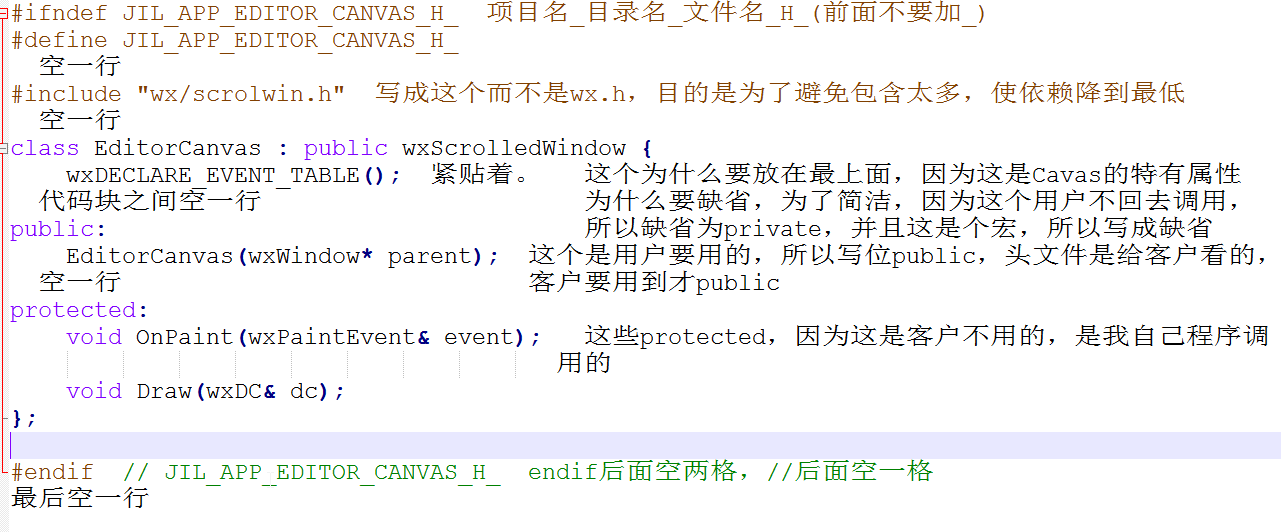
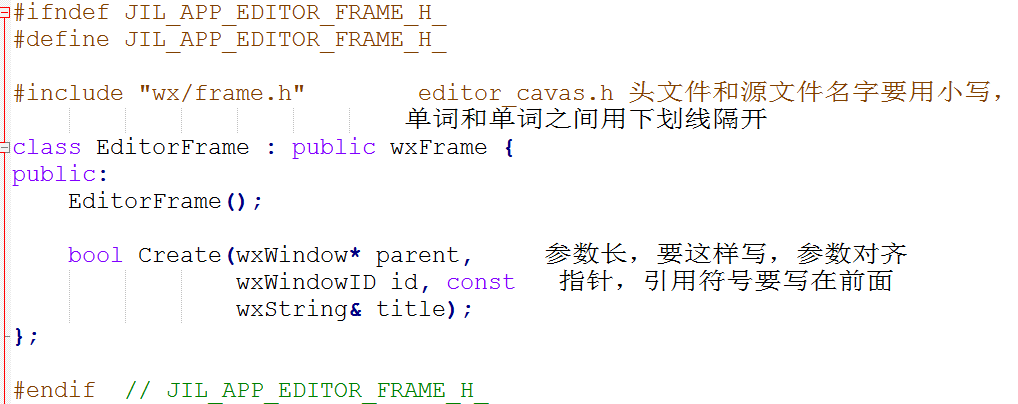
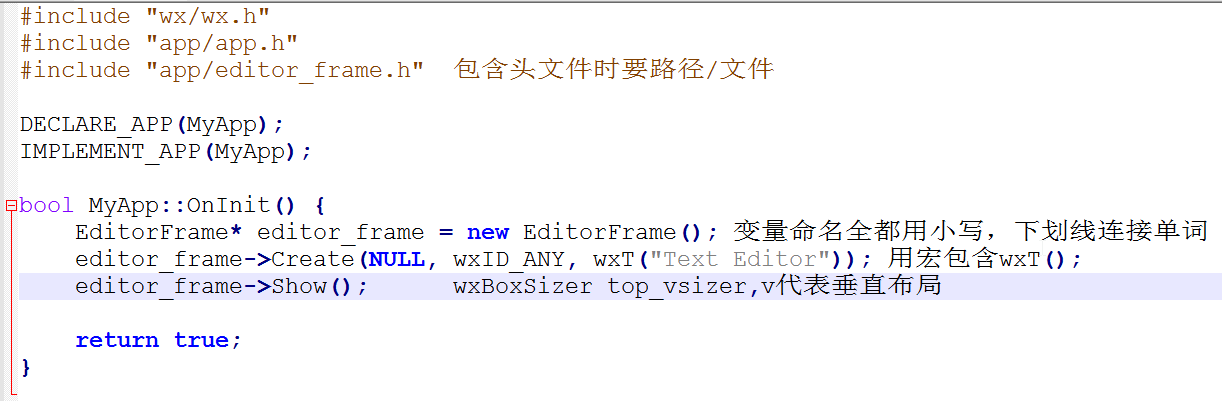
# （一）2016.4.29

1.代码规范

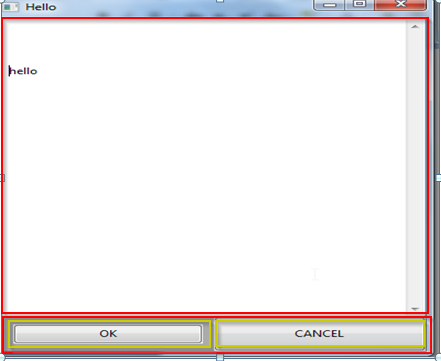








2.wxBoxSizer



bool MyApp::OnInit() {

MyFrame\* myFrame = new MyFrame(wxT("Hello"));

wxTextCtrl\* text = new wxTextCtrl(myFrame, wxID\_ANY, "hello", wxDefaultPosition, wxSize(480, 320), wxTE\_MULTILINE);

wxButton\* leftButton = new wxButton(myFrame, wxID\_OK, "OK");

wxButton\* rightButton = new wxButton(myFrame, wxID\_CANCEL, "CANCEL");

wxBoxSizer\* topSizer = new wxBoxSizer(wxVERTICAL);

wxBoxSizer\* buttonSizer = new wxBoxSizer(wxHORIZONTAL);

