# VS-QT程序总结

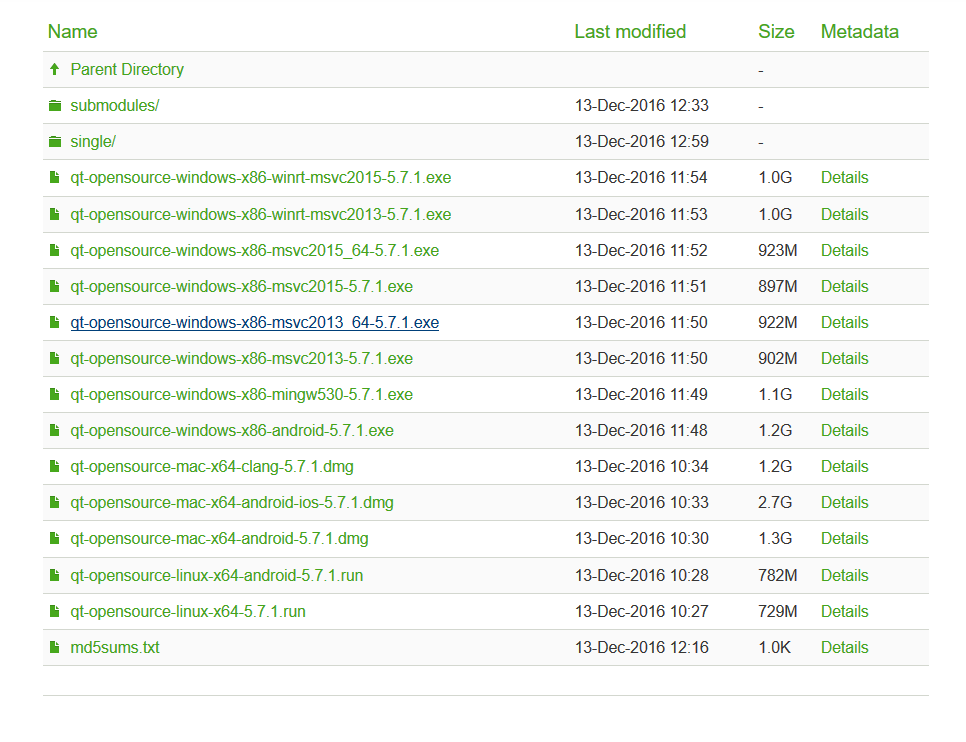
# 1．安装配置

## 1.QT的安装和配置

⑴下载QT : <http://qt-project.org/downloads>

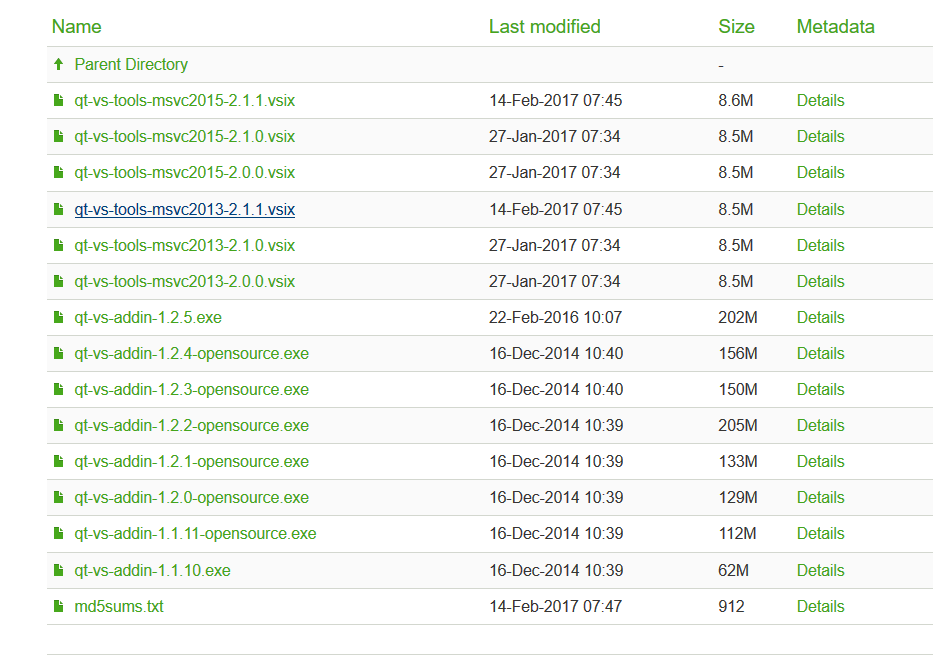
这里用的是QT5.7.1，当然也可以选择其他版本，进入到相应网站：

<http://download.qt.io/official_releases/qt/5.7/5.7.1/>



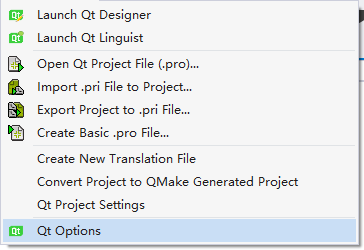
这里有较多的版本，我们用的VS是2013版的电脑是64位的，所以我们就选择上图选中的那一项即可。下载完以后进行相应的按照即可，按照没有什么需要特别注意的，默认安装即可。

⑵ QT-VS插件下载安装 ：<http://download.qt.io/archive/vsaddin/>



如上图所示，由于我们选用的是vs2013，那么这里我们就选一个针对vs2013的最新插件即可，即上图中选中的那一项下载安装即可。

⑶ 全部安装好以后，启动VS，在VS标签栏就会多一个QT VS TOOLS的选项，然后点击，找到QT Options选项，进入选项。

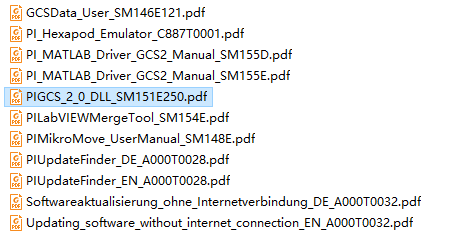


然后点击add按钮，添加路径即可，我们这里的路径是: C:\Qt5.7.1\5.7\msvc2013 也就是说不管哪个版本，添加至QT路径的msvc2013文件夹下即可。接下来我们就在VS下可以顺利使用QT了。

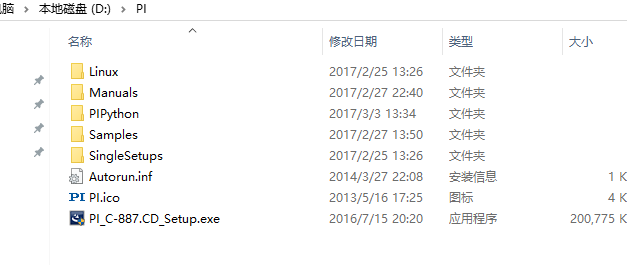
## 2. PI六轴平台的相关配置



⑴ 简单介绍： PI公司的这个平台控制起来相对比较方便，它给我们提供了很多控制和操作平台的API库函数。API函数手册在下图中的PIGCS\_2\_0\_DLL\_SM151E250中，这里主要做的是对平台的控制和轨迹规划和位姿返回，它里面的东西还是比较多的，感兴趣的同学可以深入研究。



⑵软件的安装 PI公司提供了平台对应的软件，大家可以下载玩玩，但是我们主要的目的是获得安装后的一些库，在光盘的目录中有一个安装程序，按默认安装即可。



安装完以后在：

C:\Users\Public\PI\PI\_Programming\_Files\_PI\_GCS2\_DL(如下图所示)中会有相应的库，这里我们需要的是下面5个文件

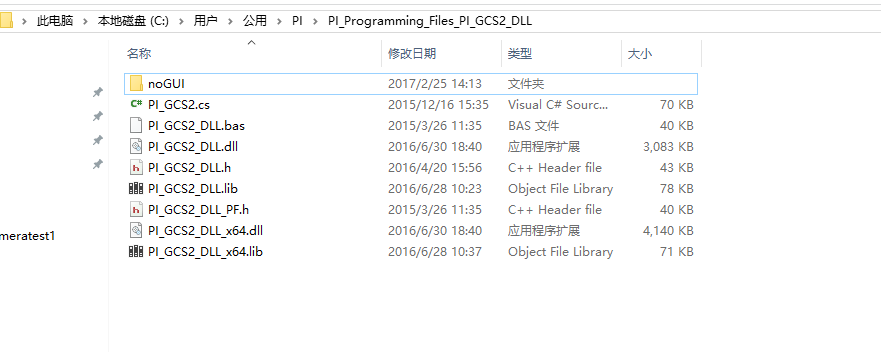
- PI\_GCS2\_DLL.h

- PI\_GCS2\_DLL.lib

- PI\_GCS2\_DLL.dll

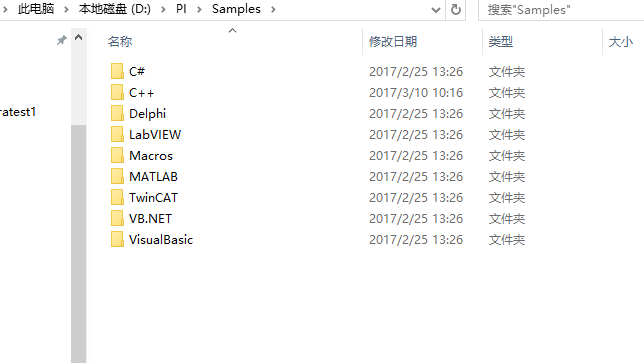
- PI\_GCS2\_DLL\_x64.lib

- PI\_GCS2\_DLL\_x64.dll



我们会需要这5个文件。

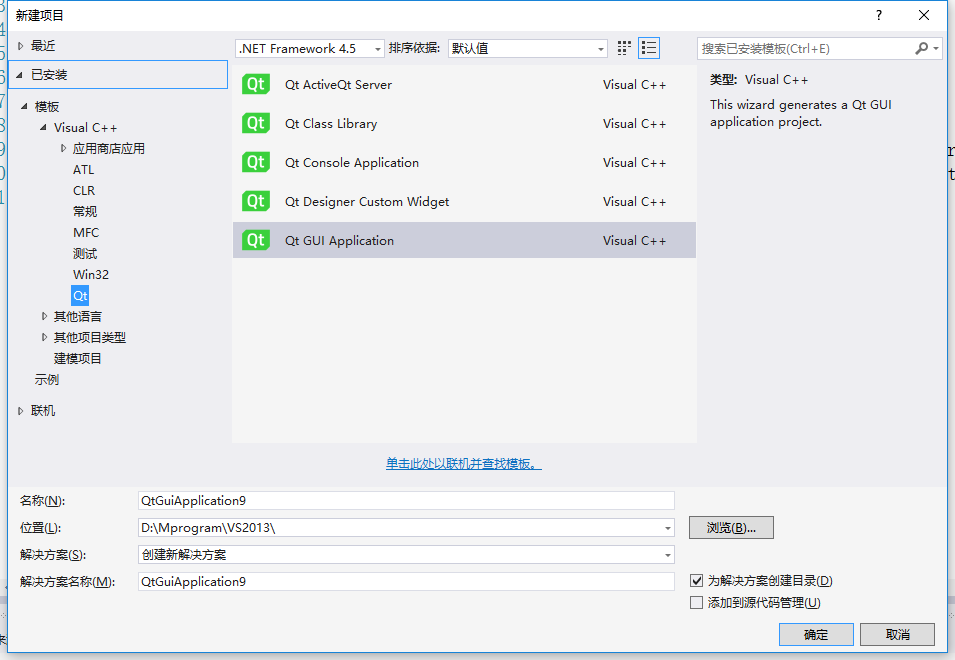
⑶使用方法 PI公司给出了一些简单的例程，在其光盘的如下图的目录中，有C#，C++，MATLAB等，都是很简单的历程，对相关语言了解的同学会很快就能看懂它的程序。



