wxWidgets跨平台GUI开发

Jeson

[jetstorm@163.com](mailto:jetstorm@163.com)

2015.07.30

CC BY-SA 3.0

<http://creativecommons.org/licenses/by-sa/3.0/>

本作品采用知识共享署名-相同方式共享 3.0 中国大陆许可协议进行许可。

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

<http://creativecommons.org/licenses/by-sa/3.0/>

This is a human-readable summary of (and not a substitute for) the [**license**](http://creativecommons.org/licenses/by-sa/3.0/legalcode).

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material

for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

* **Attribution** — You must give [**appropriate credit**](http://creativecommons.org/licenses/by-sa/3.0/), provide a link to the license, and[**indicate if changes were made**](http://creativecommons.org/licenses/by-sa/3.0/). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
* **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [**same license**](http://creativecommons.org/licenses/by-sa/3.0/) as the original.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

# 环境

* 配置日期：2014.07

wxWidgets for MSW 3.0.1 / 2.8.12 （开源免费，类LGPL）

wxFormBuilder 3.05 (Beta) （开源免费GPL）

全部采用默认路径安装

* 建议

采用wxWidgets+wxFormBuilder的组合，开发工具windows下可以考虑VS，注重跨平台可以考虑CodeLite

* 其他工具（可不安装）

Codelite 6.0 w/ Mingw 4.8.1&wxCrafter 1.5: 跨平台IDE

wxCrafter免费版（功能受限））：codelite.org提供的RAD工具

DialogBlocks 5.04 （30天试用版）：RAD工具

MSYS 1.0.11 （开源免费）：windows下的小型GNU环境，用于模拟unix

* 关于RAD工具：

Rapid Application Develop: 用于辅助快速程序开发，例如可实现所见即所得的界面设计，自动生成框架代码等。

# wxWidgets

## 编译

%WXWIN%\docs\msw找到install.txt,按照说明编译。

默认生成的lib/dll库文件名Release版为 libwxXXX30u，Debug版为libwxXXX30d。

静态链接生成库路径为xxx\_lib，动态链接生成库路径为xxx\_dll。

注意使用makefile时，参数大小写敏感。如BUILD=debug

### For CodeLite (GCC)

参考install.txt

Building wxWidgets

Cygwin/MinGW Compilation

\* Using plain makefiles:

环境需求：已安装GCC, 并且环境变量中已加入Mingw路径

命令行下输入 path = %path%;[mingwpath]\bin

或者控制面板->系统->高级->环境变量->系统变量，双击path，在变量值最后添加mingw的bin路径，分号隔开。

命令行：mingw32-make -f makefile.gcc

参数:

BUILD=debug/realse

SHARED=0/1

默认为debug,静态链接,输出库文件目录lib\gcc\_lib.如果采用动态链接(shared=0),则输出目录为lib\gcc\_dll.

编译大约用半个小时。可能需要提前加入Mingw的环境路径，否则找不到mingw32-make 默认为debug编译，非共享

MS命令行：

mingw32-make -f makefile.gcc BUILD=debug SHARED=1

mingw32-make -f makefile.gcc BUILD=debug SHARED=0

mingw32-make -f makefile.gcc BUILD=release SHARED=1

mingw32-make -f makefile.gcc BUILD=release SHARED=0

### For MSVC

配置Debug/Release生成的lib文件位于lib\vc\_lib

DLL Debug/ DLL Release生成的lib和dll文件位于lib\vc\_dll

* **命令行方式：**

参考

%WXWIN%\docs\msw\install.txt

Building wxWidgets

Microsoft Visual C++ Compilation

\* From command line using the provided makefiles:

nmake -f makefile.vc BUILD=debug SHARED=1

nmake -f makefile.vc BUILD=debug SHARED=0

nmake -f makefile.vc BUILD=release SHARED=1

nmake -f makefile.vc BUILD=release SHARED=0

* **IDE方式：**

X86：支持VC6.0以上版本。

X64：支持VS2010以上版本。

%WXWIN%\build\msw打开相应dsw或sln项目文件，编译。

* **IDE方式编译提示无法打开xxx.lib**

wx\_vc9.sln（VS2008）项目依赖设置不正确，导致首次编译很多项目会提示链接不到wxtiffd.lib/ wxtiff.lib。解决方法为设置依赖项从而调整项目生成顺序，或者多次重新编译。

解决方案右键，生成顺序可以看到项目生成顺序。一般第一个是wxscintilla，第二个是gl或者stc。点依赖项，项目选择第二个项目名，依赖于勾选adv base core wxtiff，这样编译时会首先生成上述四个项目。然后解决方案右键，批生成，全选，点生成即可。23个项目4种配置总计92个项

### 编译耗时及空间占用

版本：wxWidgets for MSW (wxMSW) 3.0.3

系统：Windows XP SP3, T8300( 2.4G\*2), 2G DDR2 800 \*2, 1T 7200RPM

编译参数(MSVC 2008)：

SHARED=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RELEASE | lib大小 | pdb大小 | 总计(M) | 耗时 |
| debug | 166 | 41 | 207 | 6 min |
| release | 336 | 80 | 416 | 7 min |

SHARED=1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RELEASE | exp | ilk | lib | pdb | dll | 总计(M) | 耗时 |
| debug | 8 | 96 | 29 | 102 | 21 | 255 | 7 min |
| release | 16 | 96 | 56 | 198 | 34 | 400 | 9 min |

版本：wxWidgets for MSW (wxMSW) 3.0.2

系统：Windows 7, I7-4790(3.6G\*4\*HT), 4G DDR3 1600 \*2, Intel 530 120G SSD

编译参数(TDM-GCC 32 4.9.2)

SHARED=0 lib 302M

|  |  |
| --- | --- |
| RELEASE | 耗时 |
| debug | 12min |
| release | 14min |

SHARED=1 dll 316M

|  |  |  |
| --- | --- | --- |
| RELEASE | 仅dll大小 | 耗时 |
| debug | 205M | 12min |
| release | 52M | 14min |

## 参考资料

%WXWIN%\docs

文档及帮助：%WXWIN%\docs\htmlhelp\wx.chm

安装说明：%WXWIN%\docs\msw\install.txt (for MSW platform)

MSVC guide:

<http://wiki.wxwidgets.org/Microsoft_Visual_C%2B%2B_Guide#Visual_Studio_C.2B.2B_2008_Professional>

## Special notes for TDM-GCC 4.9 and wxWidgets 2.8.12

### compile wxWidgets in C++11 mode

#### 关键信息：

error: #error This file requires compiler and library support for the ISO C++ 2011 standard. This support is currently experimental, and must be enabled with the -std=c++11 or -std=gnu++11 compiler options.

error: 'is\_enum' in namespace 'std' does not name a template type

error: 'is\_enum' was not declared in this scope

#### 详细错误信息

g++ -c -o gcc\_mswudll\basedll\_any.o -O2 -mthreads -DHAVE\_W32API\_H -D\_\_WXMSW\_\_

-DNDEBUG -D\_UNICODE -I..\..\lib\gcc\_dll\mswu -I..\..\include -W -Wall -DW

XBUILDING -I..\..\src\tiff\libtiff -I..\..\src\jpeg -I..\..\src\png -I..\..\src\

zlib -I..\..\src\regex -I..\..\src\expat\lib -DwxUSE\_GUI=0 -DWXMAKINGDLL\_BASE -D

wxUSE\_BASE=1 -Wno-ctor-dtor-privacy -MTgcc\_mswudll\basedll\_any.o -MFgcc\_mswu

dll\basedll\_any.o.d -MD -MP ../../src/common/any.cpp

In file included from C:/TDM-GCC-64-4.9.2/lib/gcc/x86\_64-w64-mingw32/4.9.2/inclu

de/c++/type\_traits:35:0,

from ..\..\include/wx/strvararg.h:25,

from ..\..\include/wx/string.h:46,

from ..\..\include/wx/any.h:19,

from ../../src/common/any.cpp:18:

C:/TDM-GCC-64-4.9.2/lib/gcc/x86\_64-w64-mingw32/4.9.2/include/c++/bits/c++0x\_warn

ing.h:32:2: error: #error This file requires compiler and library support for the ISO C++ 2011 standard. This support is currently experimental, and must be enabled with the -std=c++11 or -std=gnu++11 compiler options.

#error This file requires compiler and library support for the \

^

In file included from ..\..\include/wx/string.h:46:0,

from ..\..\include/wx/any.h:19,

from ../../src/common/any.cpp:18:

..\..\include/wx/strvararg.h:350:18: error: 'is\_enum' in namespace 'std' does not name a template type

typedef std::is\_enum<T> is\_enum;

^

..\..\include/wx/strvararg.h:354:54: error: 'is\_enum' was not declared in this scope

enum { value = wxFormatStringSpecifierNonPodType<is\_enum::value>::value };

^

..\..\include/wx/strvararg.h:354:68: error: template argument 1 is invalid

enum { value = wxFormatStringSpecifierNonPodType<is\_enum::value>::value };

^

makefile.gcc:11370: recipe for target 'gcc\_mswudll\basedll\_any.o' failed

mingw32-make: \*\*\* [gcc\_mswudll\basedll\_any.o] Error 1

#### 编译环境

TDM-GCC 4.9.2-3

wxWidgets 3.0.2

#### 原因

代码中使用了C++11的特性，但是由于C++11处于试验阶段，所以编译器需要强制启用该特性。

#### 解决方案

compile wxWidgets in C++11 mode use CXXFLAGS="-std=gnu++11", eg:

mingw32-make -f makefile.gcc BUILD=release SHARED=1 CXXFLAGS="-std=gnu++11"

#### 影响

编译过程会比较慢。单次编译约需40分钟左右。(TDM-GCC 4.9.2-3, release, shared)

#### 参考资料

[wxWidgets-3.0.2]\docs\msw\install.txt

C++11 note: If you want to compile wxWidgets in C++11 mode, you currently have

to use -std=gnu++11 switch as -std=c++11 disables some extensions

that wxWidgets relies on. I.e. please use CXXFLAGS="-std=gnu++11".