//整个Frame用了一个垂直布局，垂直布局里面分为两块，上边是个textCtrl，下面是个水平布局

//垂直布局中，EXPAND是水平填充；水平不居中，EXPAND是垂直填充

//proportion是比例，在一个布局里面的比例，两个按钮，2:1，是在buttonSizer中的比例，两个按钮的整体大小是buttonSizer包裹的

topSizer->Add(text, 1, wxEXPAND | wxALL, 0);

buttonSizer->Add(leftButton, 2, wxALL, 10);

buttonSizer->Add(rightButton, 1, wxEXPAND );

topSizer->Add(buttonSizer, 0, wxEXPAND);

myFrame->SetSizer(topSizer);

myFrame->Show(true);

return true;

}

# （二）2016.6.22

1. operator= 要处理自我赋值if(this == &rhs) return \*this

2. CommandManager放到TextFile里面，不要放在view层，文件属性是否改变，要与command 关联，判断其现在是否改变了。存放command用list更好。list里面的指针所指的堆对象delete以后，list要clear()清掉里面的指针，否则里面的指针成为了野指针。

3.判断一个容器是否为空用Empty()更可读直观，比用Size() == 0这样的代码更好。

4.命名：config一般就泛指整个配置。configs是错误的，config是configure的缩写。如果是 options，preferences 这样的可以使用。

5.代码要空行（成员函数等），更直观。

6.头文件里面要放注释，因为头文件是用来给别人看的，源文件是自己看的，所以头文件要写注释。

7.错误：Config(const std::string file\_path); const 要引用，否则没有意义，因为改变的是临时变量。

8.config读取文件时，parse后就丢掉该文件，也就是关闭。

9.返回值不要用123等数字来标识错误，可以定义个枚举之类的来返回，更加直观。

10.map插入的时候，不要用config\_map.insert(std::pair <std::string, std::string>(key, value)); 使用make\_pair()： config\_map.insert(std::make\_pair(key, value)); 最好使用index操作符，而不是用insert，更加直观，config\_map[key] = value;

11.app中函数要调用基类的oninit 和onexit等.

12.一个函数返回引用,要用引用去接,如果用一个变量去接,就进行了拷贝.所以在调用函数时要注意返回类型。

13.构造函数初始化列表要风格统一。

14.函数参数要注意\*和&放的位置。

15.不要包含无用的include

16.错误 const wxString TextFile::GetEOL(TextFileType type)返回的是临时变量没有必要前面加const，临时变量即使改变了也不会影响原来的值。

17.一个函数只做一件事，如果里面有多件事，进行函数分割。

18. wxString::const\_iterator start = text.begin();

wxString::const\_iterator end = text.end();

for (wxString::const\_iterator p = start; p != end; p++)

类似这样的变量start和end要尽量放在for里面，放在外面，别的地方可能会用到。

1. const TextFileType& GetLineType()const返回TextFileType没有必要返回const的引用，因为TextFileType是枚举类型，也就是int值，这些基本类型没必要返回const.返回引用，除非是要直接操作该值。

20.if逻辑要写清晰，为了清晰可以把里面的判断提出作为else来减少判断嵌套。

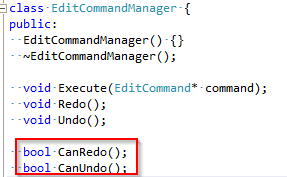
21.使wxPanel获取键盘字符输入，需要设置其style为WXWANTS\_CHARS.

22.scrollbars 改变了virtualsize之后（setvirtualsize）,之后要调用adjustScrollsize。 setvirtualsize是设置，adjustScrollsize是更新。

1. 使用友元类来实现delegate，目的是在一个对象的成员函数里面访问一个类的对象的私有成员函数，类似于观察者模式的效果。
2. Onpaint里面一定要定义wxPaintDC之类的DC，不能定义在里面的函数里面，会导致刷新问题，例如caret被遮挡后会消失一半。

# （三）2016.7.21

1. 类里面只读函数要加const。错误代码：

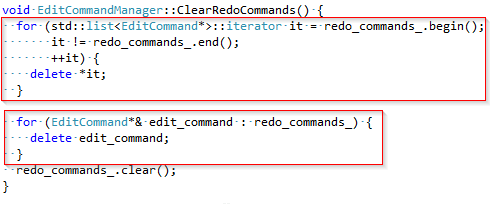


修改为：



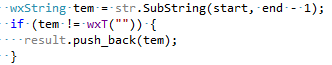
2.wxSplit这个全局函数是wxWidgets库提供用来分割字符串的。

3.可以不使用迭代器进行遍历，使用C++11的for代码更简单。

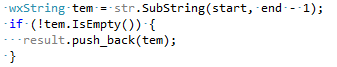


1. for循环里面要使用引用，例如for(Student& stu : stus){}。
2. tem是wxString类型，既然是一个字符串类就有判断tem是否为空的函数例如IsEmpty，而不是直接tem == wxT(“”)这样去比较，这样做代码丑陋，影响效率。

错误写法：



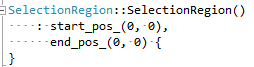
正确写法：



就像上一节提到的判断一个容器是否为空用empty()，而不是用size() == 0。

1. wxPoint初始化为(0, 0)，所以以下代码没有必要使用有参构造函数写到初始化列表里面。

错误代码：



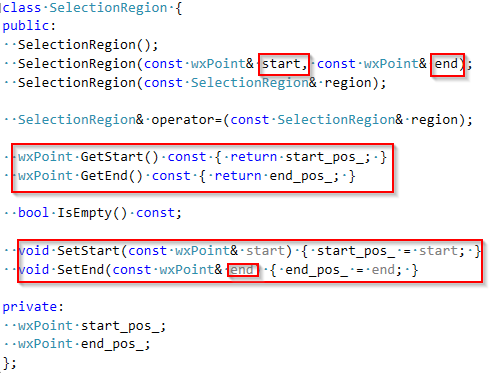
正确代码：



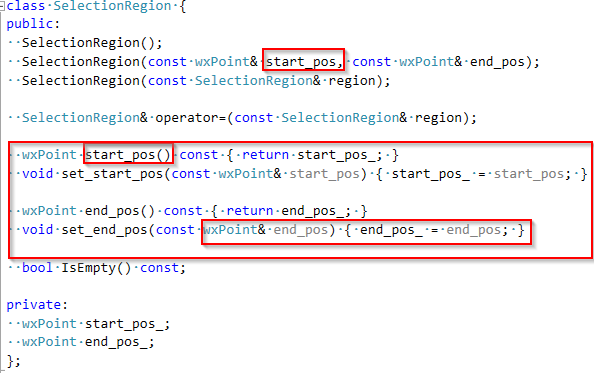
1. 以下代码共有三处错误：
2. 一致性问题，以下函数的参数最好写的和成员变量名字一样，函数参数多起个名字难以起那么多名字，也缺乏一致性，使代码可读性下降，可能出现歧义。
3. 代码排版问题，Get和Set函数要放在一起，一个东西分开放看起来很奇怪。
4. Get和Set要严格按照Google C++标准的存取函数来写，保持代码一致性以及消除歧义。例如GetStart()这个名字就不好，也许GetStart并不是获得start\_pos\_，出现了歧义，所以最好的存取函数明就是同变量名一样。

