电力系统可视化
图形开发包

（图形平台）

Power System Visualization Graph Software Development Kit

程序员开发手册

ePowerGraph SDK User Guide

V 1.3.0

January 27, 2017

目录

文档修订记录

序号	日期	更新内容
1	2016-12-01	文档框架
2	2016-12-14	初稿
3	2017-01-27	修改版
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

1 介绍

ePowerGraph SDK 图形开发包提供了基于 Qt5.x 的若干 C++程序库。在程序库中提供了 G 语言、E 语言、SVG 的解析及显示引擎，使得能够简单高效地对电力系统图模进行显示、交互及编辑。

在开发之前，请阅读《电力系统图形描述规范》、《基于 SVG 的公共图形交换格式》和《电力系统数据标记语言》，了解 G 语言、SVG 和 E 语言的基本概念。

使用本开发包要求开发者具有 Qt 的开发经验，在开发之前请前往 Qt 官网下载 Qt5.5.0 的 32 位版本：

qt-opensource-windows-x86-msvc2010-5.5.0.exe

本开发包采用 Visual Studio 2010 进行编译，因此需要下载安装 Visual Studio 2010。但依开发者的个人喜好，可选择 Visual Studio 2010 或 Qt Creator 作为 IDE。本手册以 Qt Creator 作为 IDE 进行介绍。

1.1 安装 Visual Studio 2010

下载安装 Visual Studio 2010，开发包采用 Visual Studio 2010 旗舰版进行编译，因此建议选择该版本进行安装。安装完成后 VC 可执行文件的路径为：C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\bin。

1.2 安装 Qt5.5.0

下载安装 Qt5.5.0，采用默认路径安装，安装完成后 Qt 可执行文件的路径为：C:\Qt\Qt5.5.0\5.5\msvc2010\bin 。

1.3 安装 ePowerGraph SDK

下载安装 epower_sdk_win32_qt5.5_1.2.0.exe。安装完成后新增环境变量：EPOWER_ROOT。此环境变量是开发包所在的目录。

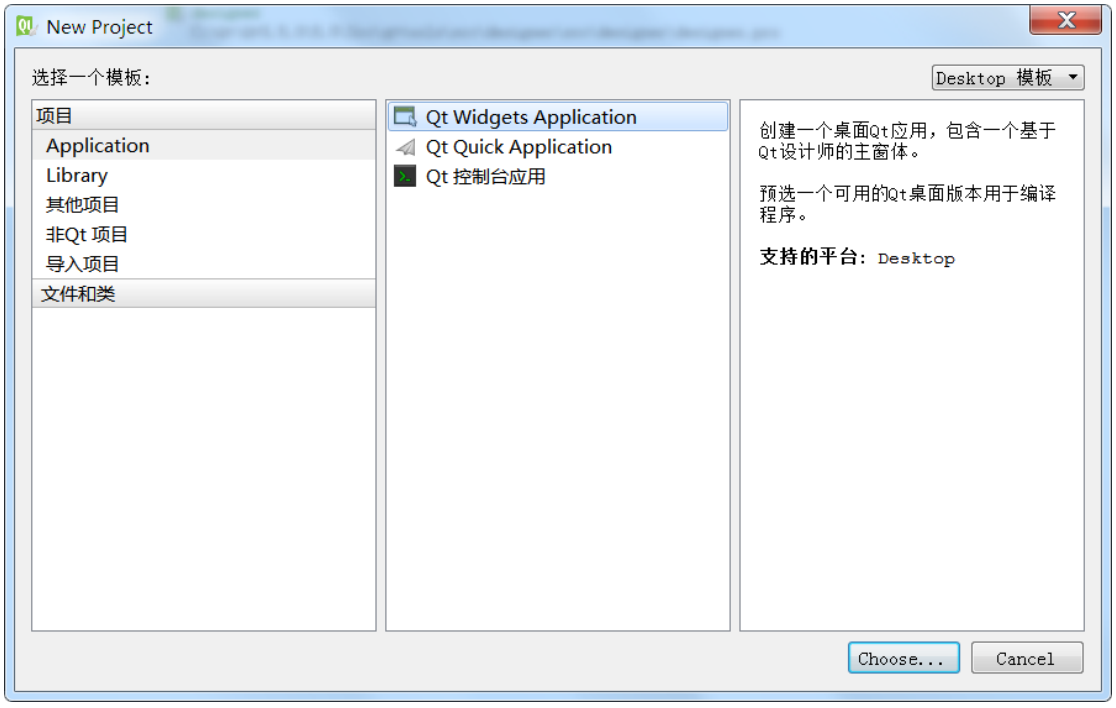
1.4 编译测试

运行“电力系统可视化图形开发包”菜单下的控制台，即$EPOWER_ROOT$\bin\epower_sdk.bat。该批处理脚本运行 VC 的环境设置脚本，并将 Qt 及 ePowerGraph SDK 的动态库路径添加到环境变量 PATH 中。如果 VS2010 和 Qt 的安装路径与上述的默认路径不一致，请打开该批处理脚本进行修改。