### compile wxWidgets with -fpermissive

#### 关键信息：

error: cast from 'void\*' to 'THREAD\_RETVAL {aka unsigned int}' loses precision [-fpermissive]

error: cast from 'wxThread::ExitCode {aka void\*}' to 'DWORD {aka long unsigned int}' loses precision [-fpermissive]

error: cast from 'wxThread::ExitCode {aka void\*}' to 'unsigned int' loses precision [-fpermissive]

#### 详细错误信息

../../src/msw/thread.cpp:525:43: error: cast from 'void\*' to 'THREAD\_RETVAL {aka unsigned int}' loses precision [-fpermissive]

rc = (THREAD\_RETVAL)thread->Entry();

^

../../src/msw/thread.cpp: In member function 'wxThreadError wxThreadInternal::WaitForTerminate(wxCriticalSection&, void\*\*, wxThread\*)':

../../src/msw/thread.cpp:845:21: error: cast from 'wxThread::ExitCode {aka void\*}' to 'DWORD {aka long unsigned int}' loses precision [-fpermissive]

if ( (DWORD)rc != STILL\_ACTIVE )

^

../../src/msw/thread.cpp: In static member function 'static bool wxThread::SetConcurrency(size\_t)':

../../src/msw/thread.cpp:1003:75: warning: format '%u' expects argument of type 'unsigned int', but argument 2 has type 'size\_t {aka long long unsigned int}' [-Wformat=]

wxLogDebug(\_T("bad level %u in wxThread::SetConcurrency()"), level);

^

../../src/msw/thread.cpp: In member function 'void wxThread::Exit(wxThread::ExitCode)':

../../src/msw/thread.cpp:1165:28: error: cast from 'wxThread::ExitCode {aka void\*}' to 'unsigned int' loses precision [-fpermissive]

\_endthreadex((unsigned)status);

^

mmakefile.gcc:8776: recipe for target 'gcc\_mswddll\basedll\_thread.o' failed

mingw32-make: \*\*\* [gcc\_mswddll\basedll\_thread.o] Error 1

#### 编译环境

TDM-GCC 4.9.2-3

wxWidgets 3.0.2

#### 原因

推测是由于MinGW4.9.2使用新的标准，采用了更加严格的错误检查，一些操作从warning变成了error。

可以使用-fpermissive

#### 解决方案

使用-fpermissive参数：

mingw32-make -f makefile.gcc BUILD=debug SHARED=1 CXXFLAGS="-fpermissive"

# Codelite

## 环境需求

wxWidgets compiled with gcc

## 环境配置

Settings -> Environment Variables

在环境变量框中添加

WXWIN=C:\wxWidgets-3.0.1

WXCFG=gcc\_lib\mswud

CXXFLAGS = -g -O0 -Wall $(Preprocessors) -std=c++11

备注：此处的CXXFLAGS会覆盖系统默认的CXXFLAGS（生成的.mk文件中）

## 导入VS项目

可以将VS项目文件导入到CodeLite中。

例如：

将已有VS2008下wxWidgets项目导入到CodeLite中。

VS2008下项目使用wxFormBuilder 3.5.0设计界面, Frame form。

CodeLite版本8.0 for MSW

* CodeLite新建wxFB项目，并删除全部默认生成的文件

Workspace -> New Project, GUI, Executable (wxWidgets + wxFB frame).

* 复制VS2008项目源代码到CodeLite项目中

将VS2008项目源代码（含.h, .cpp, .rc文件）按照原来的目录结构复制到CodeLite项目目录中。如果VS项目含有多个文件夹，复制到CodeLite项目中时其相对路径不要改变，并且所有目录都在CodeLite项目目录中。

* 将源代码及资源文件添加到CodeLite项目中

CodeLite项目管理器（左侧WorkSpace）项目名称，右键，Import File From Directory,选中项目根目录，File Extension to import中添加 ; \*.rc (同时导入rc资源文件)，确认即可。

## 常见错误

### WXWIN未配置

Please use the --prefix flag (as in wx-config --prefix=C:/wxWidgets)

or set the environment variable WXWIN (as in WXWIN=C:/wxWidgets)

to specify where is your installation of wxWidgets.

CodeLite 8.0:

Settings -> Enviroment Variables

WXWIN=C:\wxWidgets-3.0.2

### WXCFG未配置

wx-config Warning: Multiple compiled configurations of wxWidgets have been detected.

Using first detected version by default.

Please use the --wxcfg flag (as in wx-config --wxcfg=gcc\_dll/mswud)

or set the environment variable WXCFG (as in WXCFG=gcc\_dll/mswud)

to specify which configuration exactly you want to use.

### 找不到wxWidgets头文件

* 找不到wx/Setup.h

Settings -> Build Settings, Compilers, Advanced, Include Path

添加$(WXPATH)\lib\xxx\_xxx\xxx。

类似C:\wxWidgets-3.0.2\lib\gcc\_dll\mswud的目录名称，根据编译器（gcc/vc），是否静态链接(lib, dll)，是否debug会有所不同

默认情况下

$(WXPATH)\lib\gcc\_dll\mswu

$(WXPATH)\lib\gcc\_dll\mswud

$(WXPATH)\lib\gcc\_lib\mswu

$(WXPATH)\lib\gcc\_lib\mswud

四个文件夹下的wx\setup.h文件内容完全一致。

* 找不到<wx/xxx.h>：

Settings -> Build Settings, Compilers, Advanced, Include Path

添加$(WXPATH)\include。

如 C:\wxWidgets-3.0.2\include

### VS编译通过，但CodeLite报错：没有匹配的函数调用

main.cpp:226:48: error: no matching function for call to 'MainFrame::xxx(wxCommandEvent)'

src: xxx(wxCommandEvent());

但是在VS2008下编译通过。推测是由于编译器的处理不同导致。

源代码改为

wxCommandEvent evt ;

XXX(evt);

即可通过。

# MSYS

## 下载

MinGW首页上有MSYS的页面，进入MSYS页面下载。SourceFoge

## 安装

* 安装MSYS 1.0.11

默认路径。安装后会执行pi.sh脚本(..\bin\sh.exe pi.sh)并需要给出选择。

Do you wish to continue with the post install? 选Y

Do you have MinGW installed? 选Y

Where is your MinGW installation? 输入MinGW的路径(使用‘\’而非’/’)

最后提示MinGW没有make.exe

按任意键完成。

* 安装MSYS DTK 1.0.1
* 更新msysCORE 1.0.18

下载msysCORE-1.0.18-1-msys-1.0.18-bin.tar.lzma，先用7zip解压得到tar文件，之后解压到%MSYS%\1.0，覆盖相应文件。

# DialogBlocks

## 注册

采用5.03的注册码可以注册，激活过程中可能需要断开网络。或者先安装5.03，注册，然后安装5.04，后者会自动激活。

## Samples示例

安装后会在’我的文档’文件夹中生成’ DialogBlocks Samples’，其内容和安装目录中’Samples’相同，为部分示例。

## 简单流程

### 新建项目

建议每个项目使用单独的目录。DialogBlocks可以自动创建路径。

Mode选择含有C++的项即可。

配置页面单击’标准（Standard）’，勾选对应编译器<compiler>下的相应配置<configuration>

其余默认。

### 设计

注意窗口层次，某些窗口只能在特定窗口下使用，很多空间需要放在Sizer里面：

Managed Windows

MenuBar

ToolBar

Sizer

Buttons ...

### 指定App主窗口:

单击App,属性中Main window选中主窗口ID

### 添加编译项

Settings -> Configurations勾选debug and/or release

### build

提示wxWidgets和当前设置不兼容(not consistent)

需要在wxWidgets-3.0.1\include\wx\msw\setup.h修改

#define wxUSE\_MEDIACTRL 0

#define wxUSE\_GLCANVAS 0

选是（可以考虑选否）

提示setup.h已经修改，是否重新编译wxWidgets,选否。

开始编译。提示wxWidgets Debug Alert:

../../src/msw/toolbar.cpp(801): assert "Assert failure" failed in Realize():

invalid tool button bitmap.

选择cancel（不提示任何错误）。选YES会出现VS的调试窗口。

## 参考资料

<http://www.cppblog.com/zyzx/archive/2009/03/12/76348.html>

# wxFormBuilder

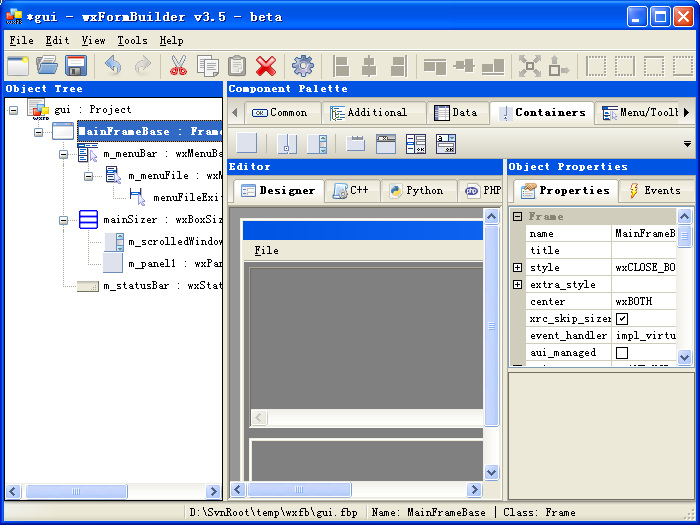
界面布局信息保存在fbp(wxFormBuilder Project)文件中，wxFormBuilder 根据项目文件生成相应的C++、Python、XRC等文件。建议随项目源码保存fbp文件，以便对项目布局进行修改。

## 优点

* 使用GNU General Public License，开源免费
* 布局不合法时会有提示。

例如Frame下直接摆放container(ScrolledWindow)时，会提示错误，并给出建议：Impossible to create the object. Did you forget to add a sizer/parent object or turn on/off an AUI management.

