Template Usage for GUI Programming

1. **Description**

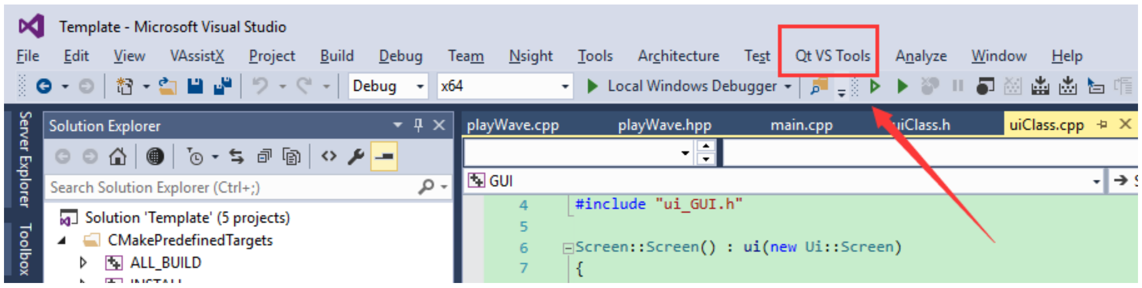
This is a template program organized by cmakelist file that defines all the necessary information of building a Visual Studio project, including configuring QT as a third-party library. With this program, your effort could be focused on code writing without needing to setup the environment manually. Particularly, you can write the functional code on the files in Computation folder, and write the GUI related code on the files in GUI folder. Thus, you only need to learn how to use QT API to implement your GUI functions.

Required Tools: Visual Studio 2015, QT 5.7, CMake (above 2.8 version).

Note that you can also implement your program in any other way you like. This template is just a recommended choice for those who has no idea on GUI programming.

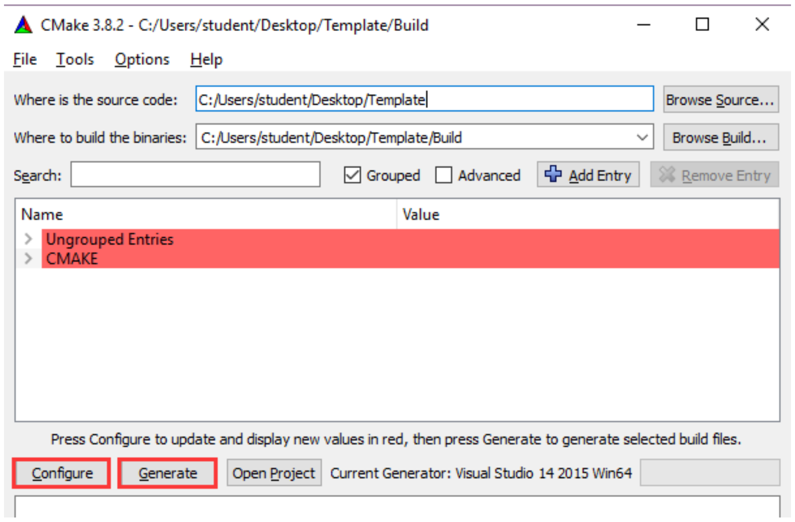
2. **Usage**

- Configure QT in Visual Studio
  Please refer to the tutorial slide: Tutorial_5.pptx.


Template - Microsoft Visual Studio
File Edit View VAssistX Project Build Debug Team Nsight Tools Architecture Test Qt VS Tools Analyze Window Help
Debug x64 Local Windows Debugger
Server Explorer Toolbox
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Template' (5 projects)
CMakePredefinedTargets
ALL_BUILD
playWave.cpp playWave.hpp main.cpp uiClass.h uiClass.cpp
GUI

```
#include "ui_GUI.h"

Screen::Screen() : ui(new Ui::Screen)
{
```

- Generate the visual studio project from source code
  Use CMake (cmake-gui) to generate a project



