

**A PROJECT REPORT ON**

**DRAWING TOOL KIT AND SCRIBBLEPAD USING QT**

BY

Ms. PRIYANKA GILL

BTECH FIRST YEAR, CSE

SRM AP UNIVERSITY

AT

**CENTER FOR AIRBORNE SYSTEM (CABS)**

**DEFENCE RESEARCH & DEVELOPMENT ORGANISATION**

**UNDER THE GUIDANCE OF**

SMT CV JANARDINI, SC “E”



**ACKNOWLEDGMENT**

Valuable inputs and helpful co-operation from many people together make an endeavour successful.

I would like to convey my deepest regards to Shri M S Easwaran, Director, Centre for Airborne Systems (CABS), Defense Research and Development Organisation(DRDO), Bangalore for providing me the most valuable opportunity to conduct a project in this esteemed organisation.

I want to extend my special regards to Smt. Rekha Rani Sinha,Sc ‘G’, Group Director, CABS, Shri S Kannan, Sc F, Division Head and Smt. M R Bhuvaneshwari, Sc ‘F’, H R Coordinator for helping and supporting in every regard.

My heartfelt gratitude towards Mrs CV Janardini, Sc ‘E’, who not only guided me with her expertise throughout the project, but also provided valuable suggestions for the improvement of this report and steered me towards a path of success.

AN OVERVIEW OF DRDO AND CABS

Defence Research & Development Organisation (DRDO) works under Department of Defense Research and Ministry of Defense. DRDO dedicatedly works on enhancing self-reliance in Defense Systems and undertakes design and development which results in world class weapon systems and equipment in accordance to the expressed needs and qualitative requirements laid down by the three forces.

DRDO is working on various areas of military technology which includes aeronautics, armaments, combat vehicles, electronics, instrumentation engineering systems, missiles, materials, naval systems, advanced computing, simulation and life sciences. DRDO while striving to meet the cutting edge weapons technology requirements provides ample spinoff benefits to the society at large thereby contributing to the nation building. DRDO consists of around 54 labs which work together to achieve their missions.

Centre for Airborne Systems (CABS) is one of the DRDO labs under Aeronautics discipline.

CABS is essentially a system house and an integration agency utilizing all available infrastructure and expertise in the country for electronic force multiplier technology focussing attention on Airborne Early warning and control systems (AEW & C).

CABS houses various unique facilities such as Lightning test facility, Avionics flight test bed,

software development facility and Planar near File Management (PNFM) facility. The Aero and Structure group in CABS is engaged in the design of structures for AEW &C project and structural modification for housing various subsystems on the Aircraft platform.

Contents

[ABSTRACT 7](#_Toc11411979)

[1. INTRODUCTION 8](#_Toc11411980)

[2. FAMILIARIZATION WITH Qt 9](#_Toc11411981)

[3. DRAWING TOOL KIT 9](#_Toc11411982)

[4. SCRIBBLE PAD 13](#_Toc11411983)

[5. CONCLUSION 16](#_Toc11411984)

[6. REFERENCES 16](#_Toc11411985)

[APPENDIX - SOURCE CODE 17](#_Toc11411986)

# ABSTRACT

*Drawing toolkit is an underlying concept upon which any graphical display software is built upon. Basic shapes of polygon, circle and lines are the building blocks to complex shapes. My project aims at building a drawing tool kit of the basic shapes using the QT tool kit. As an extension of the project I have also built a scribble pad which will give a user freedom to create his own drawing.*

# INTRODUCTION

This report details a summer internship carried out at Centre for Airborne Systems DRDO, Bangalore. I worked in Group 7 of the organization which specializes in Data Handling and Display Systems. My Group Director for the project was Mrs Rekha Rani Sinha, Scientist G and Division Head was Shri S Kannan. My guide for the project was Mrs CV Janardini. I have had immense pleasure in working with DRDO for one month. In this time period of one month I have gained enough knowledge of the organisation and the excellent work that they do. It all seems very interesting to me as I have only completed first year of Btech and I am very sure this practical knowledge provided will help me in my future studies.

I started my internship having absolutely zero knowledge on Qt, a graphical toolkit, which was the toolkit on which I had to build my project. I started studying books and tutorials on the tool kit, wrote and compiled a few of the basic programs and discovered how little code the author needed to produce a complete game, including the menus. Of course, there were a number of constructs that needed explanation, but after a short time I was hooked. The Qt library turned out not only to have a very extensive

collection of all kinds of useful widgets, but in addition had standard algorithms, data structures, and

other non-graphic classes that made programming with C++ so intuitive, in a way that I had never seen before in any other toolkit.

My task for this internship was to design and develop a drawing tool kit and subsequently extend it to cater it for a scribble pad on which a user can do free hand drawing. I developed the code using C++ on Ubuntu 18.04 Operating system using the Qt 5.0 toolkit.

# FAMILIARIZATION WITH Qt

The continuous growth of the computer software market leads to a very competitive and challenging era.

Not only does your software need to be functional and easy to use, it must also look appealing and professional to the users. In order to gain an upper hand and a competitive advantage over other software products in the market, the look and feel of your product is of utmost importance and should be taken care of early in the production stage.

