Centria UNIVERSITY OF APPLIED SCIENCES Degree Programme in Information Technology, Kokkola

Multiplatform Programming With Qt

ITK-1067

Part 1 – Getting Started

Dr Grzegorz Szewczyk
Principal Lecturer, Information Technology

Qt

Kokkola, Jan-17

Qt

Topics

- Getting started
- Practical tips for developers

ITK-1048 - Multiplatform Programming in Qt

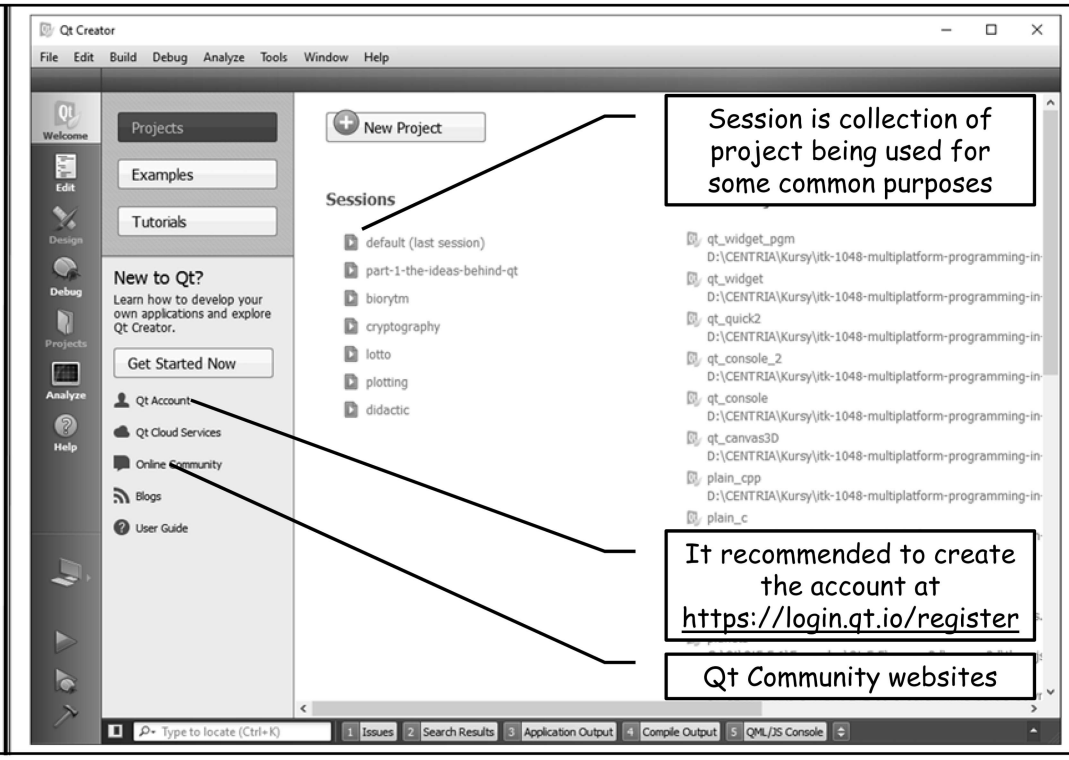
Part 1 – Getting Started

GETTING STARTED

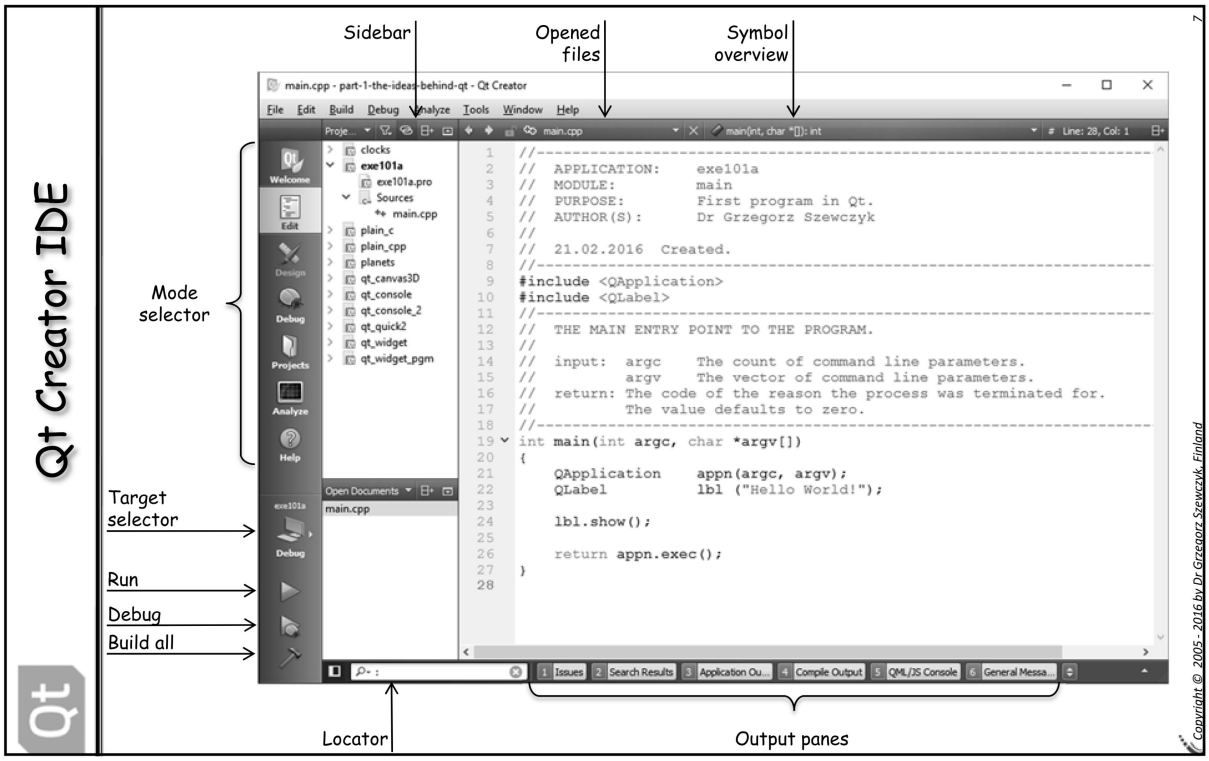
Qt Creator IDE

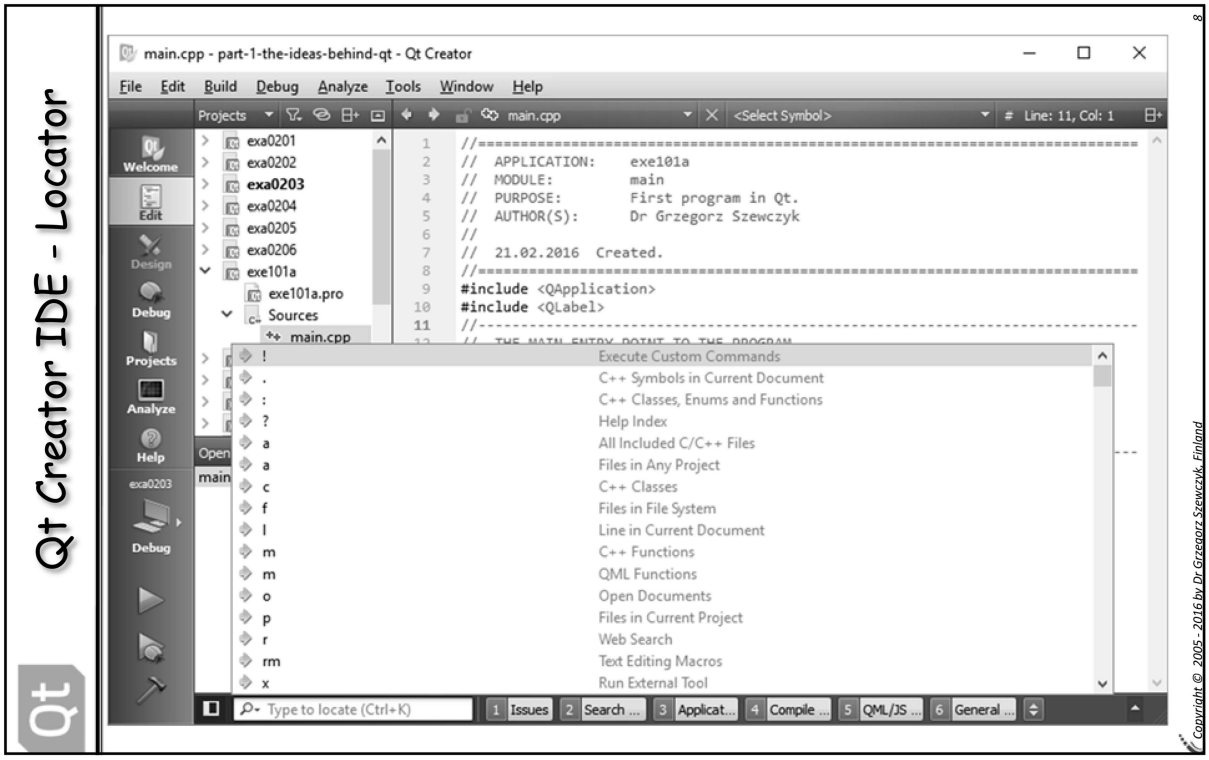
- Advanced C++ code editor
- Integrated GUI layout and forms designer
- Project and build management tools
- Integrated, context-sensitive help system
- Visual debugger
- Rapid code navigation tools
- Supports multiple platforms