有不懂得库函数可以到它的手册中进行查找。

## 3.开始建立工程

⑴ 启动VS，新建QT GUI工程，工程名称大家随意取



然后一路点击确定，next即可。

⑵将PI的相应的库添加到工程中

由上面的介绍我们知道我们需要下述5个文件

- PI\_GCS2\_DLL.h

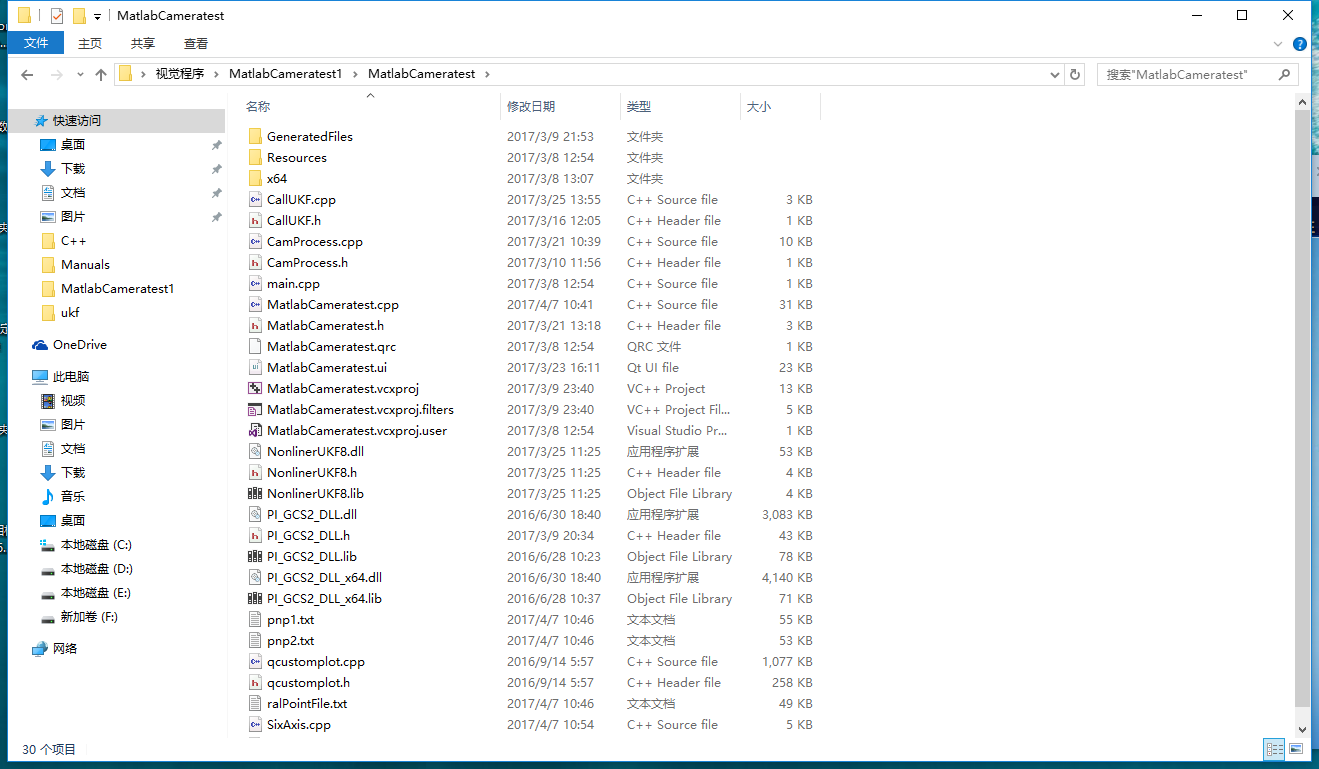
- PI\_GCS2\_DLL.lib

- PI\_GCS2\_DLL.dll

- PI\_GCS2\_DLL\_x64.lib

- PI\_GCS2\_DLL\_x64.dll

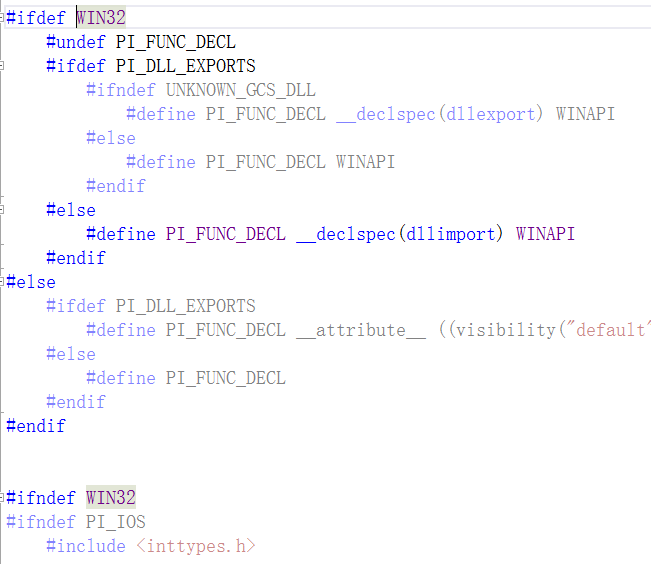
我们要将这5个文件考到我们的工程目录下



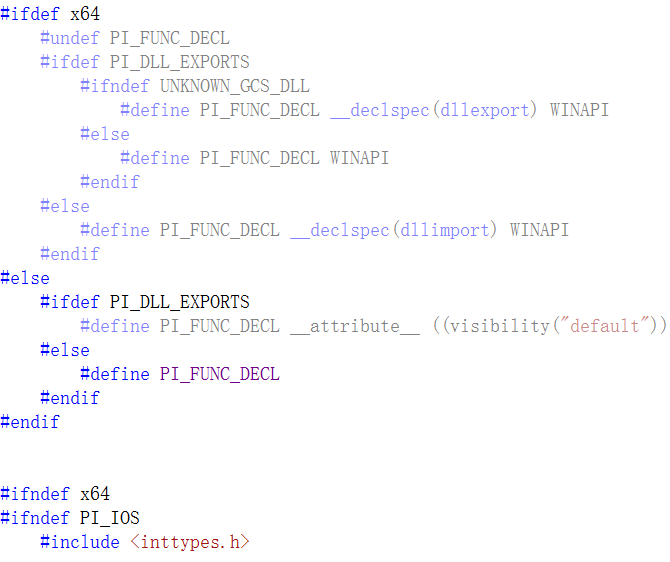
然后将需要使用相关API库函数时包含相应的头文件即可：

#include "PI\_GCS2\_DLL.h"，这里需要修改一点，打开此头文件

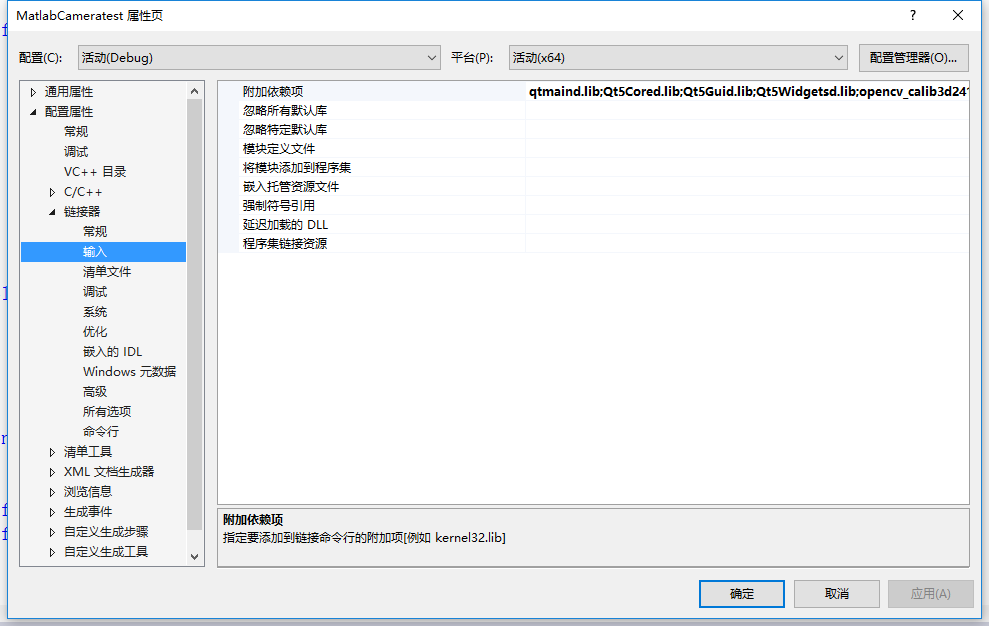
这里需要注意的是由于给的例程都是基于32位的，所以它的预定义



都是在32位下进行，而我们则是在64位下运行程序，所以需要把所有的WIN32改为x64，如下图所示。



同时在工程名上点击右键选择属性(通常是最后一项)



选择连接器->输入->附加依赖项，将PI\_GCS2\_DLL\_x64.lib添加到里面，同样由于我们是64位的程序，所以要选择带x64的库。

我们这种加载库的方式属于静态加载的方式，通过头文件调用函数，头文件通过静态库(.lib文件)找到相应的动态链接库(.dll)中的函数实现。

⑵ opencv的配置

由于我们需要做图像处理，而opencv是一个超级强大的开源图像处理库，所以我们决定使用opencv来进行相应的图像处理，opencv最新的版本是3.1的，但是我们尝试使用3.1的时候出现了各种问题，所以我们决定使用opencv2.4.11版本。Opencv2.4.11与3.1在vs中的配置略有不同，这里主要讲一下opencv2.4.11版本的配置[[1]](#footnote-1)。

i）下载安装

可以到SourceForge上下载：

<https://sourceforge.net/projects/opencvlibrary/files/opencv-win/>

目前opecv的安装包主要在SourceForge上托管，

也可以到到opencv官网下载即可：<http://opencv.org/>但是好像现在有点问题 。



选择其中的2.4.11进入下载即可。Opencv的安装程序其实也就是解压程序，解压到相应的文件夹即可。

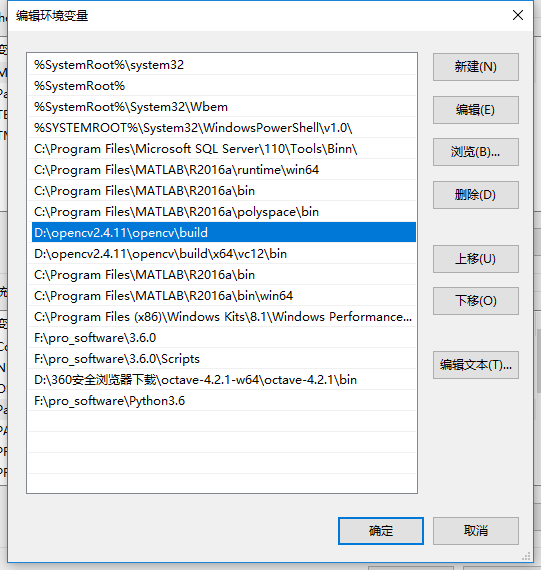
ii）设置电脑环境变量

在电脑的path路径中添加opencv的路径，根据自己的opencv的路径进行相应的设置，这里我们添加的路径是

D:\opencv2.4.11\opencv\build\

D:\opencv2.4.11\opencv\build\x64\vc12\bin

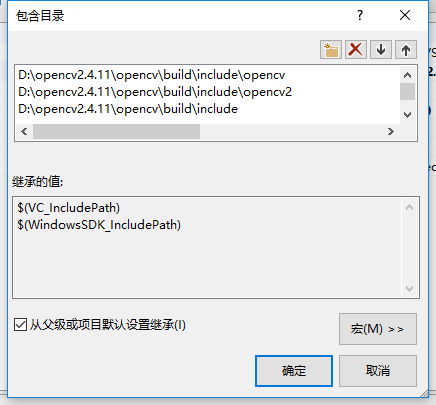
如下图所示

iii）进入VS中工程->属性->配置属性->vc++目录  
在包含目录中添加一下库：

D:\opencv2.4.11\opencv\build\include

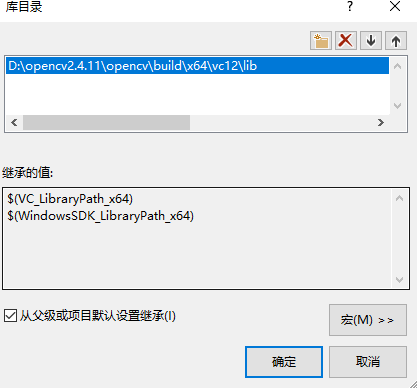
D:\opencv2.4.11\opencv\build\include\opencv