存取函数命名用小写，取：同变量名。存：值函数名为set\_varname。

错误代码：



正确代码：

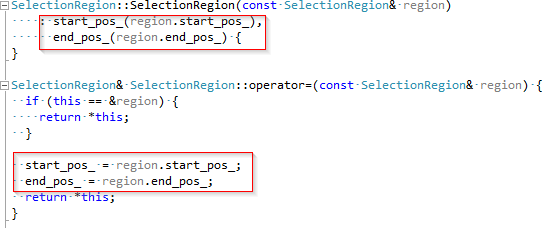


1. C++成员函数可以访问同类参数对象的私有变量，

错误代码：

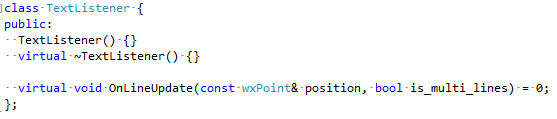


正确代码：



1. 写抽象类时，默认构造函数没什么用，可以不用写，这样看起来更像一个接口。

错误代码：

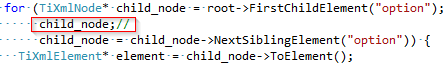


正确代码：

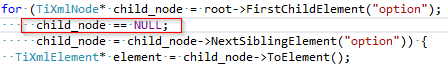
去掉构造函数。

1. Google C++ Style规定条件语句判断指针时，要用ptr == NULL，不能直接写成if(ptr)，这样做的目的是代码可读性更高，消除二义性，可以直接看出来ptr是一个指针，而不是布尔变量。

错误代码：

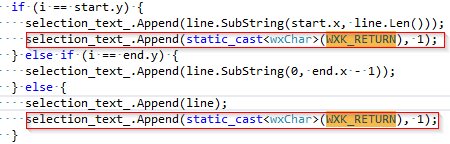


正确代码：

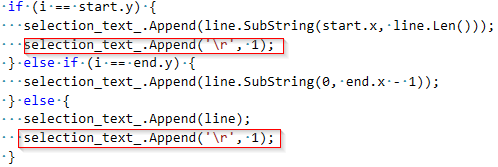


1. Ctrl+k+0，VS打开头文件。
2. WXK\_RETURN是一个全局定义的关于键盘的枚举值，直接把它当作字符来处理是不合理的，代码阅读造成了困扰，因为这个明显代表的是key code，却用到了字符串处理里面，这是一个整型值，字符串处理也需要隐式或者显式地转换，所以在什么地方就用什么样的值，字符串处理中就是用’\r’，键盘处理就用WXK\_RETURN。

错误代码：

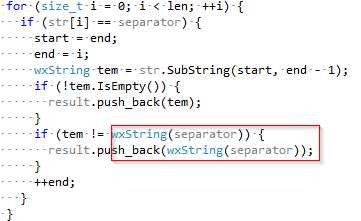


正确代码：

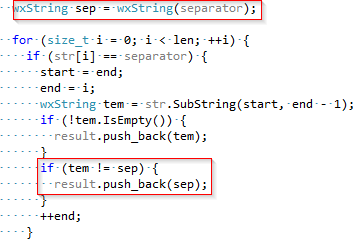


1. 在循环中不断地重复构造对象存在效率问题。例如在SplitString中，如果待分割的字符串很长，那么separator将会不断地构造这个字符串对象，严重影响效率，可以在循环外定义。

问题代码：



改善代码：



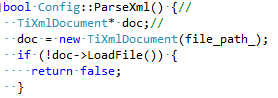
1. 类似SplitString这样的便利函数，可以用static来限制也可以放在utils文件里面作为工具函数供整个项目使用。把这样的函数放在一个类里面做私有成员函数是不合适的，因为这个函数有这个类并没有直接的关系，并不是这个类的行为方法。
2. 单元测试要写，来测试例如SplitString这样的一个算法函数，如果这个函数修改后，也可以方便的用原来的单元测试case来测试其准确性。

Gtest教程：<http://blog.csdn.net/aladdina/article/details/5508906>

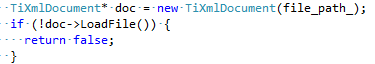
16.SelectionRegion这个类类似wxRect有起始点和终止点，在这个类里面应该去检查一下start和end哪个在前哪个在后。当然外部也可以进行对其的检查，但是这个是这个类的特性，应该自己进行检查，这样这个类的功能属性，独立性更加完整。

17.SelectionRegion这个类开区间闭区间的问题，如果是闭区间如何判断为空？因为闭区间两个点相等时[0,0]，这样恰好包含0这个点。所以肯定是选用的开区间。

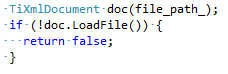
18.以下代码存在两处问题：



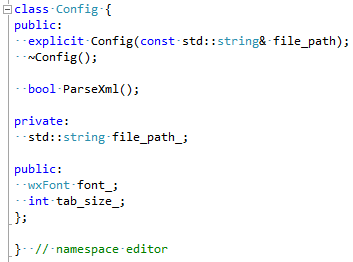
1. 代码风格：定义指针后，然后指向new出来的对象，这种风格非C非C++，应该以下的书写风格。



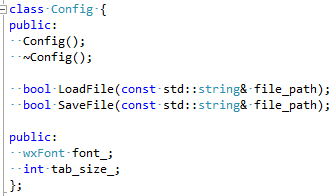
(B)堆对象问题：为什么要将TiXmlDocument这个对象new出来，后面还要delete。最初的想法是，TinyXml这个解析器会将这个文件缓存到内存，所以为了不让其栈内存溢出所以定义了new对象。这种想法十分错误。TiXmlDocument是一个类，就像String这样的类一样，它们内部维护了一个指针来指向堆内存，所以即使缓存也是讲数据存到堆内存，缓存数据的大小堆这个对象的大小是没有影响的。可以用sizeof来获得这个对象的大小。既然这是一个类，内部就有办法去维护数据的缓存。所以以前考虑将vector，list，string定义为堆对象还是栈对象是没有必要的，因为它们存的数据是类里面来实现维护的。



19.以下代码是一个配置类，目的是将从配置文件里面读取数据存到自己的属性里面。配置文件读取后直接丢掉，没必要保存成一个对象。以下写法同样没有必要去保存一个文件路径名。因为文件的读取和保存不是到一个文件里面，所以保存路径后，如果要保存到其他文件里面还需要重新构造对象从而传进去路径才能保存。这样做是没有必要的，这是一个配置类，最重要的属性就是配置内容了，而一个路径不是它关心的，所以这个数据模型没有必要将路径作为其私有成员。所以在载入和保存两个方法时传入路径就可以了，提高了灵活度。