Qt allows us to easily design our program's user interface through a method that most people are familiar with. Qt not only provides us with a powerful user interface toolkit called Qt Designer, which enables us to design our user interface without writing a single line of code, but it also allows advanced users to customize their user interface components through a simple scripting language called Qt Style Sheets.

I first familiarized with the basic structure of the Qt toolkit, together with its most important specific properties. Then I started writing some small applications. Qt is built in the following key concepts:

a. Complete abstraction of the GUI which makes its porting on different target platform easier

b. Signals and Slots - This allows communication between objects easier to implement; the widgets can send signals containing event information which can be received by other controls using slots.

c. Metaobject compiler provides programming features not available natively in C++.

d. Language bindings - Qt can be used in several programming languages like Python, Javascript, C++ etc.

Qt works on a variety of platforms from Unix flavours like Android to Microsoft platforms, Apple and other embedded platforms.

It is available under the free software GPL license. Qt comes with its own set of tools like Qt Creator, a cross-platform IDE for C++. The Qt designer's GUI layout/design functionality is integrated into the IDE.

# DRAWING TOOL KIT

The Basic Drawing example consists of two classes:

1. Window is the application's main window displaying a RenderArea widget in addition to several parameter widgets.
2. RenderArea is a custom widget that renders multiple copies of the currently active shape.

WINDOW CLASS DEFINITION

The Window class inherits QWidget, and is the application's main window displaying a Render Area widget in addition to several parameter widgets.

#ifndefWINDOW\_H

#defineWINDOW\_H

#include<QWidget>

classQCheckBox;

classQComboBox;

classQLabel;

classQSpinBox;

classRenderArea;

classWindow:publicQWidget

{

Q\_OBJECT

public:

Window();

privateslots:

void**shapeChanged**();

void**penChanged**();

void**brushChanged**();

private:

RenderArea\*renderArea;

QLabel\*shapeLabel;

QLabel\*penWidthLabel;

QLabel\*penStyleLabel;

QLabel\*penCapLabel;

QLabel\*penJoinLabel;

QLabel\*brushStyleLabel;

QLabel\*otherOptionsLabel;

QComboBox\*shapeComboBox;

QSpinBox\*penWidthSpinBox;

QComboBox\*penStyleComboBox;

QComboBox\*penCapComboBox;

QComboBox\*penJoinComboBox;

QComboBox\*brushStyleComboBox;

QCheckBox\*antialiasingCheckBox;

QCheckBox\*transformationsCheckBox;

};

#endif//WINDOW\_H

RENDER AREA CLASS DEFINITION

The Render Area class definition inherits QWidget , and renders multiple copies of the currently active shape using a QPainter

#ifndefRENDERAREA\_H

#defineRENDERAREA\_H

#include<QBrush>

#include<QPen>

#include<QPixmap>

#include<QWidget>

classRenderArea:publicQWidget

{

Q\_OBJECT

public:

enumShape{Line,Points,Polyline,Polygon,Rect,RoundedRect,Ellipse,Arc,

Chord,Pie,Path,Text,Pixmap};

RenderArea(QWidget\*parent=0);

QSize***minimumSizeHint***()constoverride;

QSize***sizeHint***()constoverride;

publicslots:

void**setShape**(Shapeshape);

void**setPen**(constQPen&pen);

void**setBrush**(constQBrush&brush);

void**setAntialiased**(boolantialiased);

void**setTransformed**(booltransformed);

protected:

void***paintEvent***(QPaintEvent\*event)override;

private:

Shapeshape;

QPenpen;

QBrushbrush;

boolantialiased;

booltransformed;

QPixmappixmap; }; #endif//RENDERAREA\_H

*The output of the project is as shown in the Figure below*

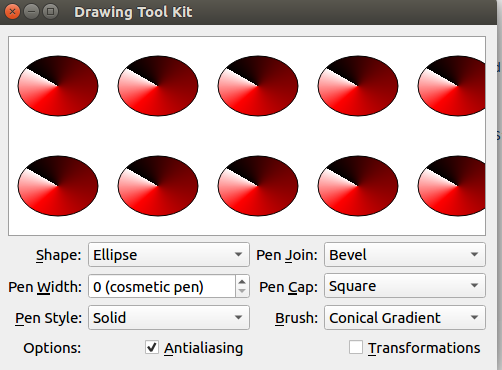


Figure : Drawing toolkit for ellipse

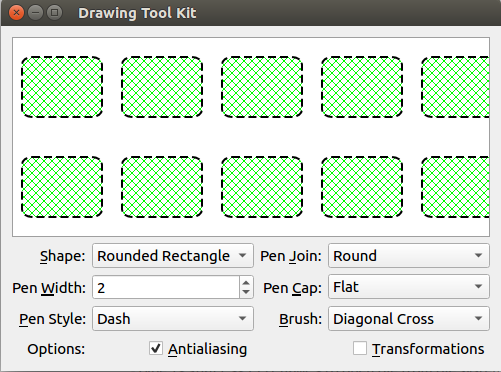


Figure :Drawing Toolkit for Rectangle

# SCRIBBLE PAD

The Scribble Pad demonstrates how to use QPainter to draw image in real time, as well as to repaint widgets.