Qt Creator IDE

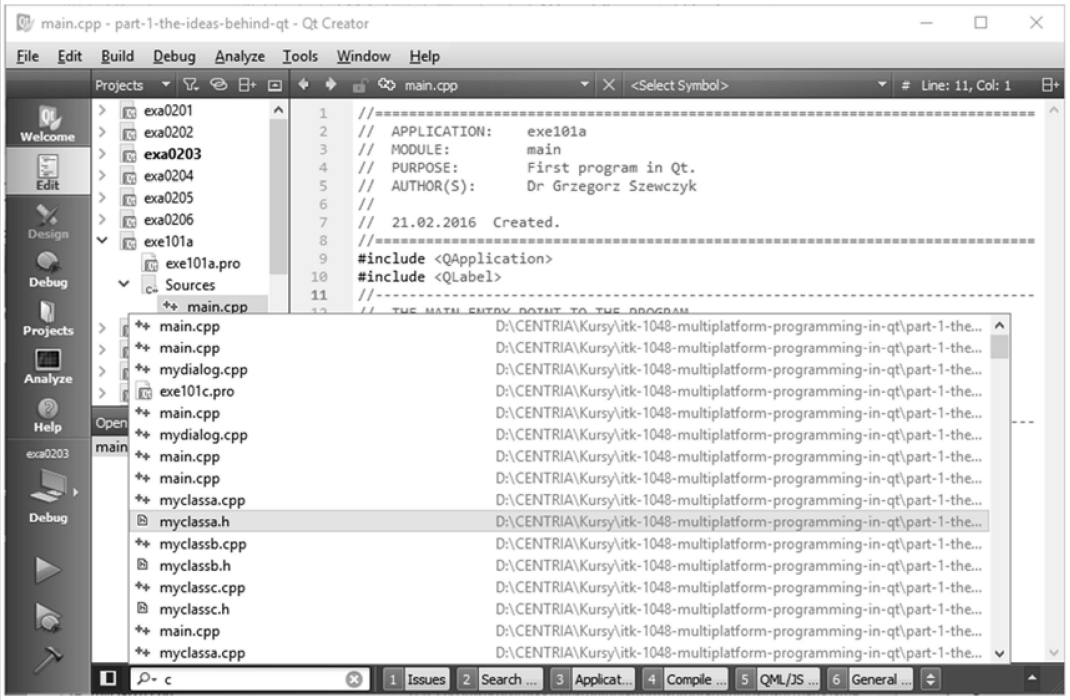


Help of Qt Creator IDE

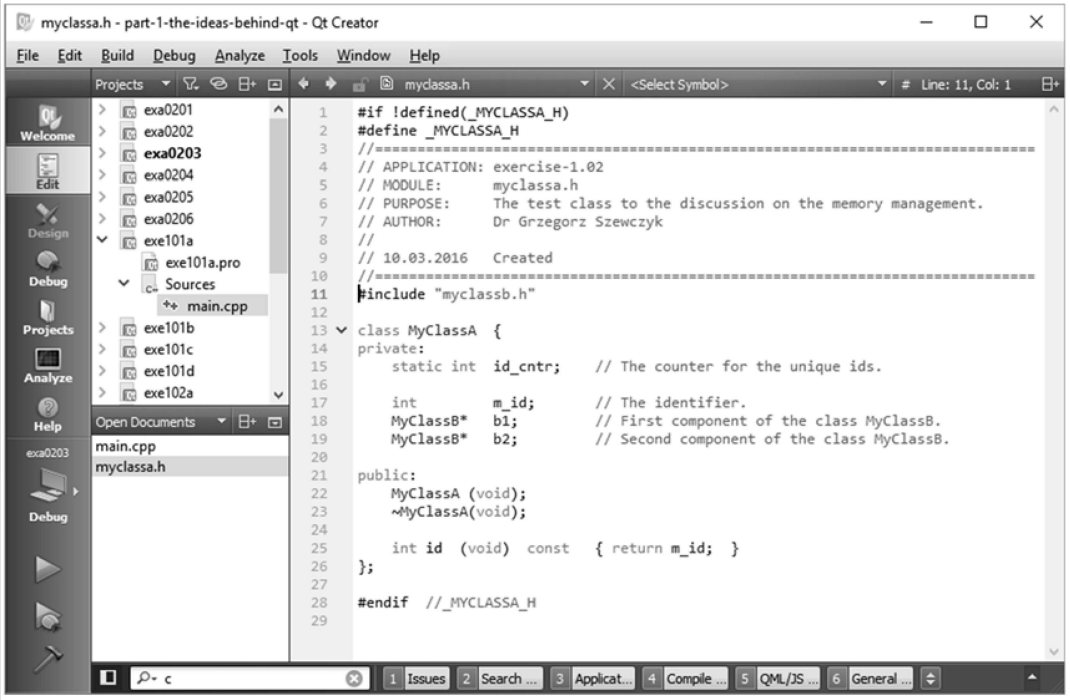

Qt Creator IDE
Sidebar
Opened files
Symbol overview
Mode selector
Target selector
Run
Debug
Build all
Locator
Output panes
main.cpp - part-1-the-ideas-behind-qt - Qt Creator
Copyright © 2005 - 2016 by Dr.Grzegorz Szewczyk, Finland


Qt Creator IDE - Locator
main.cpp - part-1-the-ideas-behind-qt - Qt Creator
Execute Custom Commands
C++ Symbols in Current Document
C++ Classes, Enums and Functions
Help Index
All Included C/C++ Files
Files in Any Project
C++ Classes
Files in File System
Line in Current Document
C++ Functions
QML Functions
Open Documents
Files in Current Project
Web Search
Text Editing Macros
Run External Tool
Copyright © 2005 - 2016 by Dr.Grzegorz Szewczyk, Finland

Qt Creator IDE - Locator



Qt Creator IDE - Locator



Exercise 1.01A

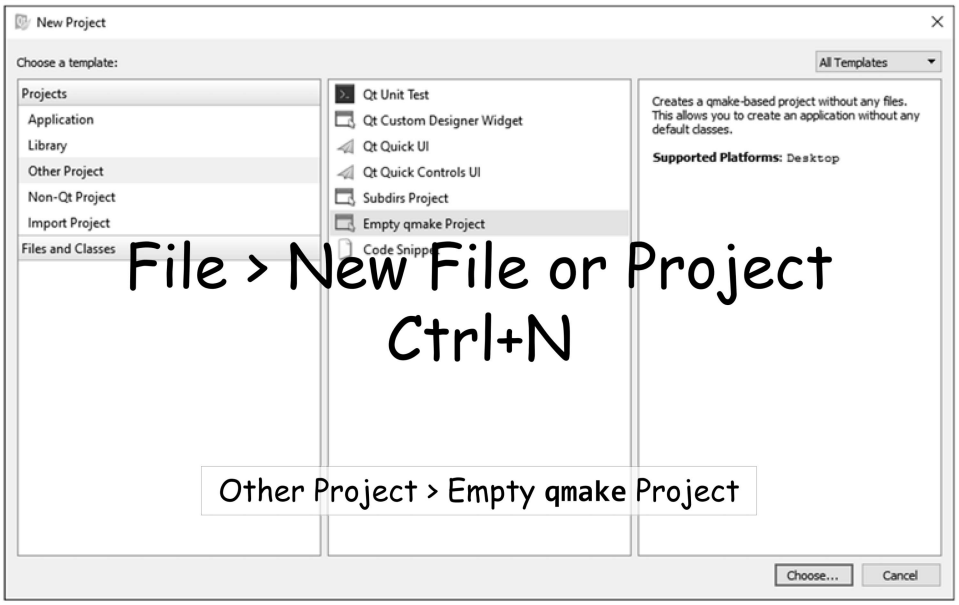
Design and develop application that uses an object of class QLabel to display the "Hello World!" text. The application should be developed in the programming mode.

