Qt – Overview and development environment

Timo Strömmer, May 24, 2010

Contents

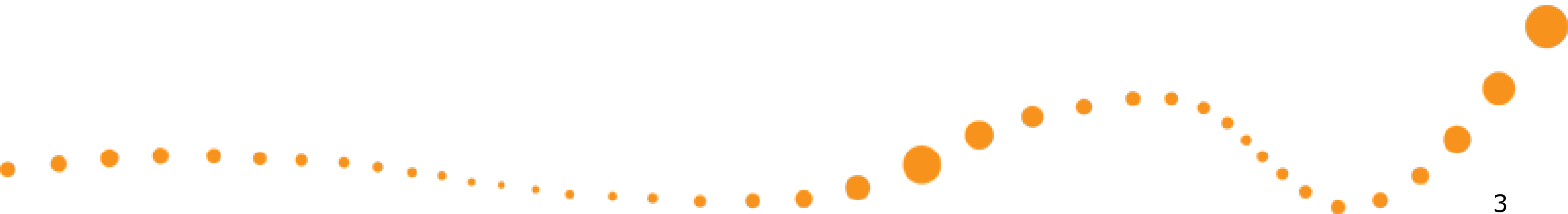
<symbio>

- Quick start
  - QtCreator demonstration
- Qt overview
  - Project basics and building
  - Qt modules walkthrough
- QtCreator overview
  - Session manager
  - Project properties
  - Code editor
  - Integrated help

Contents

<symbio>

- Beyond project file basics
  - Shared libraries
  - Managing larger projects
  - Platform-specific issues



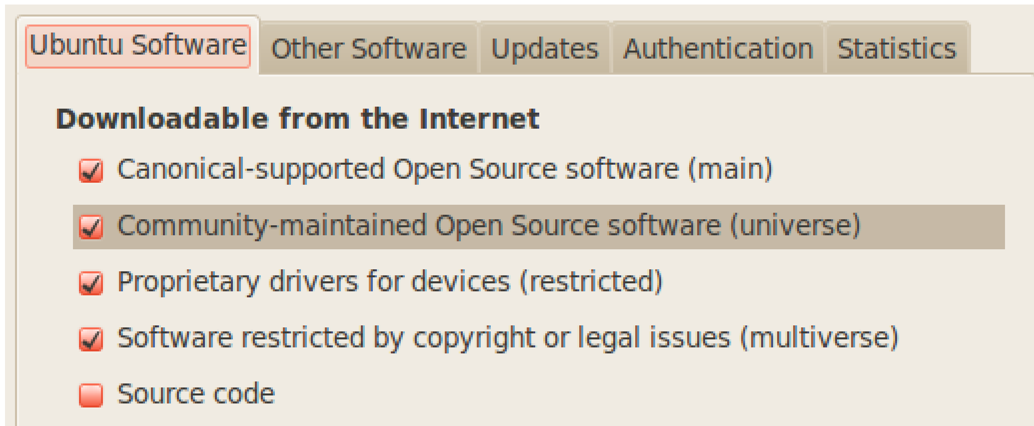
symbio

Quick start

Creating a hello world project with QtCreator

Installation

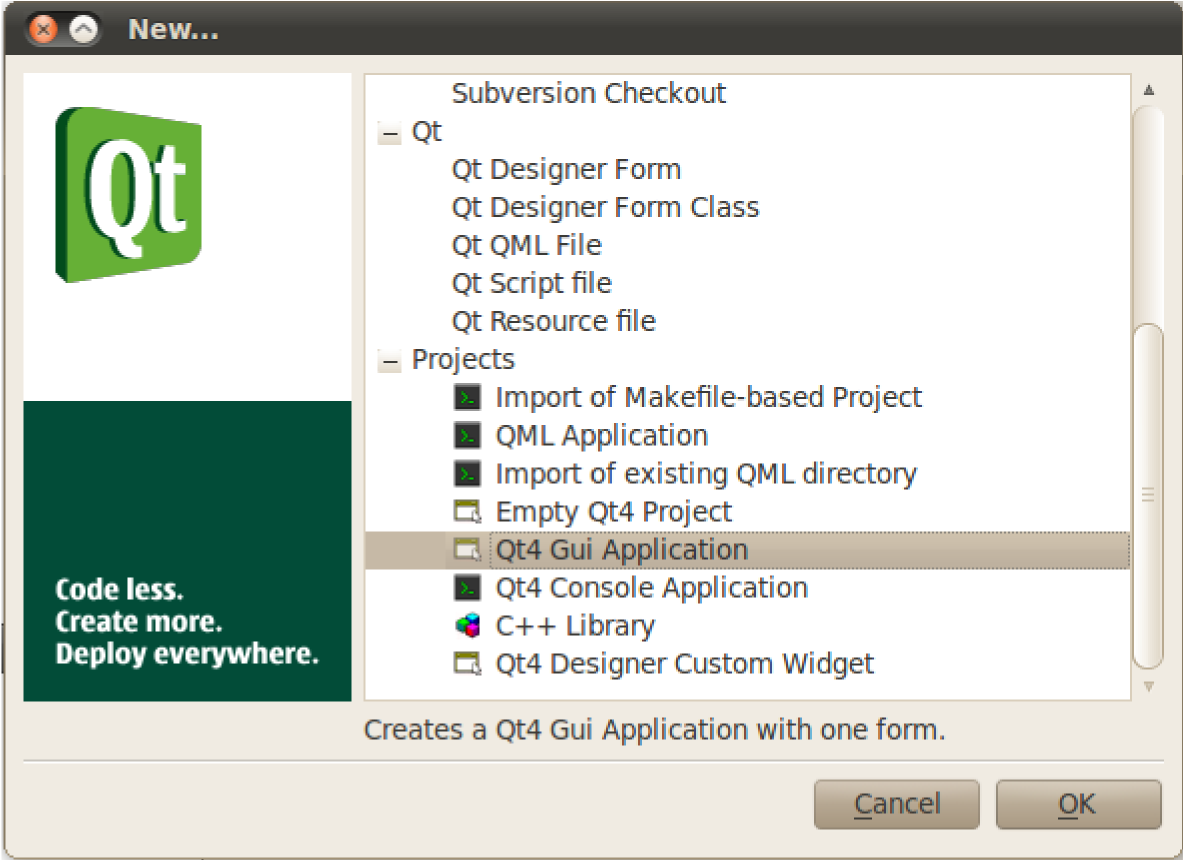
- Ubuntu 10.04 repositories have Qt 4.6.2:
  - sudo apt-get install qtcreator build-essential
- Need to enable universe repository from Software Sources