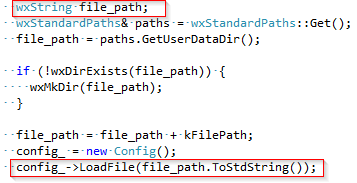


改善代码：

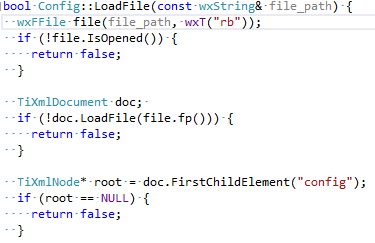


1. 用FILE\*来跳过保存文件路径的字符串互相转换导致的编码问题。

现在存在这么一个问题，如下代码。file\_path是一个wxString，而config类的LoadFile接受的是std::string。所以需要调用wxString的ToStdString来进行转换，但是这里面存在一个问题，也就是编码的问题，ToStdString还是ToStdWString这是一个问题，可能导致乱码问题。所以一定要绕开这个问题。LoadFile里面需要std::string的参数因为里面的TinyXml解析器的载入文件需要传进一个std::string类型的路径，所以避开这个问题的方法，就是找TiXmlDocument其他的构造方法，可以发现可以传进FILE\*作为读取文件的构造方法。所以在wxWidgets开发中，首先用wxString存储路径，然后用wxFFile来打开文件，从而获得FILE\*，然后将FILE\*传给TiXmlDocument等对象来进行对象的构造等，就有效的绕开了不同的字符串相互转化的问题。

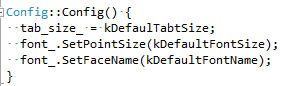


改善代码：

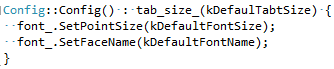


1. 构造函数要写初始化列表

错误代码：



改善代码：



22.CommandManager没必要存在，CommandManager放在TextFile里面，在TextScrollWindow里面通过TextFile来获取到CommandManager，然后再用CommandManager来执行Execute()，Redo()，Undo()，其实可以直接在TextScrollWindow里面执行，这样就减少了一个不必要的接口。

# （四）2016.7.26

1. 注释要写完整的句子，首字母大写，句号，该大写的字母大写，例如Carestream，CSI等。



1. 变量要初始化。例如 int a = 0;
2. TODO的写法，todo大写后面空格然后写名字，只写名字不写姓，然后冒号空格。

2016-07-26 15_38_58-LearnBoost - Microsoft Visual Studio2016-07-26 15_38_15-LearnBoost - Microsoft Visual Studio

# （五）2016.9.14

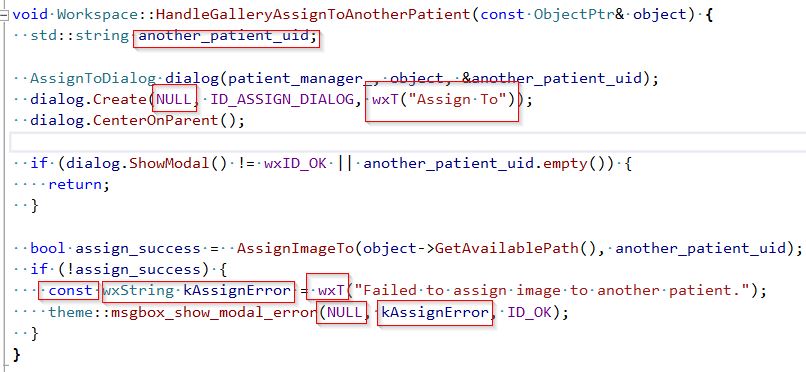
1.给dialog指定一个父窗口。

2.有时候在小的作用域里面，没必要用const，命名也无需定义为有真实意义的。

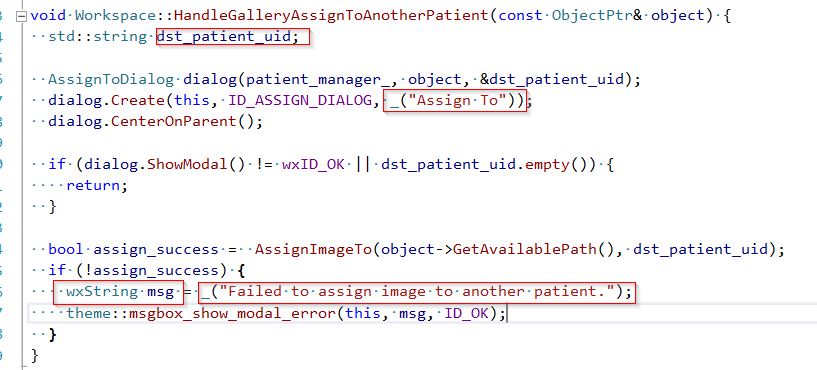
3.在界面上的显示的字符串要经过翻译，所用应该用\_(""),而wxT("")。普通的逻辑处理，例如路径什么的是不需要翻译的。

4.常用的缩写dst，src。

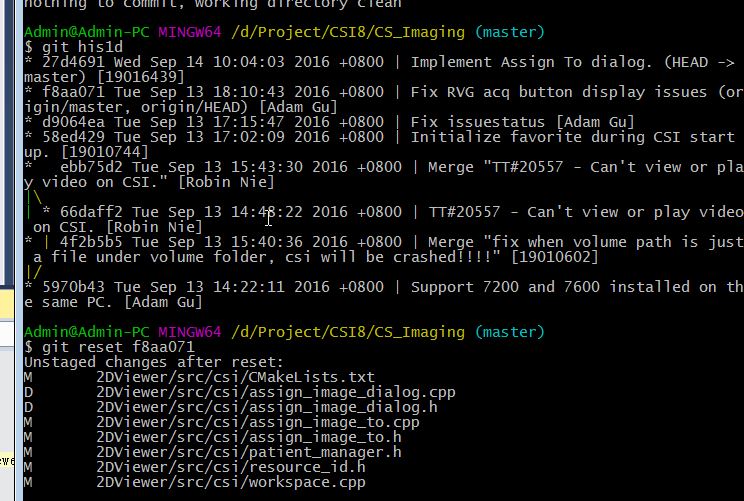
错误代码：



改正代码：

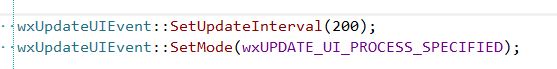


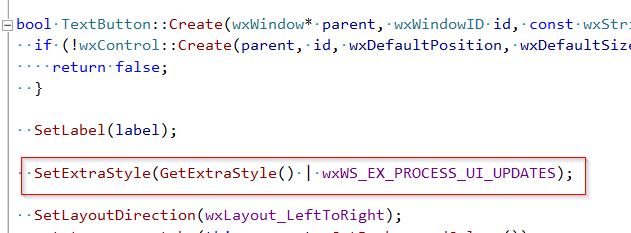
1. git返回到commit之前的状态。



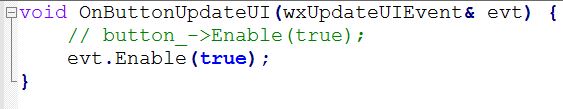
# （五）2016.9.20

1. wxUpdateUIEvent 可以设置时间间隔等。自定义控件如果要使用这个特性，需要设置Style：wxWS\_EX\_PROCESS\_UI\_UPDATES。