The Paint Window consists of two classes:

1.ScribbleArea is a custom widget that displays a QImage and allows user to draw on it.

2.MainWindow provides a menu above the Scribble Area.

SCRIBBLEAREA CLASS DEFINATION

#ifndefSCRIBBLEAREA\_H

#defineSCRIBBLEAREA\_H

#include<QColor>

#include<QImage>

#include<QPoint>

#include<QWidget>

classScribbleArea:publicQWidget

{

Q\_OBJECT

public:

ScribbleArea(QWidget\*parent=0);

bool**openImage**(constQString&fileName);

bool**saveImage**(constQString&fileName,constchar\*fileFormat);

void**setPenColor**(constQColor&newColor);

void**setPenWidth**(intnewWidth);

bool**isModified**()const{returnmodified;}

QColor**penColor**()const{returnmyPenColor;}

int**penWidth**()const{returnmyPenWidth;}

publicslots:

void**clearImage**();

void**print**();

protected:

void***mousePressEvent***(QMouseEvent\*event)override;

void***mouseMoveEvent***(QMouseEvent\*event)override;

void***mouseReleaseEvent***(QMouseEvent\*event)override;

void***paintEvent***(QPaintEvent\*event)override;

void***resizeEvent***(QResizeEvent\*event)override;

private:

void**drawLineTo**(constQPoint&endPoint);

void**resizeImage**(QImage\*image,constQSize&newSize);

boolmodified;

boolscribbling;

intmyPenWidth;

QColormyPenColor;

QImageimage;

QPointlastPoint;

};

#endif

MAINWINDOW CLASS DEFINATION

#ifndefMAINWINDOW\_H

#defineMAINWINDOW\_H

#include<QList>

#include<QMainWindow>

classScribbleArea;

classMainWindow:publicQMainWindow

{

Q\_OBJECT

public:

MainWindow();

protected:

void***closeEvent***(QCloseEvent\*event)override;

privateslots:

void**open**();

void**save**();

void**penColor**();

void**penWidth**();

void**about**();

private:

void**createActions**();

void**createMenus**();

bool**maybeSave**();

bool**saveFile**(constQByteArray&fileFormat);

ScribbleArea\*scribbleArea;

QMenu\*saveAsMenu;

QMenu\*fileMenu;

QMenu\*optionMenu;

QMenu\*helpMenu;

QAction\*openAct;

QList<QAction\*>saveAsActs;

QAction\*exitAct;

QAction\*penColorAct;

QAction\*penWidthAct;

QAction\*printAct;

QAction\*clearScreenAct;

QAction\*aboutAct;

QAction\*aboutQtAct;

};

#endif

*The output of the code is:*

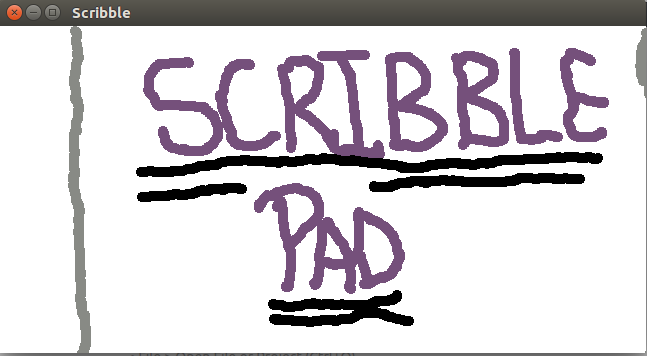


Figure : Scribble pad - example 1

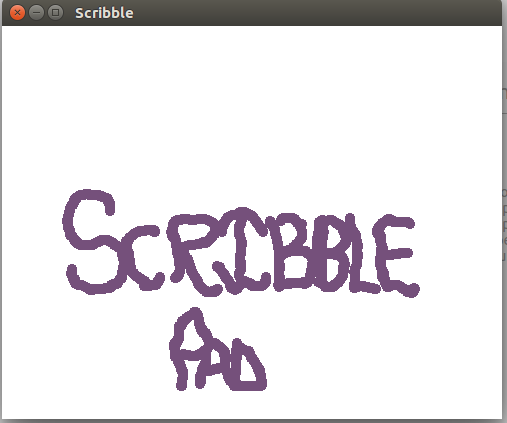


Figure : Scribble Pad - example 2

# CONCLUSION

Drawing tool kit and Scribble pad have been successfully developed. This can be further extended or used in graphical display systems for the defense establishments as an overlay over a map where the user can draw his area of interest on the map in the area that he is surveying. He can also use the scribble pad program with a stylus to enter his comments or descriptions.