D:\opencv2.4.11\opencv\build\include\opencv2



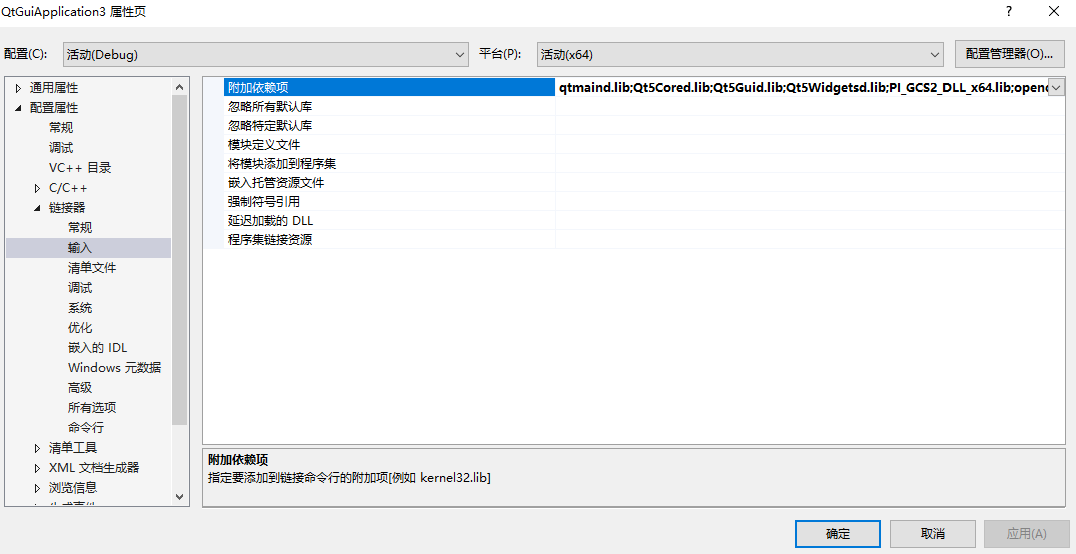
在库目录中包含以下库：

D:\opencv2.4.11\opencv\build\x64\vc12\lib



iv）进入工程属性->链接器->输入->附加依赖性

DEBUG模式下添加lib目录下文件名有d的lib文件。



opencv\_calib3d2411d.lib

opencv\_contrib2411d.lib

opencv\_core2411d.lib

opencv\_features2d2411d.lib

opencv\_flann2411d.lib

opencv\_gpu2411d.lib

opencv\_highgui2411d.lib

opencv\_imgproc2411d.lib

opencv\_legacy2411d.lib

opencv\_ml2411d.lib

opencv\_nonfree2411d.lib

opencv\_objdetect2411d.lib

opencv\_ocl2411d.lib

opencv\_photo2411d.lib

opencv\_stitching2411d.lib

opencv\_superres2411d.lib

opencv\_ts2411d.lib

opencv\_video2411d.lib

opencv\_videostab2411d.lib

release模式下去掉各个库最后的d即可

opencv\_calib3d2411.lib

opencv\_contrib2411.lib

opencv\_core2411.lib

opencv\_features2d2411.lib

opencv\_flann2411.lib

opencv\_gpu2411.lib

opencv\_highgui2411.lib

opencv\_imgproc2411.lib

opencv\_legacy2411.lib

opencv\_ml2411.lib

opencv\_nonfree2411.lib

opencv\_objdetect2411.lib

opencv\_ocl2411.lib

opencv\_photo2411.lib

opencv\_stitching2411.lib

opencv\_superres2411.lib

opencv\_ts2411.lib

opencv\_video2411.lib

opencv\_videostab2411.lib

## 4. matlab与vs混合编程

我们这里用的是vs2013+matlab2016a，测试可用，如果是其他版本的话就不一定了。此部分也较为简单，但是一定要按照步骤仔细配置，否则也会出现各种问题，这里给出一篇博客：

<http://blog.csdn.net/u013920434/article/details/52077788>，按其操作即可。

## 5. qtcustomplot[[2]](#footnote-2)的使用

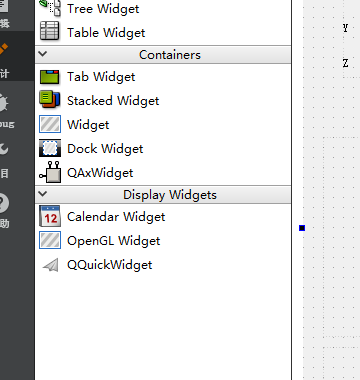
Qtcustomplot[[3]](#footnote-3)是一个强大的坐标轴控件，我们只需将它的头文件和源文件添加到我们的工程目录中即可使用，使用也比较简单。

点击下列链接可以下载源文件和文档。

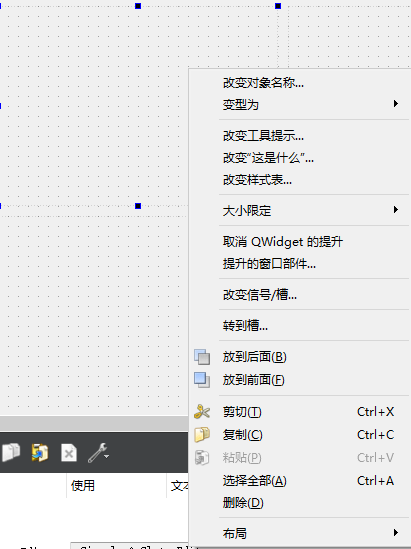
* [QCustomPlot首页](http://www.qcustomplot.com/)
* [QCustomPlot文档](http://www.qcustomplot.com/index.php/tutorials/settingup)
* [QCustomPlot下载](http://www.qcustomplot.com/index.php/download)

针对VS-QT，需要在“项目属性 -> 配置属性 -> C/C++ -> 常规 -> 附加包含目录”里面添加Qt5PrintSupportd.lib，还有其他问题可参考<http://blog.csdn.net/ch2065/article/details/28446959>或者自己上网寻找问题。

另外在ui界面画图使用的是widget控件

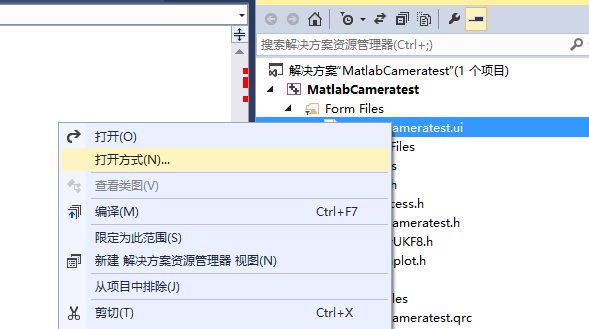


然后在控件上右键点击，将Qwidget控件提升为qcustomplot，这里是已经提升的控件，然后给控件命名就可以调用该控件了。

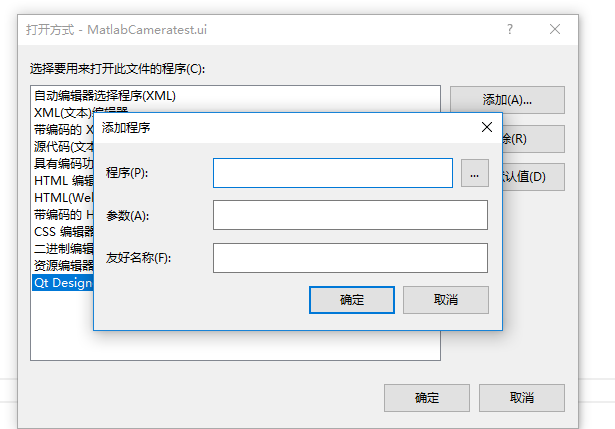


这里还有一个问题就是vs中的ui文件运行多次以后会有打不开的问题，网上的解决方案是：

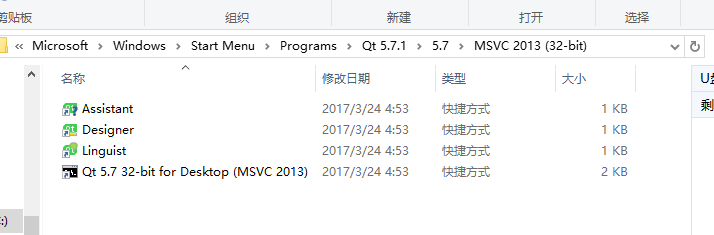
<http://blog.csdn.net/racaljk/article/details/18961429>，但是我试了不管用，这里有两种解决方案，首先可以通过直接打开desinger软件，然后再在vs中打开ui文件，其次我觉得可能是ui文件识别不到这个版本的designer软件了，可以再下另外版本的qt，然后再C:\Users\hehe\AppData\Roaming\Microsoft\Windows\Start Menu\Programs目录下找到相应的qt版本