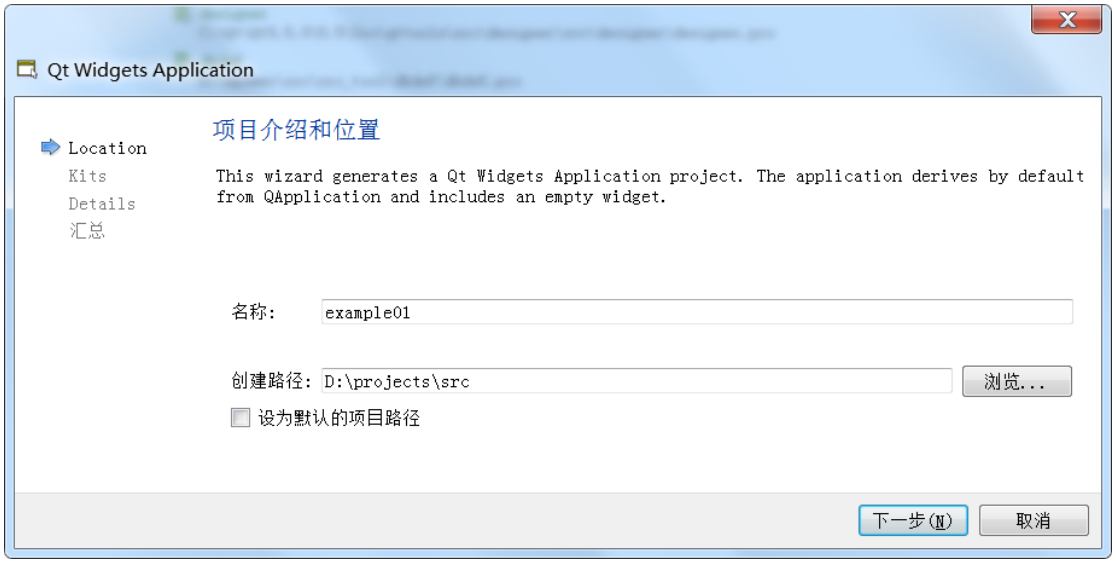
在控制台下启动 QtCreator。如下：

>qtcreator

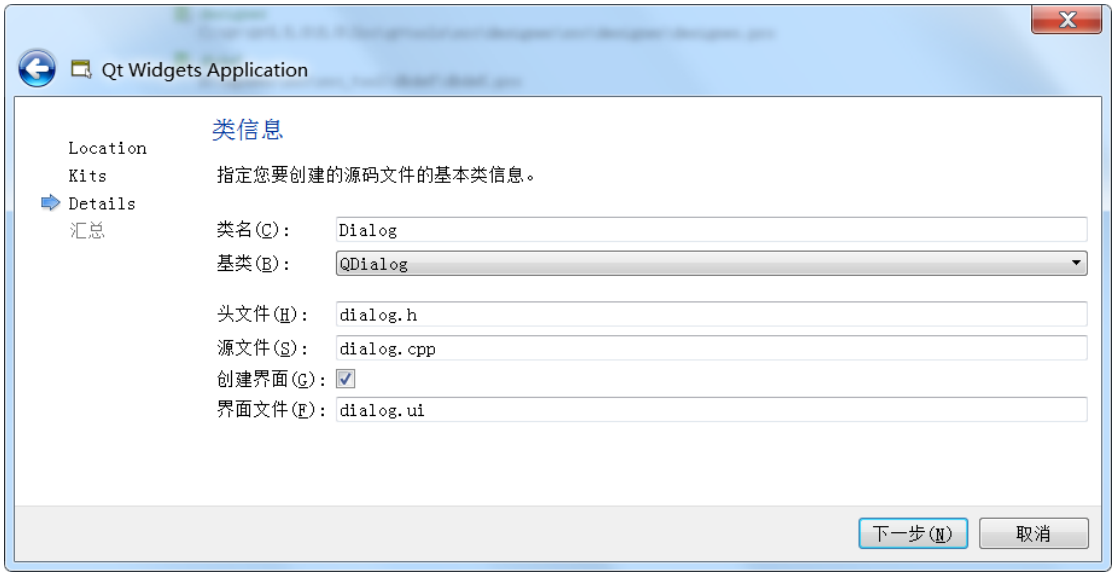
- 新建一个项目（example01），选择 Application->Qt Widgets Application:



- 设置项目名称及位置：



- 选择类名及基类 QDialog：

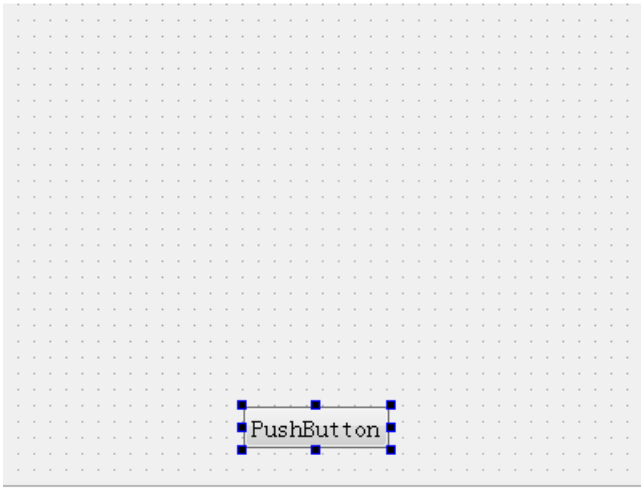


- 添加外部库

  手动修改 pro 文件，添加 ePowerGraph SDK 中的 Qt 扩展包 auxlib 的头文件及库文件：

```
QT          += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = example01
TEMPLATE = app
INCLUDEPATH += $$(EPOWER_ROOT)/src/auxlib
CONFIG(debug, debug|release):LIBS += $$(EPOWER_ROOT)/lib/auxlibd.lib
CONFIG(release, debug|release):LIBS += $$(EPOWER_ROOT)/lib/auxlib.lib
SOURCES += main.cpp\
           dialog.cpp
HEADERS   += dialog.h
FORMS     += dialog.ui
```

- 在 dialog.ui 中添加一个按钮：



- 在 dialog.h 中添加如下代码：

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>

namespace Ui {
class Dialog;
}

class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();

private slots:
    void sayHello();

private:
    Ui::Dialog *ui;
};

#endif // DIALOG_H
```

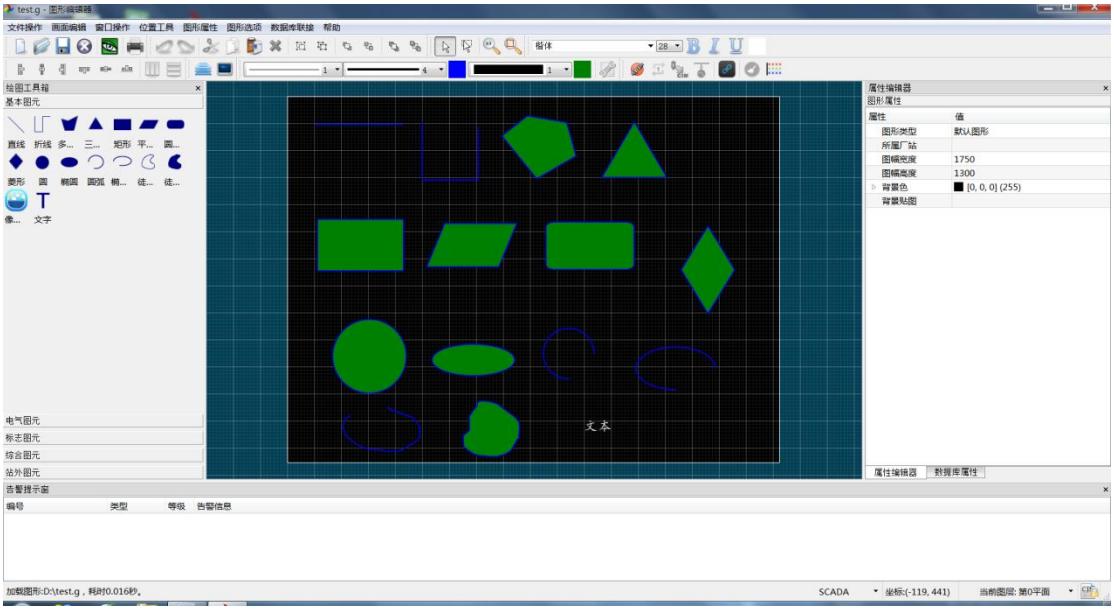
- 在 dialog.cpp 中添加如下代码：

```
#include "dialog.h"
#include "ui_dialog.h"
#include "ztooltip.h"

Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
    connect(ui->pushButton, SIGNAL(clicked(bool)),this, SLOT(sayHello()));
}

Dialog::~Dialog()
{
    delete ui;
}

void Dialog::sayHello()
{
    ZToolTip::showTip(this, "hello ePowerGraph SDK!", 5000);
}
```

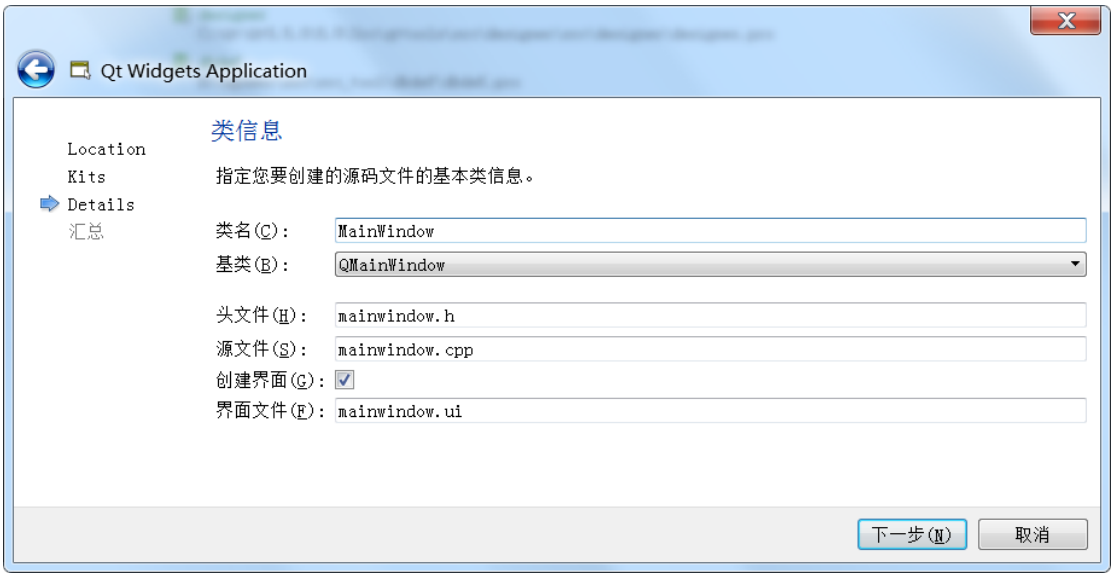
- 编译运行，如果点击按钮后得到如下的运行结果则开发环境配置成功。

2 显示图形文件

电力系统 G 语言图形是承载电力系统运行信息的一种图形格式。该格式中包含了电力系统设备及相互连接关系的描述。首先打开图形编辑器绘制如下的基础图形文件，并保存为 test.g。



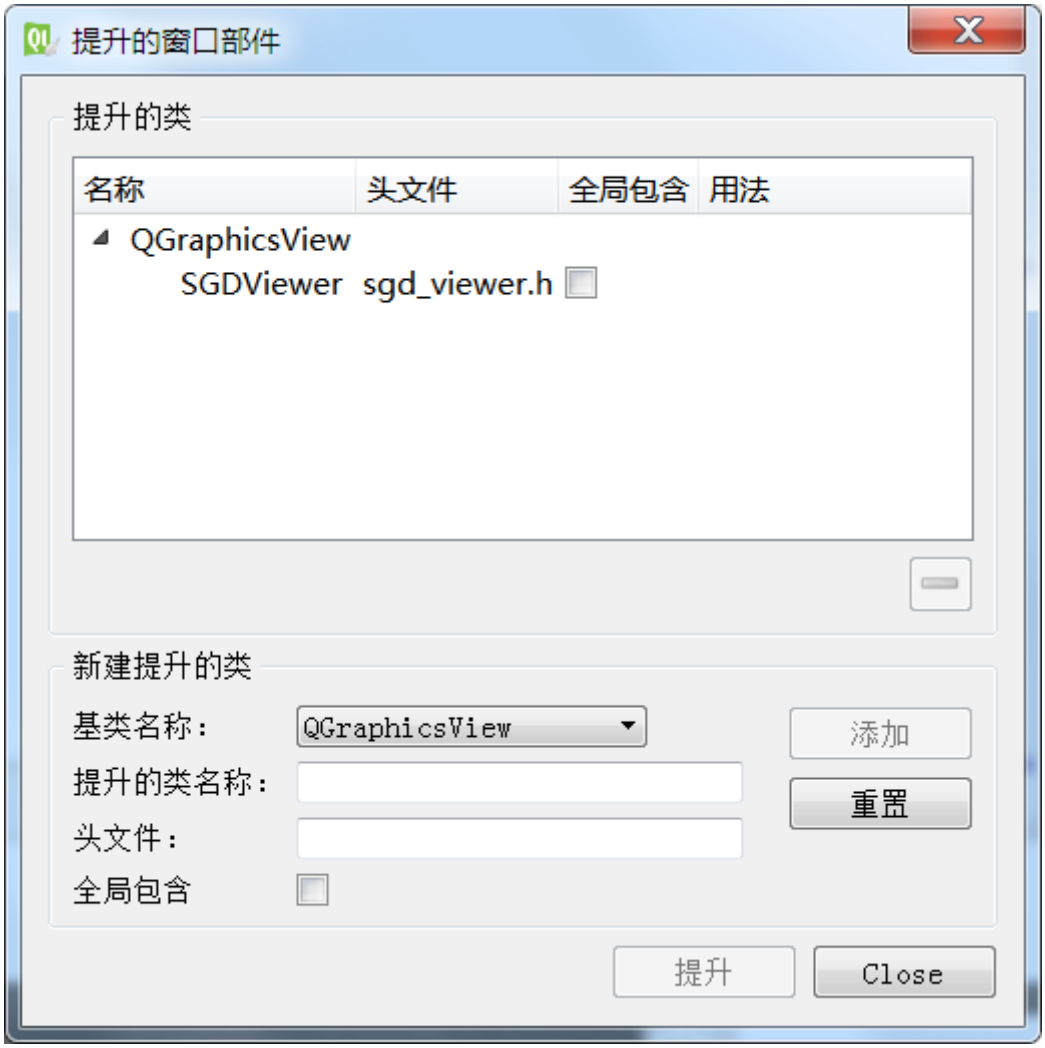
- 创建基于 QMainWindow 的项目（example02）。



- 打开 pro 文件，添加 ePowerGraph SDK 的头文件及库文件。因为此处需要用到电力系统图形的相关功能，因此除了 Qt 扩展包（auxlib）外，还需要添加电力系统图形库 sgdlib。

```
QT          += core gui xml

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = example02
TEMPLATE = app

INCLUDEPATH += $$(EPOWER_ROOT)/src/auxlib
INCLUDEPATH += $$(EPOWER_ROOT)/src/sgdlib

