GUI programming – using GTK or Qt

Aim :

To write GUI programs using FOSS tools in Linux.

This exercise consists of developing GUI programs using the Qt toolkit.

A GUI Hello World

Create a new directory for the program.

[fosslab@fosslab ~]$mkdir qthello

Go to the newly created directory.

[fosslab@fosslab ~]$cd qthello

[fosslab@fosslab qthello]$

create the file in the qthello directory.

[fosslab@fosslab qthello]$gedit qthello.cpp

Type the following code, save and close gedit

```
#include <QApplication>
#include <QPushButton>
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
QPushButton helloButton("Hello World");
helloButton.resize(80, 20);
helloButton.show();
return app.exec();
}
```

Once the file has been saved, use the following commands to compile and execute the program.

```
[fosslab@fosslab qthello]$ qmake-qt4 -project
[fosslab@fosslab qthello]$ qmake-qt4
[fosslab@fosslab qthello]$ make
[fosslab@fosslab qthello]$ ./qthello
```

Window with a Button

Create a new directory for the program.

[fosslab@fosslab ~]$mkdir qtbutton

Go to the newly created directory.

[fosslab@fosslab ~]$cd qtbutton

[fosslab@fosslab qthello]$

create the file in the qthello directory.

[fosslab@fosslab qthello]$gedit qtbutton.cpp

Type the following code, save and close gedit

```
#include <QtGui>
int main(int argv, char **args)
```

```
{
QApplication app(argv, args);
        QTextEdit *textEdit = new QTextEdit;
        QPushButton *quitButton = new QPushButton("&Quit");
        QObject::connect(quitButton, SIGNAL(clicked()), qApp, SLOT(quit()));
        QVBoxLayout *layout = new QVBoxLayout;
        layout->addWidget(textEdit);
        layout->addWidget(quitButton);
        QWidget window;
       window.setLayout(layout);
        window.show();
        return app.exec();
}
```

once the file has been saved, use the following commands to compile and execute the program.

```
[fosslab@fosslab qthello]$ qmake-qt4 -project
[fosslab@fosslab qthello]$ qmake-qt4
[fosslab@fosslab qthello]$ make
[fosslab@fosslab qthello]$ ./qtbutton
```

Using Layouts.
Create a new directory for the program.

```
[fosslab@fosslab ~]$mkdir qtlayout
```

Go to the newly created directory.

```
[fosslab@fosslab ~]$cd qtlayout
[fosslab@fosslab qtlayout]$
```
create the file in the qtlayout directory.
```
[fosslab@fosslab qtlayout]$gedit qtlayout.cpp
```

Type the following code, save and close gedit

```
//qtlayout.cpp
#include <QtGui>
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
QWidget window;
//Create a label an a single line text box
QLabel *label = new QLabel("Name:");
QLineEdit *lineEdit = new QLineEdit();
//Create a layout. Add the label and the lineedit to it.
```

```
QHBoxLayout *layout = new QHBoxLayout();
layout->addWidget(label);
layout->addWidget(lineEdit);
//Apply the layout to the main window.
//Since the widgets are part of the layout,
//they are now children of the window.
window.setLayout(layout);
window.setWindowTitle("Window layout");
window.show();
return app.exec();
}
```

once the file has been saved, use the following commands to compile and execute the program.

```
[fosslab@fosslab qtlayout]$ qmake-qt4 -project
[fosslab@fosslab qtlayout]$ qmake-qt4
[fosslab@fosslab qtlayout]$ make
[fosslab@fosslab qtlayout]$ ./qtlayout
```

Signals and Slots

This program demonstrates the use of signals and slots to make two widgets interact with each other.