Where is the source code: C:/Users/student/Desktop/Template

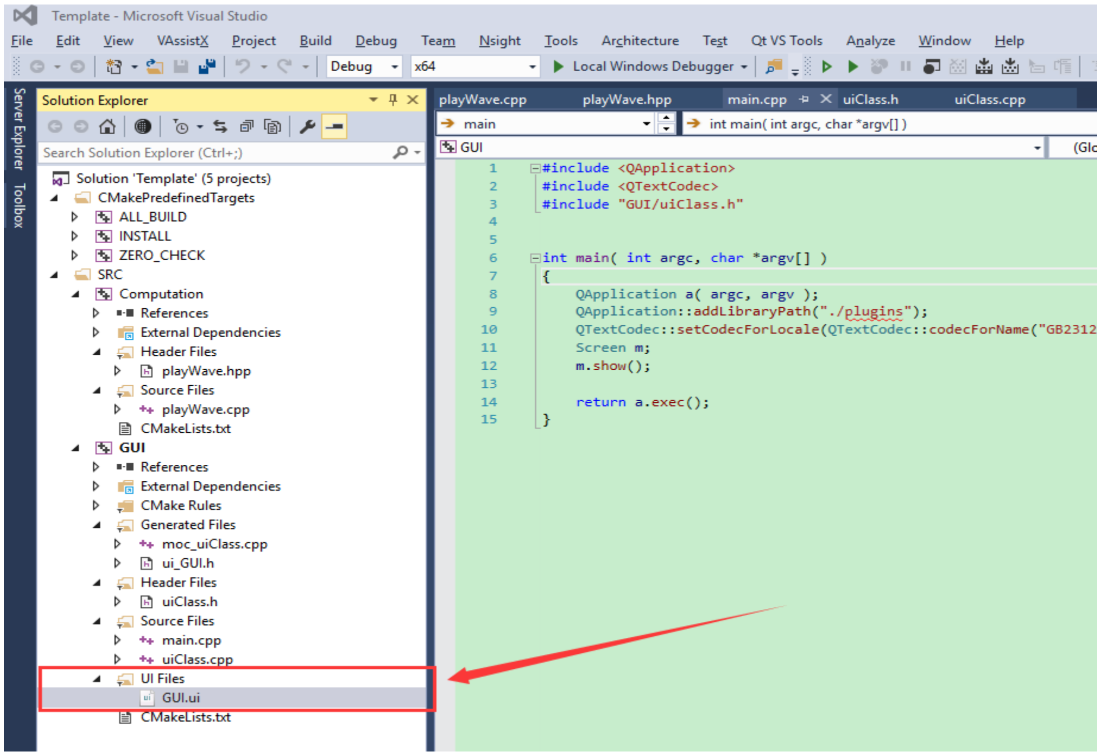
Where to build the binaries: C:/Users/student/Desktop/Template/Build

Search: [✓ Grouped] Advanced

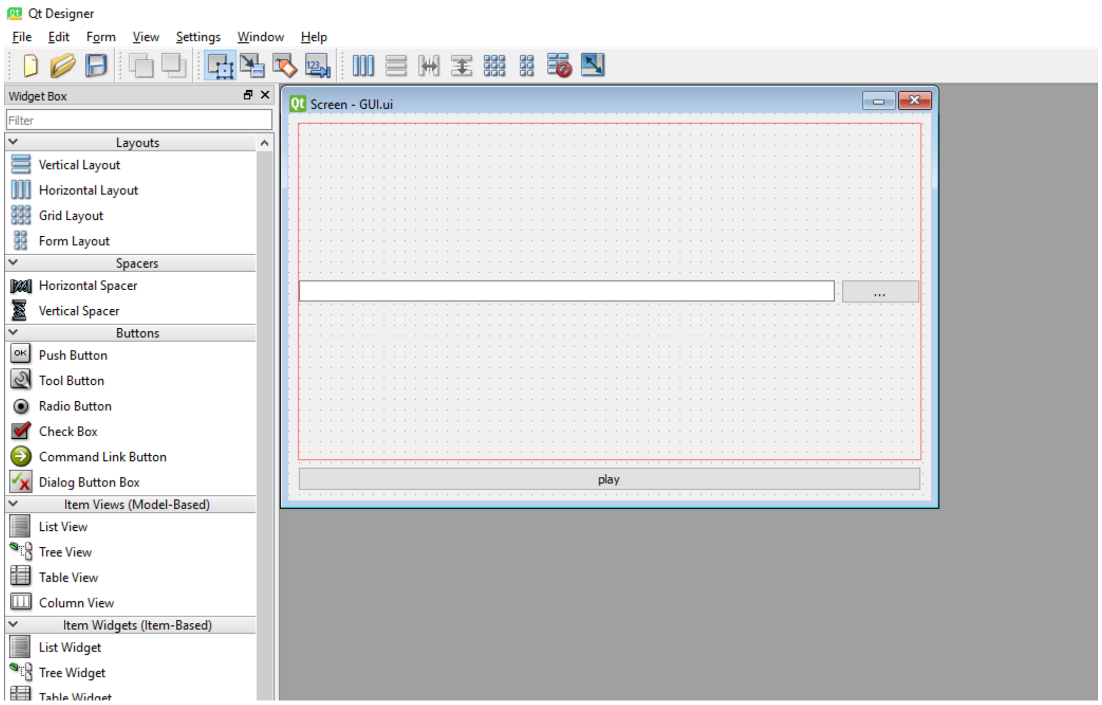
Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 14 2015 Win64