# REFERENCES

1. Molkentin, D. (2007). **The book of Qt 4: The art of building Qt applications**, Munich:Open source press
2. About DRDO .Retrieved from <http://www.wikipedia.org>
3. Qt Documentation .Retrieved from httpa://doc.qt.io/

# APPENDIX - SOURCE CODE

**1. Drawing tool kit**

**1) main.cpp**

#include"window.h"

#include<QApplication>

int**main**(intargc,char\*argv[])

{

QApplicationapp(argc,argv);

Windowwindow;

window.show();

returnapp.exec();

}

**2) window.cpp**

#include"renderarea.h"

#include"window.h"

#include<QtWidgets>

constintIdRole=Qt::UserRole;

Window::**Window**()

{

renderArea=newRenderArea;

shapeComboBox=newQComboBox;

shapeComboBox->addItem(tr("Polygon"),RenderArea::Polygon);

shapeComboBox->addItem(tr("Rectangle"),RenderArea::Rect);

shapeComboBox->addItem(tr("RoundedRectangle"),RenderArea::RoundedRect);

shapeComboBox->addItem(tr("Ellipse"),RenderArea::Ellipse);

shapeComboBox->addItem(tr("Pie"),RenderArea::Pie);

shapeComboBox->addItem(tr("Chord"),RenderArea::Chord);

shapeComboBox->addItem(tr("Path"),RenderArea::Path);

shapeComboBox->addItem(tr("Line"),RenderArea::Line);

shapeComboBox->addItem(tr("Polyline"),RenderArea::Polyline);

shapeComboBox->addItem(tr("Arc"),RenderArea::Arc);

shapeComboBox->addItem(tr("Points"),RenderArea::Points);

shapeComboBox->addItem(tr("Text"),RenderArea::Text);

shapeComboBox->addItem(tr("Pixmap"),RenderArea::Pixmap);

shapeLabel=newQLabel(tr("&Shape:"));

shapeLabel->setBuddy(shapeComboBox);

penWidthSpinBox=newQSpinBox;

penWidthSpinBox->setRange(0,20);

penWidthSpinBox->setSpecialValueText(tr("0(cosmeticpen)"));

penWidthLabel=newQLabel(tr("Pen&Width:"));

penWidthLabel->setBuddy(penWidthSpinBox);

penStyleComboBox=newQComboBox;

penStyleComboBox->addItem(tr("Solid"),static\_cast<int>(Qt::SolidLine));

penStyleComboBox->addItem(tr("Dash"),static\_cast<int>(Qt::DashLine));

penStyleComboBox->addItem(tr("Dot"),static\_cast<int>(Qt::DotLine));

penStyleComboBox->addItem(tr("DashDot"),static\_cast<int>(Qt::DashDotLine));

penStyleComboBox->addItem(tr("DashDotDot"),static\_cast<int>(Qt::DashDotDotLine));

penStyleComboBox->addItem(tr("None"),static\_cast<int>(Qt::NoPen));

penStyleLabel=newQLabel(tr("&PenStyle:"));

penStyleLabel->setBuddy(penStyleComboBox);

penCapComboBox=newQComboBox;

penCapComboBox->addItem(tr("Flat"),Qt::FlatCap);

penCapComboBox->addItem(tr("Square"),Qt::SquareCap);

penCapComboBox->addItem(tr("Round"),Qt::RoundCap);

penCapLabel=newQLabel(tr("Pen&Cap:"));

penCapLabel->setBuddy(penCapComboBox);

penJoinComboBox=newQComboBox;

penJoinComboBox->addItem(tr("Miter"),Qt::MiterJoin);

penJoinComboBox->addItem(tr("Bevel"),Qt::BevelJoin);

penJoinComboBox->addItem(tr("Round"),Qt::RoundJoin);

penJoinLabel=newQLabel(tr("Pen&Join:"));

penJoinLabel->setBuddy(penJoinComboBox);

brushStyleComboBox=newQComboBox;

brushStyleComboBox->addItem(tr("LinearGradient"),

static\_cast<int>(Qt::LinearGradientPattern));

brushStyleComboBox->addItem(tr("RadialGradient"),

static\_cast<int>(Qt::RadialGradientPattern));

brushStyleComboBox->addItem(tr("ConicalGradient"),

static\_cast<int>(Qt::ConicalGradientPattern));

brushStyleComboBox->addItem(tr("Texture"),static\_cast<int>(Qt::TexturePattern));

brushStyleComboBox->addItem(tr("Solid"),static\_cast<int>(Qt::SolidPattern));

brushStyleComboBox->addItem(tr("Horizontal"),static\_cast<int>(Qt::HorPattern));

brushStyleComboBox->addItem(tr("Vertical"),static\_cast<int>(Qt::VerPattern));

brushStyleComboBox->addItem(tr("Cross"),static\_cast<int>(Qt::CrossPattern));

brushStyleComboBox->addItem(tr("BackwardDiagonal"),static\_cast<int>(Qt::BDiagPattern));

brushStyleComboBox->addItem(tr("ForwardDiagonal"),static\_cast<int>(Qt::FDiagPattern));

brushStyleComboBox->addItem(tr("DiagonalCross"),static\_cast<int>(Qt::DiagCrossPattern));

brushStyleComboBox->addItem(tr("Dense1"),static\_cast<int>(Qt::Dense1Pattern));