CONFIG(debug, debug|release):LIBS += $$(EPOWER_ROOT)/lib/auxlibd.lib
CONFIG(debug, debug|release):LIBS += $$(EPOWER_ROOT)/lib/sgdlibd.lib

CONFIG(release, debug|release):LIBS += $$(EPOWER_ROOT)/lib/auxlib.lib
CONFIG(release, debug|release):LIBS += $$(EPOWER_ROOT)/lib/sgdlib.lib

SOURCES += main.cpp\
           mainwindow.cpp

HEADERS   += mainwindow.h

FORMS      += mainwindow.ui
```

- 打开 mainwindow.ui，在主界面上添加一个 Graphics View，并进行布局。将添加的 Graphics View 提升为 SGDViewer。注意头文件名称为 sgd_viewer.h。sgdlib 库中所有的头文件均有前缀 sgd。这个命名约定对 sgdlib 库中所有头文件均有效。


提升的窗口部件
提升的类
名称
头文件
全局包含
用法
QGraphicsView
SGDViewer
sgd_viewer.h
新建提升的类
基类名称：
QGraphicsView
提升的类名称：
头文件：
全局包含
添加
重置
提升
Close

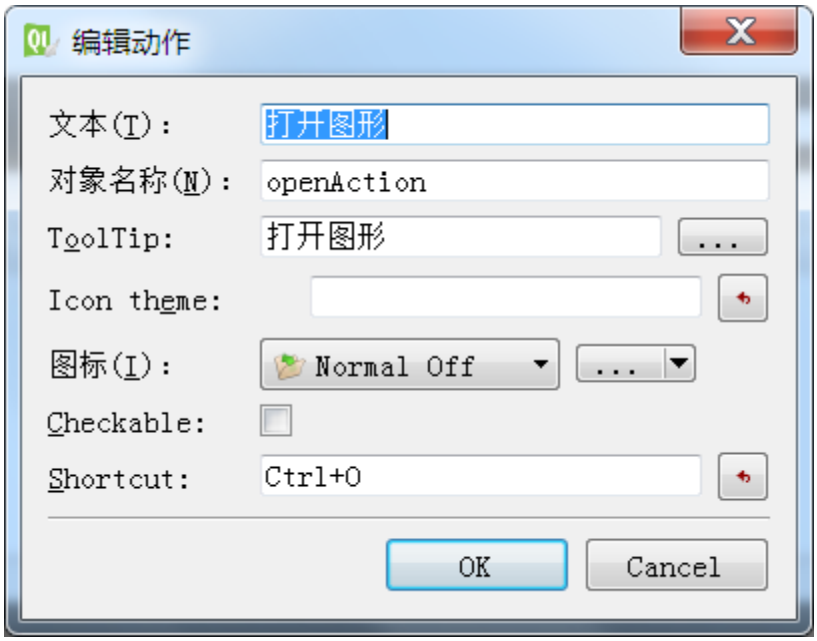
- 在头文件 mainwindow.h 中添加图形场景对象指针 m_scene。头文件中的成员变量一般添加前缀 m_，并且会尽量采用前置声明。

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class SGDScene;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow*    ui;
    SGDScene*          m_scene;
};

#endif // MAINWINDOW_H
```

- 在 mainwindow.cpp 的构造函数中对图形场景对象进行初始化，并将当前图形场景与视图进行关联。

```
#include "sgd_scene.h"
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    m_scene = new SGDScene(this);
    ui->graphicsView->setScene(m_scene);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

- 添加菜单【文件】->【打开图形】和工具栏图标



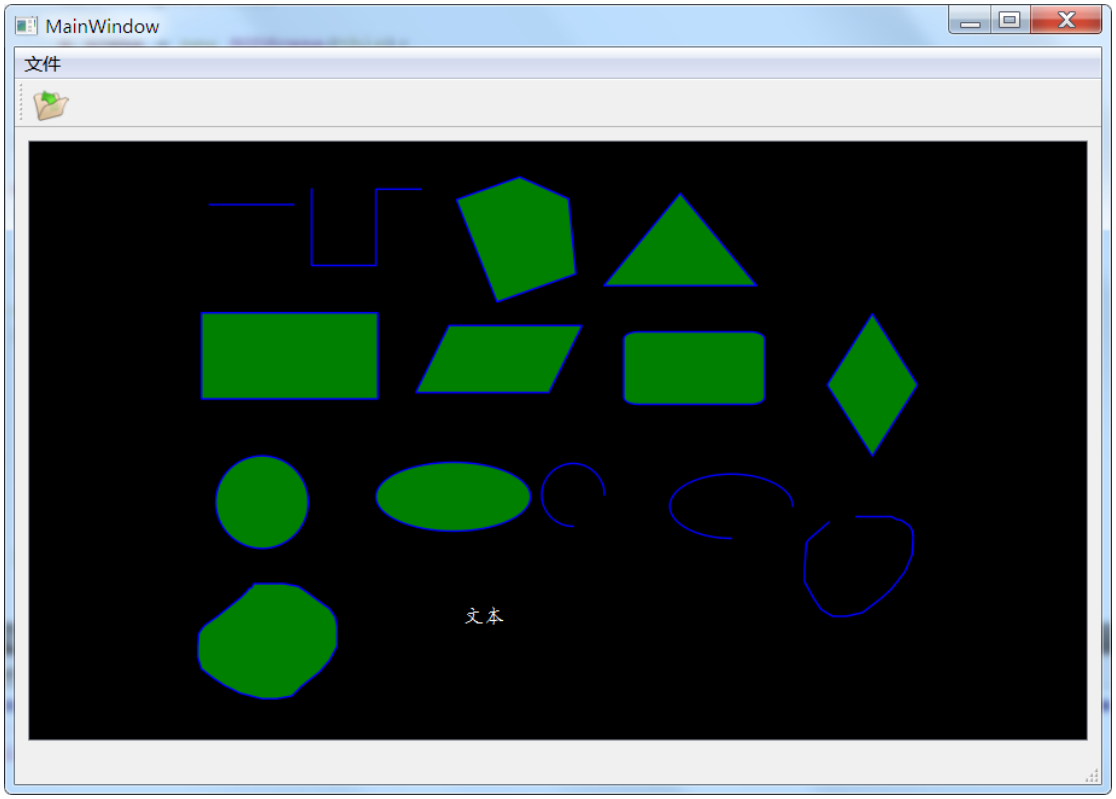
- 在 mainwindow.h 中增加一个私有槽函数：

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class SGDScene;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void open();

private:
    Ui::MainWindow *ui;
    SGDScene*           m_scene;
};
```