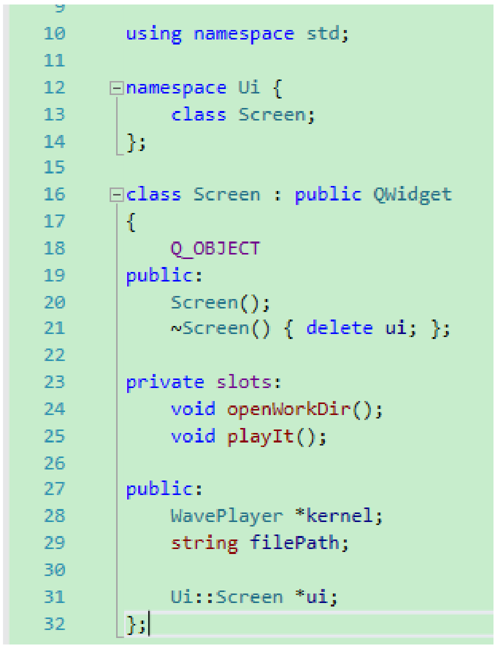
- Design the appearance of GUI
  In the project opened in VS2015, double click on the file: "GUI.ui", and you can design it in the interface of QT Designer.



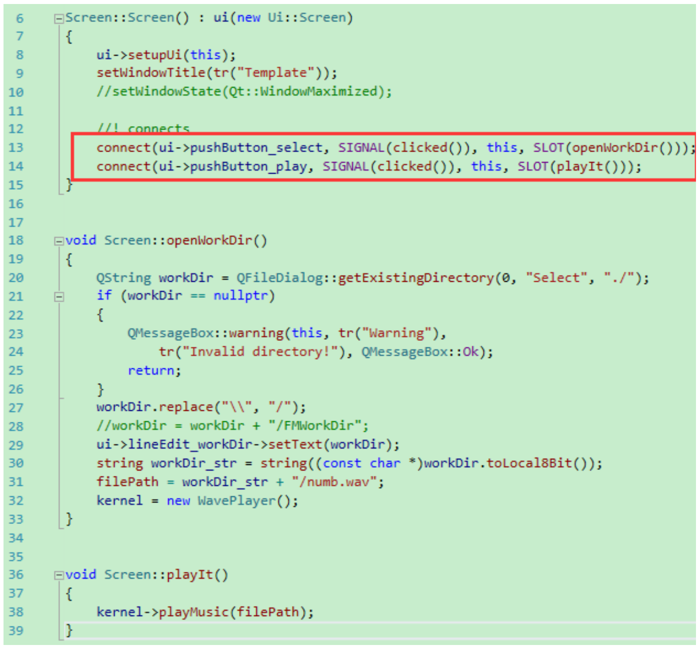
```
#include <QApplication>
#include <QTextCodec>
#include "GUI/uiClass.h"


int main( int argc, char *argv[] )
{
    QApplication a( argc, argv );
    QApplication::addLibraryPath("./plugins");
    QTextCodec::setCodecForLocale(QTextCodec::codecForName("GB2312
    Screen m;
    m.show();

    return a.exec();
}
```