Notes:

- Refer to the documentation of **QLabel** in Help or on http://doc.qt.io/.
- Take advantage on the examples being found in handouts.

… to be continued on the computer

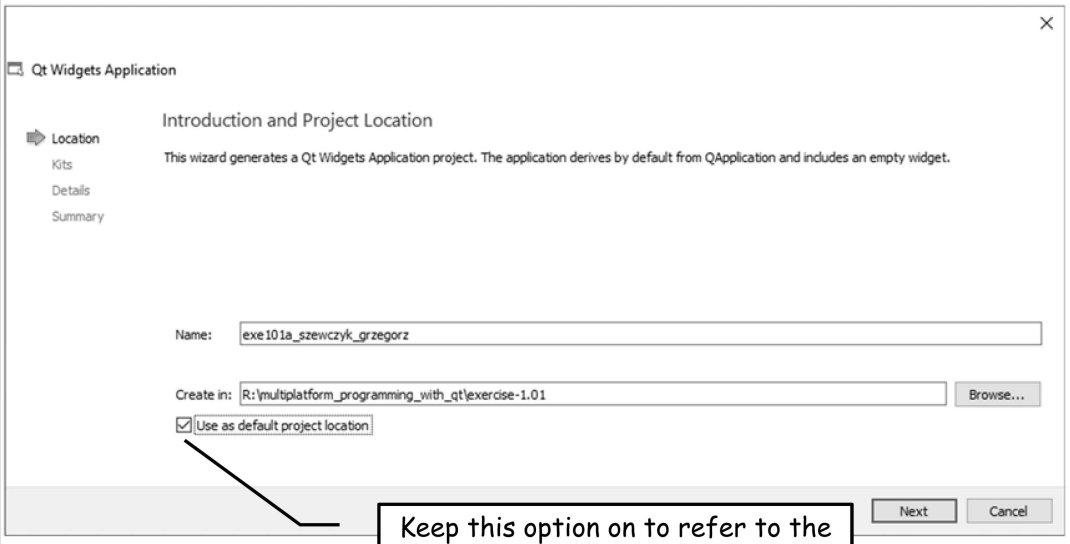
Exercise 1.01A (cont.)



File > New File or Project
Ctrl+N

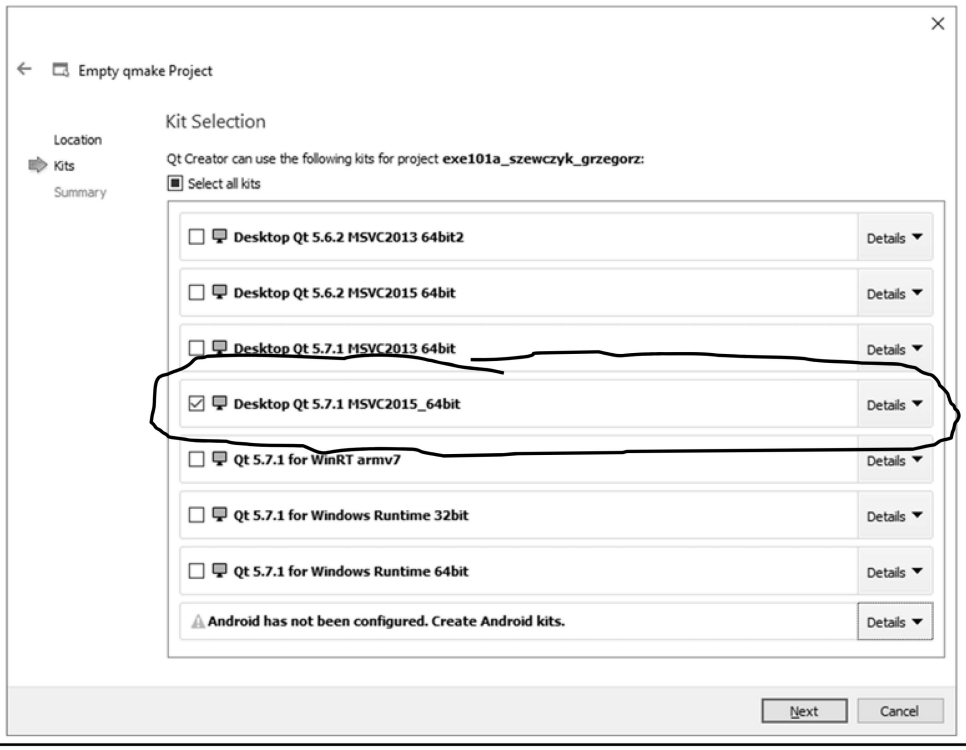
Other Project > Empty **qmake** Project

Exercise 1.01A (cont.)

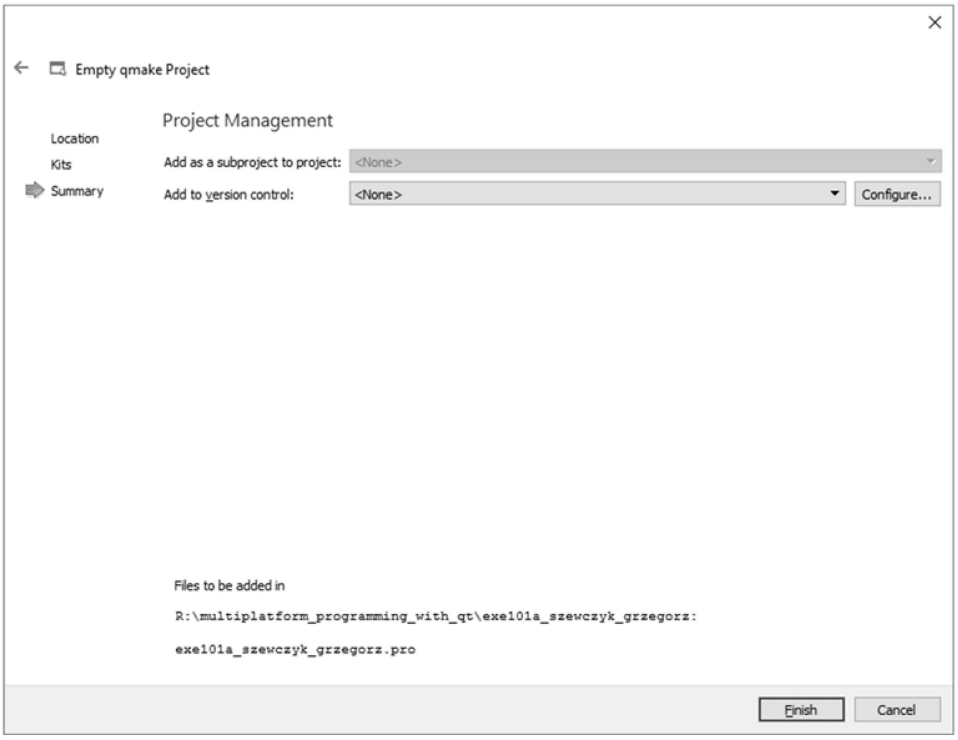


Keep this option on to refer to the same directory once subsequent project are to be created.

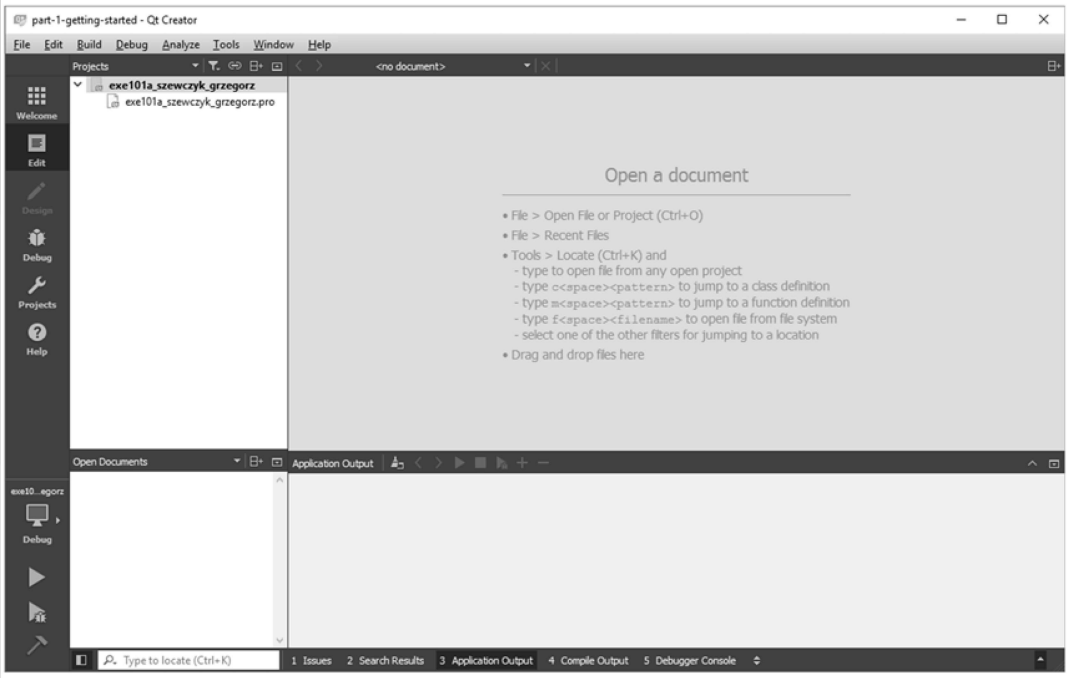
Exercise 1.01A (cont.)