- 添加菜单的响应代码，在构造函数中将【打开图形】的动作与槽 open()相关联。并在 open()中加载选中的图形文件。load()函数的第一个参数为图形文件的路径，第二个参数为 true，表示该路径将添加到导航路径中，第三个参数为 true，表示图形文件加载后将立即刷新数据。第二个和第三个参数的作用稍后介绍。

```
#include <QFileDialog>
#include "sgd_scene.h"
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    m_scene = new SGDScene(this);
    ui->graphicsView->setScene(m_scene);

    connect(ui->openAction, SIGNAL(triggered(bool)), this, SLOT(open()));
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::open()
{
    QString fileName = QFileDialog::getOpenFileName(this,
                                                     "打开",
                                                     ".",
                                                     "G 语言图形文件(*.g) ;;SVG 图形文件(*.svg)");
   if( !fileName.isEmpty() )
    {
        m_scene->load( fileName, true, true );
    }
}
```

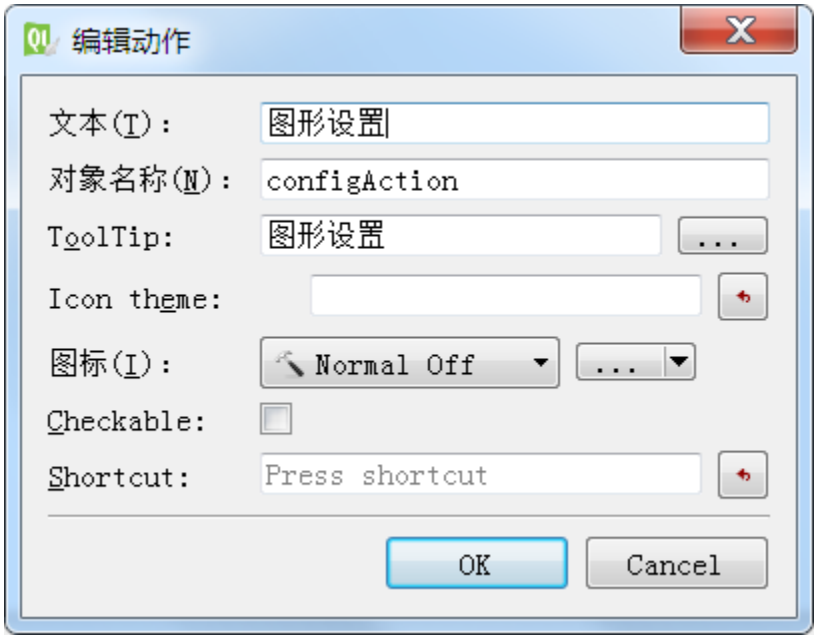
- 编译运行后，打开 test.g 图形文件，结果如下：



3 图形设置

电力系统 G 语言图形中像素图、大部分电气图元等均采用引用图元的方式。在 G 语言图形文件中，画面跳转时的文件名及引用图元文件名均不含文件目录，因此需要配置 G 语言图形中所用到的图片、图元及图形所在的目录。

- 在上一个项目的基础上新建项目（example03），添加菜单【选项】->【图形设置】和工具栏图标。如下图所示：



- 修改 mainwindow.h，添加新的槽函数 config()，代码如下：

```
private slots:
    void open();
    void config();
```

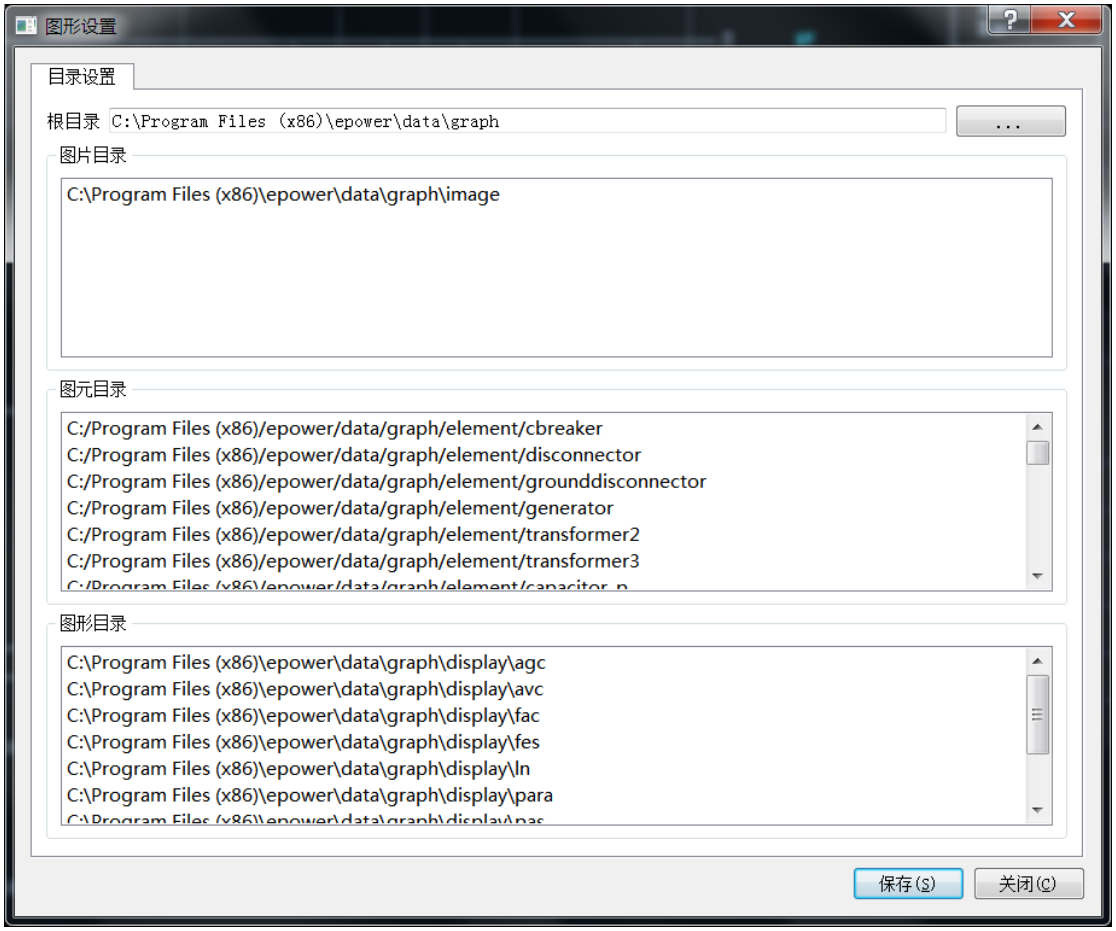
- 修改 mainwindow.cpp，在构造函数中关联信号与槽：

```
connect(ui->openAction, SIGNAL(triggered(bool)), this, SLOT(open()));
connect(ui->configAction, SIGNAL(triggered(bool)), this, SLOT(config()));
```