右键点击ui文件，选择打开方式



点击添加按钮，然后再程序右侧将上述目录找到：



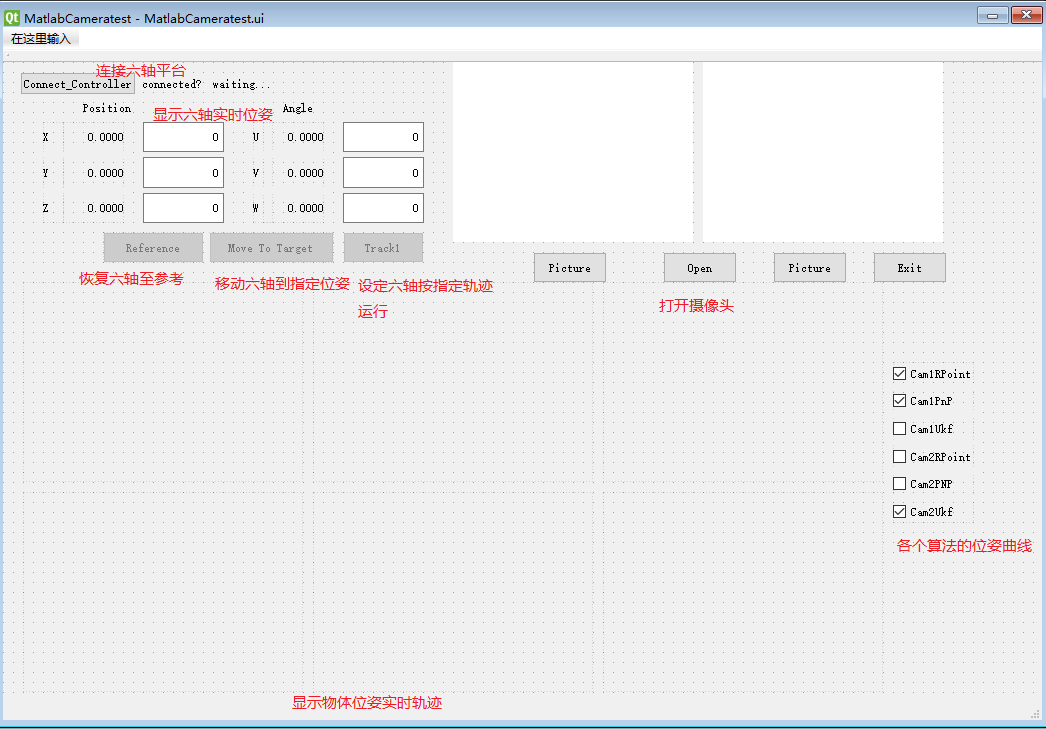
然后找到相应的desinger软件就可以打开了。

# 2. 程序介绍

这里主要介绍程序的大致思路，程序里也会有必要的注释。

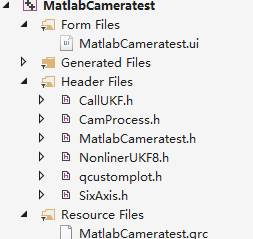
## QT按钮及槽函数的添加

我们的界面整体如下



在VS-QT中添加按钮和槽函数的方法如下：

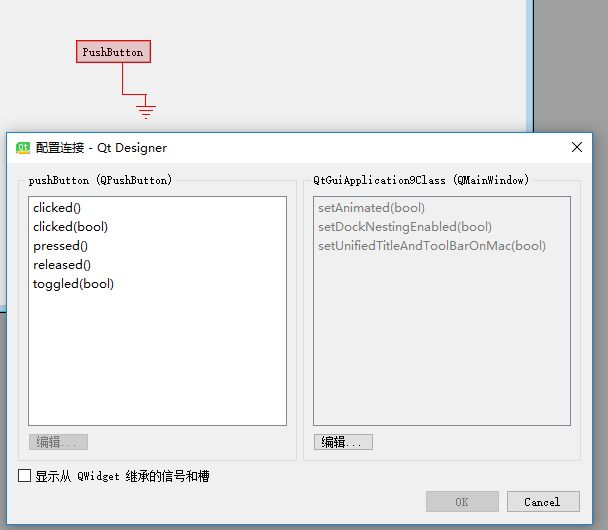
首先打开项目中的ui文件，这里是MatlabCameratest.ui,如下图所示：



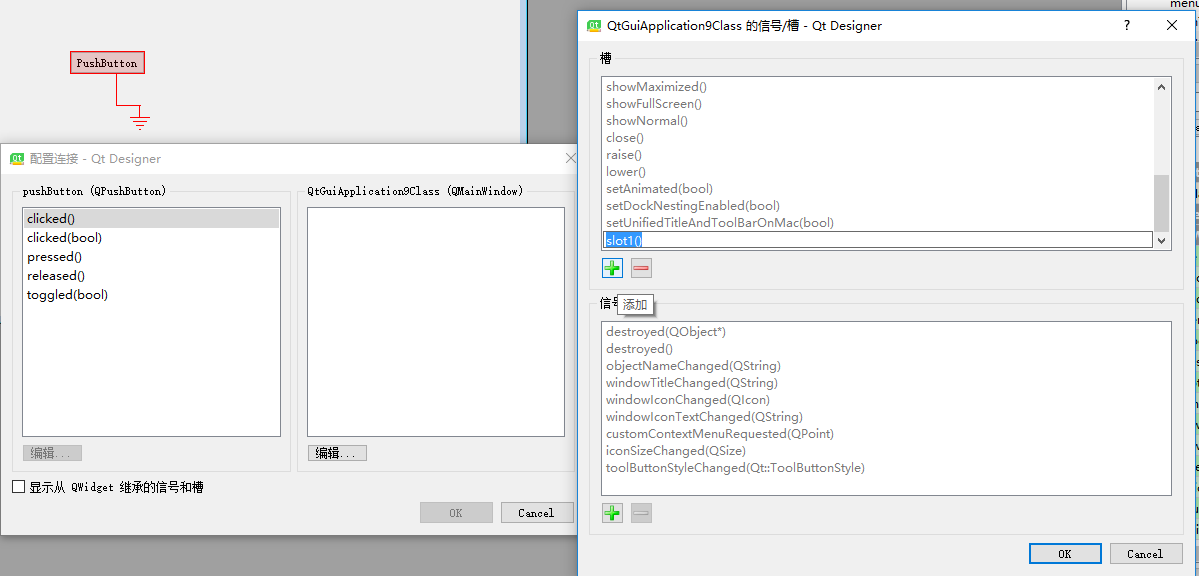
进入ui文件以后，可以在左侧栏看到各种控件，我们只需要把各种控件添加到界面上，修改我们想要的名字。然后点击标签栏上的编辑信号/槽按钮，下图中的第一个。



然后点住相应的按钮并拖出一条线，如下图所示：



通常的按钮事件为clicked()或者clicked(bool)，如clicked()事件，点击clicked()，然后点击编辑，点击绿色的加号，就可以添加槽函数，函数名字任意指定，如slot1()。



然后需要在界面类的头文件中包含你定义的槽函数，如void slot1();这个函数不是在public 或者private属性下，需要自己再定义一个

Public slots;private slots或者protected slots。

然后在cpp文件中就可以实现该函数，该函数就是鼠标点击按钮的触发事件。

## 2. 控制六轴平台类（SixAxis类）

SixAxis类是一个线程类[[4]](#footnote-4)，考虑把它作为一个线程类的主要目的是为了让物体的位姿显示，物体图像显示和六轴平台同时运动。

**【Qt中创建线程的方法】**

只需要子类化QThread并重新实现它的run()函数就可以了。run()是个纯虚函数，是线程执行的入口，在run()里出现的代码将会在另外线程中被执行。run()函数是通过start()函数来实现调用的。

部分函数实现代码：

### Connect()，ConnectFirstFoundHexapodViaTCPIP()：函数功能：连接高精度六轴平台

void SixAxis::Connect()

{

ID = ConnectFirstFoundHexapodViaTCPIP();

if (ID>-1)

{

SixAxisConnectFlag = true;

}

else

{

throw(ID);

}

}

首先通过ConnectFirstFoundHexapodViaTCPIP()函数尝试连接高精度六轴平台，如果成功连接上了，则返回一个int型的非负数，并设置标志位SixAxisConnectFlag为true，否则抛出异常。

long SixAxis::ConnectFirstFoundHexapodViaTCPIP()

{

char szFoundDevices[10000];

//printf("searching TCPIP devices...\n");

if (PI\_EnumerateTCPIPDevices(szFoundDevices, 9999, "") <= 0)

{

return -1;

}

char\* szAddressToConnect = NULL;

int port = 0;

char \* pch = strtok(szFoundDevices, "\n");

while (pch != NULL)

{

\_strupr(pch);

if (

((strstr(pch, "F-HEX") != NULL) && (strstr(pch, "LISTENING") != NULL))

|| ((strstr(pch, "HEXAPOD") != NULL) && (strstr(pch, "LISTENING") != NULL))

|| ((strstr(pch, "F-206") != NULL) && (strstr(pch, "LISTENING") != NULL))

|| ((strstr(pch, "M-8") != NULL) && (strstr(pch, "LISTENING") != NULL))

|| ((strstr(pch, "C-887") != NULL) && (strstr(pch, "LISTENING") != NULL))

)

{

char\* colon = strstr(pch, ":");

if (colon == NULL)

{

continue;

}

\*colon = '\0';

char\* bracket = strstr(pch, "(");

if (bracket == NULL)

{

continue;

}

szAddressToConnect = new char[strlen(bracket + 1) + 1];

strcpy(szAddressToConnect, bracket + 1);

bracket = strstr(colon + 1, ")");

if (bracket == NULL)

{

continue;

}

\*bracket = '\0';

port = atoi(colon + 1);

break;

}

pch = strtok(NULL, "\n");

}

if (szAddressToConnect != NULL)

{

//printf("trying to connect with %s, port %d\n", szAddressToConnect, port);

int iD = PI\_ConnectTCPIP(szAddressToConnect, port);

delete[]szAddressToConnect;

return iD;

}

return -1;

}

此函数中使用PI\_EnumerateTCPIPDevices()函数来搜寻电脑中所有的已连接的设备，然后通过匹配高精度六轴平台的设备名称找到六轴平台的设备名字并将其存放在szAddressToConnect中，然后通过PI\_ConnectTCPIP()函数连接到六轴平台。

### ReferenceIfNeeded：函数功能：将高精度六轴平台移动至原点(也叫参考点)

bool SixAxis::ReferenceIfNeeded(int ID, char\* axis)

{

BOOL bReferenced;

BOOL bFlag;

if (!PI\_FRF(ID, axis))

return false;

bFlag = false;

while (bFlag != TRUE)

{

//使用命令PI\_IsControllerReady检查参考点标定是否已经完成，如果没有完成的话不能退出while循环。

if (!PI\_IsControllerReady(ID, &bFlag))

return false;

// }

}

return true;

}

在该函数中，首先使用PI\_FRF()库函数使目标移动至原点，形参ID代表六轴平台，axis代表六个轴，然后通过PI\_IsControllerReady()库函数检查参考点标定是否已经完成，如果没有完成的话不能退出while循环。

### 3、Track1：函数功能：使目标根据设定轨迹运动

void SixAxis::Track1()

{

double position[6] = { 5, 2, 3, 0, 0, 0 };

double position1[6] = { 8, 9, 6, 3, 5, 2 };

double position2[6] = { 12, 7, 5, 9, 3, 5 };

double position3[6] = { 9, 8, 8, 10, 5, 5 };

double position4[6] = { 10, 10, 10, 8, 4, 3 };

double position5[6] = { 14, 7, -6, -5, 5, 5 };

double position6[6] = { 8, 5, 3, 8, 2, 3 };

double position7[6] = { 0, 2, 5, 5, -2, 6 };

BOOL bIsMoving = TRUE;

PI\_MOV(ID, "x y z u v w", position);

while (bIsMoving == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving);

}

BOOL bIsMoving1 = TRUE;

PI\_MOV(ID, "x y z u v w", position1);

while (bIsMoving1 == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving1);

}

BOOL bIsMoving2 = TRUE;

PI\_MOV(ID, "x y z u v w", position2);

while (bIsMoving2 == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving2);

}

BOOL bIsMoving3 = TRUE;

PI\_MOV(ID, "x y z u v w", position3);

while (bIsMoving3 == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving3);

}

BOOL bIsMoving4 = TRUE;

PI\_MOV(ID, "x y z u v w", position4);

while (bIsMoving4 == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving4);

}

BOOL bIsMoving5 = TRUE;

PI\_MOV(ID, "x y z u v w", position5);

while (bIsMoving5 == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving5);

}

BOOL bIsMoving6 = TRUE;