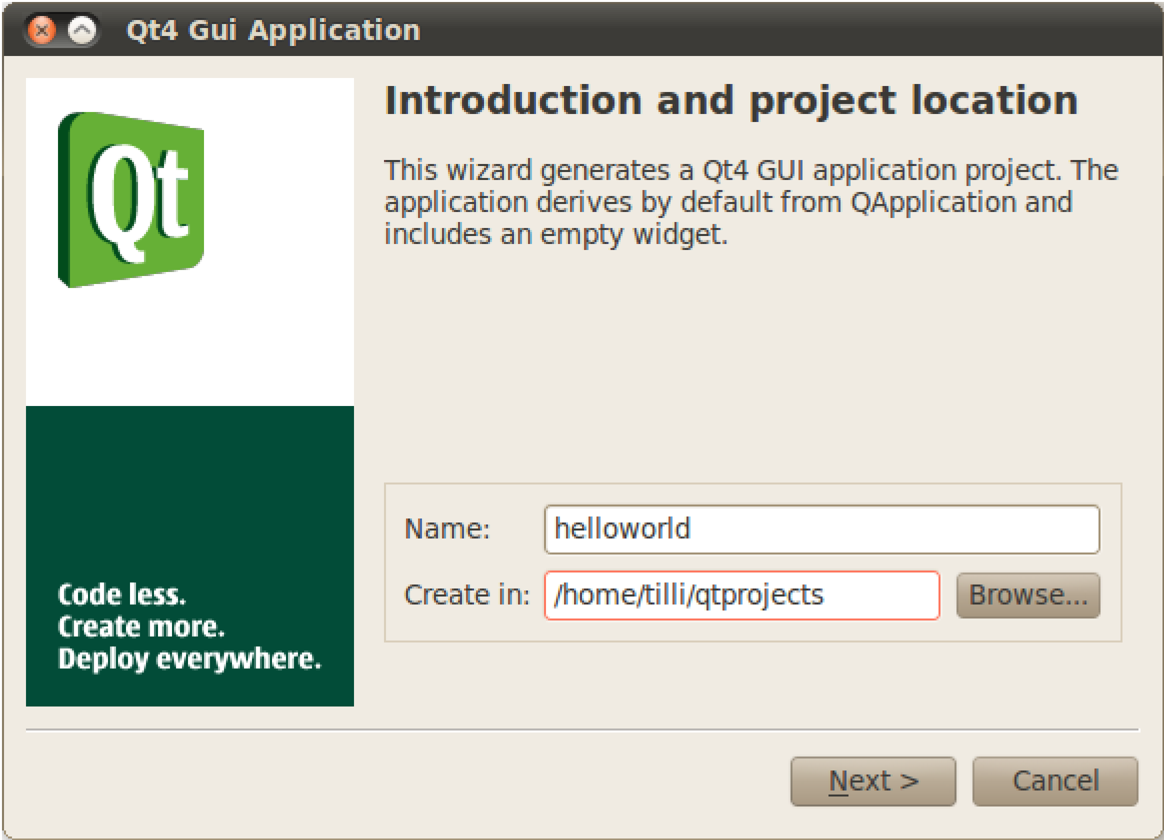
Quick start

- Run qtcreator
- Select *File / New File or Project*

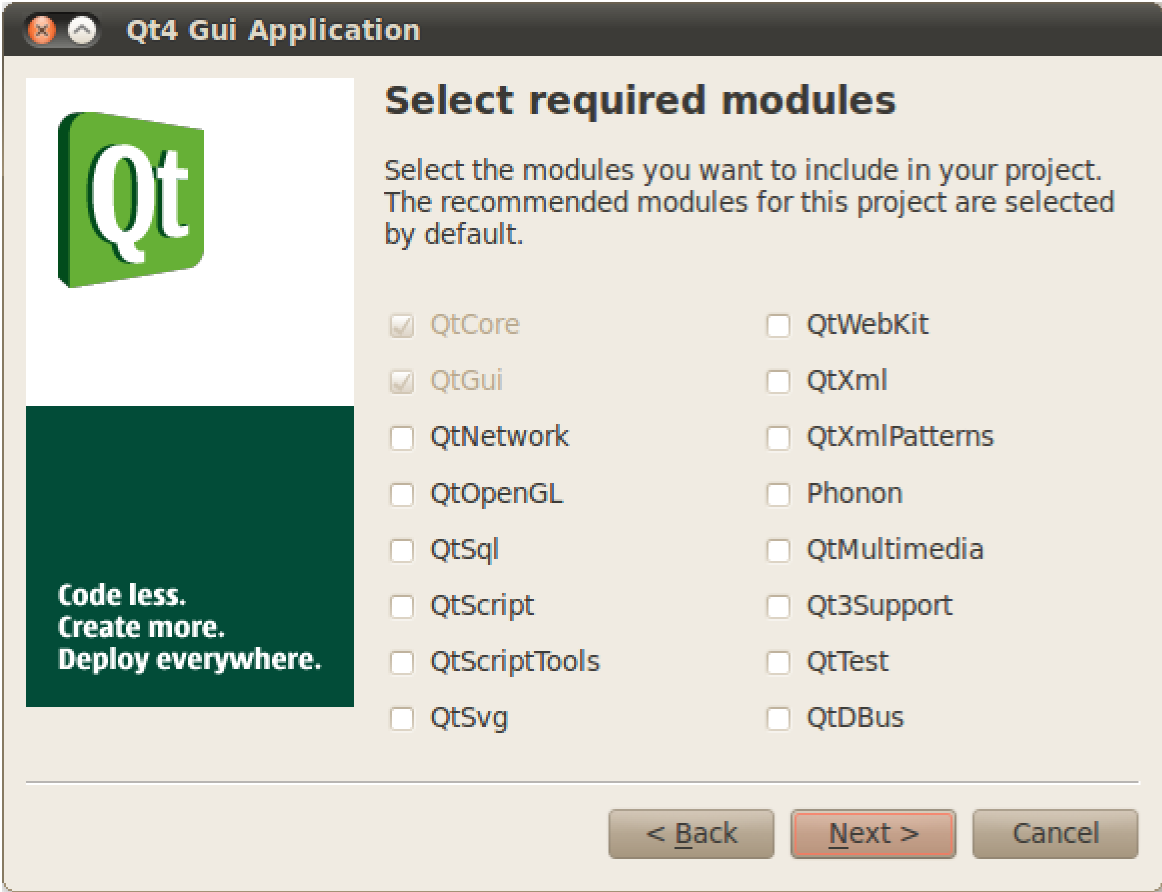
Quick start