The entire application is divided into 3 files:

1. communicate.h
2. communicate.cpp
3. main.cpp

Create a new directory for the program.

```
[fosslab@fosslab ~]$mkdir qtsignals
```

Go to the newly created directory.

```
[fosslab@fosslab ~]$cd qtsignals
[fosslab@fosslab qtsignals]$
```

create the communicate.h file in the qtsignals directory.

```
[fosslab@fosslab qtsignals]$gedit qommunicate.h
```

Type the following code and save gedit

```
//communicate.h
#include <QWidget>
#include <QApplication>
#include <QPushButton>
#include <QLabel>
class Communicate : public QWidget
{
//The Q_OBJECT macro causes the moc tool to initialise
//code for signals and slots, run time type information
//and dynamic property system
Q_OBJECT
```

```
public:
Communicate(QWidget *parent = 0);
//add a lot which allows widget communications
private slots:
void add();
private:
QLabel *label;
};
```

create the communicate.cpp file in the qtsignals directory.

[fosslab@fosslab qtsignals]$gedit qommunicate.cpp

Type the following code and save gedit

```
//communicate.cpp
#include "communicate.h"
#include <QDesktopWidget>
Communicate::Communicate(QWidget *parent)
: QWidget(parent)
{
resize(180, 140);
QPushButton *plus = new QPushButton("+", this);
plus->setGeometry(50, 40, 50, 30);
label = new QLabel("0", this);
label->setGeometry(120, 40, 20, 30);
//Connect the clicked event of the button to
//the add method of the class
connect(plus, SIGNAL(clicked()), this, SLOT(add()));
}
void Communicate::add()
{
//Change the text displayed in the label
int val = label->text().toInt();
val++;
label->setText(QString::number(val));
}
```

create the main.cpp file in the qtsignals directory.

[fosslab@fosslab qtsignals]$gedit main.cpp

Type the following code and save gedit

```
//main.cpp
#include "communicate.h"
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
Communicate window;
window.setWindowTitle("Communicate");
```

```
window.show();
return app.exec();
}
```
To compile and execute the program.

```
[fosslab@fosslab qtsignals]$ qmake-qt4 -project
[fosslab@fosslab qtsignals]$ qmake-qt4
[fosslab@fosslab qtsignals]$ make
[fosslab@fosslab qtsignals]$ ./qtsignals
```

Menus and Toolbars.

This program will display a menu which can be used to close the progra
The entire application is divided into 3 files:
1. mymenu.h
2. mymenu.cpp
3. main.cpp

Create a new directory for the program.

```
[fosslab@fosslab ~]$mkdir qtmenu
```

Go to the newly created directory.

```
[fosslab@fosslab ~]$cd qtmenu
[fosslab@fosslab qtmenu]$
```
create the mymenu.h file in the qtmenu directory.
```
[fosslab@fosslab qtmenu]$gedit mymenu.h
```

Type the following code and save gedit
```
//mymenu.h
#include <QMainWindow>
class MyMenu : public QMainWindow
{
public:
MyMenu(QWidget *parent = 0);
};
```

create the mymenu.cpp file in the qtmenu directory.

```
[fosslab@fosslab qtmenu]$gedit mymenu.cpp
```

Type the following code and save gedit

```
//mymenu.cpp
#include "mymenu.h"
#include <QMenu>
#include <QMenuBar>
#include <QApplication>
```

```
MyMenu::MyMenu(QWidget *parent)
: QMainWindow(parent)
{
//create the quit action object
QAction *quit = new QAction("&Quit", this);
//create the file menu
QMenu *file;
file = menuBar()->addMenu("&File");
//add the quit action to the new menu
file->addAction(quit);
//connect the triggered signal from the quit action menu
//to the global quit method which closes the application
connect(quit, SIGNAL(triggered()), qApp, SLOT(quit()));
}
```

create the main.cpp file in the qtmenu directory.

[fosslab@fosslab qtmenu]$gedit main.cpp

Type the following code and close gedit

```
//main.cpp
#include "mymenu.h"
#include <QDesktopWidget>
#include <QApplication>
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
MyMenu window;
window.setWindowTitle("My menu");
window.show();
return app.exec();
}
```

To compile and execute the program.

```
[[fosslab@fosslab qtmenu]$ qmake-qt4 -project
[fosslab@fosslab qtmenu]$ qmake-qt4
[fosslab@fosslab qtmenu]$ make
[fosslab@fosslab qtmenu]$ ./qtmenu
```