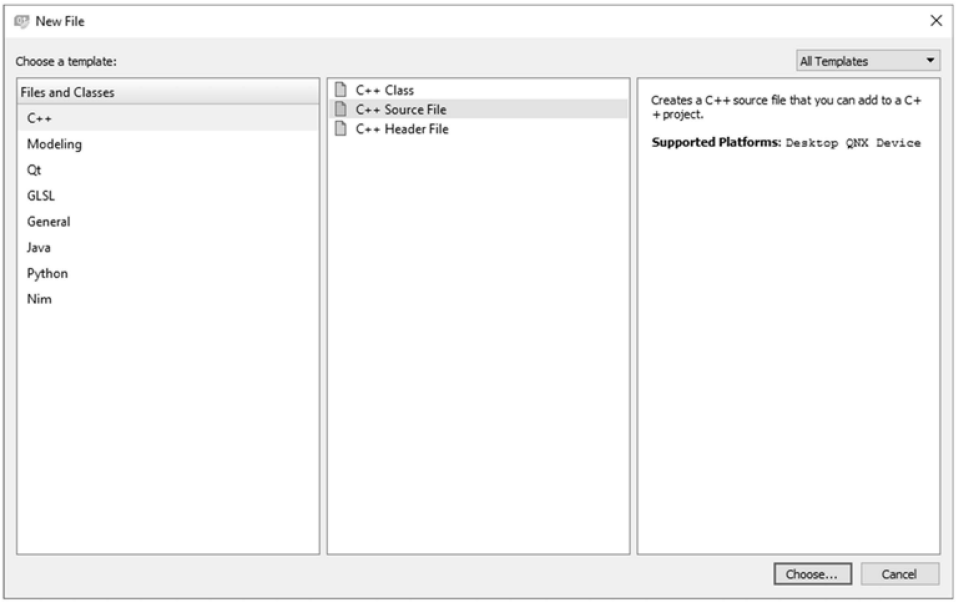
Exercise 1.01A (cont.)



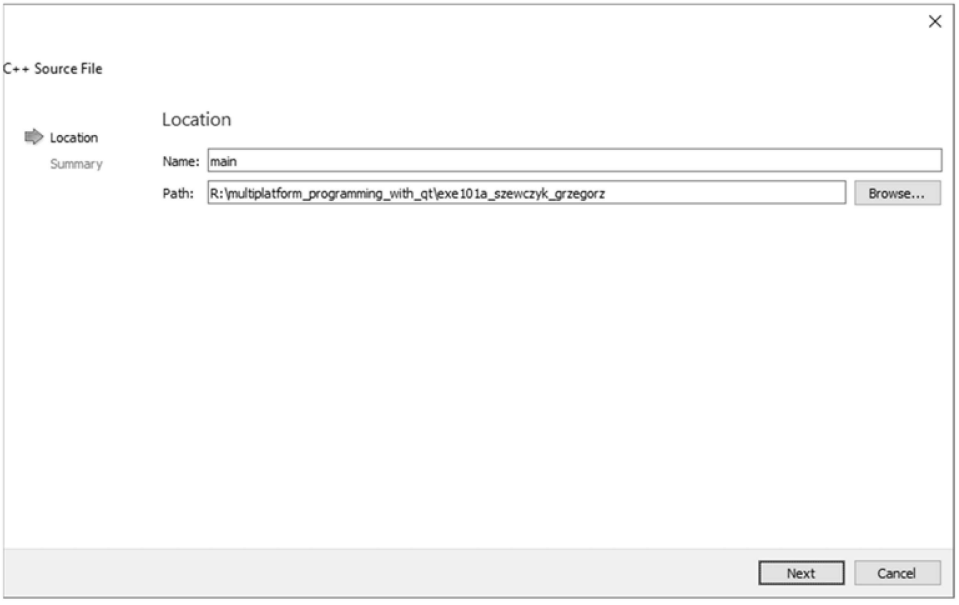
Exercise 1.01A (cont.)



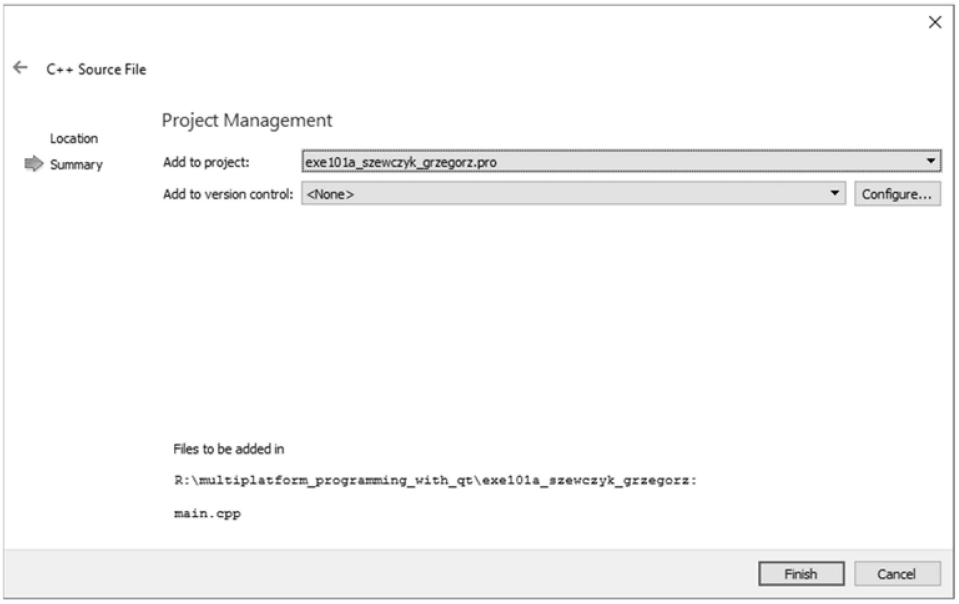
Exercise 1.01A (cont.)



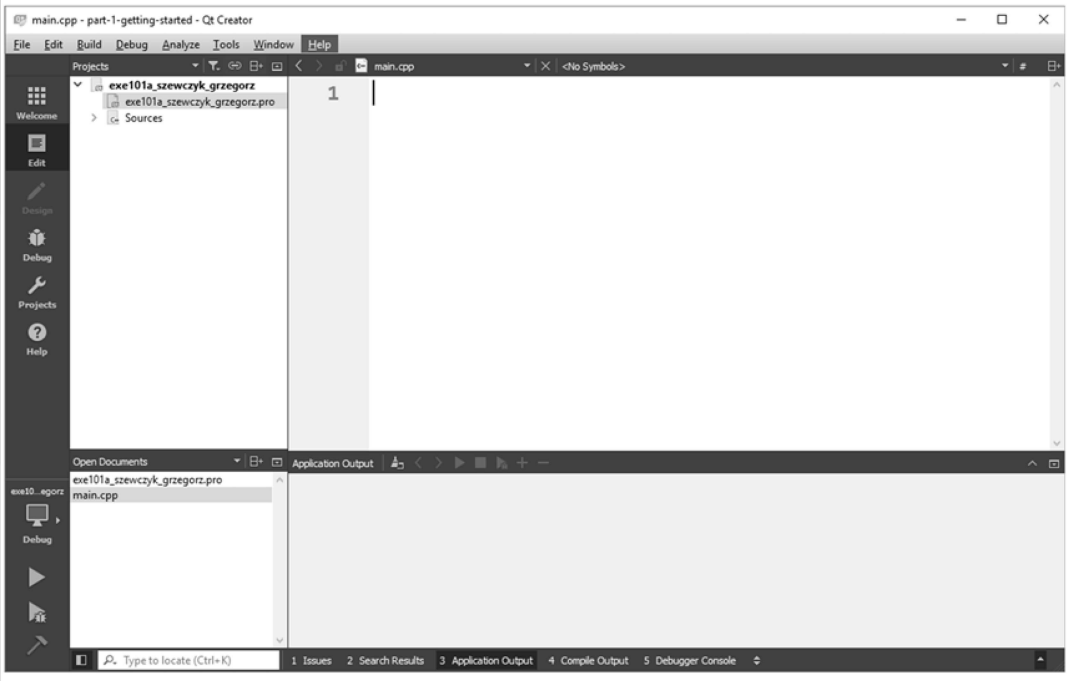
Exercise 1.01A (cont.)