Quick start

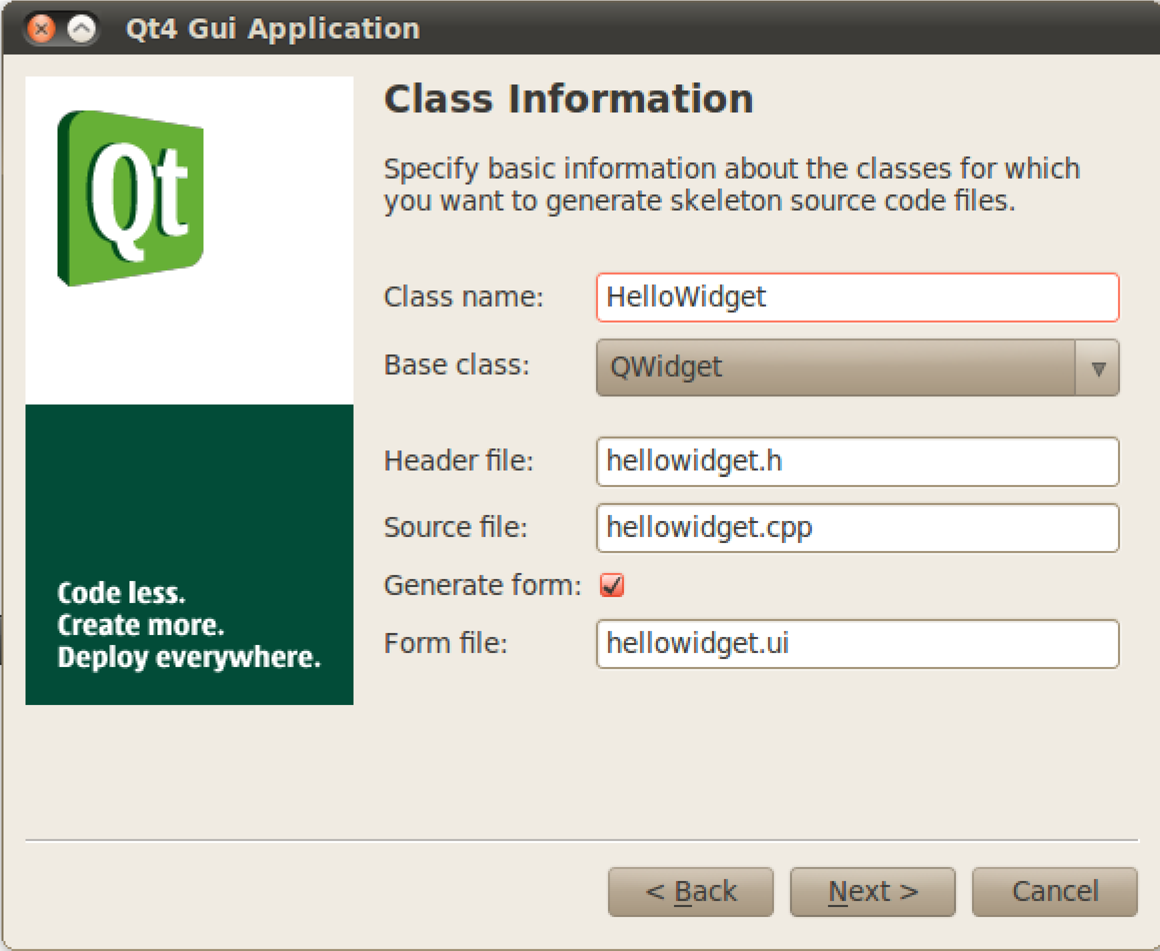


Quick start

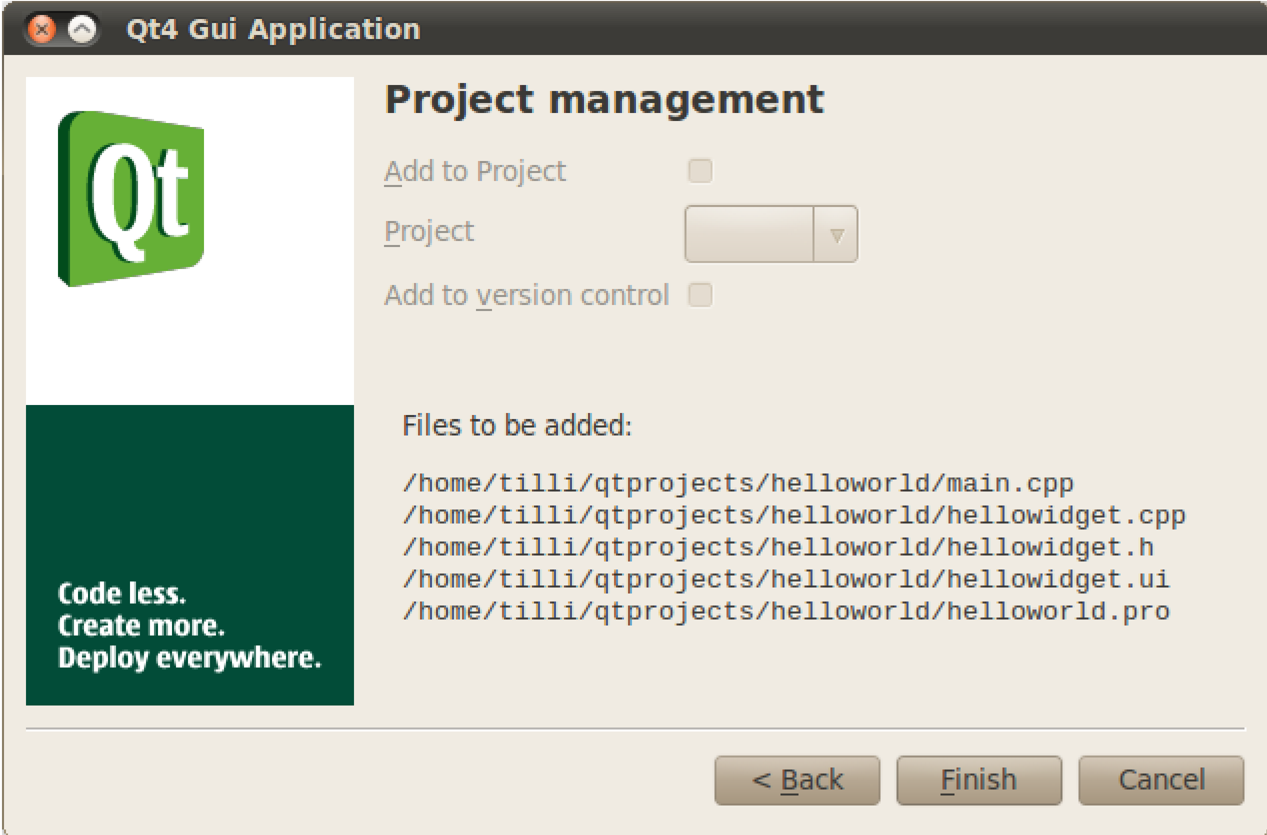


Quick start

<symbio>

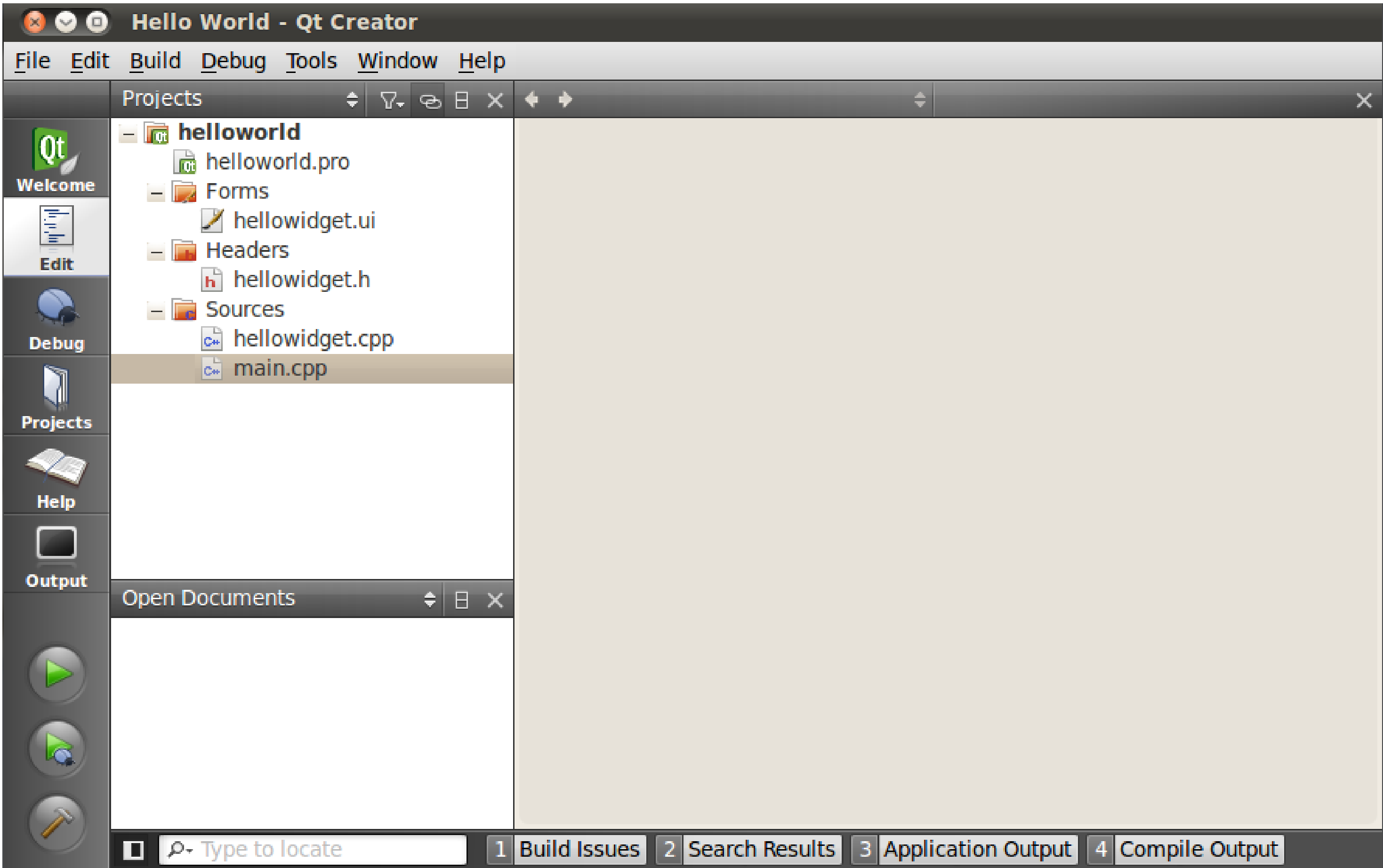