brushStyleComboBox->addItem(tr("Dense2"),static\_cast<int>(Qt::Dense2Pattern));

brushStyleComboBox->addItem(tr("Dense3"),static\_cast<int>(Qt::Dense3Pattern));

brushStyleComboBox->addItem(tr("Dense4"),static\_cast<int>(Qt::Dense4Pattern));

brushStyleComboBox->addItem(tr("Dense5"),static\_cast<int>(Qt::Dense5Pattern));

brushStyleComboBox->addItem(tr("Dense6"),static\_cast<int>(Qt::Dense6Pattern));

brushStyleComboBox->addItem(tr("Dense7"),static\_cast<int>(Qt::Dense7Pattern));

brushStyleComboBox->addItem(tr("None"),static\_cast<int>(Qt::NoBrush));

brushStyleLabel=newQLabel(tr("&Brush:"));

brushStyleLabel->setBuddy(brushStyleComboBox);

otherOptionsLabel=newQLabel(tr("Options:"));

antialiasingCheckBox=newQCheckBox(tr("&Antialiasing"));

transformationsCheckBox=newQCheckBox(tr("&Transformations"));

connect(shapeComboBox,SIGNAL(activated(int)),

this,SLOT(shapeChanged()));

connect(penWidthSpinBox,SIGNAL(valueChanged(int)),

this,SLOT(penChanged()));

connect(penStyleComboBox,SIGNAL(activated(int)),

this,SLOT(penChanged()));

connect(penCapComboBox,SIGNAL(activated(int)),

this,SLOT(penChanged()));

connect(penJoinComboBox,SIGNAL(activated(int)),

this,SLOT(penChanged()));

connect(brushStyleComboBox,SIGNAL(activated(int)),

this,SLOT(brushChanged()));

connect(antialiasingCheckBox,SIGNAL(toggled(bool)),

renderArea,SLOT(setAntialiased(bool)));

connect(transformationsCheckBox,SIGNAL(toggled(bool)),

renderArea,SLOT(setTransformed(bool)));

QGridLayout\*mainLayout=newQGridLayout;

mainLayout->setColumnStretch(0,1);

mainLayout->setColumnStretch(3,1);

mainLayout->addWidget(renderArea,0,0,1,4);

mainLayout->addWidget(shapeLabel,2,0,Qt::AlignRight);

mainLayout->addWidget(shapeComboBox,2,1);

mainLayout->addWidget(penWidthLabel,3,0,Qt::AlignRight);

mainLayout->addWidget(penWidthSpinBox,3,1);

mainLayout->addWidget(penStyleLabel,4,0,Qt::AlignRight);

mainLayout->addWidget(penStyleComboBox,4,1);

mainLayout->addWidget(penCapLabel,3,2,Qt::AlignRight);

mainLayout->addWidget(penCapComboBox,3,3);

mainLayout->addWidget(penJoinLabel,2,2,Qt::AlignRight);

mainLayout->addWidget(penJoinComboBox,2,3);

mainLayout->addWidget(brushStyleLabel,4,2,Qt::AlignRight);

mainLayout->addWidget(brushStyleComboBox,4,3);

mainLayout->addWidget(otherOptionsLabel,5,0,Qt::AlignRight);

mainLayout->addWidget(antialiasingCheckBox,5,1,1,1,Qt::AlignRight);

mainLayout->addWidget(transformationsCheckBox,5,2,1,2,Qt::AlignRight);

setLayout(mainLayout);

shapeChanged();

penChanged();

brushChanged();

antialiasingCheckBox->setChecked(true);

setWindowTitle(tr("DrawingToolKit"));

}

voidWindow::**shapeChanged**()

{

RenderArea::Shapeshape=RenderArea::Shape(shapeComboBox->itemData(

shapeComboBox->currentIndex(),IdRole).toInt());

renderArea->setShape(shape);

}

voidWindow::**penChanged**()

{

intwidth=penWidthSpinBox->value();

Qt::PenStylestyle=Qt::PenStyle(penStyleComboBox->itemData(

penStyleComboBox->currentIndex(),IdRole).toInt());

Qt::PenCapStylecap=Qt::PenCapStyle(penCapComboBox->itemData(

penCapComboBox->currentIndex(),IdRole).toInt());

Qt::PenJoinStylejoin=Qt::PenJoinStyle(penJoinComboBox->itemData(

penJoinComboBox->currentIndex(),IdRole).toInt());

renderArea->setPen(QPen(Qt::black,width,style,cap,join));

}

voidWindow::**brushChanged**()

{

Qt::BrushStylestyle=Qt::BrushStyle(brushStyleComboBox->itemData(

brushStyleComboBox->currentIndex(),IdRole).toInt());

if(style==Qt::LinearGradientPattern){

QLinearGradientlinearGradient(0,0,100,100);

linearGradient.setColorAt(0.0,Qt::white);

linearGradient.setColorAt(0.2,Qt::blue);

linearGradient.setColorAt(1.0,Qt::black);

renderArea->setBrush(linearGradient);