Exercise 1.01A (cont.)



Exercise 1.01A (cont.)



Exercise 1.01A (cont.)

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication  appn(argc, argv);
    QLabel        lbl ("Hello World!");

    lbl.show();

    return appn.exec();
}
```

main.cpp

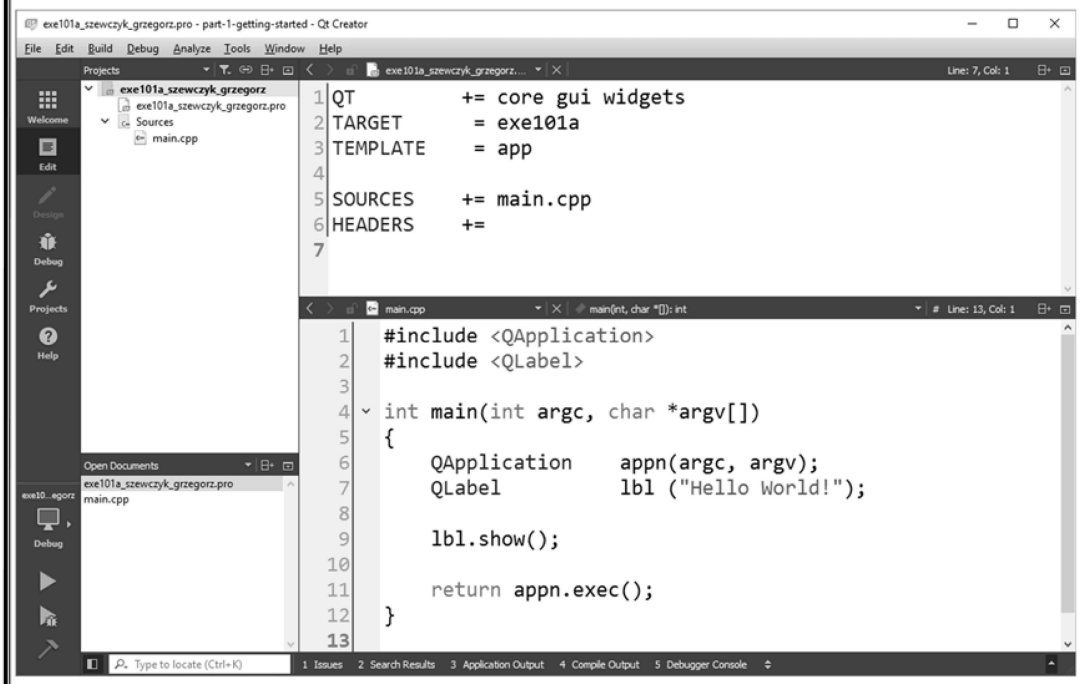
```
QT          += core gui widgets
TARGET       = exe101a
TEMPLATE     = app

SOURCES     += main.cpp
HEADERS     +=
```

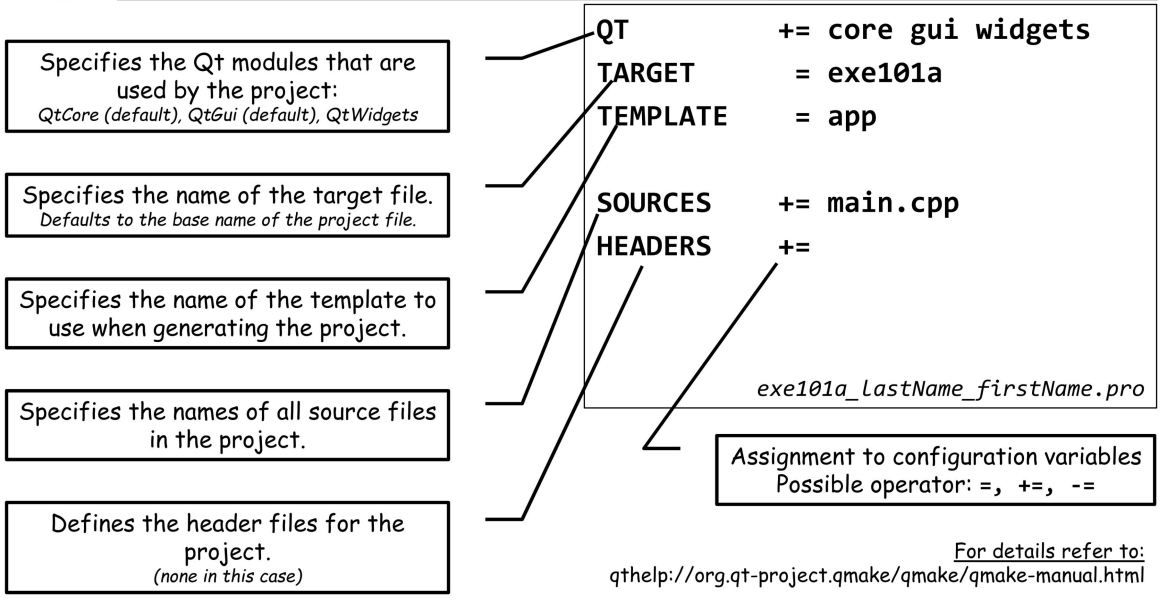
exe101a_lastName_firstName.pro

- Program consists of two files
  - main.cpp – containing statements and data.
  - *exe101a_lastName_firstName*.pro – containing configuration data of the project

Exercise 1.01A (cont.)



Exercise 1.01A (cont.)



For details refer to:
qthelp://org.qt-project.qmake/qmake/qmake-manual.html

Creating Project Files

- Project files contain all the information required by ***qmake*** to build an application, library, or plugin.
- Generally, it uses a series of declarations to specify the resources in the project, but support for simple programming constructs enables to describe different build processes for different platforms and environments.

Project File Elements

Variable	Contents
CONFIG	General project configuration options.
DESTDIR	The directory in which the executable or binary file will be placed.
FORMS	A list of UI files to be processed by the user interface compiler (uic).
HEADERS	A list of filenames of header (.h) files used when building the project.
INCLUDEPATH	Specifies the #include directories which should be searched when compiling the project.
RESOURCES	A list of resource (.qrc) files to be included in the final project. See the The Qt Resource System for more information about these files.
SOURCES	A list of source code files to be used when building the project.
TEMPLATE	The template to use for the project. This determines whether the output of the build process will be an application, a library, or a plugin.

Exercise 1.01A (cont.)

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication  appn(argc, argv);
    QLabel        lbl ("Hello World!");

    lbl.show();

    return appn.exec();
}
```

main.cpp

Libraries

Command line arguments
(like in C and C++)

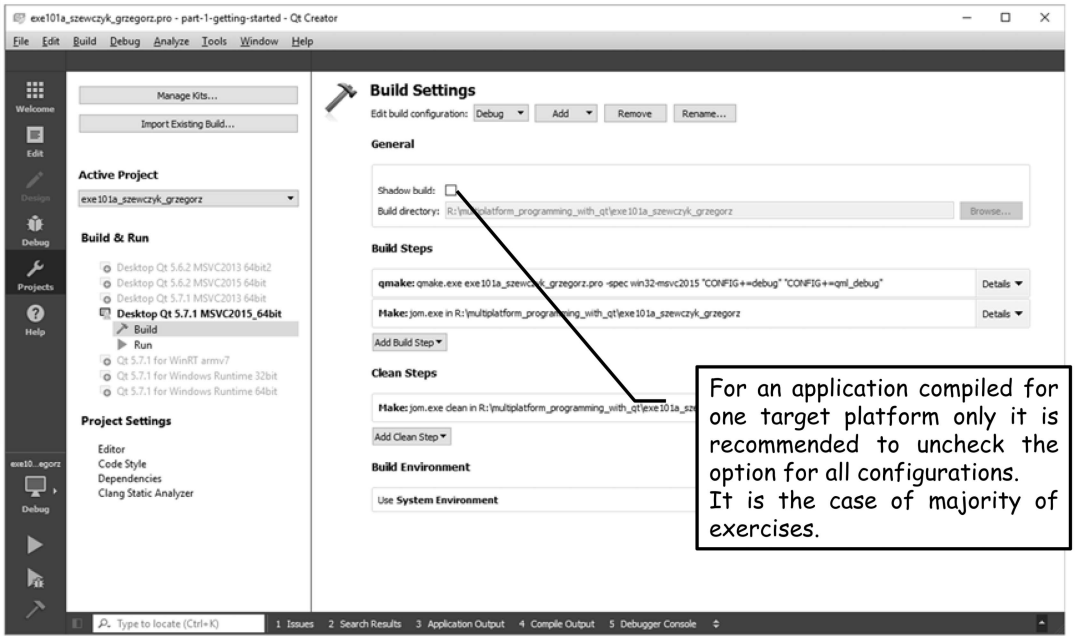
Application object.
(It is obligatory for each application.)