## 界面介绍



* 菜单

File -> Import XRC: 导入XRC资源文件。XRC是基于XML的资源布局文件。

File -> Generate Code: 按照project的file（文件名，如gui）和Frame的name（框架基类名，如MainFrameBase）生成C++代码。可基于该类或其继承类编写程序，无需XRC文件。

Tools -> Generate Inherited Class: 生成继承类。一般使用继承类作为程序框架，大部分程序代码都基于该类。勾选并单击基类列表，Class Details下输入继承类名（如MainFrame）及文件名（如main）。点OK即可生成文件。

* Object Tree
* 对象树，显示布局对象及其树状关系。选中某控件可以修改其属性（Object Properties），在Component Palette中点击相应控件可插入到当前选中的控件或其父控件下（根据控件相互间的关系）。Alt+方向键（或者使用Edit菜单）可以调整控件排列的顺序。
* Component Palette

组件板。显示所有可用的组件，并且按照类别排序

* Editor

编辑器。Designer设计器，可以看到界面的样子。C++，Python等则显示对应语言的相应代码，可以直接将代码复制到相应文件中使用，也可以通过File菜单或工具栏按钮生成相应C++文件。全部只读，由wxFormBuilder根据设计的界面自动生成。

* Object Properties

显示Object Tree或Designer中选中的控件的属性。顶层的sizer是没有flag等属性，只有处于sizer中的sizer才有。几个常用属性：

* + Project

file指定生成的文件名

first\_id 资源起始ID号

* + Frame

name 框架名，也是框架基类的类名

* + Sizer

orient 方向，用于控制sizer承载控件的排布方向

* + 通用

id 资源ID

size 初始大小，分W;H两个方向，-1为默认，可单独设置

minimum\_size 最小大小

maximum\_size 最大大小

window\_style 窗口风格

proportion 缩放(stretch)因子，整数，0表示不缩放。用于sizer在缩放时安排不同控件的缩放比例。受最大最小值限制。

* + 常见窗口风格

wxALWAYS\_SHOW\_SB 总是显示滚动条

wxHSCROLL 水平滚动条 （VSCROLL为竖直滚动条）

wxSUNKEN\_BORDER 默认或凹陷边框

wxRAISED\_BORDER 突出边框

wxTAB\_TRAVERSAL TAB停靠

* + 常见标志flag(也可通过工具栏按钮修改)

wxEXPAND / wxGROW 充满sizer

wxSHAPED 保持长宽比并充满sizer

wxALL 允许所有边框

## 简单**示例**

* 新建项目

打开程序后系统自动生成一个默认项目，或者File -> New Project新建项目。

为了方便，也可以通过CodeLite新建一个机遇wxFormBuilder的项目，并使用wxFormBuilder打开生成的fbp进行编辑（推荐）：

1. WorkSpace -> New Workspace

输入Workspace name, 勾选Crete the Workspace under a separate directory

1. WorkSpace -> New Project, GUI, Excutable (wxWidgets + wxFB frame)

输入Project name, 勾选Crete the project under a separate directory，其余默认。

1. WorkSpace浏览器找到resources/gui.fbp，使用wxFormBuilder打开。提示打开的文件为老版本时选是（会修改原文件）。

* 设计

1. 生成的模板已设置project的file为gui，Frame的name为MainFrameBase，并且具有一个menuBar，一个mainSizer和一个statusBar。
2. Object Tree选中m\_menuBar, m\_menuFile, 在Component Palette中点击Menu/Toolbar, wxMenuItem添加到m\_menuFile中。

设置properties：

name: m\_menuTest

Label: &Test

Id: wxIDM\_FILE\_TEST

Shortcut: alt+T

设置Events

OnMenuSelection: OnClickFileTest

功能：在File菜单中添加了一个Test菜单，菜单快捷键T，全局快捷键alt+T，响应函数OnClickFileTest。

1. Object Tree选中mainSizer，在Component Palette中点击Containers, wxScrolledWindow添加到mainSizer中(m\_ScrolledWindow1)。

设置minimum\_size为200;-1。表示最小宽度200，高度不限。

window\_style 为wxALWAYS\_SHOW\_SB|wxHSCROLL|wxSUNKEN\_BORDER。

1. 在Component Palette中点击Containers, wxPanel添加到mainSizer中（m\_panel1）。

设置minimum\_size和maximum\_size为-1;50。限制高度为50。

window\_style 为wxRAISED\_BORDER|wxTAB\_TRAVERSAL

1. 根据需要，可以选中m\_panel1，添加sizer之后放入一些button。

* 生成代码

File -> Generate Code生成相应代码。默认生成gui.h和gui.cpp。

* 添加事件响应函数

CodeLite中编辑main，实现gui中虚函数OnClickFileTest。例如可以弹出对消息框：

wxMessageBox(wxT("OnClickFileTest"));

* 编译

CodeLite中Build并运行。可以看到设计的界面，支持缩放。选中菜单File -> Test或按alt+T，则会弹出消息框。

* 使用建议

CodeLite默认继承关系MainFrame (main.cpp) -> MainFrameBase (gui.cpp) -> wxFrame (wxWidgets)。界面编辑使用wxFormBuilder，生成代码进入gui.cpp。用户代码在MainFrame (main.cpp)中实现。实现了界面设计和用户代码的分离。fbp文件随源代码发布，项目改动时可以通过wxFormBuilder修改，生成gui文件，如果原布局没有改动，则用户代码不需修改。

## 布局备注

支持AUI的框架/容器，可以直接布局控件，不能直接使用sizer

不支持AUI的框架/容器，需要先布局sizer，然后在sizer里面布局控件。

部分容器可以套嵌，但是需要满足上述要求。比如wxPanel可以布局一个sizer，然后在sizer里面布局aui控件。

wxAuiNoteBook可以直接布局wxPanel，每个wxPanle为一个标签页

Frame框架不论是否支持AUI，都可以布局wxManuBar, wxToolBar, wxStatusBar。

## 常见问题

### 控件文字为乱码

* 环境

Windows 7 x64 SP1, VS2008, wxFormBuilder 3.5.0, wxWidgets 3.0.2

* 现象

使用wxFormBuilder生成的代码，在VS2008下查看正常，但是编译后控件文字显示为乱码。考虑是编码问题。

XP下为发现该现象（未确认）。

编辑gui.cpp，将显示为乱码的文字赋值给wxString，调试时字串显示为乱码。

赋值时使用使用wxString::FromUTF8可以正确解析字符串。

将生成的gui.cpp（未使用wxString::FromUTF8）另存为，选择编码：Unicode (UTF-8 带签名) - 代码页65001，则显示正常。

* 相关源码

(以下wxFormBuilder源代码解析部分为个人推测)

codewrite.cpp // 文件写入类

FileCodeWriter::WriteBuffer() // 将生成的代码写入文件

if ( m\_useMicrosoftBOM ) // 判断是否需要写入MS BOM头

cpppanel.cpp // cpp文件生成类

CppPanel::OnCodeGeneration() // 生成cpp代码

PProperty pUseMicrosoftBOM = project->GetProperty( wxT( "use\_microsoft\_bom" ) // 获取use\_microsoft\_bom参数

wxFormBuilder程序文件夹中全文搜索use\_microsoft\_bom，在xml文件夹中default.xml包含相关项。打开该文件，报告文件解析错误：Couldn’t find end of Start Tag property line 101，找到101行

<property name="window\_name" type="wxString"help="The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows."/>

可以看到type="wxString"help=之间少了一个空格，修改为

type="wxString" help=即可

该文件为wxFormBuilder的选项文件，用于控制Object Properties等的显示内容。所以wxFormBuilder中应该由相应选项，勾选即可解决。

* 解决

wxFormBuilder，Object Tree选中项目，Object Properties, Properties中勾选use\_microsoft\_bom，然后Generate Code即可。问题解决。

# VC使用wxWidgets

## 环境

wxWidgets 3.0.1 vc\_dll 或 vc\_lib

Visual Studio 2008 sp1

vc\_dll采用动态链接方式，程序体积比较小，但是需要wxWidgets的运行时库。

vc\_lib采用静态链接方式，程序体积比较大，但是不需要wxWidgets的运行时库。

## 属性配置

* C/C++ -> 常规 -> 附加包含目录

C:\wxWidgets-3.0.1\include

C:\wxWidgets-3.0.1\include\msvc

* 链接器 -> 常规 -> 附加库目录

C:\wxWidgets-3.0.1\lib\vc\_lib (静态链接)

或

C:\wxWidgets-3.0.1\lib\vc\_dll (动态dll链接，具体参见“使用动态链接”一节)

* 附注：

可以考虑将wxWidgets包含目录加入到全局属性中。

1. 添加环境变量WXPATH。控制面板->系统->高级->环境变量->（根据需要选择用户变量或系统变量）新建。变量名：WXPATH，变量值：C:\wxWidgets-3.0.1（wxWidgets安装根目录）.
2. VS2008：工具->选项->项目和解决方案->VC++目录->显示以下内容的目录->包含文件。添加$(WXPATH)\include和$(WXPATH)\include\msvc
3. VS2008：工具->选项->项目和解决方案->VC++目录->显示以下内容的目录->源文件。添加$(WXPATH)\src。用于Intellisense检索。
4. VAssistX -> Visual Assist X Options -> Performance -> Rebuild重建VAX数据库。
5. 设置wxWidgets库目录:   
   $(WXPATH)\lib\vc\_lib (静态链接)  
   $(WXPATH)\lib\vc\_dll (动态链接)  
   如果如果以后所有项目都设采用同一种链接方式，则在全局属性设置该目录，否则，该需要对每一个项目单独设置。

全局方式：VS2008->工具->选项->项目和解决方案->VC++目录->显示以下内容的目录->库文件

单独设置：VS2008->项目属性->链接器->常规->附加库目录

## 使用动态链接

* 实现方法：

1. 定义WXUSINGDLL宏

项目属性，配置属性，C/C++，预处理器，预处理器定义

添加宏定义WXUSINGDLL;

1. 附加库目录使用dll库

对于VC, 链接器 -> 常规 -> 附加库目录设置为$ (WXPATH)\lib\vc\_dll

* 获取编译后程序所依赖的库

对于VS2008，可以使用dumpbin.exe获取可执行程序所依赖的库文件列表

使用“Visual Studio 2008 命令提示行”或设置dumpbin的路径(VS2008安装路径)\VC\bin，运行命令 dumpbin /dependents exefile.exe即可。例如：

> dumpbin /dependents arttest.exe

Microsoft (R) COFF/PE Dumper Version 9.00.30729.01

Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file arttest.exe

File Type: EXECUTABLE IMAGE

Image has the following dependencies:

wxmsw30ud\_core\_vc\_custom.dll

wxbase30ud\_vc\_custom.dll

MSVCR90D.dll

KERNEL32.dll

Summary

1000 .data

B000 .idata

C000 .rdata

8000 .rsrc

11000 .text

* VC编译后自动显示依赖库(VS2008)

项目属性，生成事件，生成后事件，dumpbin /dependents $(TargetPath)