}elseif(style==Qt::RadialGradientPattern){

QRadialGradientradialGradient(50,50,50,70,70);

radialGradient.setColorAt(0.0,Qt::white);

radialGradient.setColorAt(0.2,Qt::green);

radialGradient.setColorAt(1.0,Qt::black);

renderArea->setBrush(radialGradient);

}elseif(style==Qt::ConicalGradientPattern){

QConicalGradientconicalGradient(50,50,150);

conicalGradient.setColorAt(0.0,Qt::white);

conicalGradient.setColorAt(0.2,Qt::red);

conicalGradient.setColorAt(1.0,Qt::black);

renderArea->setBrush(conicalGradient);

}elseif(style==Qt::TexturePattern){

renderArea->setBrush(QBrush(QPixmap(":/images/brick.png")));

}else{

renderArea->setBrush(QBrush(Qt::green,style));

}

}

**3) renderarea.cpp**

#ifndefRENDERAREA\_H

#defineRENDERAREA\_H

#include<QBrush>

#include<QPen>

#include<QPixmap>

#include<QWidget>

classRenderArea:publicQWidget

{

Q\_OBJECT

public:

enumShape{Line,Points,Polyline,Polygon,Rect,RoundedRect,Ellipse,Arc,

Chord,Pie,Path,Text,Pixmap};

RenderArea(QWidget\*parent=0);

QSize***minimumSizeHint***()constoverride;

QSize***sizeHint***()constoverride;

publicslots:

void**setShape**(Shapeshape);

void**setPen**(constQPen&pen);

void**setBrush**(constQBrush&brush);

void**setAntialiased**(boolantialiased);

void**setTransformed**(booltransformed);

protected:

void***paintEvent***(QPaintEvent\*event)override;

private:

Shapeshape;

QPenpen;

QBrushbrush;

boolantialiased;

booltransformed;

QPixmappixmap;

};

#endif//RENDERAREA\_H

#include"renderarea.h"

#include<QPainter>

RenderArea::**RenderArea**(QWidget\*parent)

:QWidget(parent)

{

shape=Polygon;

antialiased=false;

transformed=false;

pixmap.load(":/images/qt-logo.png");

setBackgroundRole(QPalette::Base);

setAutoFillBackground(true);

}

QSizeRenderArea::***minimumSizeHint***()const

{

returnQSize(100,100);

}

QSizeRenderArea::***sizeHint***()const

{

returnQSize(400,200);

}

voidRenderArea::**setShape**(Shapeshape)

{

this->shape=shape;

update();

}

voidRenderArea::**setPen**(constQPen&pen)

{

this->pen=pen;

update();

}

voidRenderArea::**setBrush**(constQBrush&brush)

{

this->brush=brush;

update();

}

voidRenderArea::**setAntialiased**(boolantialiased)

{

this->antialiased=antialiased;

update();

}

voidRenderArea::**setTransformed**(booltransformed)

{

this->transformed=transformed;

update();

}

voidRenderArea::***paintEvent***(QPaintEvent\*/\*event\*/)

{

staticconstQPointpoints[4]={

QPoint(10,80),

QPoint(20,10),

QPoint(80,30),

QPoint(90,70)

};

QRectrect(10,20,80,60);

QPainterPathpath;

path.moveTo(20,80);

path.lineTo(20,30);

path.cubicTo(80,0,50,50,80,80);

intstartAngle=20\*16;

intarcLength=120\*16;

QPainterpainter(this);

painter.setPen(pen);

painter.setBrush(brush);

if(antialiased)

painter.setRenderHint(QPainter::Antialiasing,true);

for(intx=0;x<width();x+=100){

for(inty=0;y<height();y+=100){

painter.save();

painter.translate(x,y);

if(transformed){

painter.translate(50,50);

painter.rotate(60.0);

painter.scale(0.6,0.9);

painter.translate(-50,-50);

}

switch(shape){

caseLine:

painter.drawLine(rect.bottomLeft(),rect.topRight());

break;

casePoints:

painter.drawPoints(points,4);

break;

casePolyline:

painter.drawPolyline(points,4);

break;

casePolygon:

painter.drawPolygon(points,4);

break;

caseRect:

painter.drawRect(rect);

break;

caseRoundedRect:

painter.drawRoundedRect(rect,25,25,Qt::RelativeSize);

break;

caseEllipse:

painter.drawEllipse(rect);

break;

caseArc:

painter.drawArc(rect,startAngle,arcLength);

break;

caseChord:

painter.drawChord(rect,startAngle,arcLength);

break;

casePie:

painter.drawPie(rect,startAngle,arcLength);

break;

casePath:

painter.drawPath(path);

break;

caseText:

painter.drawText(rect,

Qt::AlignCenter,

tr("Qtby\nTheQtCompany"));

break;

casePixmap:

painter.drawPixmap(10,10,pixmap);

}

painter.restore();

}

}

painter.setRenderHint(QPainter::Antialiasing,false);

painter.setPen(palette().dark().color());

painter.setBrush(Qt::NoBrush);

painter.drawRect(QRect(0,0,width()-1,height()-1));