QLabel widget object - the main object of ther application.
*(Usually it is object inherited from **QDialog** or **QWindows**)*

Makes **QLabel** object and therefore the text visible

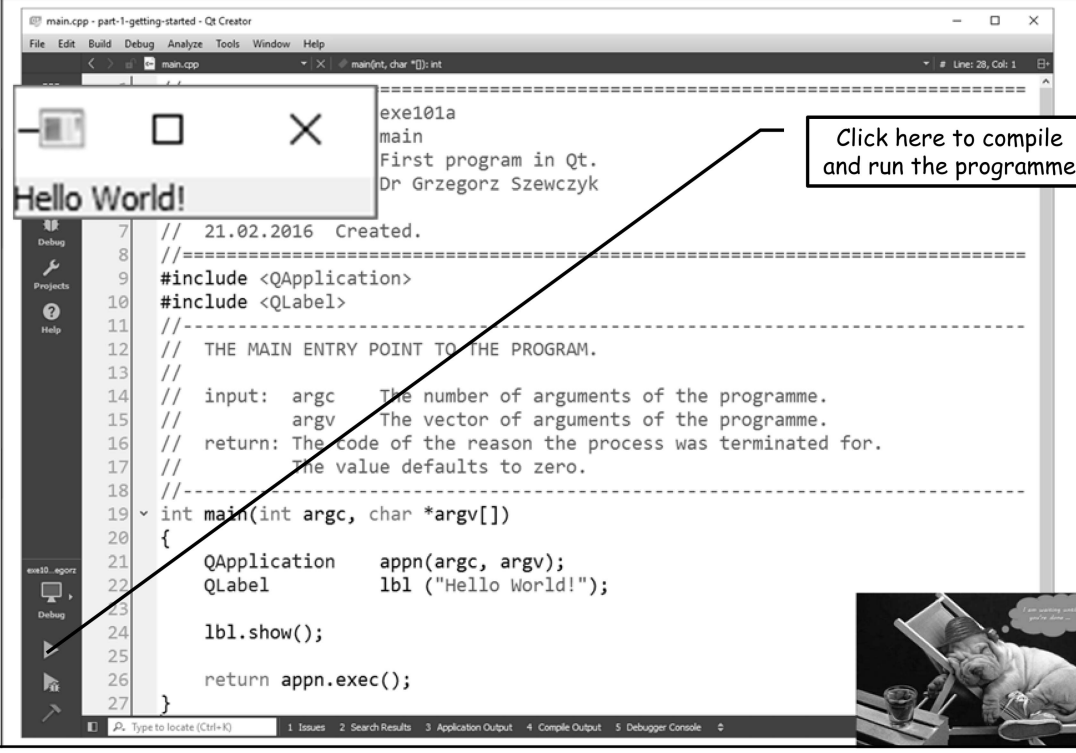
Enters the main event loop and waits until **exit()** is called, then returns the value that was set to **exit()**.

Exercise 1.01A (cont.)



For an application compiled for one target platform only it is recommended to uncheck the option for all configurations.
It is the case of majority of exercises.

Exercise 1.01A (cont.)



Click here to compile and run the programme

Exercise 1.01B

Design and develop application that uses an object of class QLabel to display the "Hello World!" text. The application should be developed in the graphical mode.

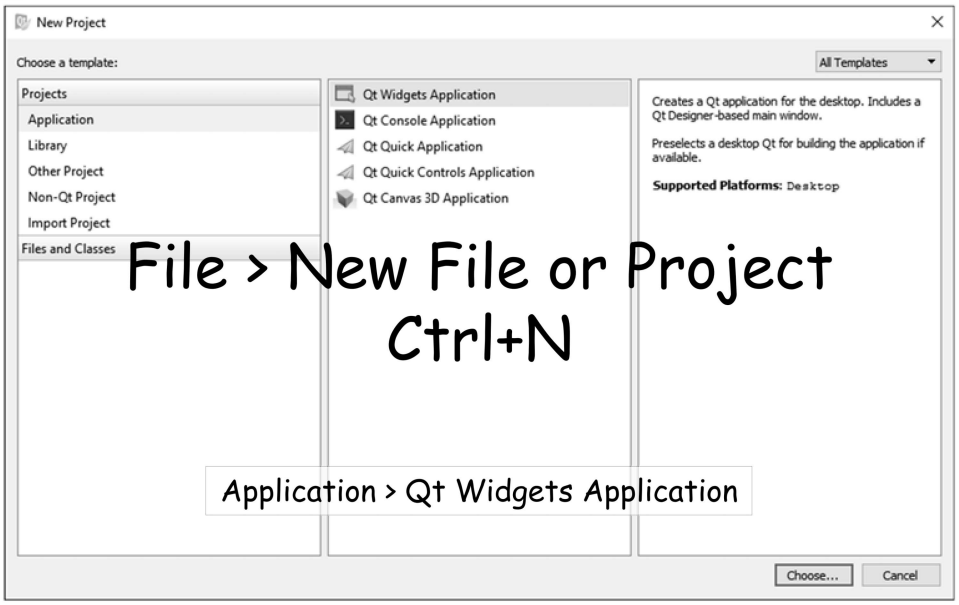
In Designer, adjust the properties of the QLabel object in this way that the size of the text font will be 36 points and the text will be centralised horizontally and vertically in the window.

Notes:

- Refer to the documentation of **QLabel** in Help or on http://doc.qt.io/.
- Take advantage on the examples being found in handouts.

… to be continued on the computer

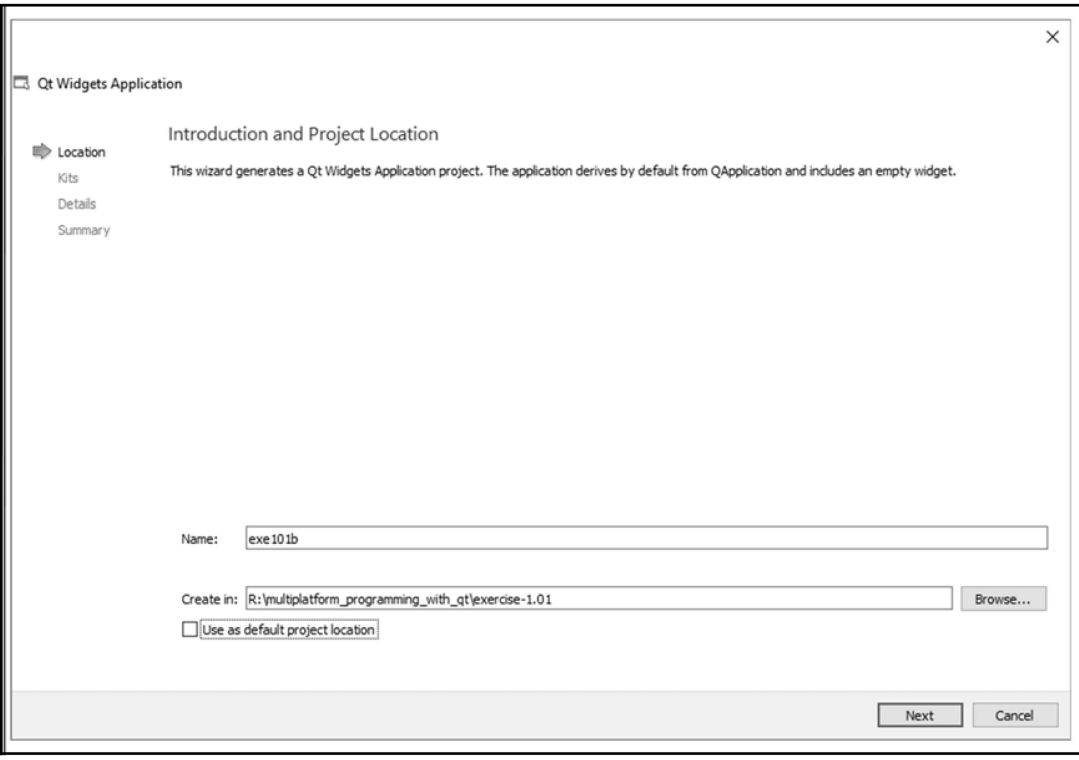
Exercise 1.01B (cont.)