则VC会在编译完成后在输出窗口（输出来源：生成）显示上述依赖信息。但是由于VC已经将输出重定向到输出窗口（包括生成后事件的输出），因此上述设置使用 > 重定向输出到文件(dumpbin /dependents $(TargetPath) > xxx.txt，此时xxx.txt只会被创建，但没有内容)。

## 示例：纯代码

* **新建”win32项目”，空项目**

新建cpp文件，源代码摘自docs\htmlhelp\wx.chm

Documentation -> Programming Guides -> Hello World Example

建议增加头文件包含msvc/wx/setup.h，该文件会自动链接wx的lib库：

#include <msvc/wx/setup.h>

注意！！

如果新建”win32控制台应用程序”，但是wxWidgets代码是基于wxFrame的，会导致程序链接过程中找不到\_main在\_\_\_tmainCRTStartup中。

* **代码**



#include <wx/wxprec.h>

#ifndef WX\_PRECOMP

#include <wx/wx.h>

#endif

// link wxlib

#include <msvc/wx/setup.h>

class MyApp: public wxApp

{

public:

virtual bool OnInit();

};

class MyFrame: public wxFrame

{

public:

MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size);

private:

void OnHello(wxCommandEvent& event);

void OnExit(wxCommandEvent& event);

void OnAbout(wxCommandEvent& event);

wxDECLARE\_EVENT\_TABLE();

};

enum

{

ID\_Hello = 1

};

// event table

wxBEGIN\_EVENT\_TABLE(MyFrame, wxFrame)

EVT\_MENU(ID\_Hello, MyFrame::OnHello)

EVT\_MENU(wxID\_EXIT, MyFrame::OnExit)

EVT\_MENU(wxID\_ABOUT, MyFrame::OnAbout)

wxEND\_EVENT\_TABLE()

// app implementation

wxIMPLEMENT\_APP(MyApp);

FILE\* pFile = NULL;

bool MyApp::OnInit()

{

MyFrame \*frame = new MyFrame( "Hello World", wxPoint(50, 50), wxSize(450, 340) );

frame->Show( true );

pFile = fopen("d:\\log.txt", "w+");

if (pFile)

{

wxLog\* logger = new wxLogStderr(pFile);

wxLog\* plogold = wxLog::GetActiveTarget();

wxLog::SetActiveTarget(logger);

wxLog::SetActiveTarget(plogold);

}

return true;

}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)

: wxFrame(NULL, wxID\_ANY, title, pos, size)

{

wxMenu \*menuFile = new wxMenu;

menuFile->Append(ID\_Hello, "&Hello...\tCtrl-H",

"Help string shown in status bar for this menu item");

menuFile->AppendSeparator();

menuFile->Append(wxID\_EXIT);

wxMenu \*menuHelp = new wxMenu;

menuHelp->Append(wxID\_ABOUT);

wxMenuBar \*menuBar = new wxMenuBar;

menuBar->Append( menuFile, "&File" );

menuBar->Append( menuHelp, "H&elp" );

SetMenuBar( menuBar );

CreateStatusBar();

SetStatusText( "Welcome to wxWidgets!" );

}

void MyFrame::OnExit(wxCommandEvent& event)

{

wxLogNull lognull;

fclose(pFile);

Close( true );

}

void MyFrame::OnAbout(wxCommandEvent& event)

{

wxMessageBox( "This is a wxWidgets' Hello world sample",

"About Hello World",

wxOK | wxICON\_INFORMATION );

}

void MyFrame::OnHello(wxCommandEvent& event)

{

wxTrace("Hello world from wxWidgets!");

wxLogMessage("Hello world from wxWidgets!");

}

## 示例：RAD

* 新建win32空项目
* 使用wxFormBuilder生成fbp文件及源文件，并添加到项目。

1. 生成fbp项目文件及源代码

可以借助使用CodeLite生成模板，WorkSpace -> New Project选择Executable (wxWidgets + wxFB frame / dialog)生成新的项目。这样做的好处是可以自动生成所需框架base类及main代码，并且产生了DECLARE\_APP和IMPLEMENT\_APP代码。

1. 添加到VS项目

将生成的gui.fbp, gui.h, gui.cpp, main.h, main.cpp添加到项目

1. VS中关联wxFormBuilder和fbp文件

VS解决方案资源管理器，选中gui.fbp，右键 -> 打开方式 -> 添加wxFormBuilder并设为默认。

1. 使用wxFormBuilder进行界面编辑

VS中双击gui.fbp，打开wxFormBuilder对界面进行编辑。可添加菜单项(wxMenuItem)及响应函数(Events)等。编辑完成后保存。

1. 生成代码。

wxformBuilder -> File -> Generate Code (F8)生成gui.h和gui.cpp。该文件为框架基类MainFrameBase定义文件。不建议修改。在主程序main中完成MainFrameBase相应响应函数（参照gui.h中的声明）。

* 编译并执行/调试

## 常见错误

### MyApp.obj 无法解析的外部符号 wxEventTable等

* 错误原因：

1. 未链接接wxWidgets库
2. 链接wxWidgets DLL库但未定义宏WXUSINGDLL

* 解决方法：

1. 使用静态lib库链接
2. 使用动态DLL库，并且添加宏定义WXUSINGDLL

* 详细错误信息：

1>------ 已启动生成: 项目: wxTest3, 配置: Debug Win32 ------

1>正在编译...

1>MyApp.cpp

1>正在编译资源清单...

1>Microsoft (R) Windows (R) Resource Compiler Version 6.1.6723.1

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>正在链接...

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "protected: static struct wxEventTable const wxFrame::sm\_eventTable" (?sm\_eventTable@wxFrame@@1UwxEventTable@@B)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "protected: static class wxAppConsole \* wxAppConsoleBase::ms\_appInstance" (?ms\_appInstance@wxAppConsoleBase@@1PAVwxAppConsole@@A)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "class wxMBConv \* wxConvLibcPtr" (?wxConvLibcPtr@@3PAVwxMBConv@@A)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "public: static unsigned int const wxString::npos" (?npos@wxString@@2IB)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "char const \* const wxStatusLineNameStr" (?wxStatusLineNameStr@@3QBDB)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "wchar\_t const \* const wxEmptyString" (?wxEmptyString@@3PB\_WB)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "char const \* const wxFrameNameStr" (?wxFrameNameStr@@3QBDB)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "class wxSize const wxDefaultSize" (?wxDefaultSize@@3VwxSize@@B)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "class wxPoint const wxDefaultPosition" (?wxDefaultPosition@@3VwxPoint@@B)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "char const \* const wxLOG\_COMPONENT" (?wxLOG\_COMPONENT@@3PBDB)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "private: static bool wxLog::ms\_doLog" (?ms\_doLog@wxLog@@0\_NA)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "private: static unsigned long wxThread::ms\_idMainThread" (?ms\_idMainThread@wxThread@@0KA)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "int const wxEVT\_NULL" (?wxEVT\_NULL@@3HB)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "class wxEventTypeTag<class wxCommandEvent> const wxEVT\_MENU" (?wxEVT\_MENU@@3V?$wxEventTypeTag@VwxCommandEvent@@@@B)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "bool wxTrapInAssert" (?wxTrapInAssert@@3\_NA)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "void (\_\_cdecl\* wxTheAssertHandler)(class wxString const &,int,class wxString const &,class wxString const &,class wxString const &)" (?wxTheAssertHandler@@3P6AXABVwxString@@H000@ZA)

1>MyApp.obj : error LNK2001: 无法解析的外部符号 "protected: static class wxAppConsole \* (\_\_cdecl\* wxAppConsoleBase::ms\_appInitFn)(void)" (?ms\_appInitFn@wxAppConsoleBase@@1P6APAVwxAppConsole@@XZA)

1>D:\Jeson\Src\wxTest3\Debug\wxTest3.exe : fatal error LNK1120: 17 个无法解析的外部命令

1>生成日志保存在“file://d:\Jeson\Src\wxTest3\wxTest3\Debug\BuildLog.htm”

1>wxTest3 - 18 个错误，0 个警告

========== 生成: 成功 0 个，失败 1 个，最新 0 个，跳过 0 个 ==========

### wxmsw30ud\_core.lib :无法解析的外部符号\_\_imp\_\_InitCommonControls等

* 原因：

wxWidgets的MSW实现使用了windows的api，但是静态链接时未连接lib库

* 解决方法：

查找MSDN InitCommonControls，可以找到相关信息。

#pragma comment(lib, "comctl32.lib")

* 详细错误信息：

1>------ 已启动生成: 项目: wxTest3, 配置: Debug Win32 ------

1>正在编译...

1>MyApp.cpp

1>正在链接...

1>wxmsw30ud\_core.lib(app.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_InitCommonControls@0，该符号在函数 "public: virtual bool \_\_thiscall wxApp::Initialize(int &,wchar\_t \* \*)" (?Initialize@wxApp@@UAE\_NAAHPAPA\_W@Z) 中被引用

1>wxmsw30ud\_core.lib(spinbutt.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_CreateUpDownControl@48，该符号在函数 "public: bool \_\_thiscall wxSpinButton::Create(class wxWindow \*,int,class wxPoint const &,class wxSize const &,long,class wxString const &)" (?Create@wxSpinButton@@QAE\_NPAVwxWindow@@HABVwxPoint@@ABVwxSize@@JABVwxString@@@Z) 中被引用

1>wxmsw30ud\_core.lib(ownerdrw.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_Destroy@4，该符号在函数 "int \_\_cdecl wxDrawStateBitmap(struct HDC\_\_ \*,struct HBITMAP\_\_ \*,int,int,unsigned int)" (?wxDrawStateBitmap@@YAHPAUHDC\_\_@@PAUHBITMAP\_\_@@HHI@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Destroy@4

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Destroy@4

1>wxmsw30ud\_core.lib(ownerdrw.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_Draw@24，该符号在函数 "int \_\_cdecl wxDrawStateBitmap(struct HDC\_\_ \*,struct HBITMAP\_\_ \*,int,int,unsigned int)" (?wxDrawStateBitmap@@YAHPAUHDC\_\_@@PAUHBITMAP\_\_@@HHI@Z) 中被引用

1>wxmsw30ud\_core.lib(listctrl.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Draw@24

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Draw@24

1>wxmsw30ud\_core.lib(ownerdrw.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_Add@12，该符号在函数 "int \_\_cdecl wxDrawStateBitmap(struct HDC\_\_ \*,struct HBITMAP\_\_ \*,int,int,unsigned int)" (?wxDrawStateBitmap@@YAHPAUHDC\_\_@@PAUHBITMAP\_\_@@HHI@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Add@12