- 修改 mainwindow.cpp,实现 config()函数。并在创建图形场景前加载配置。

```
void MainWindow::config()
{
    SGDConfigDialog dlg(SGDConfigDialog::PAGE_DIR, this);
    dlg.exec();
}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    SGDConfig::loadSettings();
    m_scene = new SGDScene(this);
    ui->graphicsView->setScene(m_scene);

    connect(ui->openAction, SIGNAL(triggered(bool)), this, SLOT(open()));
    connect(ui->configAction, SIGNAL(triggered(bool)), this, SLOT(config()));
    connect(ui->goPrevAction,SIGNAL(triggered(bool)), m_scene,
SLOT(gotoPreviousGraph()));
    connect(ui->goNextAction,SIGNAL(triggered(bool)), m_scene,
SLOT(gotoNextGraph()));
    connect(ui->zoomInAction,SIGNAL(triggered(bool)), ui->graphicsView,
SLOT(zoomIn()));
    connect(ui->zoomOutAction,SIGNAL(triggered(bool)), ui->graphicsView,
SLOT(zoomOut()));
    connect(ui->dragAction, SIGNAL(toggled(bool)), ui->graphicsView,
SLOT(scrollHandDrag(bool)));
}
```

- 编译运行后点击【图形设置】查看默认的目录配置。需根据实际的目录进行配置，在电力系统可视化图形建模系统的安装目录下有预装的图元及图片目录，可将根目录设置为如下路径：



4 画面导航

除了标志调用中的画面跳转外，在浏览图形的过程中还需提供上一幅、下一幅的画面导航功能。

在上一项目的基础上新建项目(example04)，添加菜单【画面】->【上一幅】、【下一幅】，并增加相应的工具栏图标。

在项目 example02 中，在加载图形文件时已将图形文件添加到导航列表中。在构造函数中添加如下代码即可实现画面导航功能。

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    m_scene = new SGDScene(this);
    ui->graphicsView->setScene(m_scene);

    connect(ui->openAction, SIGNAL(triggered(bool)), this, SLOT(open()));
    connect(ui->configAction, SIGNAL(triggered(bool)), this, SLOT(config()));
    connect(ui->goPrevAction,SIGNAL(triggered(bool)), m_scene, SLOT(gotoPreviousGraph()));
    connect(ui->goNextAction,SIGNAL(triggered(bool)),m_scene, SLOT(gotoNextGraph()));
}
```

5 缩放拖动图形

在浏览图形时可采用鼠标滚轮缩放图形，在没有鼠标滚轮的情况下，需提供菜单进行图形的缩放和拖动。

在上一项目的基础上新建项目(example05)，添加菜单【图形】->【放大】、【缩小】、【拖动】，并增加相应的工具栏图标。

在构造函数中添加如下代码即可实现图形缩放及拖动功能。

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    m_scene = new SGDScene(this);
    ui->graphicsView->setScene(m_scene);

    connect(ui->openAction, SIGNAL(triggered(bool)), this, SLOT(open()));
    connect(ui->configAction, SIGNAL(triggered(bool)), this, SLOT(config()));
    connect(ui->goPrevAction,SIGNAL(triggered(bool)), m_scene, SLOT(gotoPreviousGraph()));
    connect(ui->goNextAction,SIGNAL(triggered(bool)), m_scene, SLOT(gotoNextGraph()));
    connect(ui->zoomInAction,SIGNAL(triggered(bool)), ui->graphicsView, SLOT(zoomIn()));
    connect(ui->zoomOutAction,SIGNAL(triggered(bool)), ui->graphicsView, SLOT(zoomOut()));
    connect(ui->dragAction, SIGNAL(triggered(bool)), ui->graphicsView, SLOT(scrollHandDrag(bool)));
}
```

6 遥测数据

除了显示静态的图形文件外，更多有用的信息需要通过动态数据才能展示出来。G 语言图形规范中展示动态数据的图元有很多，如动态文本、工况图元、状态图元、断路器、刀闸、表格、曲线等。本示例将展示如何将遥测数据通过动态文本显示在静态画面上。

在 G 语言规范中，大多数展示动态数据的图元都有一个共同属性：设备 ID。设备 ID 一般采用 8 字节编码，字节 0-1 表示表号，字节 2-3 表示列号，字节 4-7 表示行号。通过该设备 ID 可定位到表中的指定数据。如下所示：

0	1	2	3	4	5	6	7

动态文本所要显示的数据与具体的应用有关，图形平台提供了三种机制来更新数据：

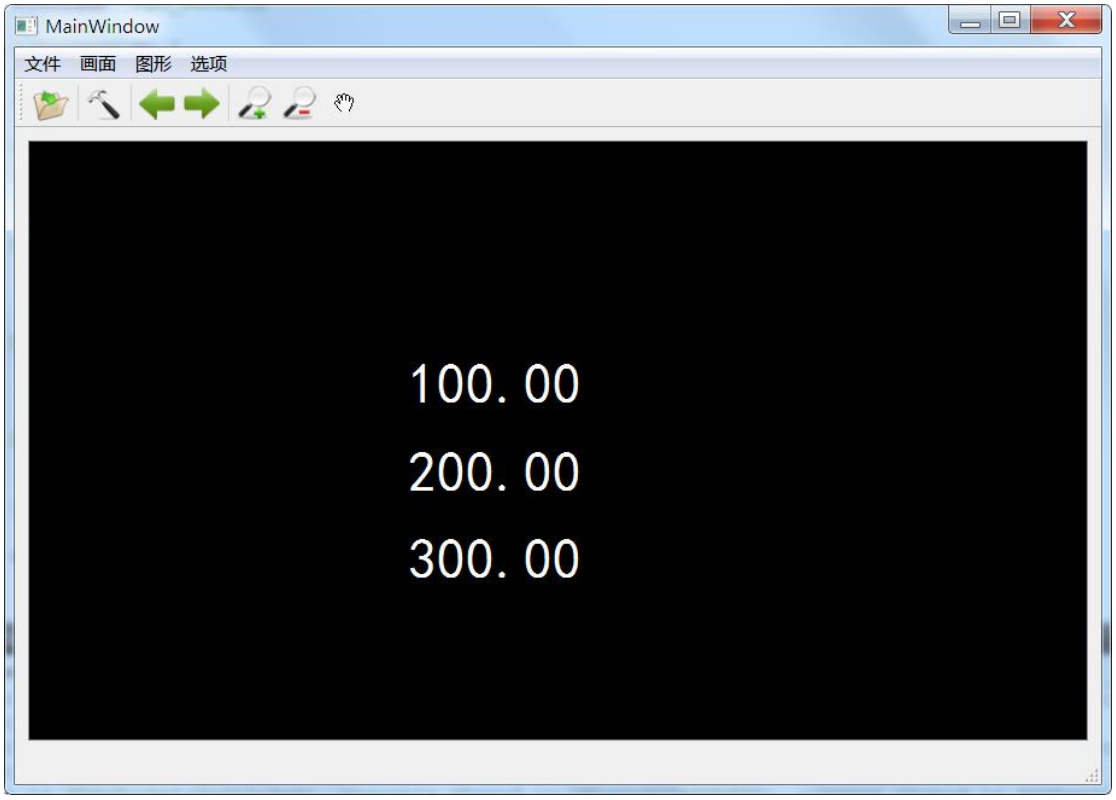
1) 从图形库内置的数据缓存中获取数据。当加载图形文件时自动从数据缓存中读取数据并显示。在示例 example02 中，调用 load()函数时传入的第三个参数为 true，则在加载图形文件时更新图形中的动态文本等图元的数据。
2) 从数据接口插件中获取数据。如果在数据缓存中无法获取到动态文本的数据，则会尝试从数据接口插件中获取。
3) 采用网络数据包来刷新画面。图形场景中内置了网络数据包的接收服务，当有新数据时可采用发送网络数据更新事件来实现本地所有画面的同步或异步更新。