Quick start



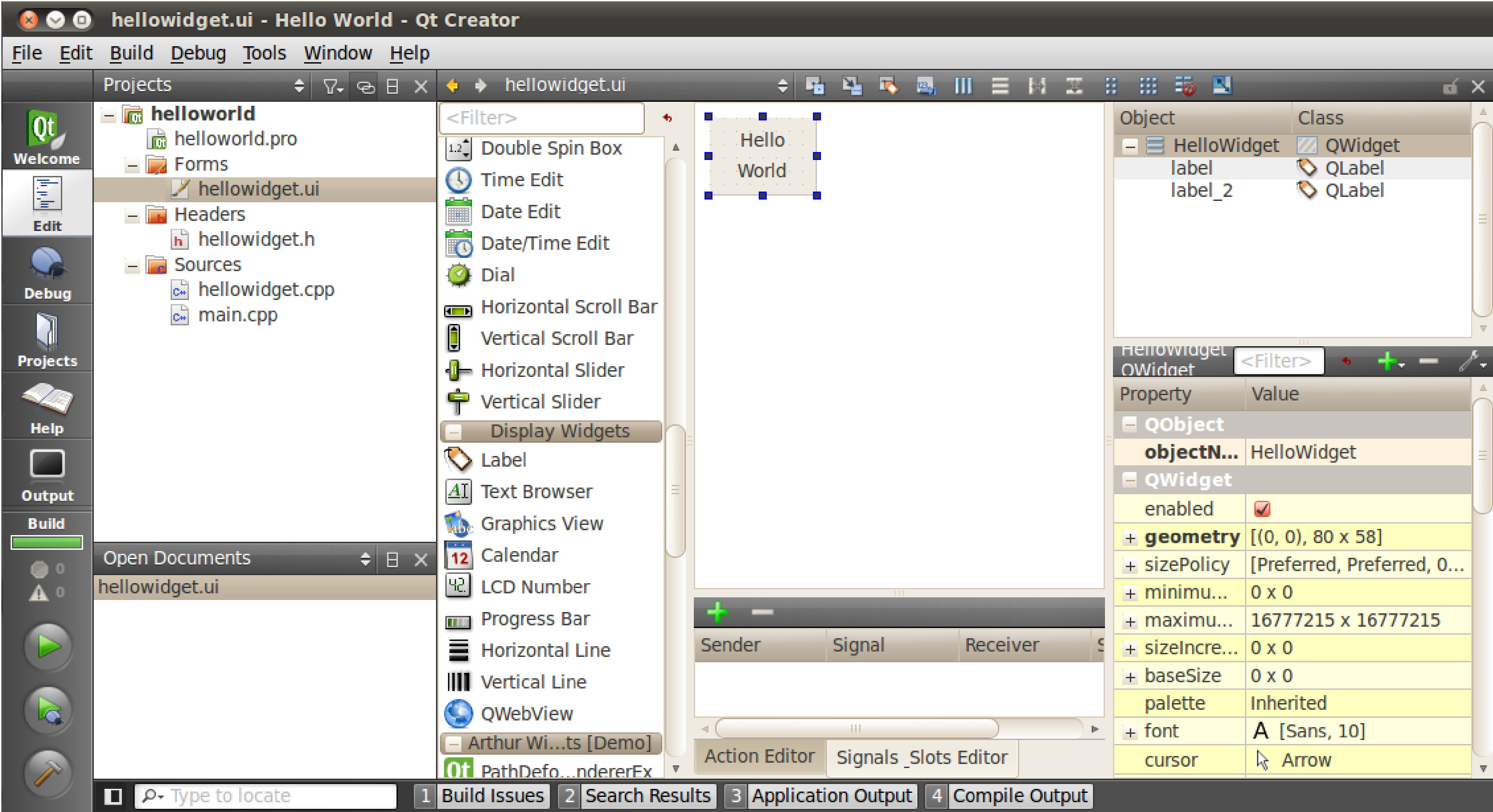
Quick start

<symbio>



Quick start

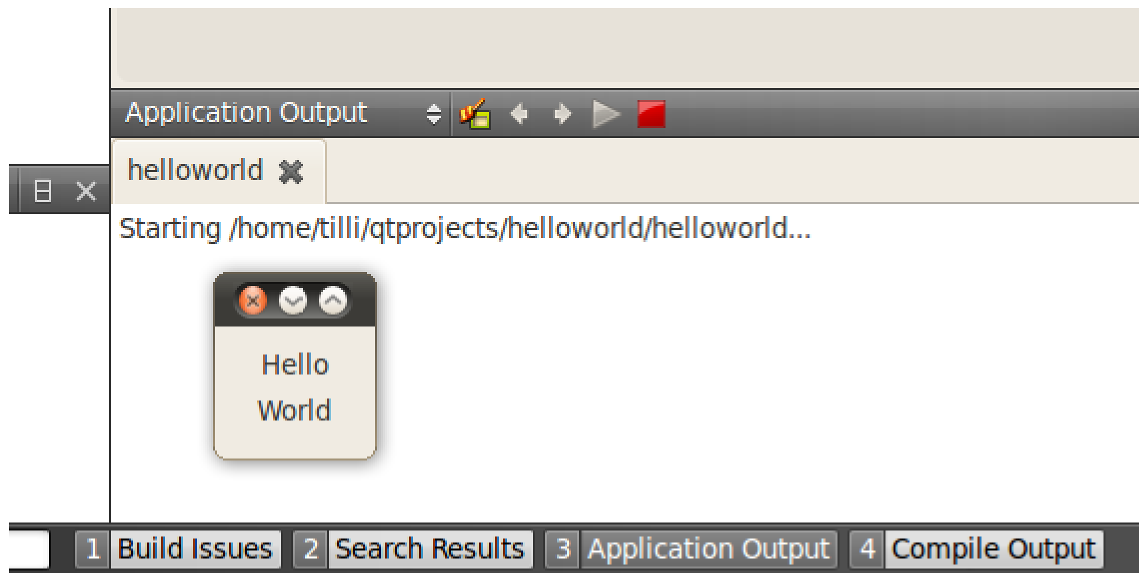
<symbio>



Quick start

<symbio>

- Build with Ctrl+B, run with Ctrl+R



Excercise

<symbio>

- Try it out, create a GUI helloworld project
  - Add some widgets with UI designer
- Build and run