}

**2. Scribble Pad**

**1) main.cpp**

#include"mainwindow.h"

#include<QApplication>

int**main**(intargc,char\*argv[])

{

QApplicationapp(argc,argv);

MainWindowmainwindow;

mainwindow.show();

returnapp.exec();

}

**2) mainwindow.cpp**

#include<QtWidgets>

#include"mainwindow.h"

#include"scribblearea.h"

MainWindow::**MainWindow**()

{

scribbleArea=newScribbleArea;

setCentralWidget(scribbleArea);

createActions();

createMenus();

setWindowTitle(tr("Scribble"));

resize(500,500);

}

voidMainWindow::***closeEvent***(QCloseEvent\*event)

{

if(maybeSave()){

event->accept();

}else{

event->ignore();

}

}

voidMainWindow::**open**()

{

if(maybeSave()){

QStringfileName=QFileDialog::getOpenFileName(this,

tr("OpenFile"),QDir::currentPath());

if(!fileName.isEmpty())

scribbleArea->openImage(fileName);

}

}

voidMainWindow::**save**()

{

QAction\*action=qobject\_cast<QAction\*>(sender());

QByteArrayfileFormat=action->data().toByteArray();

saveFile(fileFormat);

}

voidMainWindow::**penColor**()

{

QColornewColor=QColorDialog::getColor(scribbleArea->penColor());

if(newColor.isValid())

scribbleArea->setPenColor(newColor);

}

voidMainWindow::**penWidth**()

{

boolok;

intnewWidth=QInputDialog::getInt(this,tr("Scribble"),

tr("Selectpenwidth:"),

scribbleArea->penWidth(),

1,50,1,&ok);

if(ok)

scribbleArea->setPenWidth(newWidth);

}

voidMainWindow::**about**()

{

QMessageBox::about(this,tr("AboutScribble"),

tr("<p>The<b>Scribble</b>exampleshowshowtouseQMainWindowasthe"

"basewidgetforanapplication,andhowtoreimplementsomeof"

"QWidget'seventhandlerstoreceivetheeventsgeneratedfor"

"theapplication'swidgets:</p>"));

}

voidMainWindow::**createActions**()

{

openAct=newQAction(tr("&Open..."),this);

openAct->setShortcuts(QKeySequence::Open);

connect(openAct,SIGNAL(triggered()),this,SLOT(open()));

foreach(QByteArrayformat,QImageWriter::supportedImageFormats()){

QStringtext=tr("%1...").arg(QString(format).toUpper());

QAction\*action=newQAction(text,this);

action->setData(format);

connect(action,SIGNAL(triggered()),this,SLOT(save()));

saveAsActs.append(action);

}

printAct=newQAction(tr("&Print..."),this);

connect(printAct,SIGNAL(triggered()),scribbleArea,SLOT(print()));

exitAct=newQAction(tr("E&xit"),this);

exitAct->setShortcuts(QKeySequence::Quit);

connect(exitAct,SIGNAL(triggered()),this,SLOT(close()));

penColorAct=newQAction(tr("&PenColor..."),this);

connect(penColorAct,SIGNAL(triggered()),this,SLOT(penColor()));

penWidthAct=newQAction(tr("Pen&Width..."),this);

connect(penWidthAct,SIGNAL(triggered()),this,SLOT(penWidth()));

clearScreenAct=newQAction(tr("&ClearScreen"),this);

clearScreenAct->setShortcut(tr("Ctrl+L"));

connect(clearScreenAct,SIGNAL(triggered()),

scribbleArea,SLOT(clearImage()));

aboutAct=newQAction(tr("&About"),this);

connect(aboutAct,SIGNAL(triggered()),this,SLOT(about()));

aboutQtAct=newQAction(tr("About&Qt"),this);

connect(aboutQtAct,SIGNAL(triggered()),qApp,SLOT(aboutQt()));

}

voidMainWindow::**createMenus**()

{

saveAsMenu=newQMenu(tr("&SaveAs"),this);

foreach(QAction\*action,saveAsActs)

saveAsMenu->addAction(action);

fileMenu=newQMenu(tr("&File"),this);

fileMenu->addAction(openAct);

fileMenu->addMenu(saveAsMenu);

fileMenu->addAction(printAct);

fileMenu->addSeparator();

fileMenu->addAction(exitAct);

optionMenu=newQMenu(tr("&Options"),this);

optionMenu->addAction(penColorAct);

optionMenu->addAction(penWidthAct);

optionMenu->addSeparator();

optionMenu->addAction(clearScreenAct);

helpMenu=newQMenu(tr("&Help"),this);

helpMenu->addAction(aboutAct);

helpMenu->addAction(aboutQtAct);

menuBar()->addMenu(fileMenu);

menuBar()->addMenu(optionMenu);

menuBar()->addMenu(helpMenu);

}

boolMainWindow::**maybeSave**()

{

if(scribbleArea->isModified()){

QMessageBox::StandardButtonret;

ret=QMessageBox::warning(this,tr("Scribble"),

tr("Theimagehasbeenmodified.\n"

