ( dlg->ShowModal() == wxID_OK )
{
      wxMessageBox("You live in " + dlg->GetStringSelection());
}
dlg->Destroy();
```

wxGetSingleChoice

wxGetSingleChoice 显示 wxSingleChoiceDialog并返回选择的字符串。

```
wxString wxGetSingleChoice(const wxString& message,
                           const wxString& caption,
                           const wxArrayString& choices,
                           wxWindow * parent= (wxWindow *) NULL,
                           int x= -1,
                           int y= -1,
                           bool centre= TRUE,
                           int width= wxCHOICE_WIDTH,
                           int height= wxCHOICE_HEIGHT);

wxString wxGetSingleChoice(const wxString& message,
                           const wxString& caption,
                           int n,
                           const wxString * choices,
                           wxWindow * parent= (wxWindow *) NULL,
                           int x= -1,
                           int y= -1,
                           bool centre= TRUE,
                           int width= wxCHOICE_WIDTH,
                           int height= wxCHOICE_HEIGHT);
```

wxGetSingleChoiceIndex

wxGetSingleChoiceIndex 显示 wxSingleChoiceDialog 并返回选择的索引。当没有任何选项被选择时返回-1。

```
int wxGetSingleChoiceIndex(const wxString& message,
                           const wxString& caption,
                           const wxArrayString& choices,
```

```
                            wxWindow * parent= (wxWindow *) NULL,
                            int x= -1,
                            int y= -1,
                            bool centre= TRUE,
                            int width= wxCHOICE_WIDTH,
                            int height= wxCHOICE_HEIGHT);

int wxGetSingleChoiceIndex(const wxString& message,
                            const wxString& caption,
                            int n,
                            const wxString * choices,
                            wxWindow * parent= (wxWindow *) NULL,
                            int x= -1,
                            int y= -1,
                            bool centre= TRUE,
                            int width= wxCHOICE_WIDTH,
                            int height= wxCHOICE_HEIGHT);
```

wxGetSingleChoiceData

wxGetSingleChoiceData 显示wxSingleChoiceDialog 并返回与所选项关联的数据。

```
void* wxGetSingleChoiceData(const wxString& message,
                             const wxString& caption,
                             const wxArrayString& choices,
                             void ** client_data,
                             wxWindow * parent= (wxWindow *) NULL,
                             int x= -1,
                             int y= -1,
                             bool centre= TRUE,
                             int width= wxCHOICE_WIDTH,
                             int height= wxCHOICE_HEIGHT);

void* wxGetSingleChoiceData(const wxString& message,
                             const wxString& caption,
                             int n,
                             const wxString * choices,
                             void ** client_data,
                             wxWindow * parent= (wxWindow *) NULL,
                             int x= -1,
                             int y= -1,
                             bool centre= TRUE,
                             int width= wxCHOICE_WIDTH,
                             int height= wxCHOICE_HEIGHT);
```

注意

wxCHOICE_WIDTH 的定义是 200， *wxCHOICE_HEIGHT* 的定义是 150.

第五章 对话框

目录 **Table of Contents**

对话框是一个包含标题栏的窗口。这个窗口可以移动，可以包含控件与其它窗口。对话框可以是模态或非模态的。模态对话框在显示时程序里的其它窗口被锁，一直到对话框退出才能解锁。

wxDialog

WxDialog是所有对话框的基类。你可以自定义一个继承自wxDialog的对话框。

构造函数 Constructor