- Implement your GUI code
  Refer to the template code, such as creating connection between signal and slot function, and defining the corresponding slot functions. Of course, you can implement more advanced features. Please try to use Google and QT Assistant (a QT API usage tool)

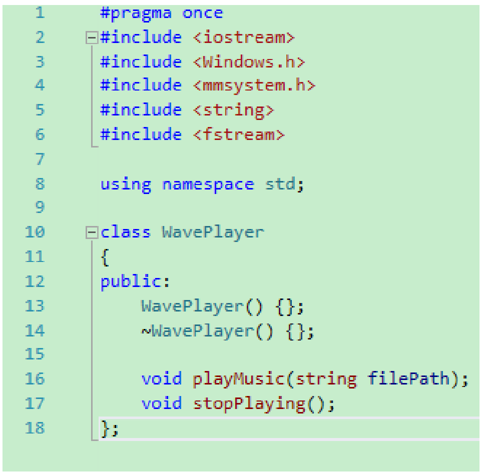


```cpp
using namespace std;

namespace Ui {
    class Screen;
};

class Screen : public QWidget
{
    Q_OBJECT
public:
    Screen();
    ~Screen() { delete ui; };

private slots:
    void openWorkDir();
    void playIt();

public:
    WavePlayer *kernel;
    string filePath;

    Ui::Screen *ui;
};
```

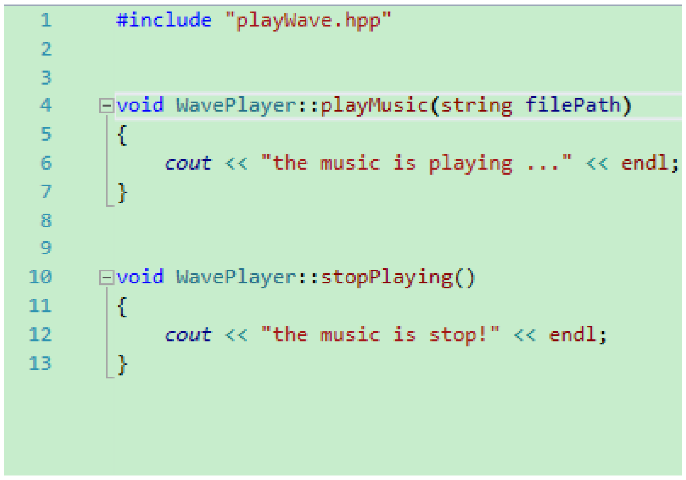


```cpp
Screen::Screen() : ui(new Ui::Screen)
{
    ui->setupUi(this);
    setWindowTitle(tr("Template"));
    //setWindowState(Qt::WindowMaximized);

    //! connects
    connect(ui->pushButton_select, SIGNAL(clicked()), this, SLOT(openWorkDir()));
    connect(ui->pushButton_play, SIGNAL(clicked()), this, SLOT(playIt()));
}


void Screen::openWorkDir()
{
    QString workDir = QFileDialog::getExistingDirectory(0, "Select", "./");
    if (workDir == nullptr)
    {
        QMessageBox::warning(this, tr("Warning"),
            tr("Invalid directory!"), QMessageBox::Ok);
        return;
    }
    workDir.replace("\\", "/");
    //workDir = workDir + "/FMWorkDir";
    ui->lineEdit_workDir->setText(workDir);
    string workDir_str = string((const char *)workDir.toLocal8Bit());
    filePath = workDir_str + "/numb.wav";
    kernel = new WavePlayer();
}


void Screen::playIt()
{
    kernel->playMusic(filePath);
}
```

- Implement your computation code
  Write your code in the files in Computation folder. They have nothing to do with the GUI issues. They are the purely functional codes, e.g., functions to play the WAV format music.



```cpp
#pragma once
#include <iostream>
#include <Windows.h>
#include <mmsystem.h>
#include <string>
#include <fstream>

using namespace std;

class WavePlayer
{
public:
    WavePlayer() {};
    ~WavePlayer() {};

    void playMusic(string filePath);
    void stopPlaying();
};
```



```cpp
#include "playWave.hpp"


void WavePlayer::playMusic(string filePath)
{
    cout << "the music is playing ..." << endl;
}


void WavePlayer::stopPlaying()
{
    cout << "the music is stop!" << endl;
}
```