1. wxUpdateUIEvent的处理函数中，可以使用event.Enable(true);而避免了获取控件来进行Enable。



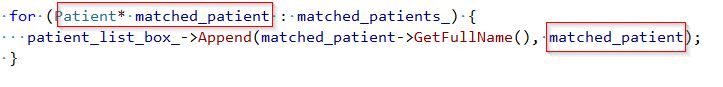
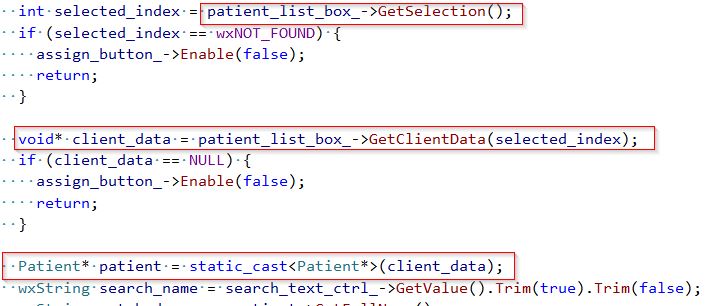
1. 比较字符串可以使用IsSameAs(const wxString& s, bool caseSensitive=true)。



1. csdm用的是UTF-8，wxString需要转换。

2016-09-20 17_10_26-CSI - Microsoft Visual Studio

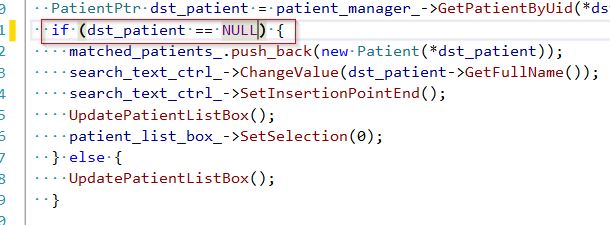
5.继承wxItemContainer的控件是可以携带数据的。例如wxListBox。



# （五）2016.9.23

1. 智能指针判断不需要写成ptr==NULL。

错误代码：



修改：

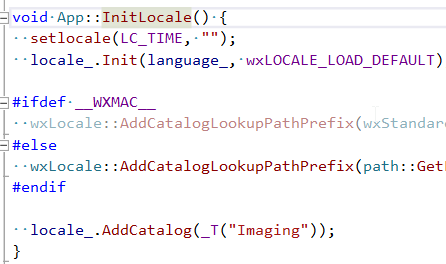
2016-09-23 16_24_19-CSI - Microsoft Visual Studio

2.Connect绑定事件要放在Create的最后，防止出现问题。



3.static const wxString KString(\_T("ABV"));

\_T是一个翻译函数。程序一开始的时候会在app中进行本地化设置，然后在以后调用\_T翻译函数时，进行对应的翻译。但是如果定义为static后，static变量会在main函数调用之前创建static变量，也就是main函数之前调用\_T,但是此时本地化还没有初始化，所以写成\_T是没有起到作用的。所以要写成#define。

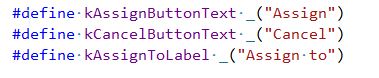


2016-09-26 09_08_05-CSI - Microsoft Visual Studio

错误代码：

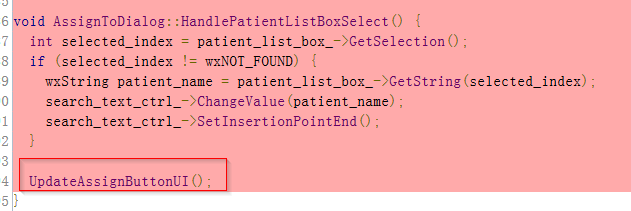
2016-09-23 16_23_53-CSI - Microsoft Visual Studio

修改：

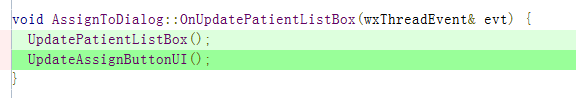


4. 更新病人列表和更新按钮是两个不同的行为，所以将更新按钮嵌入到更新病人列表是不合理的。

错误代码：

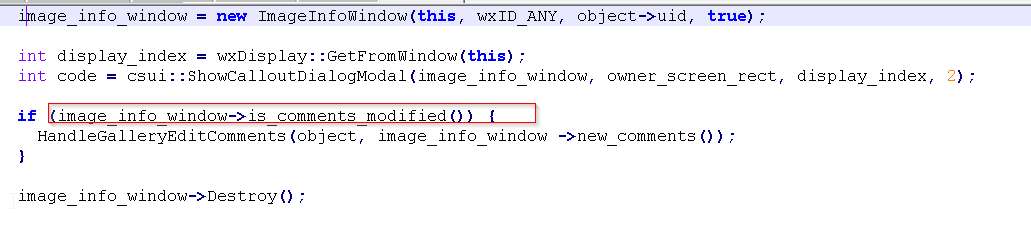


修改：



# （六）2016.10.20

1. New出来一个dialog，虽然窗口关闭了，但是在delete之前，这个对象还是存在的，也就是里面的成员还是有值的，也就是说窗口关闭并不等于dialog这个对象被delete了。

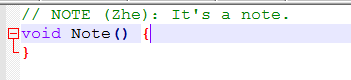


# （七）2016.12.8

1. 绘图一定要被动地去画，也就是在OnPaint中，不要用ClientDc等去主动地画，会出现问题。
2. Refresh()之后调用Update()可以立即刷新，进行OnPatint，否则会进入消息队列，等待刷新。

# （八）2016.12.15

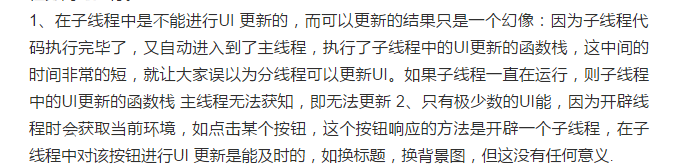
1. 代码注释note要大写。Eg:



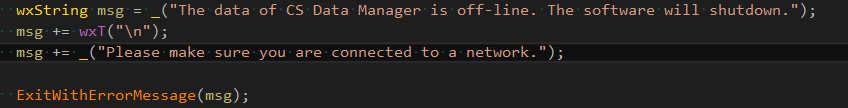
1. 只有主线程（UI线程）可以刷新UI，也就是做跟界面相关的动作。子线程（工作线程）或者说非UI线程不能做界面相关的动作，例如弹出个对话框之类的。在子线程弹出对话框是会导致crash的。解决主线程和工作线程之间通信的办法，就是在子线程中向消息队列发消息。

<http://blog.csdn.net/hebezai/article/details/10383323>

<http://www.jianshu.com/p/493590fa8e5b>



1. ProcessEvent的作用是便利消息队列，然后执行该消息响应函数，是直接调用，并不是把执行权立刻交给主线程去运行。在子线程中依然是子线程中处理。
2. 需要翻译的字符串，有换行符时，要用wxT()来写，因为换行符不好翻译。



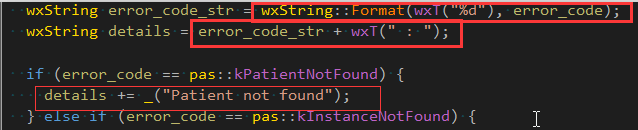
# （九）2016.12.27

1. fatal用来表示不可恢复的，重大的，崩溃性的错误。

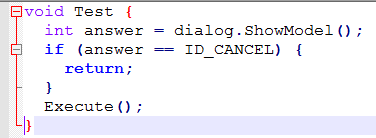


1. 并不是所有的字符串都需要翻译\_(“”)，单个的字母，数字，缩写（例如FMS），标点符号，

换行符(\n\r)等不用翻译，用wxT()即可，直接显示给用户看的，且有实际意义的才需要翻译。



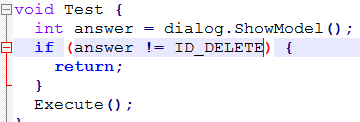
1. dialog退出时返回值的判断，最好不要用”==”什么就返回，否则就继续执行这样的逻辑。容易出错，例如，



出错的原因很明显，dialog非确定退出时不一定返回的ID\_CANCEL，也可能是ID\_NO等等。

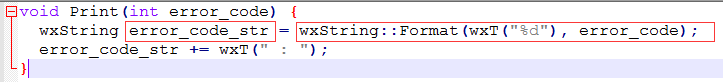
并且dialog确定时返回的也不一定是ID\_OK，也可能是ID\_YES等。所以这个地方一定要

根据对话框的按钮的实际情况来判断返回值。以下的写法更好，



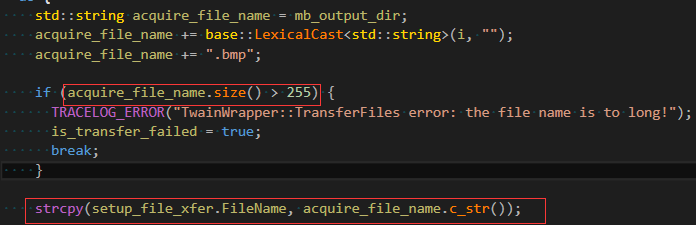
1. 为了避免命名重复可以利用变量类型来作为区分的后缀。例如字符串可以加”str”。

将其他类型的变量转为wxString时，可以用函数wxString::Format。



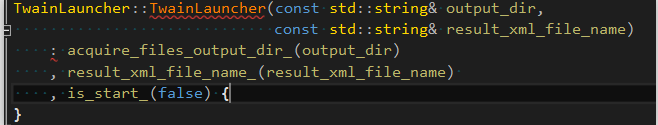
1. strcpy(buf, src)。Strcpy函数会把字符串的结尾’\0’一并复制到buf中，所以buf除了有效字符之外，还要留有一个字节存放’\0’。也就是说在strcpy之前，一定要检查buf的空间是否足够存放目标字符串，如果不判断，就进行存放是一个很危险的动作，会访问非法地址。并且std::string.size()函数返回的有效字符的长度，不包含’\0’.

所以应该如下写，

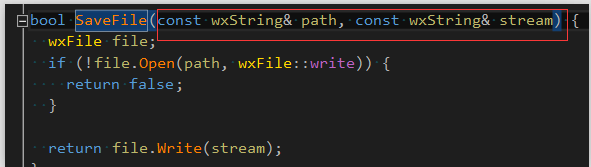


6.CSI中的文件名，路径等操作都用的是wxString，并且程序中除非与外部接口相关与业务很相关的用std::string。用wxString还是很方便的，因为它提供了很多库函数，并且可以很方便地转化为其他类型其他编码的字符串。

错误写法：

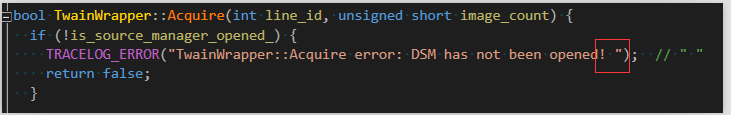


正确写法：

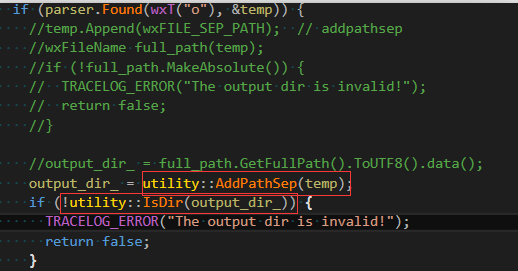


1. 不要随意加空格，即使在log或者注释里面，会显得很不专业。

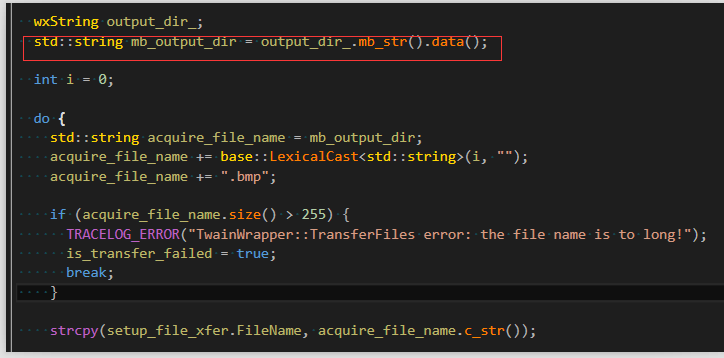
错误代码：



1. 写程序时要加一些注释，一些逻辑，算法的注释，说明。否则过一段时间，就看不懂了。
2. 处理一个路径要判断下它是否已经存在”\\”了。可以用CSI中的函数进行判断。