新建图形文件，绘制三个动态文本，为了演示的方便，此处未采用标准的 8 字节编码 ID，而是简单地将设备 ID 设置为 1、2、3，图形文件保存为 yc.g。

新建项目 example06,基类选为 QDialog,在 pro 中加入 auxlib、sgdlib 两个库的头文件及库文件。添加两个按钮，一个按钮为“缓存数据”，一个按钮为“刷新数据”。头文件 dialog.h 中添加一个成员变量：SGDMemDb　　m_memDb;并添加对应的头文件 sgd_memdb.h。dialog.cpp 实现代码如下：

```
#include "sgd_event.h"
#include "dialog.h"
#include "ui_dialog.h"
Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);

    connect(ui->pushButton, SIGNAL(clicked(bool)), this, SLOT(cacheData()));
    connect(ui->pushButton_2, SIGNAL(clicked(bool)), this, SLOT(refreshData()));
}
Dialog::~Dialog()
{
    delete ui;
}
void Dialog::cacheData()
{
    QHash<QString, QString> data;
    data.insert("1", "100");
    data.insert("2", "200");
    data.insert("3", "300");
    m_memDb.setData(data);
}
void Dialog::refreshData()
{
    SGDEventSender   sender;
    sender.sendEvent( SGDEvent(SGDEvent::EVENT_YC_UPDATED, "1", QString::number(rand()%1000)) );
    sender.sendEvent( SGDEvent(SGDEvent::EVENT_YC_UPDATED, "2", QString::number(rand()%1000)) );
    sender.sendEvent( SGDEvent(SGDEvent::EVENT_YC_UPDATED, "3", QString::number(rand()%1000)) );
}
```

运行 example06.exe 后，点击按钮【缓存数据】。保持 example06.exe 运行以确保缓存数据不被释放，然后再运行 example05.exe，打开 yc.g 后会看到三个动态文本图元均已取到数据。如下图所示：



点击按钮【刷新数据】，将向 example05.exe 的当前画面发送随机的遥测数据。多运行几次 example05.exe，刷新数据后可同步更新当前打开的所有画面。

7 遥信数据

G 语言图形中断路器、刀闸、工况图元、状态图元、其他、光字牌、保护图元等均可展示遥信数据。编辑图形文件 yx.g，绘制上述图元，并将设备 ID 依次设置 1、2、3、4、5、6、7。

新建项目 example07，基类为 QDialog，并添加两个按钮【缓存数据】和【刷新数据】。主要的实现代码为：

```
#include "sgd_event.h"

void Dialog::cacheData()
{
    QHash<QString, QString> data;
    for(int i = 0; i < 7; ++i)
    {
        data.insert(QString::number(i+1), "1");
    }
    m_memDb.setData(data);
}

void Dialog::refreshData()
{
    SGDEventSender   sender;
    for(int i = 0; i < 7; ++i)
    {
        sender.sendEvent( SGDEvent(SGDEvent::EVENT_YX_UPDATED,
                                   QString::number(i+1),
                                   QString::number(rand()%2)) );
    }
}
```

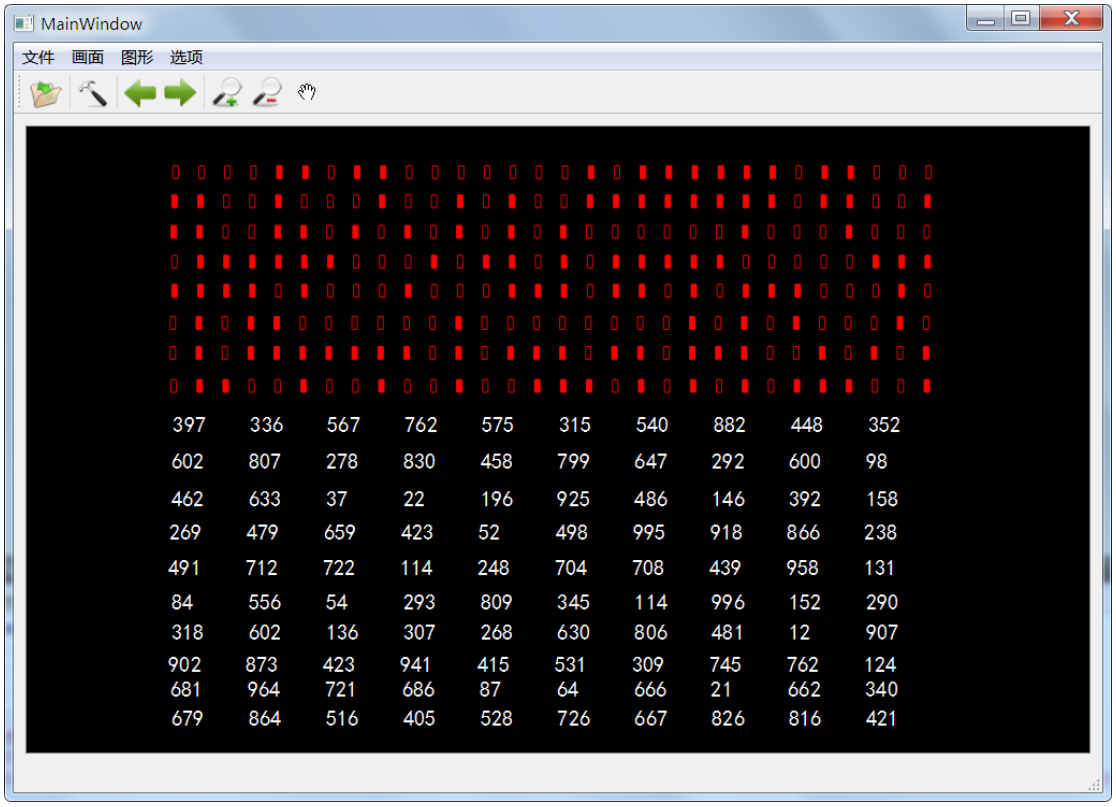
8　批量更新数据

上两节的示例中均采用了同步发送数据更新事件的方式来刷新数据，当更新少量的量测数据时此方法适用。但是当需要批量更新一批数据时，就会导致程序阻塞，影响人机交互。批量更新数据有两种方式：