PI\_MOV(ID, "x y z u v w", position6);

while (bIsMoving6 == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving6);

}

BOOL bIsMoving7 = TRUE;

PI\_MOV(ID, "x y z u v w", position7);

while (bIsMoving7 == TRUE)

{

PI\_IsMoving(ID, "x", &bIsMoving7);

}

}

在该函数中，首先使用数组依次存放设定的轨迹，每次移动都包括6个自由度的移动，设定好估计以后，我们在每一次移动时需要设定一次标志位，并在构造函数中将标志位设置为true，调用PI\_MOV函数移动六轴平台，然后调用PI\_IsMoving函数，当六轴平台移动时，参数&bIsMoving7会使bIsMoving7为true。因此程序会在while循环里继续运行，直到六轴平台移动结束。

## 3. 图像处理类（CamProcess类）

部分函数实现代码：

### 1、ProcessImage：函数功能：图像处理，提取特征点坐标，并绘制特征点图像

CamProcess::ImgAndData CamProcess::ProcessImage(const Mat &frame)//提取特征点图像坐标系下坐标

{

Mat hsvImage;

QImage QImg;

ImgAndData mreturn;

Mat frame1;

cv::cvtColor(frame, hsvImage, CV\_BGR2HSV);//将图像格式转换为hsv

cv::erode(hsvImage, hsvImage, Mat());//腐蚀图像

cv::dilate(hsvImage, hsvImage, Mat());//膨胀图像

cv::erode(hsvImage, hsvImage, Mat());

inRange(hsvImage, Scalar(110, 50, 0), Scalar(135, 255, 255), hsvImage);

//选取固定颜色区间，二值化图像

//Mat resizeImage;

//resize(hsvImage, resizeImage, Size(320, 240), 0, 0, INTER\_LINEAR);

//QImg = cvMat2QImage1(resizeImage);

Mat GrayImage;

GrayImage = hsvImage;

std::vector<std::vector<Point>> contours;

// std::vector<Vec4i> hierarchy;

findContours(hsvImage, contours, CV\_RETR\_LIST, CV\_CHAIN\_APPROX\_NONE);

//提取轮廓

Mat image2;

image2 = GrayImage;

Mat ResultImage(hsvImage.size(), CV\_8U, Scalar(255));

std::vector<std::vector<Point>>::iterator itc = contours.begin();

unsigned int cmin = 70;

unsigned int cmax = 500;

//选择所需轮廓大小的区间，去除过大或过小的轮廓

while (itc != contours.end())

{

if (itc->size() < cmin || itc->size() > cmax)

itc = contours.erase(itc);

else

++itc;

}

float radius;

Point2f center;

Coor temp;

Coor temp1;

Coor temp2;

Coor temp3;

Coor temp4;

//下面5组数据用于存放将坐标分组后的信息，使用了vector容器

std::vector<Coor> mydata;

std::vector<Coor> mydata1;

std::vector<Coor> mydata2;

std::vector<Coor> mydata3;

std::vector<Coor> mydata4;

if (contours.size() > 0)

{

minEnclosingCircle(contours[0], center, radius);

//用圆去逼近已筛选的轮廓，确定特征圆

}

itc = contours.begin();

mydata.erase(mydata.begin(), mydata.end());

while (itc != contours.end())

{

Moments mom = cv::moments(Mat(\*itc++));

circle(ResultImage, Point(mom.m10 / mom.m00, mom.m01 / mom.m00), 2, Scalar(0), 2);

temp.x = mom.m10 / mom.m00;

temp.y = mom.m01 / mom.m00;

mydata.push\_back(temp);

//对所提取的圆进行圆心坐标求取并保存到mydata中

}

Mat resizeImage;

cv::resize(ResultImage, resizeImage, Size(240, 180), 0, 0, INTER\_LINEAR);

QImg = cvMat2QImage1(resizeImage);

vector<Coor>::iterator it1;

double sumx = 0;

double sumy = 0;

//我们提取特征点的个数是8，判断是否为8个特征点，这里其实也应该有个异常判断

if (contours.size() == 8)

{

//将8个特征点的x，y坐标相加并求取平均值

for (it1 = mydata.begin(); it1 != mydata.end(); it1++)

{

sumx += (\*it1).x;

sumy += (\*it1).y;

}

sumx = sumx / 8; sumy = sumy / 8;

//将坐标分组上下左右的4组，每组有两个坐标值

for (it1 = mydata.begin(); it1 != mydata.end(); it1++)

{

if ((\*it1).x - sumx>0)

{

if ((\*it1).y - sumy>0)

{

temp1.x = (\*it1).x;

temp1.y = (\*it1).y;

mydata1.push\_back(temp1);

}

else

{

temp2.x = (\*it1).x;

temp2.y = (\*it1).y;

mydata2.push\_back(temp2);

}

}

else

{

if ((\*it1).y - sumy>0)

{

temp3.x = (\*it1).x;

temp3.y = (\*it1).y;

mydata3.push\_back(temp3);

}

else

{

temp4.x = (\*it1).x;

temp4.y = (\*it1).y;

mydata4.push\_back(temp4);

}

}

}

}

//我们分组后每组的特征点的个数是2，判断是否为2个特征点，这里其实也应该有个异常判断

if (mydata1.size() == 2)

{

//由于我们的坐标原点是在图像左上角，所以不能仅仅判断坐标x值的大小，这样会出错，应该判断x坐标与x坐标平均值差的大小

//将中心点也就是坐标均值比较远的点设置为第一个点，放入m\_observe\_z[0]，m\_observe\_z[0]

if ((fabs(mydata1.at(0).x - sumx)) > (fabs(mydata1.at(1).x - sumx)))

{

m\_observe\_z[0] = mydata1.at(0).x;

m\_observe\_z[1] = mydata1.at(0).y;

m\_observe\_z[2] = mydata1.at(1).x;

m\_observe\_z[3] = mydata1.at(1).y;

}

else

{

m\_observe\_z[0] = mydata1.at(1).x;

m\_observe\_z[1] = mydata1.at(1).y;

m\_observe\_z[2] = mydata1.at(0).x;

m\_observe\_z[3] = mydata1.at(0).y;

}

}

//将中心点也就是坐标均值比较远的点设置为第一个点，放入m\_observe\_z[4]，m\_observe\_z[5]

if (mydata2.size() == 2)

{

if ((fabs(mydata2.at(0).x - sumx)) > (fabs(mydata2.at(1).x - sumx)))

{

m\_observe\_z[4] = mydata2.at(0).x;

m\_observe\_z[5] = mydata2.at(0).y;

m\_observe\_z[6] = mydata2.at(1).x;

m\_observe\_z[7] = mydata2.at(1).y;

}

else

{

m\_observe\_z[4] = mydata2.at(1).x;

m\_observe\_z[5] = mydata2.at(1).y;

m\_observe\_z[6] = mydata2.at(0).x;

m\_observe\_z[7] = mydata2.at(0).y;

}

}

//将中心点也就是坐标均值比较远的点设置为第一个点，放入m\_observe\_z[6]，m\_observe\_z[9]

if (mydata3.size() == 2)

{

if ((fabs(mydata3.at(0).x - sumx)) > (fabs(mydata3.at(1).x - sumx)))

{

m\_observe\_z[8] = mydata3.at(0).x;

m\_observe\_z[9] = mydata3.at(0).y;

m\_observe\_z[10] = mydata3.at(1).x;

m\_observe\_z[11] = mydata3.at(1).y;

}

else

{

m\_observe\_z[8] = mydata3.at(1).x;

m\_observe\_z[9] = mydata3.at(1).y;

m\_observe\_z[10] = mydata3.at(0).x;

m\_observe\_z[11] = mydata3.at(0).y;

}

}

//将中心点也就是坐标均值比较远的点设置为第一个点，放入m\_observe\_z[12]，m\_observe\_z[13]

if (mydata4.size() == 2)

{

if ((fabs(mydata4.at(0).x - sumx)) > (fabs(mydata4.at(1).x - sumx)))

{

m\_observe\_z[12] = mydata4.at(0).x;

m\_observe\_z[13] = mydata4.at(0).y;

m\_observe\_z[14] = mydata4.at(1).x;

m\_observe\_z[15] = mydata4.at(1).y;

}

else

{

m\_observe\_z[12] = mydata4.at(1).x;

m\_observe\_z[13] = mydata4.at(1).y;

m\_observe\_z[14] = mydata4.at(0).x;

m\_observe\_z[15] = mydata4.at(0).y;

}

}

mreturn.mQImg = QImg;

mreturn.mData = m\_observe\_z;

//将处理后的图片，和获取到的坐标数据返回

return mreturn;

}

此函数中首先对输入的图片进行形态学处理中的膨胀和腐蚀：erode()和dilate()等，然后利用inRange()函数二值化图像，最后通过findContours()函数找到图像中的特征点轮廓，并排除比较大的和比较小的轮廓。此函数中利用提取图像中心距的办法提取图像中各个轮廓的中心坐标并绘制出来。最后将图片和坐标数据保存到mreturn中并返回。

# 3．摄像头（工业摄像头）

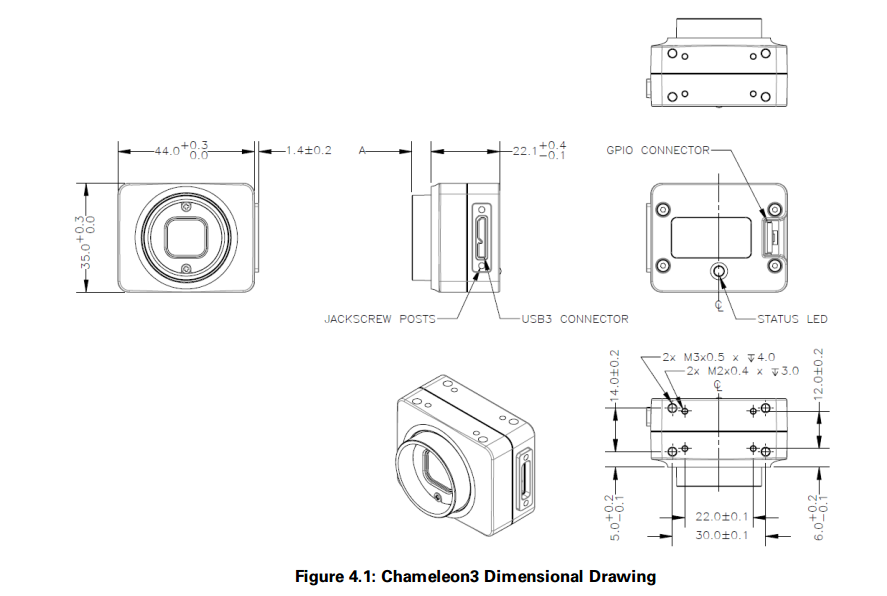
摄像头自带有光盘，但是一进入光盘以后就会很卡，复制出来也会卡死，而且好像光盘里提供的不知道是什么鬼东西，跟本摄像头貌似无关，十分误导人。所以就到它的官网看了：<https://www.ptgrey.com/>

官网支持栏有下载链接(如下图)，然后输入我们的摄像头型号和相应的操作系统进行下载我们需要的文档和软件。本以为下载这些就够了，可是在使用时还是会到图像转换的问题，还以为要自己转换呢，想想都难，然后又到官网看了看，有技术支持说明：