```
wxDialog(wxWindow* parent,
         wxWindowId id,
         const wxString& title,
         const wxPoint& pos= wxDefaultPosition,
         const wxSize& size= wxDefaultSize,
         long style= wxDEFAULT_DIALOG_STYLE,
         const wxString& name= "dialogBox");
```

- *parent* 父窗口的指针，当对话框没有父窗口时使用NULL
- *id* 独立的窗口ID。如果不需要它使用-1。
- *title* 显示在标题栏里的标题。
- *pos* 对话框的位置。使用wxDefaultPosition时为默认位置。
- *size* 窗口的大小。默认大小为wxDefaultSize。
- *style* 对话框的风格。is the style of the dialog.

表 **5.1. wxDialog** 的风格。

wxDEFAULT_DIALOG_STYLE	wxSYSTEM_MENU 与 wxCAPTION 的组合。在 Unix 系统上 wxSYSTEM_MENU 是不使用的。
wxCAPTION	显示标题。
wxMINIMIZE	与 wxICONIZE 相同
wxMINIMIZE_BOX	可以最小化.
wxMAXIMIZE	以最大化显?尽?(只适用于 Windows)The
wxMAXIMIZE_BOX	可以最大化。
wxSTAY_ON_TOP	对话框将出现在其它窗口之上。(仅适用于 Windows)
wxSYSTEM_MENU	显示一个系统菜单。
wxSIMPLE_BORDER	无边框(仅用于 GTK 与 Windows)
wxRESIZE_BORDER	可拖拽缩放的边框。(仅用于 Unix)

- *name* 窗口的名字。这个参数与一个选项相关联。这允许用户为一个单独的窗口设置 Motif资源值。

对话框编程

当你要在程序中使用对话框时，wxWindows 需要一个资源文件。在你不需要额外的图标时，它的使用就象你将在例 5.1 中看到的那样简单。

例 **5.1.** 一个简单的资源文件。

```
#include "wx/msw/wx.rc"
```

例 5.2 展示了在早先开发的简单文本编辑器中显示关于信息对话框的定义。例 5.3 是它的实现。

例 **5.2. AboutDialog.h –** 关于对话框的定义。

```
#ifndef _ABOUTDIALOG_H
#define _ABOUTDIALOG_H
class AboutDialog : public wxDialog
{
public:
   /**
    * 构造函数
    */
   AboutDialog(wxWindow *parent);
   /**
    * 析构函数
    */
   virtual ~AboutDialog() { }
   /**
    * 设置对话框中的文本。
```

```
     */
    void SetText(const wxString& text);
private:
    wxStaticText *m_pInfoText;
    wxButton *m_pOkButton;
};
#endif
```

例 **5.3. AboutDialog.cpp -** 关于对话框的实现

```
// For compilers that supports precompilation , includes "wx/wx.h"
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
       #include "wx/wx.h"
#endif
#include "AboutDialog.h"
AboutDialog:: AboutDialog(wxWindow *parent)
       : wxDialog(parent, -1, "About Simple Text Editor", wxDefaultPosition,
                          wxSize(200, 200), wxDEFAULT_DIALOG_STYLE)
{
    m_pInfoText = new wxStaticText(this, -1, "", wxPoint(5, 5),
                                                  wxSize(100, 100), wxALIGN_CENTRE);
    m_pOkButton = new wxButton(this, wxID_OK, "Ok", wxPoint(5, 40));
}
void AboutDialog::SetText(const wxString& text)
{
    m_pInfoText->SetLabel(text);
}
```

在例子中没有事件处理的代码，它不是必须的，因为所有的事都是通过 wxWindows 默认的事件处理机制完成的。当用户点击由 wxID_OK 或 wxID_CANCEL 标识出的按钮时对话框是自动退出的。例 5.4 是当用户点击关于菜单时用于显示对话框的代码。

例 **5.4.**显示对话框。

```
void TextFrame::OnMenuInfoAbout(wxCommandEvent &event)
{
    AboutDialog *dlg = new AboutDialog(this);
    dlg->SetText("(c) 2001 S.A.W. Franky Braem\nSimple Text Editor\n");
    dlg->ShowModal();
}
```

图 **5.1.** 关于对话框。

重点

如果不使用例子中的 OK 或 Cancel 按钮代码时，默认的事件处理是不使用的。这时要撤消对话框必须在 EVT_CLOSE 事件中调用 Destroy。当对话框没被撤消时程序仍然运行。

Sizers

上个例子中创建的对话框布局可能不是用户所希望的，让文本与按钮都位于中央可能会更好一些。另外如果文本的大小改变会出现什么？如何让文本下的按钮位于中央呢？

wxWindows 提供了两种方法来解决布局的问题。在wxWindows的第一个版本中使用constraints，从第二版开始使用sizers。虽然constraints在当前版本中仍然可以使用，但sizers是wxWindows进行控件布局的首选。WxSizer是所有sizers的抽象基类。WxSizer不能直接使用，必须用一个派生类来代替，通常的派生类有： wxBoxSizer, wxStaticBoxSizer, wxNotebookSizer, wxGridSizer 与 wxFlexGridSizer.

Sizers 使用的布局算法可以与 Java、GTK.....之中的任何一个来比较。基本的原理是每个 sizer 的项目报告它所请求的最小大小，父窗口的大小按需要进行改变。这表示父窗口的大小不能由开发者来指定，当窗口包含一个 sizer 时会要求 sizer 返回一个建议大小，sizer 则去询问它所包含的子构件的建议大小，这些构件可以是一个窗口，其它 sizer 或者空白空间。父窗口根据回答自动计算大小。

这让 sizer 十分适合写多平台的程序。考虑下面的问题：在 Unix, Motif, Windows 中的字体大小并不完全相同。你应该如何设置对话框的原始大小呢？解决方法是让 sizer 来为你处理这个问题。

注意

当窗口有一个 sizer 时布局算法并不是自动调用的。要在窗口显示之前调用包含 sizer 窗口上的 Layout 方法。在窗口缩放时也必须同时调用 Layout。这可以通过调用包含 tue 参数的 SetAutoLayout 来省略。

Sizers 和 窗口