File > New File or Project
Ctrl+N

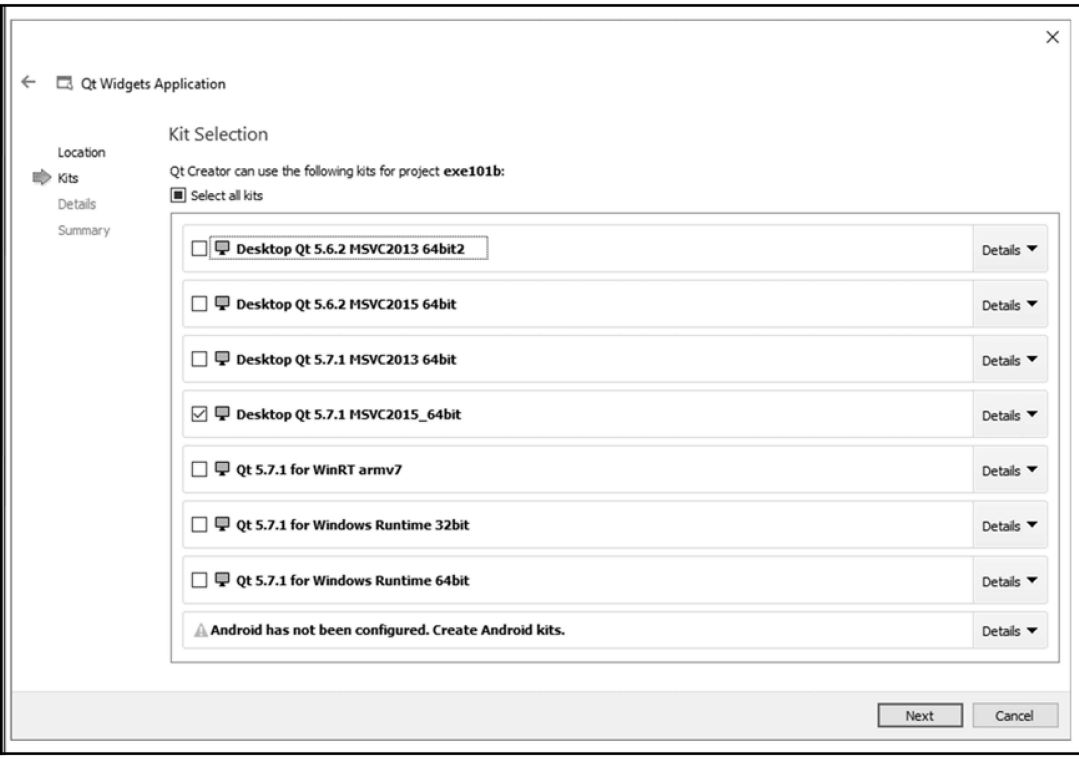
Application > Qt Widgets Application

Exercise 1.01B (cont.)



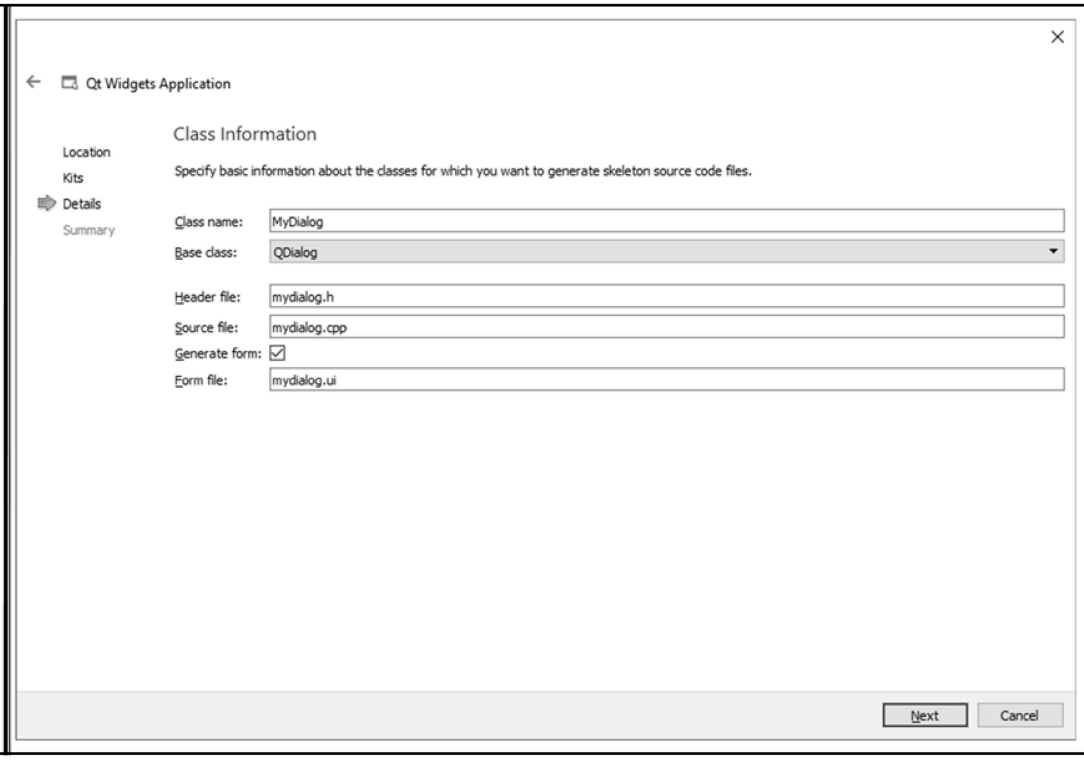
31

Exercise 1.01B (cont.)



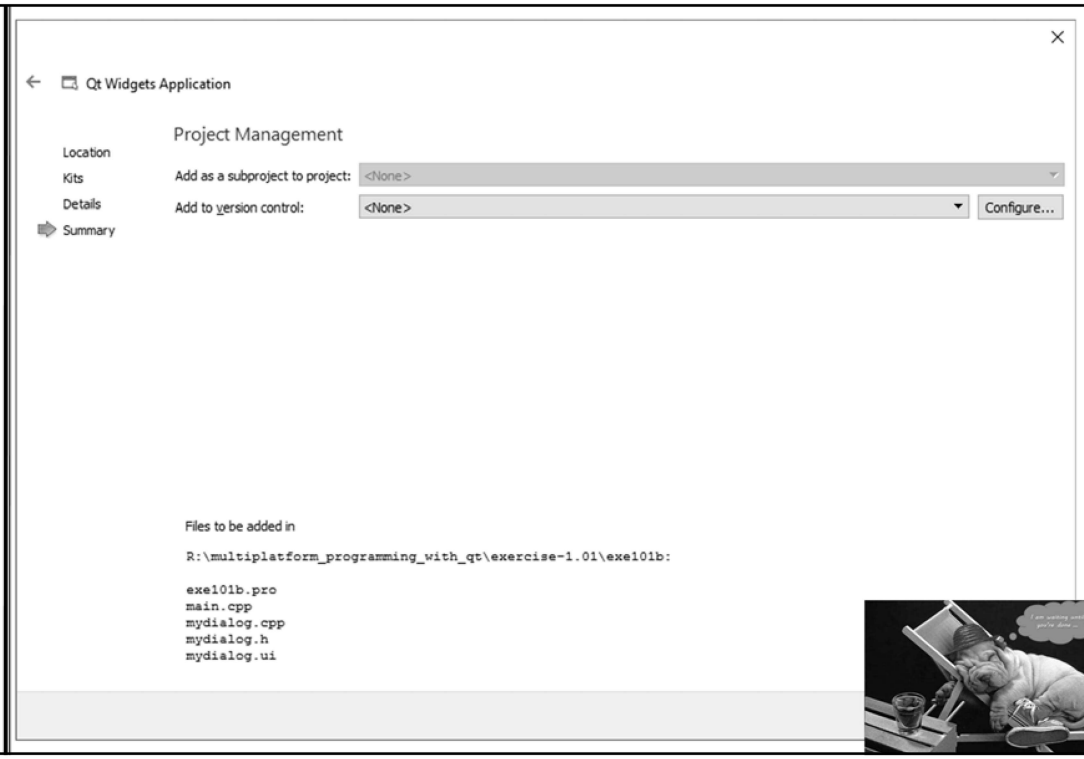
32

Exercise 1.01B (cont.)