<https://www.ptgrey.com/tan/10898>，恰巧有关于opencv的文档，当然还有其他的相关文档。给产品的时候没有个文档说明真是扯啊。



## 物理结构及尺寸

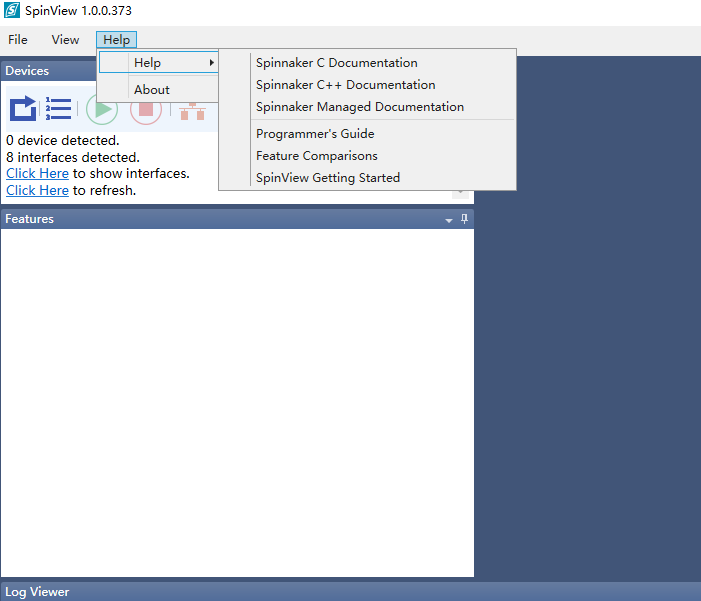


## Spinnaker C++ Programmer's Guide Contents（C++编程指导）

链接：

[file:///C:/Program%20Files/Point%20Grey%20Research/Spinnaker/doc/API/C++/html/page2.html](file:///C:\Program%20Files\Point%20Grey%20Research\Spinnaker\doc\API\C++\html\page2.html)

也可以在SpinView的help中打开；如下图所示：



### Architecture of Spinnaker API

Spinnaker API is built around the GenICam standard, which offers a generic programming interface for various cameras and interfaces. Spinnaker is an extension of Spinnaker. Spinnaker provides quick and easy access to your camera.( Spinnaker API基于GenICam标准，为多种摄像头和接口提供了常用的编程接口，Spinnaker是Spinnaker的一种扩展，Spinnaker为使用摄像头提供了简单快速的方式)

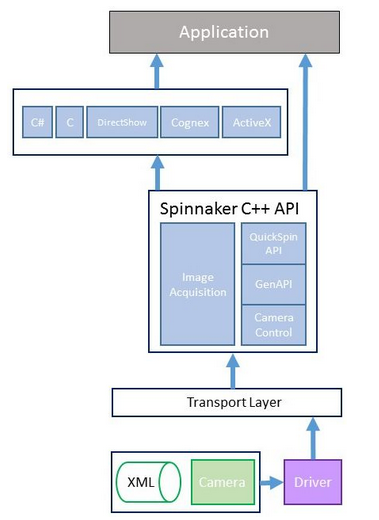
Spinnaker API includes two major components:

**Image Acquisition**

This is the acquisition engine that is responsible for setting up image buffers and image grabbing.

**Image Configuration**

This is the configuration engine that is responsible for controlling your camera. This component consist of quick spin API, which is a wrapper that makes GenAPI easy to use.(这一部分是控制你的摄像机的配置引擎，这部分由quick spin API组成，spin API是使GenAPI更容易使用包装)



### Examples(例程)

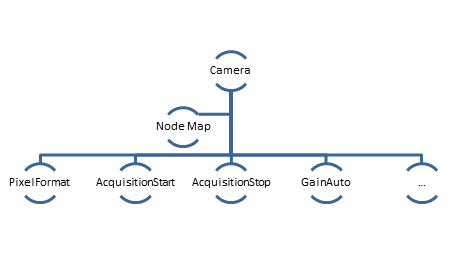
Included with the Spinnaker SDK are a number of source code examples to help you get started. These examples are provided for C, C++, C#, and VB.NET languages and are precompiled for your convenience.

The table below describes the available Spinnaker SDK examples.

| **Spinnaker Example** | **Description** |
| --- | --- |
| Acquisition | Enumerate, start acquisition, and grab images |
| Trigger\* | Trigger |
| ImageFormatControl\* | Configure a custom image size and format |
| Exposure\* | Configure a custom exposure time |
| AcquisitionMultipleCamera | How to capture images from multiple cameras simultaneously |
| SaveToAVI | Save images in AVI format |
| ImageEvents | Image events |
| Sequencer | Capture multiple images with different parameters in a sequence |
| SpinSimpleGUI\_MFC | Graphical User Interface for evaluating and setting camera parameters |
| ChunkData | How to get chunk data on an image, either from the nodemap or from the image itself |
| DeviceEvents | Create a handler to access device events |
| Enumeration\* | Enumerate interfaces and cameras |
| EnumerationEvents | Explore arrival and removal events on interfaces and the system |
| GenTLInfo\_QuickSpin | How to access node information from interfaces and cameras |
| Logging | Create a logging event handler |
| LookupTable | Configure lookup tables for the customization and control of individual pixels |
| NodeMapCallback | Create, register, use, and unregister callbacks |
| NodeMapInfo | How to retrieve node map information |
| \*Also available in QuickSpin | |  |

### Nodes

Every GenICam compliant camera has an XML description file. The XML describes camera registers, their interdependencies, and all other information needed to access high-level features by means of low level register read and write operations. These features include Gain, Exposure Time, Image Format, and others. The elements of a camera description file are represented as software objects called Nodes. A Nodes map is a list of nodes created dynamically at run time.(每一种GenICam相机都有一个XML描述文件，这个XML文件描述了相机寄存器，它们的依赖关系以及其他需要去访问高级特征的信息，这些信息都是通过低级的寄存器读写操作获得的。这些特征包裹增益，曝光时间，图片格式等等。一种相机描述文件的元素通过节点的形式呈现。一个节点图就是运行时动态创建的一组节点)



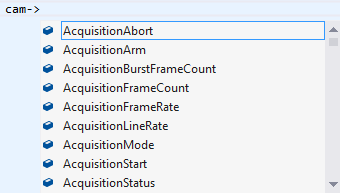
To access camera properties such as setting image width:

|  |  |
| --- | --- |
| **C++ GenAPI** | GenApi::INodeMap & nodeMap = cam.GetNodeMap();  CIntegerPtr width = nodeMap.GetNode("Width");  width->SetValue(new\_width\_val); |

### QuickSpin API and Accessing Camera Parameters

Generic programming with GenICam requires developers to know feature names before using them. Spinnaker provides the QuickSpin API, which helps developers write their applications hassle free. The QuickSpin API consists of a list of static functions integrated into the Camera class.(一般的GenICam编程要求开发者使用时需要知道特征名字，而这相对比较麻烦，Spinnaker提供了QuickSpin API，QuickSpin API帮助开发者简单方便的写他们的应用程序，QuickSpin API包括一些集成在Camera类中的静态功能)

All camera parameters can be accessed through the camera pointer object.



Most camera parameters (all items in camera.h) can be accessed using the QuickSpin API.(大多数相机参数都可以使用QuickSpin API获取)

For parameters not handled by QuickSpin API, you can access them via GenAPI. GenAPI is an open source API for configuring GenICam cameras. GenAPI is maintained by the European Machine Vision Association.(对于不能通过QuickSpin API获取的参数，你可以通过GenAPI获取，GenAPI配置GenICam摄像机的开源API，GenICam由欧洲机器视觉协会维持)

**Note:**QuickSpin API and GenAPI are available for C++ and C# only. They are not available for C.

Below is an example comparison of inquiring camera gain via GenICam API (GenAPI) and QuickSpin API.

|  |  |
| --- | --- |
| **C++ GenAPI** | Spinnaker::GenApi::INodeMap & nodeMap = cam->GetNodeMap(); CFloatPtr GainNode = nodeMap.GetNode("Gain"); Float GainVal = GainNode->GetValue(); |
| **C++ QuickSpin API** | float quickGainVal = cam->Gain.GetValue(); |

### C# Graphical User Interface API

For applications that want to take advantage of Spinnaker's graphical user elements, graphical user interface (GUI) controls are available. GUI controls are divided into static and dynamic categories. Static GUI controls include the CameraSelectionDialog, display window, and property grid window. Dynamic GUI control capability is built into the camera's firmware. Therefore, new firmware will have the ability to add GUI controls to the same application, without recompiling.(对于想要利用Spinnaker图像用户元素的应用，GUI控制是可以获取的，GUI控制被分为动态和静态种类，静态GUI控制包括相机选择对话框，显示窗口，和属性敞口。动态GUI 控制被建立到相机的硬件中，因此，新的硬件将有添加GUI控制到同样的应用，无需再次编译。)

|  |  |
| --- | --- |
| **Static GUI Dialogs** | |
| //To show image drawing window  GUIFactory AcquisitionGUI = new GUIFactory ();  AcquisitionGUI.ConnectGUILibrary(cam);  ImageDrawingWindow AcquisitionDrawing = AcquisitionGUI.GetImageDrawingWindow();  AcquisitionDrawing.Connect(cam);  AcquisitionDrawing.Start();  AcquisitionDrawing.ShowModal(); | C:\Program Files\Point Grey Research\Spinnaker\doc\API\C++\html\drawing-window.jpg |
| //To show camera selection window  GUIFactory AcquisitionGUI = newGUIFactory ();  AcquisitionGUI.ConnectGUILibrary(cam);  CameraSelectionWindow camSelection = AcquisitionGUI.GetCameraSelectionWindow();  camSelection.ShowModal(true); | C:\Program Files\Point Grey Research\Spinnaker\doc\API\C++\html\selection-window.jpg |
| //To show property grid window  GUIFactory AcquisitionGUI = new GUIFactory ();  AcquisitionGUI.ConnectGUILibrary(cam);  PropertyGridWindow propWindow = AcquisitionGUI.GetPropertyGridWindow();  propWindow.Connect(cam);  propWindow.ShowModal(); | C:\Program Files\Point Grey Research\Spinnaker\doc\API\C++\html\grid-window.jpg |

|  |  |
| --- | --- |
| **Dynamic GUI Control** | |
| GUIFactory dynamicGUI = new GUIFactory ();  dynamicGUI.ConnectGUILibrary(cam);  // Get dialog name via dynamicGUI.GetDialogNameList()  Window dlg = dynamicGUI.GetDialogByName(dialogName);  dlg.Owner = Window .GetWindow(this );  dlg.Show(); | C:\Program Files\Point Grey Research\Spinnaker\doc\API\C++\html\dynamic-window.jpg |

### Camera XML

The camera's XML file contains information such as feature naming, register mapping, and dependencies between features. It is typical for GenICam-compliant software to cache the XML file for quicker access to the camera's definition. Spinnaker caches the XML file in a binary format to achieve better performance.(相机的XML问价包含特征命名，寄存器映射以及特征之间的依赖。缓存XML文件是GenICam-compliant软件特有的，是为了更快的访问摄像机的定义。为了实现更好的性能Spinnaker以二进制方式缓存XML文件)

Camera XML files are located in:

C:\ProgramData\Spinnaker\XML

### Recommended Environment

Spinnaker supports the following list of operating systems and development environments.

|  |  |
| --- | --- |
| OS Compatibility (32- and 64-bit) | Windows XP  Windows 7  Windows 8.1  Windows 10 |
| Language Support | C  C++  C#  VB.NET |
| Compiler Support | Visual Studio 2010  Visual Studio 2013  Visual Studio 2015 |
| Interface Support | USB3 Vision 1.0 |