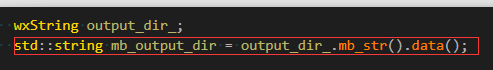


1. 不要在循环里面做过多重复的事，有些可以一步完成的事最好放到循环外面去做，以此来提高效率。



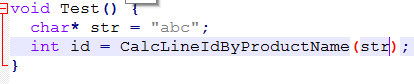
例如上述代码，mb\_output\_dir，可以在外部做，不需要去循环重复的做一件事，则要注意将其放到循环外面去做。

1. windows里面定义char，一般默认为多字节char，所以用wxString进行转化时，要转为多字节char。可以用函数mb\_str()。例如

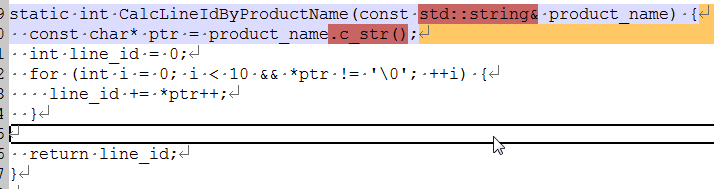


wxString通过mb\_str().data()，转化为多字节char\*赋值给std::string ,则该std::string里面存放的是多字节的char类型。并且在后面的代码mb\_output\_dir += “.bmp”。”.bmp”默认的还是多字节的，因为在windows变成里面其默认的就是多字节的，如果用宽字节，则用\_T(“”)， L””，等来修饰。

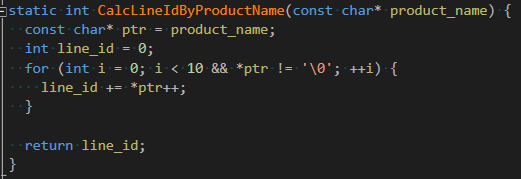
1. 进行函数参数类型选择时，可以根据传进来的实参类型进行选择，可以避免对象构造的可以尽量避免，可以提高效率。例如如下的例子，实参为char\*，函数参数用std::string则会导致一次对象的构造，是没用的，所以函数参数选择char\*更好。



错误代码：

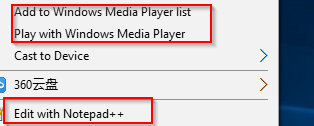


正确代码：



# （十）2017.3.26

1. 右键菜单大小写要一致，单词首字母大写，介词小写。例如“Assign to another patient”要写为“Assign to Another Patient”。



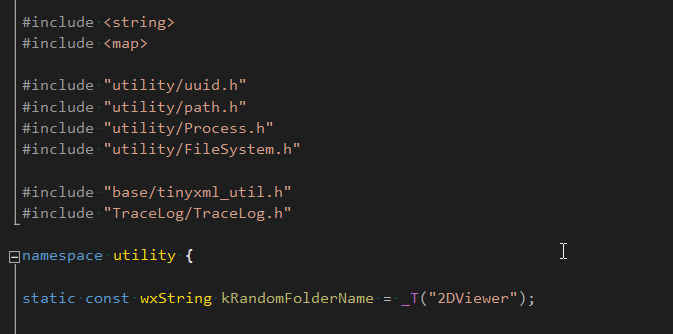
1. success是名词不能当bool量，可以用successful或者failed。Eg：bool successful。
2. 类似utility这样的公共库，要尽量减少依赖，尤其不要依赖对其他主要项目的依赖。
3. utility中的dicom\_img.cpp中产生的临时文件放在了临时目录“2DViewer”中。这是不对的。

Utility是公共库，并不是只属于CSI的，所以放在“2DViewer”中不合适。应该放在自己的文件夹中，例如临时文件夹“utility”“dicom\_img”等。

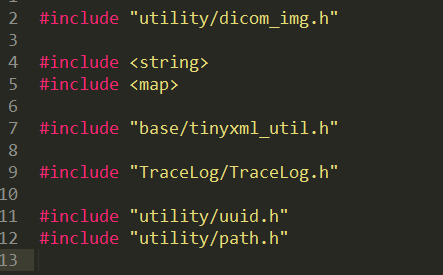
1. base库是纯c++写的库，utility依赖了base库。头文件的摆放位置应该为：

自己的头文件，纯c++库的头文件，第三方库的头文件，自己定义的头文件。

错误代码：

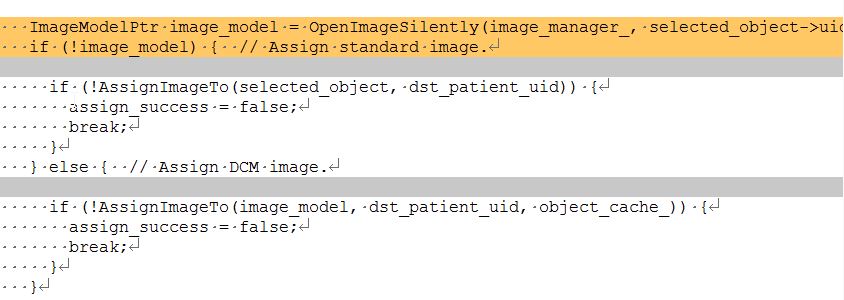


正确代码：

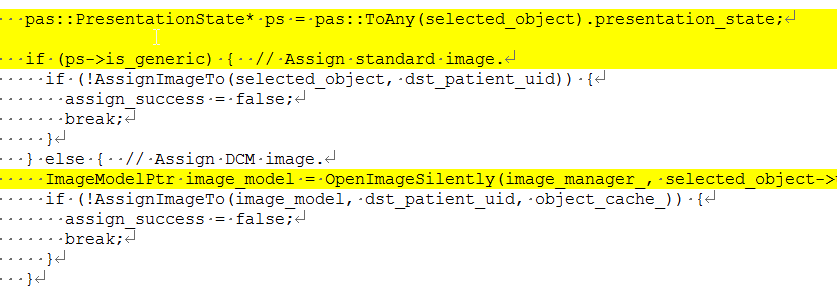


1. 不要用theme::msg\_warning\_dialog。用csi/Common.h中的ShowMsgWarning和ShowMsgError。
2. PS里面有generic和非generic，所以处理PS图片时，要区分generic。