<symbio>

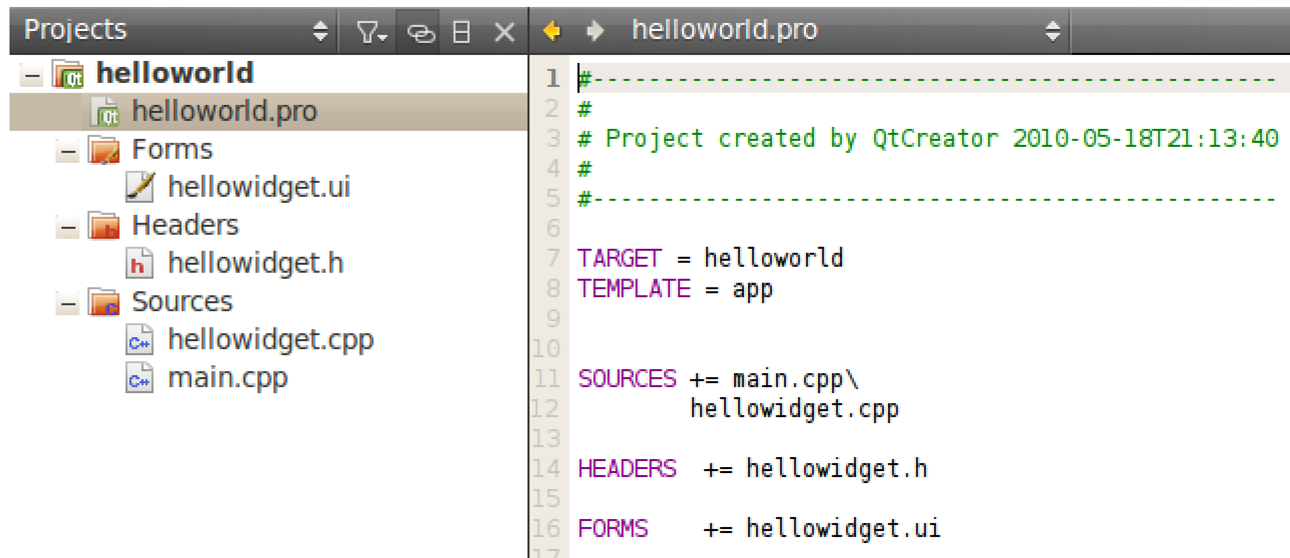
Qt projects

Basics

Qt project file

<symbio>

- A *.pro* file with same name as the directory it sits in
- Processed by *qmake* to generate platform-specific build files



Qt project basics

<symbio>

- Project name and type
  - TARGET, TEMPLATE
- Project files
  - SOURCES, HEADERS, FORMS
- Project configuration
  - CONFIG, QT

Project templates

- Basic TEMPLATE types: app, lib, subdirs
  - Executable files (console or GUI) are created with the *app* type
    - GUI is default, console needs *CONFIG += console*
  - Libraries (static and shared) are created with *lib* type
    - Shared default, static needs *CONFIG += staticlib*
  - Sub-directory template is used to structure large projects into hierarchies

Project name

<symbio>

- Project TARGET specifies the output file name
  - *TARGET = helloworld*
- Affected by template and platform
  - Executable name (*name*, *name*.exe etc.)
  - Library name (*libname.so*, *name.dll* etc.)

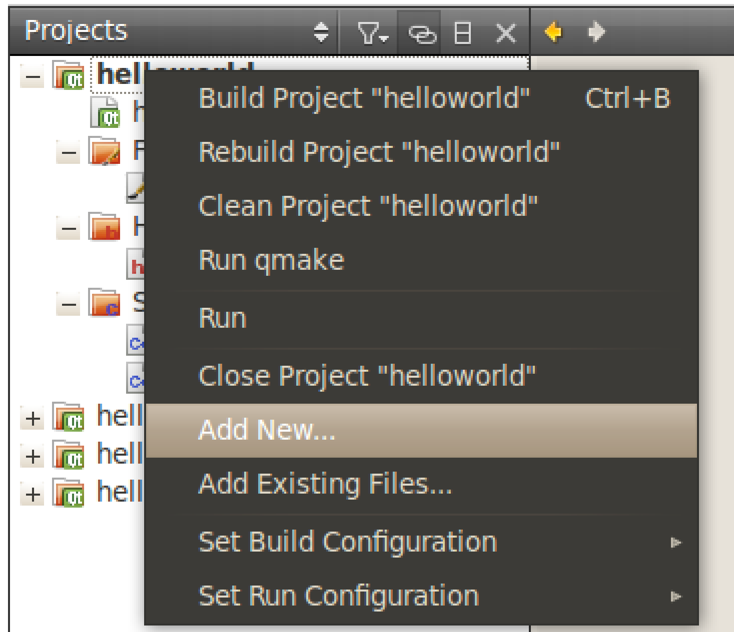
Project files

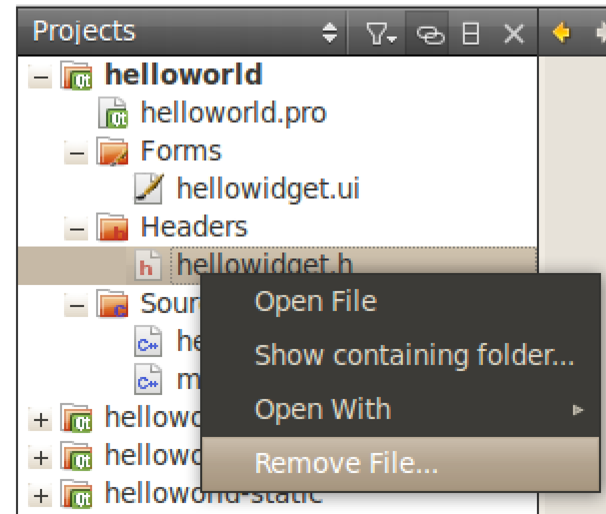
- SOURCES are obviously needed
- HEADERS also, as they are processed by meta-object compiler
- UI form data (.ui files) are included with FORMS directive

Sources and headers

<symbio>

- QtCreator updates the directives in *.pro* file in most cases
  - Add and remove but no rename