33

Exercise 1.01B (cont.)



34

Exercise 1.01B (cont.)

```
#include <QApplication>
#include "mydialog.h"

int main(int argc, char *argv[])
{
   Qapplication  appn(argc, argv);
   MyDialog      dlg;

   dlg.show();

   return appn.exec();
}
```

```
QT          += core gui widgets
TARGET       = exe101b
TEMPLATE     = app

SOURCES     += main.cpp \
               mydialog.cpp
HEADERS     += mydialog.h
FORMS       += mydialog.ui
```


Exercise 1.01B (cont.)

This character enables continuation of the assignment statement in the next row

Specifies the form containing the design of the UI

```
QT          += core gui widgets
TARGET       = exe101b
TEMPLATE     = app

SOURCES     += main.cpp \
               mydialog.cpp
HEADERS     += mydialog.h
FORMS       += mydialog.ui
```

Exercise 1.01B (cont.)

```
#include <QApplication>
#include "mydialog.h"

int main(int argc, char *argv[])
{
   QApplication  appn(argc, argv);
   MyDialog      dlg;

   dlg.show();

   return appn.exec();
}
```

The file containing the class of the main object of the application.

Main object of the application. It plays the similar role like the QLabel in the Task A

Exercise 1.01B (cont.)

```
#ifndef MYDIALOG_H
#define MYDIALOG_H

#include <QDialog>

namespace Ui {
    class MyDialog;
}

class MyDialog : public QDialog
{   Q_OBJECT
    Ui::MyDialog   *ui;

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

};

#endif // MYDIALOG_H
```

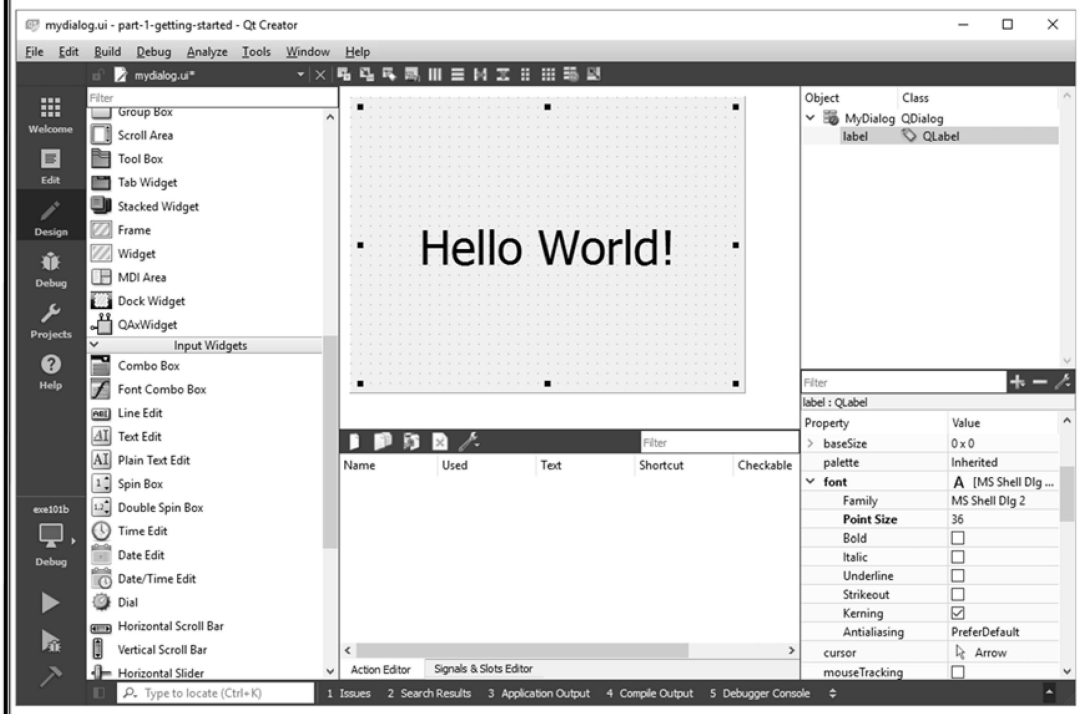
The Qt class generated from the form designed graphically in Designer

```
#include "mydialog.h"
#include "ui_mydialog.h"

// Class constructor
//
MyDialog::MyDialog(QWidget* parent) :
    QDialog(parent),
    ui(new Ui::MyDialog)
{
    ui->setupUi(this);
}

// Class destructor
//
MyDialog::~MyDialog()
{
    delete ui;
}
```

Exercise 1.01B (cont.)



Exercise 1.01C

Create the application in the same way like in the Task B.

Next, set the name of the **QLabel** object to **lblGreetings**.

Programmatically set the text of the message and adjust the properties of the object in the way that the text font size will be 36 points and the will be centralised horizontally and vertically in the window.

Notes:

- Refer to the documentation of **QLabel** and **QFont** in Help or on http://doc.qt.io/.
- Take advantage on the examples being found in handouts.

... to be continued on the computer

Exercise 1.01C (cont.)

```
MyDialog::MyDialog(QWidget *parent) : QDialog(parent), ui(new
Ui::MyDialog), fontSizePoint(36)
{
    ui->setupUi(this);
    ui->lblGreetings->setText(tr("Hello World!"));
    ui->lblGreetings->setFont(
            QFont(ui->lblGreetings->font().family(), fontSizePoint)
    );
    ui->lblGreetings->setAlignment(Qt::AlignCenter);
}
```

Method to set a text.

Method to set the font of a text.

The function facilitates the translation of a text, returning a translated version.
(*It will be discussed later*)

A font object

Method to set a mode of the text alignment.

Exercise 1.01D

Take advantage on the Task C and amend application of Task A to make its UI looking very similar to the UI of the application developed in Task C

ITK-1048 - Multiplatform Programming in Qt

Part 1 – Practical Tips for Developers

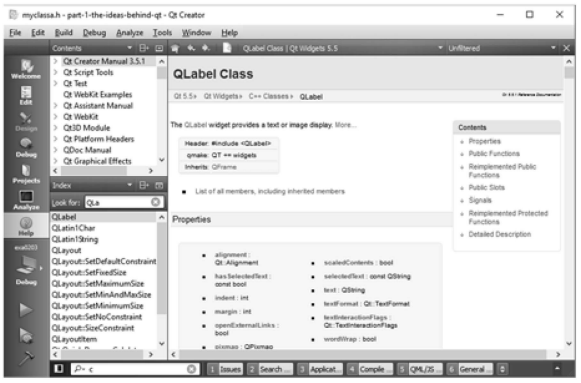
PRACTICAL TIPS FOR DEVELOPERS

How Much C++ Do You Need To Know?

- Objects and classes
  - Declaring a class,
  - Inheritance,
  - calling member functions
  - etc.
- Polymorphism
  - That is virtual methods
- Operator overloading
- Templates
  - For the container classes only
- No ...
  - ... RTTI
  - ... sophisticated templates
  - ... exceptions thrown
  - ...

Qt Documentation

- Reference Documentation
  - All classes documented
  - Contains tons of examples
- Collection of Howto's and Overviews
- A set of Tutorials for Learners



Finding the Answers

- Documentation in Qt Assistant (or QtCreator)
- Qt's examples: $QTDIR/examples
- Qt developer network: http://qt-project.org/
- Qt Centre Forum: http://www.qtcentre.org/
- KDE project source code (cross-referenced): http://lxr.kde.org/ .
- Online communities: http://qt-project.org/wiki/OnlineCommunities

Use the source!

Qt's source code is easy to read, and can answer questions the reference manual cannot answer!

Modules and Include files

- Qt Modules
  - QtCore, QtGui, QtWidgets, QtXml, QtSql, QtNetwork, QtTest ...
- Enable Qt Module in qmake .profile:

  QT += network

Qt Modules Documentation

Modules and Include files

- Default: ***qmake*** projects use QtCore and QtGui
  - Any Qt class has a header file.

    #include <QLabel>

    #include <QtWidgets/QLabel>
  - Any Qt Module has a header file.

    #include <QtGui>

Includes and Compilation Time

- *Module includes*
  #include <QtGui>
- *Precompiled header and the compiler*
  - *If not supported may add extra compile time*
  - *If supported may speed up compilation*
  - *Supported on: Windows, Mac OS X, Unix*

Precompiled headers Documentation

Includes and Compilation Time

- Class includes
  #include <QLabel>
- Reduce compilation time
  - Use class includes (#include <QLabel>)
  - Forward declarations (class QLabel;)
- Place module includes before other includes.