### Instantiate Cameras

Before you can instantiate a camera, you must create and initialize a system object. The System Singleton object is used to retrieve the list of interfaces (USB 3.0 or GigE) and cameras available. You must call ReleaseInstance() at the end of your program to free up the system object.（在你开始实例化一个相机之前，你必须常见和出事或一个系统对象，这个单例模式的系统对象被用于获取USB3.0接口以及相机，在你编程结束以后你必须调用ReleaseInstance()函数释放你的系统对象）

Multiple cameras can only be instantiated one at a time.（多相机只能被编程一次一个）

|  |  |
| --- | --- |
| **Instantiate multiple cameras (C++)** | // Retrieve singleton reference to system object  SystemPtr system = System::GetInstance();  CameraList camList = system->GetCameras(); unsigned int numCameras = camList.GetSize();  for (unsigned int i = 0; i < numCameras; ++i) {     CameraPtr pCamera = cameraList.GetByIndex(i);     pCamera->Init();  }  // Release system  system->ReleaseInstance(); |

### Enumeration(枚举)

The snippet below detects the number of cameras connected and enumerates them from an index.

|  |  |
| --- | --- |
| **Spinnaker C++ GenAPI** | SystemPtr system = System::GetInstance(); CameraList camList = system->GetCameras(); unsigned int numCameras = camList.GetSize(); CameraPtr pCam = NULL; for (int i = 0; i < numCameras; i++) {     pCam = camList.GetByIndex(i);       pCam->Init(); } |

### Asynchronous Hardware Triggering(异步硬件触发)

The snippet below does the following:

* Enables Trigger Mode(使能触发模式)
* Configures GPIO0/Line0 as the trigger input source(配置GPIO0或者Line0作为触发源)
* Specifies the trigger signal polarity as an active high (rising edge) signal（指定触发信号为高信号触发，即上升沿触发）

|  |  |
| --- | --- |
| **Spinnaker C++ QuickSpin API** | Cam->TriggerMode.SetValue(Spinnaker::TriggerModeEnums::TriggerMode\_On);  Cam->TriggerSource.SetValue(Spinnaker::TriggerSourceEnums::TriggerSource\_Line0);  Cam->TriggerSelector.SetValue(Spinnaker::TriggerSelectorEnums::TriggerSelector\_FrameStart);  Cam->TriggerActivation.SetValue(Spinnaker::TriggerActivationEnums::TriggerActivation\_RisingEdge); |
| **Spinnaker C++ GenAPI** | CEnumerationPtr triggerMode = nodeMap.GetNode("TriggerMode");  triggerMode->SetIntValue(triggerMode->GetEntryByName("On")->GetValue());  CEnumerationPtr triggerSource = nodeMap.GetNode("TriggerSource");  triggerSource->SetIntValue(triggerSource->GetEntryByName("Line0")->GetValue());  CEnumerationPtr triggerSelector = nodeMap.GetNode("TriggerSelector");  triggerSelector->SetIntValue(triggerSelector->GetEntryByName("FrameStart")->GetValue());  CEnumerationPtr triggerActivation = nodeMap.GetNode("TriggerActivation");  triggerActivation->SetIntValue(triggerActivation->GetEntryByName("RisingEdge")->GetValue()); |

### Setting Black Level(DC偏移)

BlackLevel is the GenICam feature that represents the DC offset that is applied to the video signal. This example compares the mechanism used to set this feature in both environments.

|  |  |
| --- | --- |
| **Spinnaker C++ QuickSpin API** | // Brightness is called black level in GenICam cam->BlackLevelAuto.SetValue(Spinnaker::BlackLevelAutoEnums::BlackLevelAuto\_Off);  //Set the absolute value of brightness to 1.5%. cam.BlackLevel.SetValue(1.5); |
| **Spinnaker C++ GenAPI** | CBooleanPtr blackLevelEnabled = nodeMap.GetNode("BlackLevelEnabled"); blackLevelEnabled->SetValue("True");  CFloatPtr blackLevel = nodeMap.GetNode("BlackLevel"); blackLevel->SetValue(1.5); |

### Setting Exposure Time(设置曝光时间)

ExposureTime refers to the amount of time that the camera's electronic shutter stays open. This example sets your camera's exposure/shutter time to 20 milliseconds.

|  |  |
| --- | --- |
| **Spinnaker C++ QuickSpin API** | // Turn off auto exposure cam->ExposureAuto.SetValue(Spinnaker::ExposureAutoEnums::ExposureAuto\_Off);  //Set exposure mode to "Timed" cam->ExposureMode.SetValue(Spinnaker::ExposureModeEnums::ExposureMode\_Timed);  //Set absolute value of shutter exposure time to 20000 microseconds cam->ExposureTime.SetValue(20000); |
| **Spinnaker C++ GenAPI** | CEnumerationPtr exposureAuto = nodeMap.GetNode("ExposureAuto"); exposureAuto->SetIntValue(exposureAuto->GetEntryByName("Off")->GetValue());  CEnumerationPtr exposureMode = nodeMap.GetNode("ExposureMode"); exposureMode->SetIntValue(exposureMode->GetEntryByName("Timed")->GetValue());  CFloatPtr exposureTime = nodeMap.GetNode("ExposureTime"); exposureTime->SetValue(20000); |

### Setting Gain(设置增益)

The following code snippet adjusts gain to 10.5 dB.

|  |  |
| --- | --- |
| **Spinnaker C++ QuickSpin API** | //Turn auto gain off  cam->GainAuto.SetValue(Spinnaker::GainAutoEnums::GainAuto\_Off);  //Set gain to 10.5 dB  cam->Gain.SetValue(10.5); |
| **Spinnaker C++ GenAPI** | CEnumerationPtr gainAuto = nodeMap.GetNode("GainAuto"); gainAuto->SetIntValue(gainAuto->GetEntryByName("Off")->GetValue());  CFloatPtr gainValue = nodeMap.GetNode("Gain"); gainValue->SetValue(10.5); |

### Setting Gamma

The following code snippet adjusts gamma to 1.5.

|  |  |
| --- | --- |
| **Spinnaker C++ QuickSpin API** | // Set the absolute value of gamma to 1.5  cam.Gamma.SetValue(1.5); |
| **Spinnaker C++ GenAPI** | CFloatPtr gamma = nodeMap.GetNode("Gamma"); gamma->SetValue(1.5); |

### Setting White Balance(设置白平衡)

The following code snippet adjusts the white balance's red and blue channels.

|  |  |
| --- | --- |
| **Spinnaker C++ QuickSpin API** | //Set auto white balance to off  cam->BalanceWhiteAuto.SetValue(Spinnaker::BalanceWhiteAutoEnums::BalanceWhiteAuto\_Off);  //Select blue channel balance ratio  cam->BalanceRatioSelector.SetValue(Spinnaker::BalanceRatioSelectorEnums::BalanceRatioSelector\_Blue);  //Set the white balance blue channel to 2  CFloatPtr BalanceRatio = nodeMap.GetNode("BalanceRatio"); BalanceRatio->SetValue(2);  //Set the white balance red channel to 2  cam->BalanceRatioSelector.SetValue(Spinnaker::BalanceRatioSelectorEnums::BalanceRatioSelector\_Red); BalanceRatio->SetValue(2); |
| **Spinnaker C++ GenAPI** | CEnumerationPtr balanceWhiteAuto = nodeMap.GetNode("BalanceWhiteAuto"); balanceWhiteAuto->SetIntValue(balanceWhiteAuto->GetEntryByName("Off")->GetValue());  CEnumerationPtr balanceRatioSelector = nodeMap.GetNode("BalanceRatioSelector"); balanceRatioSelector->SetIntValue(balanceRatioSelector->GetEntryByName("Blue")->GetValue());  CFloatPtr balanceRatio = nodeMap.GetNode("BalanceRatio"); balanceRatio->SetValue(2);  balanceRatioSelector->SetIntValue(balanceRatioSelector->GetEntryByName("Red")->GetValue()); balanceRatio->SetValue(2); |

### Accessing Raw Bayer Data

Raw image data can be accessed programmatically via the getData method of the Spinnaker Image class. In 8 bits per pixel modes such as BayerRG8, the first byte represents the pixel at [row 0, column 0], the second byte at [row 0, column 1], and so on. Image data always starts at row zero and column zero.

|  |  |
| --- | --- |
| **Spinnaker C++ API** | // Assuming image is 640 x 480 resolution. The current pixel format as well as PixelColorFilter indicate the Bayer Tile Mapping for the camera. For example, BayerRG8 is RGGB.  ImagePtr pResultImage = cam.GetNextImage(); char\* data = (char\*)pResultImage->GetData();  // Assuming image is 640 x 480  data[0] = Row 0, Column 0 = red pixel (R) data[1] = Row 0, Column 1 = green pixel (G) data[640] = Row 1, Column 0 = green pixel (G) data[641] = Row 1, Column 1 = blue pixel (B) |

### Setting Number of Software Buffers

The following code snippet adjusts the number of image buffers that the driver initializes for buffering images on your PC to 11 (default is 10).

|  |  |
| --- | --- |
| **Spinnaker C++ API** | Spinnaker::GenApi::INodeMap & sNodeMap = cam->GetStreamNodeMap(); CIntegerPtr StreamNode = sNodeMap.GetNode(“StreamDefaultBufferCount”); INT64 bufferCount = StreamNode->GetValue(); StreamNode->SetValue(11); |

### Event Handling

Spinnaker introduces two event classes: interface events and device events.

### Interface Event

The interface event class is a new feature that is responsible for registering and deregistering user defined interface events such as device arrival and removal.（接口类是一种新的特征，它负责登记和撤销登记用户定义的接口事件，比如设备到达和移除）

|  |  |
| --- | --- |
| **Interface Event C++** | class InterfaceEventsHandler : public InterfaceEvent {  public : InterfaceEventsHandler(){}; virtual ~InterfaceEventsHandler(){};  void OnDeviceArrival() {  std::cout<< "A Camera Arrived" << std::endl;  };  void OnDeviceRemoval( uint64\_tdeviceSerialNumber ) {  std::cout<< "A Camera was removed with serial number: " << deviceSerialNumber << std::endl;  };  };  InterfaceEventsHandler handler; cam->RegisterEvent(handler); |

### Device Event

The device event class is responsible for registering and deregistering user defined device events such as start or end of exposure. （接口类是一种新的特征，它负责登记和撤销登记用户定义的设备事件，比如开始或者结束曝光）

|  |  |
| --- | --- |
| **Device Event C++** | // Select the Exposure End event  Spinnaker::GenApi:: CEnumerationPtr pEnum = nodeMap .GetNode( "EventSelector" ); pEnum->SetIntValue(pEnum->GetEntryByName( "EventExposureEnd" )->GetValue());  // Turn on the Event notification for Exposure End Event  Spinnaker::GenApi:: CEnumerationPtr pBool = nodeMap .GetNode( "EventNotification" ); pBool->SetIntValue(1);  // Once Exposure End Event is detected, the OnDeviceEvent function will be called  classDeviceEventHandler : publicDeviceEvent {  public : DeviceEventHandler(){}; ~DeviceEventHandler(){};  void OnDeviceEvent( Spinnaker::GenICam::gcstring eventNameeventId ) {  std::cout << "Got Device Event with " << eventName << " and ID=" << GetDeviceEventId() << std::endl;  }  };  // Register event handler  DeviceEventHandler allDeviceEventHandler; cam->RegisterEvent(allDeviceEventHandler); |