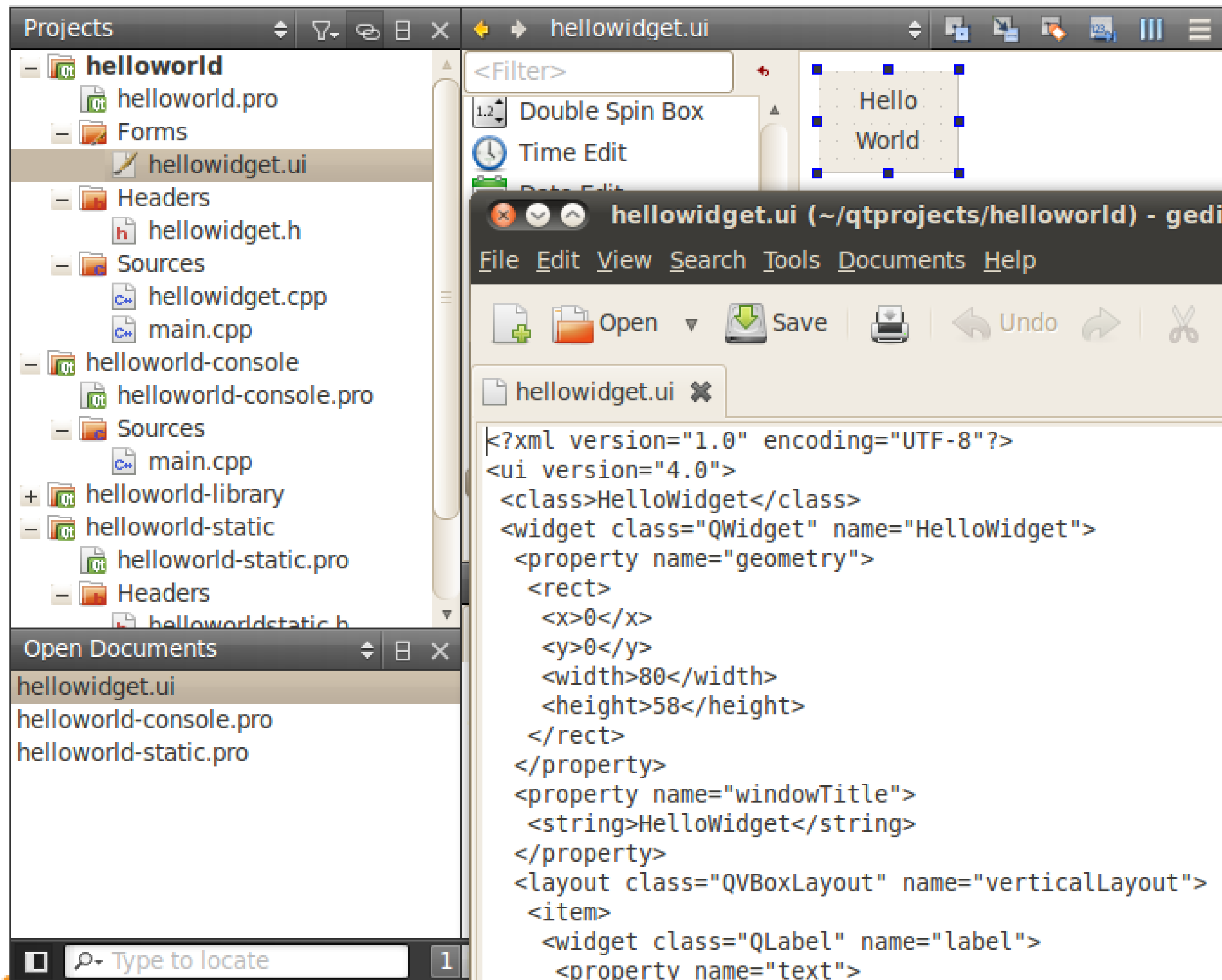


UI resources

<symbio>

- UI resource files are XML documents, which are processed by *uic* compiler during build
  - Generates C++ code from the resource and integrates it into project
- No need to edit manually, use QtCreator form editor instead

UI resources



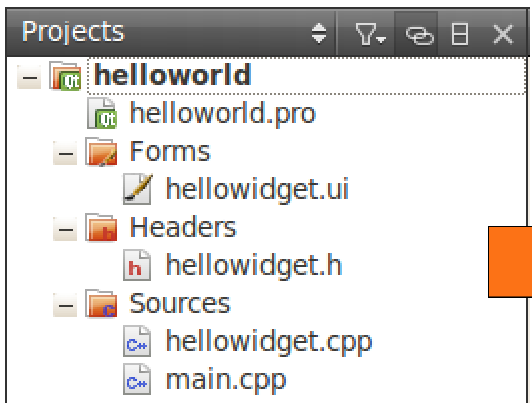
Build from command line

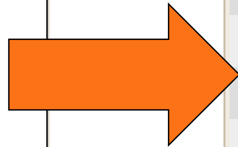
<symbio>

- Run *qmake* in the directory, which contains the *.pro* file
  - Generates the project Makefile
- Run *make* to build project
  - Runs *uic*, generates *ui_<form>.h* files
  - Runs *moc*, generates *moc_<class>.cpp* files
  - Compiles the sources to object *.o* files
  - Links the object files together and with Qt modules to produce the project target

Project output

<symbio>

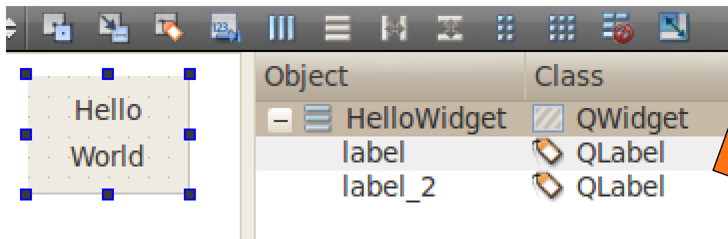


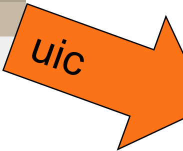


Name	Size	Type
hellowidget.cpp	437 bytes	C++ source code
hellowidget.h	330 bytes	C header
hellowidget.o	6.3 KB	object code
hellowidget.ui	967 bytes	Qt Designer file
helloworld	24.5 KB	executable
helloworld.pro	295 bytes	Qt QMake Profile
helloworld.pro.user	12.1 KB	plain text document
main.cpp	179 bytes	C++ source code
main.o	1.5 KB	object code
Makefile	7.1 KB	Makefile
moc_hellowidget.cpp	2.0 KB	C++ source code
moc_hellowidget.o	6.2 KB	object code
ui_hellowidget.h	2.2 KB	C header