```
void Add(wxWindow * window,
         int option= 0,
```

```
        int flag= 0,
        int border= 0,
        wxObject* userData= NULL);
```

Add 方法用来将一个窗口添加到sizer条目的后面。

```
void Insert(int before,
            wxWindow * window,
            int option= 0,
            int flag= 0,
            int border= 0,
            wxObject* userData= NULL);
```

Insert 用来在berfore之前插入一个窗口。

```
void Prepend(wxWindow * window,
             int option= 0,
             int flag= 0,
             int border= 0,
             wxObject* userData= NULL);
```

Prepend 用来设置一个窗口成为条目列表的第一项。

```
void Remove(wxWindow * window);
```

Remove 用于从条目列表中移除一个窗口。

- *option* 与wxBoxSizer一直使用. 0 表示条目在sizer主向(创建sizer时指定的方向)内的大小不许改变，1 表示可以在sizer主向内缩放。
- *flag* 用来设置标志可以用“或”操作符来组合使用。

表 **5.2. wxSizer** 的边框标志

wxTOP	边框在顶部
wxBOTTOM	边框在底部
wxLEFT	边框在左
wxRIGHT	边框在右
wxALL	边框在所有面上。

表 **5.3. wxSizer** 的行为标志

wxGROW or wxEXPAND	条目可以缩放
wxSHAPED	条目可以成比例缩放
wxALIGN_CENTER or wxALIGN_CENTRE	条目在中央

wxALIGN_LEFT	条目左对齐
wxALIGN_TOP	条目上对齐
wxALIGN_RIGHT	条目右对齐
wxALIGN_CENTER_HORIZONTAL	条目在水平向中央
wxALIGN_CENTER_VERTICAL	条目在垂直向的中央。

- *border* 定义边框的宽度。
- *userData* 允许一个额外对象附加到sizer条目上。可以通过派生类使用。

Sizers 与其它 sizer

Sizers 可以嵌套

```
void Add(wxSizer * sizer,
         int option= 0,
         int flag= 0,
         int border= 0,
         wxObject* userData= NULL);
```

Add 方法用来将一个sizer添加到sizer条目的后面。

```
void Insert(int before,
            wxSizer * sizer,
            int option= 0,
            int flag= 0,
            int border= 0,
            wxObject* userData= NULL);
```

Insert 在条目 *before之前插入一个sizer*。

```
void Prepend(wxSizer * sizer,
             int option= 0,
             int flag= 0,
             int border= 0,
             wxObject* userData= NULL);
```

Prepend 用来设置一个sizer成为条目列表的第一项

```
void Remove(wxSizer * sizer);
```

Remove 用于从条目列表中移除一个sizer。

更多参数信息参见“Sizers和 windows”

Sizers 和 spacer

Spacer 用来在控件之间插入空白。当 spacer 在两个条目之间时允许扩展。结果将使用第一个条目保持左对齐其它条目保持右对齐。

```
void Add(int width,
         int height,
         int option= 0,
         int flag= 0,
         int border= 0,
         wxObject* userData= NULL);
```

Add方法用来将一个spacer添加到sizer条目的后面。