"Doyouwanttosaveyourchanges?"),

QMessageBox::Save|QMessageBox::Discard

|QMessageBox::Cancel);

if(ret==QMessageBox::Save){

returnsaveFile("png");

}elseif(ret==QMessageBox::Cancel){

returnfalse;

}

}

returntrue;

}

boolMainWindow::**saveFile**(constQByteArray&fileFormat)

{

QStringinitialPath=QDir::currentPath()+"/untitled."+fileFormat;

QStringfileName=QFileDialog::getSaveFileName(this,tr("SaveAs"),

initialPath,

tr("%1Files(\*.%2);;AllFiles(\*)")

.arg(QString::fromLatin1(fileFormat.toUpper()))

.arg(QString::fromLatin1(fileFormat)));

if(fileName.isEmpty()){

returnfalse;

}else{

returnscribbleArea->saveImage(fileName,fileFormat.constData());

}

}

**3) scribblearea.cpp**

#include<QtWidgets>

#ifdefined(QT\_PRINTSUPPORT\_LIB)

#include<QtPrintSupport/qtprintsupportglobal.h>

#ifQT\_CONFIG(printdialog)

#include<QPrinter>

#include<QPrintDialog>

#endif

#include"scribblearea.h"

ScribbleArea::ScribbleArea(QWidget\*parent)

:QWidget(parent)

{

setAttribute(Qt::WA\_StaticContents);

modified=false;

scribbling=false;

myPenWidth=1;

myPenColor=Qt::blue;

}

boolScribbleArea::openImage(constQString&fileName)

{

QImageloadedImage;

if(!loadedImage.load(fileName))

returnfalse;

QSizenewSize=loadedImage.size().expandedTo(size());

resizeImage(&loadedImage,newSize);

image=loadedImage;

modified=false;

update();

returntrue;

}

boolScribbleArea::saveImage(constQString&fileName,constchar\*fileFormat)

{

QImagevisibleImage=image;

resizeImage(&visibleImage,size());

if(visibleImage.save(fileName,fileFormat)){

modified=false;

returntrue;

}else{

returnfalse;

}

}

voidScribbleArea::setPenColor(constQColor&newColor)

{

myPenColor=newColor;

}

voidScribbleArea::setPenWidth(intnewWidth)

{

myPenWidth=newWidth;

}

voidScribbleArea::clearImage()

{

image.fill(qRgb(255,255,255));

modified=true;

update();

}

voidScribbleArea::*mousePressEvent*(QMouseEvent\*event)

{

if(event->button()==Qt::LeftButton){

lastPoint=event->pos();

scribbling=true;

}

}

voidScribbleArea::*mouseMoveEvent*(QMouseEvent\*event)

{

if((event->buttons()&Qt::LeftButton)&&scribbling)

drawLineTo(event->pos());

}

voidScribbleArea::*mouseReleaseEvent*(QMouseEvent\*event)

{

if(event->button()==Qt::LeftButton&&scribbling){

drawLineTo(event->pos());

scribbling=false;

}

}

voidScribbleArea::*paintEvent*(QPaintEvent\*event)

{

QPainterpainter(this);

QRectdirtyRect=event->rect();

painter.drawImage(dirtyRect,image,dirtyRect);

}

voidScribbleArea::*resizeEvent*(QResizeEvent\*event)

{

if(width()>image.width()||height()>image.height()){

intnewWidth=qMax(width()+128,image.width());

intnewHeight=qMax(height()+128,image.height());

resizeImage(&image,QSize(newWidth,newHeight));

update();

}

QWidget::*resizeEvent*(event);

}

voidScribbleArea::drawLineTo(constQPoint&endPoint)

{

QPainterpainter(&image);

painter.setPen(QPen(myPenColor,myPenWidth,Qt::SolidLine,Qt::RoundCap,

Qt::RoundJoin));

painter.drawLine(lastPoint,endPoint);

modified=true;

intrad=(myPenWidth/2)+2;

update(QRect(lastPoint,endPoint).normalized()

.adjusted(-rad,-rad,+rad,+rad));

lastPoint=endPoint;

}voidScribbleArea::resizeImage(QImage\*image,constQSize&newSize)

{

if(image->size()==newSize)

return;

QImagenewImage(newSize,QImage::Format\_RGB32);

newImage.fill(qRgb(255,255,255));

QPainterpainter(&newImage);

painter.drawImage(QPoint(0,0),\*image);

\*image=newImage;

}voidScribbleArea::print()

{

#ifQT\_CONFIG(printdialog)

QPrinterprinter(QPrinter::HighResolution);

QPrintDialogprintDialog(&printer,this);

if(printDialog.exec()==QDialog::Accepted){

QPainterpainter(&printer);

QRectrect=painter.viewport();

QSizesize=image.size();

size.scale(rect.size(),Qt::KeepAspectRatio);

painter.setViewport(rect.x(),rect.y(),size.width(),size.height());

painter.setWindow(image.rect());

painter.drawImage(0,0,image);

}

#endif//QT\_CONFIG(printdialog)