1>wxmsw30ud\_core.lib(treectrl.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Add@12

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Add@12

1>wxmsw30ud\_core.lib(ownerdrw.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_Create@20，该符号在函数 "int \_\_cdecl wxDrawStateBitmap(struct HDC\_\_ \*,struct HBITMAP\_\_ \*,int,int,unsigned int)" (?wxDrawStateBitmap@@YAHPAUHDC\_\_@@PAUHBITMAP\_\_@@HHI@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Create@20

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_Create@20

1>wxmsw30ud\_core.lib(listctrl.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_GetIconSize@12，该符号在函数 "bool \_\_cdecl HandleSubItemPrepaint(struct tagNMLVCUSTOMDRAW \*,struct HFONT\_\_ \*,int)" (?HandleSubItemPrepaint@@YA\_NPAUtagNMLVCUSTOMDRAW@@PAUHFONT\_\_@@H@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_GetIconSize@12

1>wxmsw30ud\_core.lib(listctrl.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_GetImageCount@4，该符号在函数 "bool \_\_cdecl HandleSubItemPrepaint(struct tagNMLVCUSTOMDRAW \*,struct HFONT\_\_ \*,int)" (?HandleSubItemPrepaint@@YA\_NPAUtagNMLVCUSTOMDRAW@@PAUHFONT\_\_@@H@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_GetImageCount@4

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_AddMasked@12，该符号在函数 "public: int \_\_thiscall wxImageList::Add(class wxBitmap const &,class wxColour const &)" (?Add@wxImageList@@QAEHABVwxBitmap@@ABVwxColour@@@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_ReplaceIcon@12，该符号在函数 "public: int \_\_thiscall wxImageList::Add(class wxIcon const &)" (?Add@wxImageList@@QAEHABVwxIcon@@@Z) 中被引用

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2001: 无法解析的外部符号 \_\_imp\_\_ImageList\_ReplaceIcon@12

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_Replace@16，该符号在函数 "public: bool \_\_thiscall wxImageList::Replace(int,class wxBitmap const &,class wxBitmap const &)" (?Replace@wxImageList@@QAE\_NHABVwxBitmap@@0@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_Remove@8，该符号在函数 "public: bool \_\_thiscall wxImageList::Remove(int)" (?Remove@wxImageList@@QAE\_NH@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_SetBkColor@8，该符号在函数 "public: bool \_\_thiscall wxImageList::Draw(int,class wxDC &,int,int,int,bool)" (?Draw@wxImageList@@QAE\_NHAAVwxDC@@HHH\_N@Z) 中被引用

1>wxmsw30ud\_core.lib(imaglist.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_GetIcon@12，该符号在函数 "public: class wxIcon \_\_thiscall wxImageList::GetIcon(int)const " (?GetIcon@wxImageList@@QBE?AVwxIcon@@H@Z) 中被引用

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_UuidToStringW@8，该符号在函数 "public: \_\_thiscall Uuid::Uuid(class Uuid const &)" (??0Uuid@@QAE@ABV0@@Z) 中被引用

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_RpcStringFreeW@4，该符号在函数 "public: \_\_thiscall Uuid::~Uuid(void)" (??1Uuid@@QAE@XZ) 中被引用

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_UuidCreate@4，该符号在函数 "public: void \_\_thiscall Uuid::Create(void)" (?Create@Uuid@@QAEXXZ) 中被引用

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_UuidFromStringW@8，该符号在函数 "public: bool \_\_thiscall Uuid::Set(wchar\_t const \*)" (?Set@Uuid@@QAE\_NPB\_W@Z) 中被引用

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_SetDragCursorImage@16，该符号在函数 "public: bool \_\_thiscall wxDragImage::BeginDrag(class wxPoint const &,class wxWindow \*,bool,class wxRect \*)" (?BeginDrag@wxDragImage@@QAE\_NABVwxPoint@@PAVwxWindow@@\_NPAVwxRect@@@Z) 中被引用

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_BeginDrag@16，该符号在函数 "public: bool \_\_thiscall wxDragImage::BeginDrag(class wxPoint const &,class wxWindow \*,bool,class wxRect \*)" (?BeginDrag@wxDragImage@@QAE\_NABVwxPoint@@PAVwxWindow@@\_NPAVwxRect@@@Z) 中被引用

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_EndDrag@0，该符号在函数 "public: bool \_\_thiscall wxDragImage::EndDrag(void)" (?EndDrag@wxDragImage@@QAE\_NXZ) 中被引用

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_DragMove@8，该符号在函数 "public: bool \_\_thiscall wxDragImage::Move(class wxPoint const &)" (?Move@wxDragImage@@QAE\_NABVwxPoint@@@Z) 中被引用

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_DragEnter@12，该符号在函数 "public: bool \_\_thiscall wxDragImage::Show(void)" (?Show@wxDragImage@@QAE\_NXZ) 中被引用

1>wxmsw30ud\_core.lib(dragimag.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_ImageList\_DragLeave@4，该符号在函数 "public: bool \_\_thiscall wxDragImage::Hide(void)" (?Hide@wxDragImage@@QAE\_NXZ) 中被引用

1>D:\Jeson\Src\wxTest3\Debug\wxTest3.exe : fatal error LNK1120: 24 个无法解析的外部命令

1>生成日志保存在“file://d:\Jeson\Src\wxTest3\wxTest3\Debug\BuildLog.htm”

1>wxTest3 - 37 个错误，0 个警告

========== 生成: 成功 0 个，失败 1 个，最新 0 个，跳过 0 个 ==========

### wxmsw30ud\_core.lib: 无法解析的外部符号 \_\_imp\_\_UuidToStringW等

* 错误原因：

wxWidgets的MSW实现使用了windows的api，但是静态链接时未连接lib库。

* 解决方法：

查找MSDN UuidToString，可以找到相关信息。

#pragma comment(lib, "rpcrt4.lib")

* 参考资料

http://whatwg.blog.163.com/blog/static/544502372012080590251/

http://blog.sina.com.cn/s/blog\_6f6dfec7010195mt.html

* 详细信息：

1>------ 已启动生成: 项目: wxTest3, 配置: Debug Win32 ------

1>正在编译...

1>MyApp.cpp

1>正在链接...

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_UuidToStringW@8，该符号在函数 "public: \_\_thiscall Uuid::Uuid(class Uuid const &)" (??0Uuid@@QAE@ABV0@@Z) 中被引用

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_RpcStringFreeW@4，该符号在函数 "public: \_\_thiscall Uuid::~Uuid(void)" (??1Uuid@@QAE@XZ) 中被引用

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_UuidCreate@4，该符号在函数 "public: void \_\_thiscall Uuid::Create(void)" (?Create@Uuid@@QAEXXZ) 中被引用

1>wxmsw30ud\_core.lib(uuid.obj) : error LNK2019: 无法解析的外部符号 \_\_imp\_\_UuidFromStringW@8，该符号在函数 "public: bool \_\_thiscall Uuid::Set(wchar\_t const \*)" (?Set@Uuid@@QAE\_NPB\_W@Z) 中被引用

1>D:\Jeson\Src\wxTest3\Debug\wxTest3.exe : fatal error LNK1120: 4 个无法解析的外部命令

1>生成日志保存在“file://d:\Jeson\Src\wxTest3\wxTest3\Debug\BuildLog.htm”

1>wxTest3 - 5 个错误，0 个警告

========== 生成: 成功 0 个，失败 1 个，最新 0 个，跳过 0 个 ==========

### MSVCRTD.lib(crtexe.obj) :无法解析的外部符号 \_main，该符号在函数 \_\_\_tmainCRTStartup 中被引用

* 错误原因：

使用了错误的项目类型。比如wxFrame程序在VS中新建项目时选择了”win32控制台应用程序”而非”win2项目”。此时会导致程序连接到错误的入口函数。

* 解决方法：

1. 重新新建合适的空项目
2. 修改项目属性
3. VS2008选中项目->项目->属性->配置属性->预处理器->预处理器定义

对”win32项目”需定义\_WINDOWS

对”win32控制台应用程序”需定义\_CONSOLE

修改相应定义即可。

1. VS2008选中项目->项目->属性->配置属性->连接器->系统->子系统

修改”控制台…”为”Windows…”

* 详细信息：

1>------ 已启动生成: 项目: wxXmlExplorer, 配置: Debug Win32 ------

1>正在编译...

1>gui.cpp

1>main.cpp

1>MainFunctions.cpp

1>tinyxml2.cpp

1>正在生成代码...

1>正在编译资源清单...

1>Microsoft (R) Windows (R) Resource Compiler Version 6.1.6723.1

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>正在链接...

1>MSVCRTD.lib(crtexe.obj) : error LNK2019: 无法解析的外部符号 \_main，该符号在函数 \_\_\_tmainCRTStartup 中被引用

1>D:\SvnRoot\RdLocalMain\trunk\wxXmlEditor\bin\run\wxXmlExplorer\_Debug\_Win32.exe : fatal error LNK1120: 1 个无法解析的外部命令

1>生成日志保存在“file://D:\SvnRoot\RdLocalMain\trunk\wxXmlEditor\bin\inter\wxXmlExplorer\_Debug\_Win32\BuildLog.htm”

1>wxXmlExplorer - 2 个错误，0 个警告

========== 生成: 成功 0 个，失败 1 个，最新 0 个，跳过 0 个 ==========

### 编译资源/打开资源文件时产生错误C1015：

* 错误原因：

编译资源文件(rc文件)时，Visual Studio无法搜索到相关文件。

注意：Visual Studio搜索rc中的包含文件时，并不检索项目属性中的附加包含目录，而是搜索全局包含目录列表。

* 解决方法：

将wxWidgets路径添加到Visual Studio全局包含目录列表中。

工具，选项，项目和解决方案，显示以下内容的目录，包含文件

添加$(WXPATH)\include

* 详细信息：
* 编译资源文件win\_resources.rc时出现RC1015错误

1>正在编译...

1>gui.cpp

1>main.cpp

1>正在生成代码...

1>正在编译资源...

1>Microsoft (R) Windows (R) Resource Compiler Version 6.1.6723.1

1>Copyright (C) Microsoft Corporation. All rights reserved.

1>.\win\_resources.rc(1) : fatal error RC1015: cannot open include file 'wx/msw/wx.rc'.