```
void Insert(int before,
            int width,
            int height,
            int option= 0,
            int flag= 0,
            int border= 0,
            wxObject* userData= NULL);
```

Insert 将一个spacer插入before条目之前。

```
void Prepend(int width,
             int height,
             int option= 0,
             int flag= 0,
             int border= 0,
             wxObject* userData= NULL);
```

Prepend设置一个spacer 为条目列表的第一个条目。

更多参数细节参见 “Sizers 与 windows”

wxBoxSizer

WxBoxSizer 用来根据不同的标志将控件排成一行或一列。它可以水平或垂直伸展。

Constructor 构造函数

```
wxBoxSizer(int orient);
```

- *orient* 方向。

  表 **5.4. wxBoxSizer** 的方向

wxHORIZONTAL 水平

wxVERTICAL 垂直。

在下面的例子中，创建了一个充满对话框的 sizer。它的主向是垂直。所有其它 sizer 都是这个 sizer 的子控件，全部是水平向的。它们可以在水平方向扩展，但垂直方向则不能。第二个 sizer 用来对 OK 按钮布局，这将可以让按钮与对话框边框间保持固定的边界。在添加按钮时指定边框的大小为 20。

例 **5.5. wxBoxSizer** 的使用

```
AboutDialog::AboutDialog(wxWindow *parent)
      : wxDialog(parent, -1, "About Simple Text Editor", wxDefaultPosition,
                        wxSize(200, 200), wxDEFAULT_DIALOG_STYLE |
wxRESIZE_BORDER)
{
   wxBoxSizer *dialogSizer = new wxBoxSizer(wxVERTICAL);
   wxBoxSizer *textSizer = new wxBoxSizer(wxHORIZONTAL);
   m_pInfoText = new wxStaticText(this, -1, "", wxDefaultPosition,
                                              wxDefaultSize, wxALIGN_CENTER);
   // The text can grow horizontally.
   textSizer->Add(m_pInfoText, 1, 0);
   wxBoxSizer *buttonSizer = new wxBoxSizer(wxHORIZONTAL);
   m_pOkButton = new wxButton(this, wxID_OK, "Ok", wxPoint(-1, -1));
   // The button can grow horizontally, not vertically.
   // The button has a left and right border with size 20
   buttonSizer->Add(m_pOkButton, 1, wxLEFT | wxRIGHT, 20);
   // The sizers can't change their vertical size. They can only grow horizontally.
   dialogSizer->Add(textSizer, 0, wxGROW);
   dialogSizer->Add(buttonSizer, 0, wxGROW);
   SetSizer(dialogSizer);
   SetAutoLayout(TRUE);
   Layout();
}
```

图 **5.2.** 使用 **wxBoxSizer** 的关于对话框。

wxGridSizer

网格 sizer 是一个拥有相同大小单元的两维表格布局控件。每个单元格的宽与高就是是控件的最大宽度与最大高度。当一个网格 sizer 创建时，它需要知道需要多少行与列。如果这些参数中有一个为 0,则这个值由网格 sizer 计算得出。同样也必须指定行与列之间的间隔。

Constructor 构造函数

```
wxGridSizer(int rows,
            int cols,
            int vgap,
            int hgap);
wxGridSizer(int cols,
            int vgap= 0,
            int hgap= 0);
```

- *rows* 指定行数
- *cols* 指定列数
- *vgap* 指定列之间的间隔。
- *hgap* 指定行之间的间隔。

下面的例子展示了如何用 wxGridSizer 来实现对关于对话框中控件的布局。例子将建立丙行与一列，行与列之间的间隔是 10。

例 **5.6.wxGridSizer** 的使用