### Grabbing Images(抓取图像)

You can grab images using the GetNextImage() function. This function returns an image pointer for the current image. The image pointer should be released whenever you are done with the image. Image pointer, being a smart pointer, is automatically released when set to null or when out of scope.(你可以使用GetNextImage()函数抓取图像，这个函数返货当前图像的图像指针，这个图像指针需要被释放当你结束使用图像时。图像指针时一个智能指针，当被设置为空或者超出范围时自动释放)

|  |  |
| --- | --- |
| **Image Acquisition (C++)** | // Begin acquiring images  camBeginAcquisition();>-  ImagePtr pResultImage = cam->GetNextImage();  pResultImage->Release(); |

### Grab Result

In almost all cases, you should check to see if the grabbed image has any errors. To do so, you need to call getImageStatus().

|  |  |
| --- | --- |
| **To check for errors in the image (C++)** | ImageStatus imageStatus = pResultImage->GetImageStatus(); |
| **Available error enums** | /\*\* Status of images returned from GetNextImage() call. \*/  enum ImageStatus {      IMAGE\_NO\_ERROR = 0,  /\*\*< Image is returned from GetNextImage() call without any errors. \*/      IMAGE\_CRC\_CHECK\_FAILED, /\*\*< Image failed CRC check. \*/      IMAGE\_INSUFFICIENT\_SIZE, /\*\*< Image size is smaller than expected. \*/      IMAGE\_MISSING\_PACKETS, /\*\*< Image has missing packets \*/      IMAGE\_LEADER\_BUFFER\_SIZE\_INCONSISTENT, /\*\*< Image leader is incomplete. \*/      IMAGE\_TRAILER\_BUFFER\_SIZE\_INCONSISTENT, /\*\*< Image trailer is incomplete. \*/      IMAGE\_PACKETID\_INCONSISTENT, /\*\*< Image has an inconsistent packet id. \*/      IMAGE\_DATA\_INCOMPLETE, /\*\*< Image data is incomplete. \*/      IMAGE\_UNKNOWN\_ERROR /\*\*< Image has an unknown error. \*/ }; |

### Image Pointer Class（图像指针类）

The image pointer (ImagePtr) is a smart pointer that points to the image object. You can have multiple image pointers pointing to the same object. Smart pointers automatically manage the life time of the object that it points to. You can also re-use the same image pointer object.

Image pointer should always be assigned before using.

|  |  |
| --- | --- |
| **Image Pointer Usage** | ImagePtr pResultImage;  // Retrieve the next received image  pResultImage = cam->GetNextImage();  ImagePtr duplicateImagePtr = pResultImage; |
| **Image Pointer INCORRECT Usage** | // Incorrect usage  ImagePtr illegalImage; illegalImgae->Create( ... ); |
| **Image Pointer CORRECT Usage** | // Correct usage  ImagePtr goodImage; goodImage = Image::Create(...); |

### Error Handling（异常处理）

Spinnaker C++ uses a try catch block for exception handling.（使用try-catch进行异常处理）

|  |  |
| --- | --- |
| **Spinnaker C++ API** | //Assuming Camera& Cam  try  {  cam .Init();  } catch (Spinnaker:: Exception &e) { //Exception handling } |

### Loading and Saving Images（载入和保存图片）

Loading and saving a raw image (.raw) from disk into Spinnaker library can be achieved via Image class's smart pointer.（通过Image类的智能指针可以从磁盘载入和保存图片）

|  |  |
| --- | --- |
| **Loading and Saving Images** | int offlineImageWidth = 1280; int offlineImageHeight = 1024; int offlineOffsetX = 0; int offlineOffsetY = 0; unsigned char\* offlineData;  // Allocate buffer for image  offlineData = (unsigned char\*)malloc(sizeof(unsigned char) \* offlineImageHeight \* offlineImageWidth);  // Create empty image  ImagePtr loadImage = Image::Create(offlineImageWidth, offlineImageHeight, offlineOffsetX, offlineOffsetY, PixelFormatEnums::PixelFormat\_BayerRG8, offlineData);  FILE\* inFile;  // Load image from disk into buffer (offlineData)  inFile = fopen(filename.str().c\_str(), "rb");  fread(offlineData, 1, offlineImageHeight \* offlineImageWidth, inFile);  // Convert image to mono8 data format  loadImage->Convert(PixelFormat\_Mono8);  // Save image  loadImage->Save("offline.jpg"); |

### Chunk Data（数据块）

Chunk data is extra information that the camera can append to each image besides image data. Examples of chunk data include frame counter, image width, image height and exposure time.

Chunk data is comprised of:

* Leader
* Image Data
* Chunk Information (i.e., gain, exposure, image size)
* Trailer

|  |  |
| --- | --- |
| **C++ Enable Chunk Data** | Cam->ChunkSelector ChunkSelector.SetValue(ChunkSelectorEnums ::ChunkSelector\_ExposureTime) ;  Cam->ChunkEnable.SetValue(true);  Cam->ChunkModeActive.SetValue(true); |
| **C++ Retrieve Chunk Data** | const ChunkData& chunkData = rawImage->GetChunkData();  float64\_t currentExposure = chunkData.GetExposureTime(); |

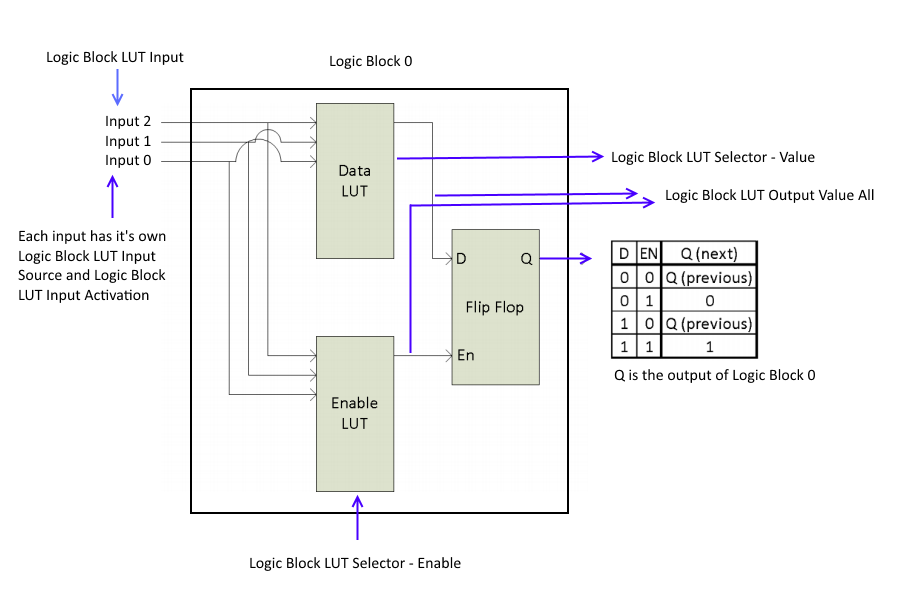
### Sequencer

The purpose of a sequencer is to allow you to programmatically control the acquisition parameters of an image sequence. You can define not only how the images are captured (i.e. the camera feature settings) but also when the camera transitions from one acquisition setting to another. This is akin to a state machine diagram where the states correspond to the sequencer set feature settings, and the transition among states corresponds to a particular event that triggers the state machine to move from one state to another.

To configure sequencer on your camera, you can use SpinView's sequencer tab. Or, to programmatically configure it, you can use the C++ Sequencer source code example that is installed along with Spinnaker SDK.

### Logic Block

A Logic Block is a collection of combinatorial logic and latches that allows the user to create new, custom signals inside the camera. Each Logic Block is comprised of 2 lookup tables (LUT) with programmable inputs, truth tables and a flip flop output. There is a LUT for both the D input (Value LUT) and the enable input (Enable LUT) of the flip flop. Both LUTs have 3 inputs and thus have 8 configuration bits for their truth table.



### Logging（日志）

Spinnaker supports five levels of logging:

* Error—failures that are non-recoverable
* Warning—failures that are recoverable without user intervention
* Notice—information about events such as camera arrival or disconnect, camera initialize, camera start/stop, or modification of a feature
* Info—information about recurring events that are generated with every image
* Debug—information that can be used to troubleshoot the system

You can define the logging level that you want to monitor. Debug levels are inclusive, that is, if you monitor debug level error, you also monitor all logging levels above it.

For a complete C++ and C# example of Logging, please see Spinnaker SDK source code examples. By default, Spinnaker SDK's SpinView application saves all logging data to:

C:\ProgramData\Spinnaker\Logs

|  |  |
| --- | --- |
| **Register Logging (C++)** | SystemPtr system = System::GetInstance();  // Register logging callback class  LogCallback callBackClass; system>RegisterLoggingEvent((Spinnaker::LoggingEvent&)callBackClass);  // Set callback priority level  system->SetLoggingEventPriorityLevel(k\_LoggingLevel);  class LogCallback : Spinnaker::LoggingEvent  {         void OnLogEvent(LoggingEventDataPtr loggingEventDataPtr)        {          ...        }  }; |

### User Set

User set is an on camera non-volatile memory space that you can use to store camera properties such as exposure and gain.

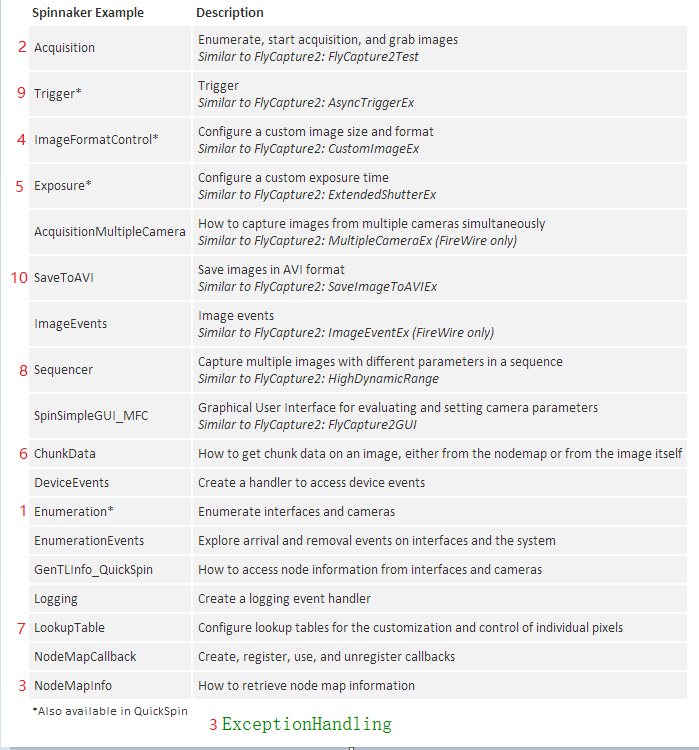
To check if user set supports the feature that you want to save, you can either query the User Set Feature Selector programmatically or run SpinView:

### C:\Program Files\Point Grey Research\Spinnaker\doc\API\C++\html\User-Set.png

## 源代码查看顺序

为了了解程序，我们需要从简到难查看，最好按照下面的程序查看，

理解了标注出来的程序后，再阅读其他没有标注的程序。



1. <http://jingyan.baidu.com/article/ff411625b1311a12e4823784.html> [↑](#footnote-ref-1)
2. <http://www.qcustomplot.com/> [↑](#footnote-ref-2)
3. <http://blog.sina.com.cn/s/blog_a6fb6cc90102v2cs.html> [↑](#footnote-ref-3)
4. <http://blog.csdn.net/xipiaoyouzi/article/details/8450704> [↑](#footnote-ref-4)