* 打开wx\msw\wx.rc时出现RC1015错误

C:\wxWidgets-3.0.2\include\wx\msw\wx.rc(18)

fatal error RC1015: cannot open include file 'wx/msw/rcdefs.h'

# 部分类用法

## wxFrame

响应事件wxEVT\_ACTIVATE可放置初始化代码

## wxDialog

响应事件wxEVT\_INIT\_DIALOG可放置初始化代码

## wxStatusBar

* 用途

显示一个状态栏

* 显示状态文本

SetStatusText("status text.");

## wxTextCtrl

* 用途

文本控件

* 显示状态文本

SetValue("text.");

## wxDataViewListCtrl

* 用途

显示数据列表，支持列序重排，列排序，特殊单元格（复选框，编辑框，进度条，含图标单元）等。

* 添加列

AppendTextColumn：文本单元

AppendToggleColumn：复选框单元

AppendProgressColumn：进度条单元

AppendIconTextColumn：含图标单元

参数：

enum wxDataViewCellMode

{

wxDATAVIEW\_CELL\_INERT,

wxDATAVIEW\_CELL\_ACTIVATABLE,

wxDATAVIEW\_CELL\_EDITABLE

};

enum wxDataViewColumnFlags

{

wxDATAVIEW\_COL\_RESIZABLE = wxCOL\_RESIZABLE,

wxDATAVIEW\_COL\_SORTABLE = wxCOL\_SORTABLE,

wxDATAVIEW\_COL\_REORDERABLE = wxCOL\_REORDERABLE,

wxDATAVIEW\_COL\_HIDDEN = wxCOL\_HIDDEN

};

enum // width

{

// special value for column width meaning unspecified/default

wxCOL\_WIDTH\_DEFAULT = -1,

// size the column automatically to fit all values

wxCOL\_WIDTH\_AUTOSIZE = -2

};

* 常用方法

DeleteAllItems()

GetItemCount()

GetValue

GetTextValue

GetToggleValue

SetTextValue

* 插入数据InsertItem

wxVector<wxVariant> vtItem;

vtItem.push\_back(nth value of row);

…

InsertItem(nCol, vtItem)

* 部分事件

wxEVT\_COMMAND\_DATAVIEW\_ITEM\_ACTIVATED

激活单元格（双击），可以实现双击单元格出现列表框等操作。

wxEVT\_COMMAND\_DATAVIEW\_ITEM\_EDITING\_DONE

单元格编辑完成

* 获取单元格编辑完成示例

当单元格编辑完成后获取该事件可实时获取编辑后的值

wxVariant value;

event.GetModel()->GetValue( value, event.GetItem(), 1 );

wxString strItemName = value;

wxString strValue = event.GetValue();

int nRow = m\_dvlistMainSetting->ItemToRow(event.GetItem());

SetStatusText(wxString::Format("%s = %s | %s = %s", strItemName,

strValue,

m\_dvlistMainSetting->GetTextValue(nRow, 1),

m\_dvlistMainSetting->GetTextValue(nRow, 2)));

return;

const wxDataViewItem & item = event.GetItem();

const wxDataViewColumn \* column =event.GetDataViewColumn();

if (column)

{

wxDataViewRenderer\* pRender = column->GetRenderer();

unsigned int col = pRender->GetOwner()->GetModelColumn();

wxVariant value;

m\_dvlistMainSetting->GetModel()->GetValue( value, item, col );

wxString str = value;

}

# 使用wxAUI

## 概览

### 简介

AUI (Advanced User Interface)目标是为用户提供先进的浮动窗口及可自定义的窗口布局。具体可参考用户手册中wxAui Overview，其框架核心类是wxAuiManager。

wxAuiManager通过所管理窗格(pane)的wxAuiPaneInfo信息决定pane的停靠及浮动行为，使用sizer机制决定窗口布局。所有重绘机制通过可替换的dock art类实现。

首先添加pane到manager，或修改已有pane信息。完成之后调用Update函数一次性更新窗口，以避免单个pane的重绘闪烁。可使用GetPane获取相应pane进行操作。

### 名词约定

Pane：窗格。使用wxAUI的时候，一个控件或容器可以作为一个pane。多个pane可以放置在同一个dock中。

Directions： dock的停靠方向。包括上、下、左、右和中间。

Positions：停靠顺序。同一dock中的pane，按照position从小到大横向并排排列。如左侧停靠则pane从上到下并排排列。

Row：不同row的dock并排排列。如左侧的dock，按照从左到右的顺序排列。

Layer：可以将layer看做洋葱结构，layer0是最内部的pane，最靠近中央的pane（一般作为内容pane使用）。外层的pane完全包裹住内层pane。

\*\*以下部分为个人定义，为便于说明而用。

Dock：停靠位。同一个停靠方向可以有一个活多个dock。如顶部停靠的窗口会占据顶部的一个横条，这个横条就成为dock。如顶部停靠有多个dock时，不同layer的dock右下到上依次排列。

停靠：dock，指pane在主窗口中，成为主窗口的一部分。

悬浮/浮动：float，指pane脱离主窗口处于悬浮状态。

横向和纵向：由于停靠位置除了中间外有4个，为了便于说明，横向表示垂直于停靠边沿的方向，纵向表示平行于停靠边沿的方向，内侧表示靠近中心，外侧表示远离中心。具体如下

|  |  |  |  |
| --- | --- | --- | --- |
|  | 横向 | 纵向 | 内侧 |
| 上方停靠 | 左右 | 上下 | 下 |
| 下方停靠 | 左右 | 上下 | 上 |
| 左侧停靠 | 上下 | 左右 | 右 |
| 右侧停靠 | 上下 | 左右 | 左 |

### 窗口布局

wxAuiManager通过相应pane的wxAuiPaneInfo信息控制pane的行为及布局。详细信息参考手册wxAuiManager类说明Layers, Rows and Directions, Positions部分的解释。

Layer和Row相似，但是作用不同：布局时按照layer –> row -> position的顺序计算位置，相同layer的不同row具有相同的横向宽度。

对于四角，由内而外，不同layer较小的优先占据，layer相同，上下位置的dock优先。

可通过wxAUI示例程序（samples\aui\auidemo）观察具体布局方式。

### 常用选项

\*\*指wxFormBuilder 3.5 beta，对象属性(Object Properties)中的AUI部分的相关选项。

aui\_name: 内部名称

caption\_visible: 标题栏是否课件

caption: 标题，显示在pane的标题栏

close/maximize/minimize/pin\_button:是否具有关闭/最大化/最小化/停靠按钮

pane\_border: 是否具有pane边界

gripper: 是否边缘自动停靠。选中时，pane靠近边界则会自动停靠。

center/default/toolbar\_pane: pane自动采用的设置样式

moveable:

resize: 停靠时pane可以通过鼠标拖动边沿调整大小。

pane\_size: 悬停后的大小

best/min/max\_size：停靠后最佳/最小/最大大小。可单独设置长/宽。如(-1,200)表示仅限制高度为200。best\_size可能受限于其他因素。

dock：dock停靠或float悬停。

dock\_fixed:

floatable: 是否可悬浮。

bottom/Top/Left/RightDockable: 是否底部/顶部/左侧/右侧可停靠。

docking: 停靠方向。可选上/下/左/右/中央。

pane\_position: 浮动窗口的位置。当窗口初始为浮动窗口时有效。

aui\_position: 参见8.1.2名词约定及8.1.3窗口布局

aui\_row: 参见8.1.2名词约定及8.1.3窗口布局

aui\_layer: 参见8.1.2名词约定及8.1.3窗口布局

## 环境

wxWidgets: 3.0.1, vc\_lib compile

IDE: Visual Studio 2008 SP1

RAD: wxFormBuilder 3.5 beta

辅助：CodeLite 6.0

## 示例

### 新建项目

参见7. VC使用wxWidgets, 7.4. RAD示例，新建wxForm项目。

### 编辑gui.fbp

* 使用wxFormBuilder进行编辑gui.fbp。
* 启用框架AUI支持

单击MainFrameBase: Frame，右侧属性中勾选aui\_managed。如果框架下已有控件，需要全部删除口才能修改该属性。

* 添加控件

直接向框架中添加控件。框架下不能直接添加sizer。通过修改pane类型及docking位置属性等来控制窗口布局。

### 编译并调试程序

## 注意事项

### 调整控件布局

* 默认情况下，上/下停靠的控件宽度布满窗口，左/右停靠的控件占用左右两侧剩余部分。
* 控件AUI属性aui\_layer控制着控件的布局。aui\_layer表明控件占用控件的优先级，数值越大优先级越高。aui\_layer优先级高的控件优先占用边缘空间。

1. aui\_layer越大越靠近边缘。

例如左侧停靠，不同优先级控件从左到右依次排列。

1. aui\_layer相等时同侧控件并排排列。

如左侧停靠，同一aui\_layer控件在同一列垂直并排排列。

1. aui\_layer相等时邻侧控件上/下侧控件优先级高于左右侧控件。

* 手动调整窗口布局

如果需要调整该布局，可以在wxFormBuilder里面直接对控件进行拖动摆放，然后根据需要修改pane\_size或pane\_position。修改这两个属性为-1;-1后不会影响大的窗口布局

### best\_size选项不起效

* 现象

主窗口设为800\*600时，控件ctrl下方停靠，best\_size设为(-1, hight)。当hight小于某一值x（接近180）时有效，而当hight大于该值时，窗口高度仍然为x。

* 推测

内部机制对dock的纵向长度有默认限制。

* 解决流程

跟踪代码，查看相应处理流程。

由于wxAuiManager通过AddPane添加pane时指定best\_size，通过Update函数重绘界面，所以主要跟踪这两个函数。

AddPane: 将pane加入到m\_panes队列，但其best\_size并未改变

Update: 在DoFrameLayout()中通过m\_uiParts绘制各部分窗口。而m\_uiParts是通过LayoutAll计算得到：

// create a layout for all of the panes

sizer = LayoutAll(m\_panes, m\_docks, m\_uiParts, false);

LayoutAll：首先根据panes信息计算docks(panes循环)；然后根据docks及pane信息计算size（docks循环中，size即为纵向宽度）。在docks循环中，如果dock的size为0，则重新计算size，此时有最大限制：