```
AboutDialog::AboutDialog(wxWindow *parent)
      : wxDialog(parent, -1, "About Simple Text Editor", wxDefaultPosition,
                        wxSize(200, 200), wxDEFAULT_DIALOG_STYLE |
wxRESIZE_BORDER)
{
   wxGridSizer *dialogSizer = new wxGridSizer(2, 1, 10, 10);
   m_pInfoText = new wxStaticText(this, -1, "", wxDefaultPosition,
                                              wxDefaultSize, wxALIGN_CENTER);
   dialogSizer->Add(m_pInfoText, 0, wxALIGN_CENTER);
   m_pOkButton = new wxButton(this, wxID_OK, "Ok", wxPoint(-1, -1));
   dialogSizer->Add(m_pOkButton, 0, wxALIGN_CENTER);
   SetSizer(dialogSizer);
   SetAutoLayout(TRUE);
   Layout();
}
```

wxFlexGridSizer

曲线网格 sizer 是一个二维表格布局控件，它所有在同一行中的单元格都有相同的宽度并且同一列中的单元格都有相同的高度。

如果你需要在对话框中的控件上加入标签，网格 sizer 是非常有用的。图 5.3 展示了一个正确使用 wxFlexGridSizer 的例子。在第一列中包含标签，最宽的那个标签决定了第一列的列宽。这可以保证控件是垂直对齐的。创建这种对话框的代码显示在例 5.7 中。创建了两行与两列，当添加控件到 sizer 时，它们是从左到右，从上到下放置的。

构造函数

```
wxFlexGridSizer(int rows,
                int cols,
                int vgap,
                int hgap);
wxFlexGridSizer(int cols,
                int vgap= 0,
                int hgap= 0);
```

参数说明参见 “wxGridSizer”

图 **5.3. wxFlexGridSizer** 的使用

例 **5.7. wxFlexGridSizer** 的使用

```
wxFlexGridSizer *dialogSizer = new wxFlexGridSizer(2, 2, 10, 10);
dialogSizer->Add(new wxStaticText(this, -1, "Username"), 0,
wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxTextCtrl(this, ID_USER_NAME), 0,
wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxStaticText(this, -1, "Password"), 0,
wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxTextCtrl(this, ID_PASSWORD), 0,
wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxStaticText(this, -1, "Retype Password"), 0,
wxALIGN_CENTRE_VERTICAL);
dialogSizer->Add(new wxTextCtrl(this, ID_PASSWORD2), 0,
wxALIGN_CENTRE_VERTICAL);
SetSizer(dialogSizer);
SetAutoLayout(TRUE);
Layout();
```

wxStaticBoxSizer

wxStaticBoxSizer 继承自wxBoxSizer。它的工作与box sizer相同。但它在sizer周围画出一个静态框。

构造函数

```
wxStaticBoxSizer(wxStaticBox* box,
                 int orient);
```

- *box* 是一个wxStaticBox指针。静态框不是通过sizer来完成的，它必须通过开发者来完成。

- *orient* box sizer的方向，参见“wxBoxSizer”

wxNotebookSizer

wxNotebookSizer 是专门在wxNotebook上工作的sizer。

Sizer 的嵌套

Sizer 是可以嵌套的。下一个例子展示了一个组合了 wxBoxSizer 与 wxFlexGridSizer 的对话框。图 5.4 显示了运行结果。WxBoxSizer 用来在对话框中其它控件下方的中央放置一个 OK 按钮。

例 **5.8. sizer** 的嵌套

```
wxBoxSizer *dialogSizer = new wxBoxSizer(wxVERTICAL);
wxFlexGridSizer *controlSizer = new wxFlexGridSizer(2, 2, 10, 10);
controlSizer->Add(new wxStaticText(this, -1, "Username"), 0,
wxALIGN_CENTRE_VERTICAL);
controlSizer->Add(new wxTextCtrl(this, -1), 0, wxALIGN_CENTRE_VERTICAL);
controlSizer->Add(new wxStaticText(this, -1, "Password"), 0,
wxALIGN_CENTRE_VERTICAL);
controlSizer->Add(new wxTextCtrl(this, -1), 0, wxALIGN_CENTRE_VERTICAL);
controlSizer->Add(new wxStaticText(this, -1, "Retype Password"), 0,
wxALIGN_CENTRE_VERTICAL);
controlSizer->Add(new wxTextCtrl(this, -1), 0, wxALIGN_CENTRE_VERTICAL);
wxBoxSizer *buttonSizer = new wxBoxSizer(wxVERTICAL);
buttonSizer->Add(new wxButton(this, wxID_OK, "Ok"));
dialogSizer->Add(controlSizer);
dialogSizer->Add(20, 20);
dialogSizer->Add(buttonSizer, 0, wxALIGN_CENTRE);
SetSizer(dialogSizer);
SetAutoLayout(TRUE);
Layout();
```

图 **5.4.Sizer** 的嵌套


Enter a new password
Username
Password
Retype Password
Ok