错误代码：

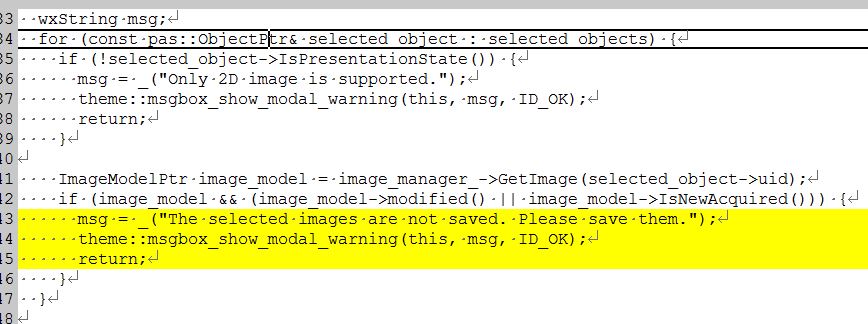


正确代码：

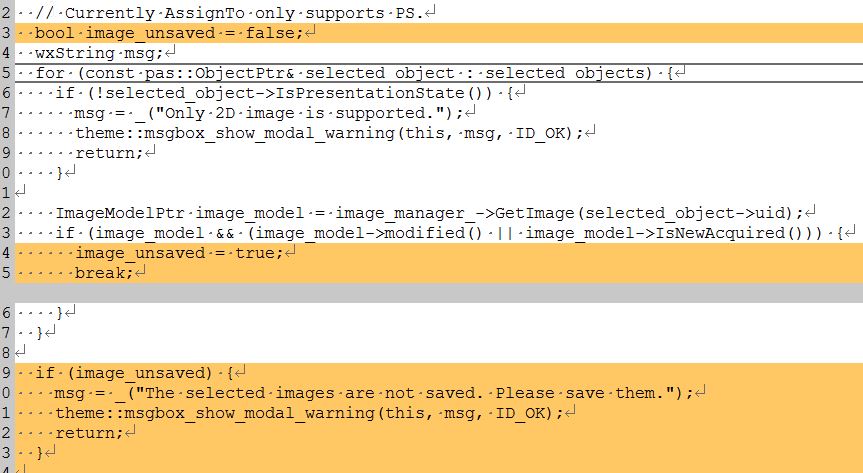


8.循环中，有时候不满足时条件时可以直接return，不必加个标志变量导致冗余。

错误代码：

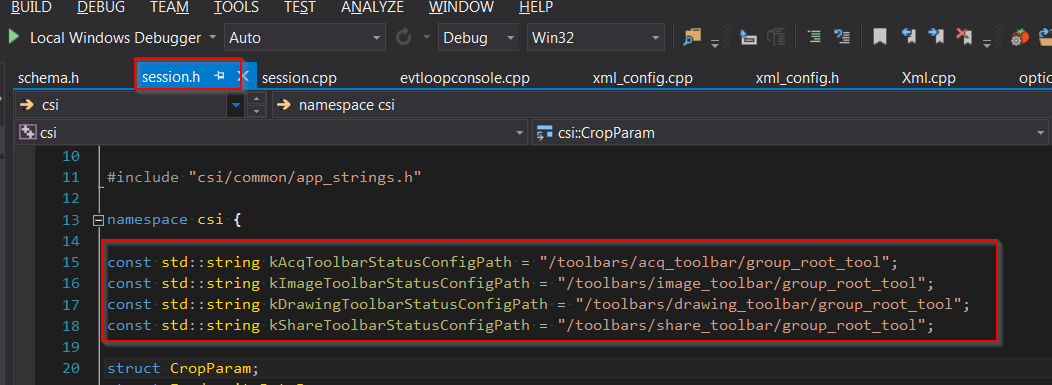


正确代码：



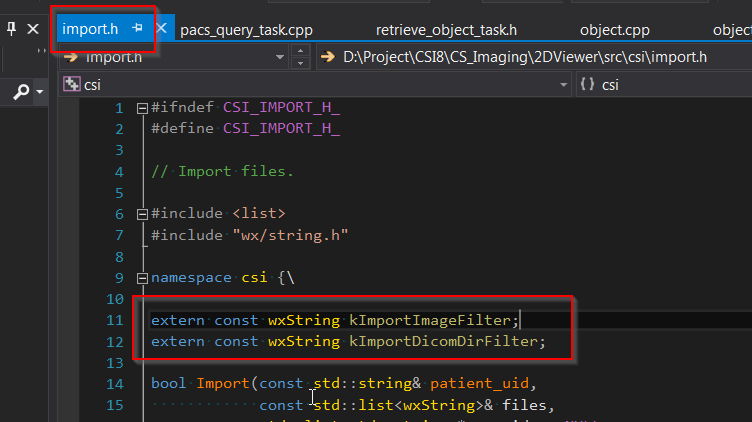
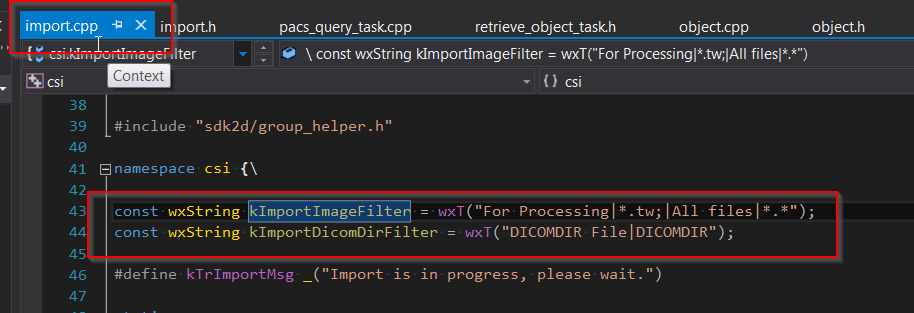
1. 全局静态变量中不能定义在头文件中，其他文件中引用时，要在其他文件中进行extern声明，这是严格的C++语法。

错误代码：



这个代码直接在头文件中定义了全局常量，为了其他文件可以使用，但是这种写法是错误的。不符合c++语法。

正确代码：



正确的写法在源文件中定义常量，也就是全局变量，为了在其他文件中引用，则在头文件中进行extern声明，目的是为了，别的文件直接载入该头文件即可使用。并且直接定义在头文件中是不合适的，因为将具体的定义暴露在了头文件中，别人使用它时，没必要得到具体的定义，暴露出去会导致危险。在头文件中不要随便暴露东西，除非时对别人有用的。

10.要遵循开放关闭原则。例如在使用对话框时，如果要加一个answer按钮“Assign”，然后再该对话框的定义中再加一个选项“ID\_ASSIGN”。这样写是错误的，违反了开放关闭原则，如果这样写，每次别人使用要加一个新选项时，都要修改对话框的定义，向里面添加新的内容，重新编译该定义，这是不对的。其定义应该只有一些不变量，例如“ID\_OK”，“ID\_CANCEL”之类的，其他的自定义应该是可扩展的，也就是不用修改定义，而是提供接口使得他人可以轻松修改！例如MsgBox该对话框就提供了AddOptionLabel对外接口，用户可以使用该接口进行扩展而不用修改其定义。