// new dock's size may not be more than the dock constraint

// parameter specifies. See SetDockSizeConstraint()

int max\_dock\_x\_size = (int)(m\_dockConstraintX \* ((double)cli\_size.x));

int max\_dock\_y\_size = (int)(m\_dockConstraintY \* ((double)cli\_size.y));

可以看到，默认m\_dockConstraintX/Y都是0.3，即默认情况下一个dock最大纵向长度为窗口长度的30%（含pane\_border边界）。这与下方停靠时pane最大高度接近180相吻合（600\*0.3=180）。

\*\*注意：此处设置的size最小值为dock的最小是而非pane，并且仅当size为0时LayoutAll才会重新计算。再次执行Update时，因为docks的size已经计算过，故而SetDockSizeConstraint再Update不会改变纵向长度限制。

* 解决方法

由于gui.cpp中完成了布局并Update，此时DockSizeConstraint使用的是默认值。而mainframe中再次Update时因为已经计算过dock的size，所以不会重新根据DockSizeConstraint计算新的size。故而在mainframe中再次Update无效。但是pane的min\_size会导致窗口布局的重绘。因此可以通过先设置相应pane的min\_size为best\_size，强制wxAUI使用best\_size重绘所有pane，然后再恢复相应pane的min\_size的方式实现。代码如下

#include <wx/aui/framemanager.h>

void ForceAuiBestSizeOnce(wxAuiManager& auiManager)

{

wxAuiPaneInfoArray& panes = auiManager.GetAllPanes();

deque<wxSize> dqSizes;

for (size\_t i = 0, pane\_count = panes.GetCount(); i < pane\_count; ++i)

{

wxAuiPaneInfo& pane = panes.Item(i);

dqSizes.push\_back(pane.min\_size);

pane.min\_size = pane.best\_size;

}

auiManager.Update();

for (size\_t i = 0, pane\_count = panes.GetCount(); i < pane\_count; ++i)

{

wxAuiPaneInfo& pane = panes.Item(i);

pane.min\_size = dqSizes[i];

}

auiManager.Update();

}

### 程序启动时工具栏不显示图标

当主框架仅有toolbar和/或menubar、statusbar时，程序启动后toolbar上面的tool图标会不显示，需要拖动一下才可以。但是如果主框架加入其它控件后即可正常。疑为bug。

# 使用wxThread

## 概览

### wxThreads类型

区别于进程，线程间共享一个内存空间，因此需要注意同步问题。根据POSIX线程API，wxWidgets有两种线程：detached和joinable，而Win32 API所有的线程都是joinable的。

默认情况下wxWidgets使用detached线程。detached线程需要创建在堆上，并且会在运行结束后删除自身（或通过调用Delete()销毁），成功创建的detached线程无需也不能调用delete销毁。可以在主线程中保留detached线程指针，通过检测指针是否为空确定线程是否在运行并通过指针对线程进行操作（如调用Delete()函数提前结束线程），线程析构函数中对该指针置NULL。对该指针的访问需要同步。

对于joinable线程，可以分配在栈空间或堆上，需要调用Wait()函数等待线程结束并释放资源，如果是在堆上，还需调用delete释放线程对象内存。

### 线程销毁

Wait()或Delete()可以优雅的结束线程。它们是通过等待线程调用TestDestory()函数或线程完成处理（如从wxThread::Entry()返回）来实现的。

Kill()是结束线程的最后手段，但并不建议使用。因为它会导致对象资源无法释放（detached线程的wxThread对象仍会被释放），并且使C运行时库处于不确定状态。

### Secondary Threads

所有非主程序线程(main application thread)的线程都成为secondary threads（类工作线程，次线程？）。

Secondary threads中的GUI调用（如wxWindow或wxBitmap相关调用）是不安全的，并可能导致程序退出。导致这一情况的原因包括依赖的本地API和wxThread没有采用和MFC等相似的GUI事件循环机制。

某些平台下，通过使用wxMutexGUIEnter()和wxMutexGUILeave()建立GUI工作区的方式来实现secondary thread中的GUI调用。但是，一般建议通过wxQueueEvent()向主线程发送消息的方式来完成GUI调用。使用过程中要注意事件类的线程安全性（包括wxString等类不是线程安全的）。

### 不要轮询状态

不建议通过轮询(poll)方式检查线程状态，可能会导致意外情况，如对detached线程使用IsRunning()函数等，建议线程结束时发出通知。

当仅需通知主线程时，可以使用wxQueueEvent()发送事件，而secondary thread中则可以通过设置变量值来获知线程退出，常见的是通过互斥量等同步方式实现。

### 工作流程

* m\_internal

wxThreadInternal\* wxThread::m\_internal是内部线程管理类，实际负责线程的创建，销毁，暂停等工作。

使用wxThreadState wxThreadInternal::m\_state保存线程的工作状态。wxThread::TestDestory()函数即通过检查该状态决定是否终止线程。

* 准备工作

继承线程类MyThread: public wxThread

重写线程函数virtual void\* MyThread::Entry()

* 父线程

创建线程类MyThread对象pThread

pThread->Create()：由m\_internal采用休眠方式创建内部线程，内部线程入口函数为wxThreadInternal::WinThreadStart

pThread->Run(): 恢复内部线程运行

* 子线程

内部线程函数：

WinThreadStart()

流程：

检查状态

通过调用DoThreadStart -> wxThread::CallEntry() -> wxThread::Entry()执行用户线程代码。

设置状态，并清理线程（区分detached / joinable线程）

## 示例

## 注意事项

# 使用wxLog

## 概览

此处的日志(logging)概念不仅包括非交互信息，还包含程序的所有输出。wxWidgets包含的日志工具集提供了基本的wxLog类，该类定义了日志对象的标准接口及其实现，并提供了一组相关函数。

使用日志仅需了解wxLogXXX()系列函数即可，并不需要掌握wxLog的知识。所有函数具有和printf或vprintf相同的语法。

* wxLogFatalError()

类似wxLogError()，但是会使用about()标准函数使程序退出，返回值3。与其他日志函数不同，该函数不能被日志对象(log target)覆盖。

which is like wxLogError(), but also terminates the program with the exit code 3 (using abort() standard function). Unlike for all the other logging functions, this function can't be overridden by a log target.

* wxLogError()

用于记录错误(error)信息。默认弹出消息提示框。

is the function to use for error messages, i.e. the messages that must be shown to the user. The default processing is to pop up a message box to inform the user about it.

* wxLogWarning()

用于记录警告(warning)信息。一般情况下用户课件，但是不会打断程序运行。

for warnings. They are also normally shown to the user, but don't interrupt the program work.

* wxLogMessage()

记录所有普通信息(normal, informational messages)。默认弹出信息提示框，但可以更改。

is for all normal, informational messages. They also appear in a message box by default (but it can be changed, see below).

* wxLogVerbose()

用于输出详细(verbose)信息。一般隐藏不显示(suppressed)，但是需要时可以被激活(active)。另一个容易混淆的函数名为wxLogInfo()。

is for verbose output. Normally, it is suppressed, but might be activated if the user wishes to know more details about the program progress (another, but possibly confusing name for the same function is wxLogInfo).

* wxLogStatus()

记录状态(status)信息。该信息会显示在活动的或指定的状态栏(status bar)中（如果有）。

is for status messages. They will go into the status bar of the active or specified (as the first argument) wxFrame if it has one.

* wxLogSysError()

主要由wxWidgets自身使用，也可以用于记录系统API调用(system call)失败信息。它会记录特定的消息文本以及系统错误代码(last system error code)及相应错误信息。该函数的第二种形式使用确定的错误代码作为第一个参数。

is mostly used by wxWidgets itself, but might be handy for logging errors after system call (API function) failure. It logs the specified message text as well as the last system error code (errno or Windows' GetLastError() depending on the platform) and the corresponding error message. The second form of this function takes the error code explicitly as the first argument.

* wxLogDebug()

用于输出调试信息(debug output)。仅在调试模式（预处理符号WXDEBUG被定义时）下起效。注意：在Windows下，必须在调试器下运行程序，或者使用第三方工具如DebugView才能看到调试信息。

is the right function for debug output. It only does anything at all in the debug mode (when the preprocessor symbol WXDEBUG is defined) and expands to nothing in release mode (otherwise). Note that under Windows, you must either run the program under debugger or use a 3rd party program such as DebugView (http://www.microsoft.com/technet/sysinternals/Miscellaneous/DebugView.mspx) to actually see the debug output.

* wxLogTrace()

和wxLogDebug类似，仅在调试版本中起效。之所以使用单独的函数是因为一般会有很多追踪(trace)信息，二者分开可以避免debug信息被trace信息淹没。同时，该函数支持将第一个参数作为追踪掩码(trace mask)，可以限制产生的信息总数。

as wxLogDebug() only does something in debug build. The reason for making it a separate function from it is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages which would be flooded in them. Moreover, the second version of this function takes a trace mask as the first argument which allows to further restrict the amount of messages generated.

### 优点

wxLog系列函数相比其他日志工具（如C标准stdio函数）更适合wxWidgets，并且具有以下优点：

1. 移植性

可以将wxLogMessage()看做printf()的替代，并且可以将其输出重定向到cout或wxTextCtrl。

1. 伸缩性

wxLog的输出可以根据其优先级被重定向或完全隐藏(suppressed)。如：仅error，仅error和warning，或者过滤所有informational信息。

1. 完整性

由于wxWidgets内部使用了wxLogError()，因此当发生内部调用错误（如wxFile::Write()出现磁盘空间不足时），不仅可以返回调用错误，还可以获知到内部错误代码及相应信息。

### Log选择

默认情况下，所有log都是允许的，包括wxWidgets内部错误信息。可以通过wxLog::EnableLogging()禁止所有log，或使用wxLogNull类临时禁止log，当它析构时自动恢复。

可以使用wxLog::SetLogLevel()禁止低于相应logLevel的log。

另外，可以通过定义wxLOG\_COMPONENT定义每一个模块(component)，并且通过wxLog::SetComponentLevel()来指定是否输出相应模块的log。注意子模块会覆盖上级设置，但同级的仍会继承上级模块设置。

### 日志对象Log Target

wxWidgets中定义继承自wxLog的类为日志对象(Log Target)，其实现了信息记录中所需调用的基类的虚函数。同一时刻只有一个log target是活动的，使用SetActiveTarget()设置，该对象将被wxLogXXX函数使用。