一是采用缓存加同步发送事件的方式；

二是采用异步发送事件的方式。

新建图形文件 yxyc.g。绘制大量断路器和动态文本，并依次将断路器和动态文本的设备 ID 进行编号。如下图所示：



参考上节示例新建工程 example08，添加两个按钮，分别实现同步刷新和异步刷新。

8.1 同步更新

同步批量更新数据时，首先将遥信遥测数据写入缓存中，然后再发送一次图形更新事件。

先运行 example05,打开 yxyc.g 图形文件,然后运行 example08，点击同步刷新可看到断路器和动态文本的变化。

```
#include "sgd_event.h"

void Dialog::refreshSync()
{
    QHash<QString, QString> measValues;
    for(int i = 1; i <= 1000; ++i)
    {
        measValues.insert(QString("yx%1").arg(i), QString::number(rand()%2));
    }
    for(int j = 1; j <= 1000; ++j)
    {
        measValues.insert(QString("yc%1").arg(j), QString::number(rand()%1000));
    }
    SGDMemDb db;
    db.append( measValues );

    SGDEventSender   sender;
    sender.sendEvent( SGDEvent(SGDEvent::EVENT_GRAPH_UPDATED, "", "") );
}
```

8.2 异步更新

异步发送事件是通过线程将指定的遥信遥测数据按照一定的时间间隔（默认为 15 毫秒）进行发送。

首先在 dialog.h 中添加头文件：

```
#include "sgd_event.h"
```

定义事件发送线程的成员变量：

```
SGDEventSendThread   m_thread;
```

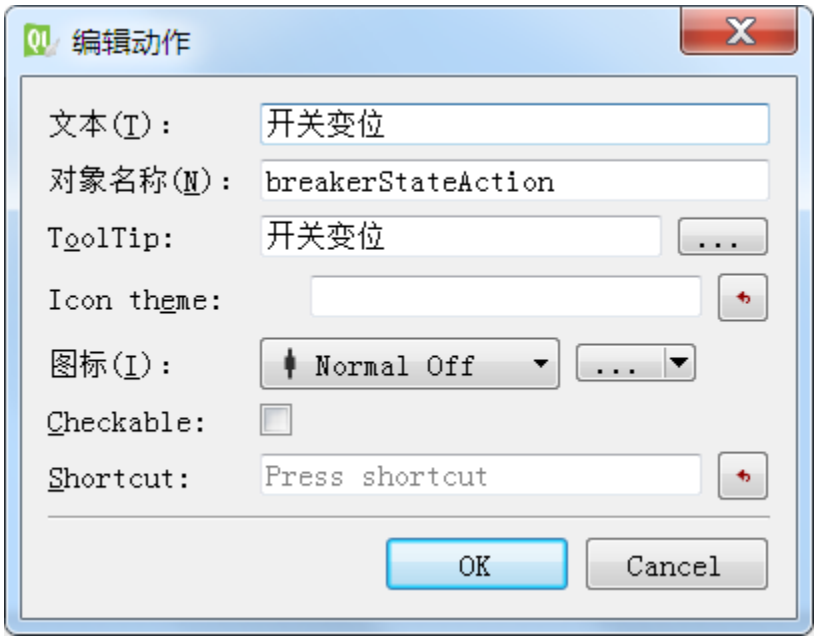
异步更新的实现代码如下：

```
void Dialog::refreshAsync()
{
    QHash<QString, QString> yxValues;
    for(int i = 1; i <= 1000; ++i)
    {
        yxValues.insert(QString("yx%1").arg(i), QString::number(rand()%2));
    }
    QHash<QString, QString> ycValues;
    for(int j = 1; j <= 1000; ++j)
    {
        ycValues.insert(QString("yc%1").arg(j), QString::number(rand()%1000));
    }
    m_thread.appendEvent( SGDEvent(SGDEvent::EVENT_YX_UPDATED, yxValues) );
    m_thread.appendEvent( SGDEvent(SGDEvent::EVENT_YC_UPDATED, ycValues) );
    m_thread.start();
}
```

9 响应鼠标事件

在 example05 的基础上新建项目 example09,在此项目中将响应鼠标右键单击事件，弹出右键菜单，并实现“开关变位”的功能。

- 新建一个动作：breakerStateAction，如下图:



- 在 mainwindow.cpp 的构造函数中将开关变位的动作与槽函数 **changeSwitchState()**连接，并将图形场景的右键菜单信号与槽函数 **requestContextMenu()**连接。

```
connect(ui->breakerStateAction, SIGNAL(triggered(bool)),
this, SLOT(changeSwitchState()));
connect(m_scene,  SIGNAL(requestContextMenu(QGraphicsItem*)),  this,
SLOT(requestContextMenu(QGraphicsItem*) ) );
```

- 如果鼠标右键点击处的图元为断路器，则弹出右键菜单。开关变位时，先获取当前开关的状态，再将开关状态进行变位。实现代码如下：

```
void    MainWindow::requestContextMenu(QGraphicsItem* item)
{
      m_currentItem = item;
      if( item != NULL )
      {
            if( qElementType(item) == SGD::CBreaker )
            {
                  QMenu contextMenu( this );
                  contextMenu.addAction( ui->breakerStateAction );
                  contextMenu.exec( QCursor::pos() );
            }
      }
}

void MainWindow::changeSwitchState()
{
      if( qElementType(m_currentItem) == SGD::CBreaker )
      {
            SGDCBreaker* cbreaker =
qgraphicsitem_cast<SGDCBreaker*>(m_currentItem);
            bool closed = cbreaker->isClosed();
            cbreaker->setClosed( !closed );
      }
}
```

10 响应键盘事件

在 example09 的基础上新建项目 example10,在此项目中将响应键盘事件，如果按键是 F5，则采用随机数刷新量测值。

- 绑定图形场景的按键信号到槽函数上：

```
connect(m_scene, SIGNAL(keyPressed(const QKeySequence&)),
this, SLOT(keyPressed(const QKeySequence&)));
```

- 当检测到按键是 F5 时，刷新当前画面：

```
void MainWindow::keyPressed(const QKeySequence& key)
{
    if( key == Qt::Key_F5 )
    {
        QHash<QString, QString> measValues;
        for(int i = 1; i <= 1000; ++i)
        {
            measValues.insert(QString("yx%1").arg(i), QString::number(rand()%2));
        }
        for(int j = 1; j <= 1000; ++j)
        {
            measValues.insert(QString("yc%1").arg(j), QString::number(rand()%1000));
        }
        SGDMemDb db;
        db.append( measValues );

        SGDEventSender   sender;
        sender.sendEvent( SGDEvent(SGDEvent::EVENT_GRAPH_UPDATED, "", "") );
    }
}
```