创建新的log target只需要继承wxLog类，并且重写其中的一个或多个函数：wxLog::DoLogRecord(), wxLog::DoLogTextAtLevel() and wxLog::DoLogText()。其中第一个功能最多，可以实现信息格式化，动态过滤，重定向等，除了wxLogFatalError()产生的信息之外都会传入该函数。

其他有用的wxLog派生类包括：

wxLogStderr: This class logs messages to a FILE \*, using stderr by default as its name suggests.

wxLogStream: This class has the same functionality as wxLogStderr, but uses ostream and cerr instead of FILE \* and stderr.

wxLogGui: This is the standard log target for wxWidgets applications (it is used by default if you don't do anything) and provides the most reasonable handling of all types of messages for given platform.

wxLogWindow: This log target provides a "log console" which collects all messages generated by the application and also passes them to the previous active log target. The log window frame has a menu allowing user to clear the log, close it completely or save all messages to file.

wxLogBuffer: This target collects all the logged messages in an internal buffer allowing to show them later to the user all at once.

wxLogNull: The last log class is quite particular: it doesn't do anything. The objects of this class may be instantiated to (temporarily) suppress output of wxLogXXX() functions. As an example, trying to open a non-existing file will usually provoke an error message, but if for some reasons it is unwanted, just use this construction:

另外，如果希望正常处理log的同时将log重定向到其它位置，可以考虑wxLogChain, wxLogInterposer, 和 wxLogInterposerTemp。

### 多线程中的log

从wxWidgets 2.9.1开始，日志函数可以在任意线程中安全调用。非主线程的log信息首先被缓存，直到主线程调用wxLog::Flush()（一般时主线程空闲时，比如处理完所有消息）时才输出。

注意：多线程下，多条信息可能会同一时刻被显示，并且log信息可能会产生顺序错乱(非timestamp顺序)。此外，wxLog::EnableLogging()和wxLogNull只会影响当前线程。

代码跟踪：

wxLogXXX产生log信息

wxLog::OnLog(src\common\log.cpp)判断当前是否处于主线程

不是主线程：

log放入gs\_bufferedLogRecords

wxWakeUpIdle()唤醒

**主线程MainLoop()**：wxEventLoopBase::ProcessIdle() -> wxAppConsoleBase::ProcessIdle() -> wxLog::FlushActive() -> wxLog::FlushThreadMessages() -> wxLog::CallDoLogNow()-> DoLogRecord()显示缓存的log

是主线程：GetMainThreadActiveTarget()获取logger(由SetActiveTarget设置） -> CallDoLogNow -> DoLogRecord()

### 自定义日志Logging Customization

通过定义新的log target可以实现log的自定义。

示例程序dialogs(samples/dialogs)演示了如何使用wxLogGui实现自定义log target。

### 使用trace掩码Trace Mask

注意：现在trace mask几乎没有必要使用，因为结合不同的log component是和log level可以实现相同的效果。关于log components可以参考10.1.2Log选择。

使用wxLogTrace结合wxLog::AddTraceMask可以很容易的实现一些特定的需求，例如trace输出开关等。标准的trace mask参见wxLogTrace()文档。

## 示例

### 框架继承自wxLog

1. 新建wxFrame项目。

参见8.3示例

1. 修改MainFrame继承。

同时继承自MainFrameBase和wxLog

1. MainFrame重载DoLogRecord()

virtual void DoLogRecord(wxLogLevel level,

const wxString& msg,

const wxLogRecordInfo& info);

1. DoLogRecord中处理log信息

level由wxLogXXX函数定义

msg为wxLogXXX函数字串参数

info由wxLogXXX函数产生，包括代码filename, line,func及logtimestamp和threadId

可在该函数中将log输出到多个目标，如wxTextCtrl及file。

1. MainFrame设置log目标

构造函数中wxLog::SetActiveTarget(this);

可先GetActiveTarget获取之前的log目标，待结束时恢复原log。这样当log目标(此处是MainFrame)被析构时，仍然可以显示log（默认是弹出对话框）。

1. 使用wxLog函数输出log

### 使用单独的Log类

1. 新建LogEntry类，继承自wxLog
2. 重写LogEntry::DoLogRecord，处理log信息
3. MainFrame构造时设置log目标

新建LogEntry实例

设置LogEntry参数(可在构造函数中包含大部分参数)

wxLog::SetActiveTarget设置log目标

1. 使用wxLog函数输出Log
2. MainFrame析构函数设置销毁LogEntry实例

wxLog::SetActiveTarget(NULL)或pOldLog替换当前log目标

销毁LogEntry实例

文件示例:



## 注意事项

### 文本控件的继承及类型转换

根据manual，wxStyledTextCtrl及wxTextCtrl都继承自wxTextEntry。但实际代码中，默认情况下wxUSE\_TEXTCTRL宏被定义，此时wxStyledTextCtrl将继承自wxTextCtrlIface而非wxTextEntry (wx/stc/stc.h)，wxTextCtrlIface则继承自wxTextEntryBase(wx/textctrl.h)。

wxTextEntryBase中的AppendText()(src/common/textentrycmn.cpp)会调用WriteText()(wx/textentry)，而后者是纯虚函数。

wxTextEntry继承自wxTextEntryBase(wx/msw/textctrl.h)。

因此，默认情况，函数传参时wxStyledTextCtrl\*无法自动转换为wxTextEntry\*，如果进行强制转换，将会导致调用WriteText()时由于该函数为纯虚函数而出现wxWidgets Debug Alert错误:

src/common/wincmn.cpp

void wxWindowBase::SetNextHandler(wxEvtHandler \*WXUNUSED(handler)):

wxFAIL\_MSG("wxWindow cannot be part of a wxEvtHandler chain");

或

src/common/object.cpp

wxObjectRefData \*

wxObject::CloneRefData(const wxObjectRefData \* WXUNUSED(data)) const:

wxFAIL\_MSG( wxT("CloneRefData() must be overridden if called!") );

* **形参定义为wxTextEntryBase\***

为了通用性，可以将需要同时以wxStyledTextCtrl\*或wxTextCtrl\*为参数的函数形参定义为wxTextEntryBase\*，编译器会自动进行动态绑定。

* **继承类转换为基类不需要进行强制转换**

如果出现提示继承类无法转换为基类的编译错误，一般是因为条件编译导致基类更改。此时应该仔细检查一下源码。

### log文件换行符

由于Windows，Mac及Unix等系统采用不同的换行符，因此写入log文件时要注意区分。有两种解决方法：

* **使用wxTextFile::GetEOL()**

src/common/ textbuf.cpp:

* **使用宏条件编译**

src/common/ textbuf.cpp:

#if defined(\_\_WINDOWS\_\_) || defined(\_\_DOS\_\_)

wxTextFileType\_Dos;

#elif defined(\_\_UNIX\_\_)

wxTextFileType\_Unix;

#elif defined(\_\_OS2\_\_)

wxTextFileType\_Os2;

#else

wxTextFileType\_None;

#error "wxTextBuffer: unsupported platform."

#endif

# 使用拖放Drag & Drop

## 概览

wxWidgets可以接受文件或文本拖放(Drag and Drop, DND)。详细信息参考手册（%WXWIN%\docs\htmlhelp\wx.chm）Drag and Drop Overview及Clipboard and Drag & Drop

## 示例: Drop File

以文件拖放为例演示窗口如何接受一个拖放的文件。窗口wxWindow具有成员SetDropTarget，可以注册拖放对象类。当有文件拖放到该窗口时，会自动调用拖放对象类的OnDropFiles函数进行处理。

### 新建拖放对象类

wxFileDropTarget继承类并改写OnDropFiles()。OnDropFiles为主要处理函数。

class MainFrame;

class MyFileDropTarget: public wxFileDropTarget

{

public:

MyFileDropTarget(MainFrame\* pFrame)

: wxFileDropTarget()

, m\_pFrame(pFrame)

{

}

protected:

// main frame pointer

MainFrame\* m\_pFrame;

virtual bool OnDropFiles (wxCoord x, wxCoord y, const wxArrayString &filenames)

{

if (filenames.GetCount())

{

// TODO: get dropped files and process

// m\_pFrame may be used.

return true;

}

return false;

}

};

### 在窗口中注册拖放对象

MainFrame构造函数中注册拖放对象。当文件被拖放到MainFrame窗口时，会自动调用拖放对象的OnDropFiles函数。

SetDropTarget(new MyFileDropTarget(this));

## 注意事项

# Samples示例项目

系统内置了80多个示例项目（wxWidgets 3.0.2），包括各种较复杂的控件的使用等。建议在碰到问题是查阅

示例项目包括：

Accessibility Sample

Animation Sample

Art Provider Sample ：使用自定义图标及wxArtProvider获取内置图标，示例项目可浏览所有内置图标（wxART\_宏定义）

Advanced User Interface Sample

Calendar Sample

Caret Sample

Collapsible Pane Sample

Combo Sample

Configuration Sample

Console Program Sample

Controls Sample

wxDataViewCtrl Sample

Clipboard Sample

Debug Reporter Sample

Dialogs Sample

Dialup Sample

Display Sample

Drag & Drop Sample

Document/View Sample

Drag Image Sample

Drawing Sample

Erase Event Sample

Event Sample

Exception Sample

External Program Execution Sample

Flash Sample

Font Sample

Grid Sample

Help Sample

HTML Sample

HTML List Box Sample

Image Sample

Internationalization Sample

Connection Sample

Joystick Sample

Key Event Sample

Layout Sample

List Control Sample

MDI Sample

Mediaplayer Sample

Memory Checking Sample

Menu Sample

MFC Sample

Minimal Sample

Native Windows Dialog Sample

Notebook Sample

OLE Automation Sample

OpenGL Sample

Owner-drawn Sample

Popup Transient Window Sample

Power Management Sample

Printing Sample

wxPropertyGrid Sample

Registry Sample

Render Sample

wxRichTextCtrl Sample

Sash Sample

Scroll Window Sample

Shaped Window Sample

Sockets Sample

Sound Sample

Splash Screen Sample

Splitter Window Sample

Status Bar Sample

wxStyledTextCtrl Sample

SVG Sample

Tab Order Sample

Task Bar Icon Sample

Text Sample

Thread Sample

Tool Bar Sample

wxTreeCtrl Sample

Types Sample

wxUIActionSimulator Sample

Validator Sample

VScrolled Window Sample

wxWebView Sample

Widgets Sample

Wizard Sample

wxWrapSizer Sample

XRC Sample