Generated files - uic

- *uic* creates C++ code based on form resource
- Project source loads the UI






```
class Ui_HelloWidget
{
public:
    QVBoxLayout *verticalLayout;
    QLabel *label;
    QLabel *label_2;

    void setupUi(QWidget *HelloWidget)

        if (HelloWidget->objectName().isEmpty())
            HelloWidget->setObjectName(QString::fromUtf8("HelloWidget"));
        HelloWidget->resize(80, 58);
        verticalLayout = new QVBoxLayout(HelloWidget);
        verticalLayout->setSpacing(6);
        verticalLayout->setContentsMargins(11, 11, 11, 11);
        verticalLayout->setObjectName(QString::fromUtf8("verticalLayout"));
        label = new QLabel(HelloWidget);
        label->setObjectName(QString::fromUtf8("label"));
        label->setAlignment(Qt::AlignCenter);
```

```
#include "ui_hellowidget.h"

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

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

Generated files - moc

<symbio>

- *Moc* creates C++ code, which provides meta-information about classes
  - Somewhat similar to java *instanceof* operator and reflection API

```
class HelloWidget : public QWidget
{
    Q_OBJECT

public:
    HelloWidget(QWidget *parent = 0);
    ~HelloWidget();
```

```
void *HelloWidget::qt_metacast(const char *_clname)
{
    if (!_clname) return 0;
    if (!strcmp(_clname, qt_meta_stringdata_HelloWidget))
        return static_cast<void*>(const_cast< HelloWidget*>(this));
    return QWidget::qt_metacast(_clname);
}

int HelloWidget::qt_metacall(QMetaObject::Call _c, int _id, void **_a)
{
    _id = QWidget::qt_metacall(_c, _id, _a);
    if (_id < 0)
        return _id;
    return _id;
}
```

Qt modules

Overview of what’s in there

Qt modules

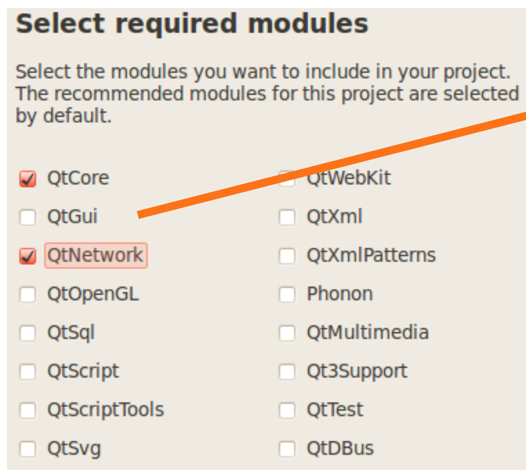
<symbio>

- Qt libraries are split into modules
- Specifying a module in *.pro* file results in it to be included into the project
  - Also loaded at runtime, so consumes resources
- Future plans?
  - Mobile devices need smaller modules to save resources

Qt modules

<symbio>

- Qt modules are configured into project files with *QT* keyword
  - *Core* and *gui* included by default
- QtCreator adds module definitions during project creation




```
QT          += network
QT          -= gui
```

Qt modules

<symbio>

- Module documentation has some general info about the module

Detailed Description

To include the definitions of both modules' classes, use the following directive:

```
#include <QtGui>
```

The QtGui module is part of the Qt GUI Framework Edition , the Qt Full Framework Edition, and the Open Source Versions of Qt .

[Previous: QtCore Module] [All Qt Modules] [Next: QtNetwork Module]

- However, in general don’t include the whole module as it increases compile time (unless using precompiled headers)

Qt modules walkthrough

<symbio>

- Qt documentation integrated to QtCreator
  - API reference -> Class and Function Documentation -> All Qt Modules

Core module

- Frameworks discussed during this course
  - Qt object model (QObject, QMetaObject)
  - Strings (QString, QByteArray)
  - Containers (QList, QMap, QHash, QLinkedList)
  - Data models (QAbstractItemModel & related)
  - Event loops (QCoreApplication, QEvent)
  - Animations (QAbstractAnimation & related)

Core module

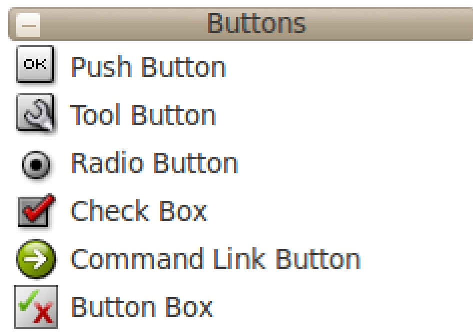
<symbio>

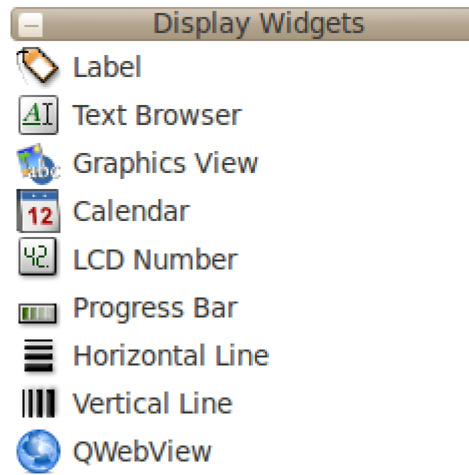
- Frameworks not discussed in this course
  - Multithreading (QFuture & related)
  - I/O devices (QIODevice, Qfile & related)
  - State machines (QStateMachine & related)

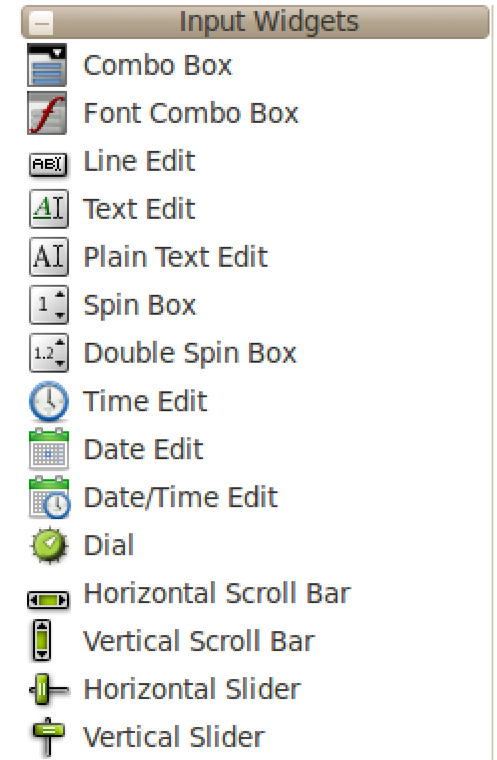
GUI module

<symbio>

- ”Traditional” widgets
  - Window is a widget without parent

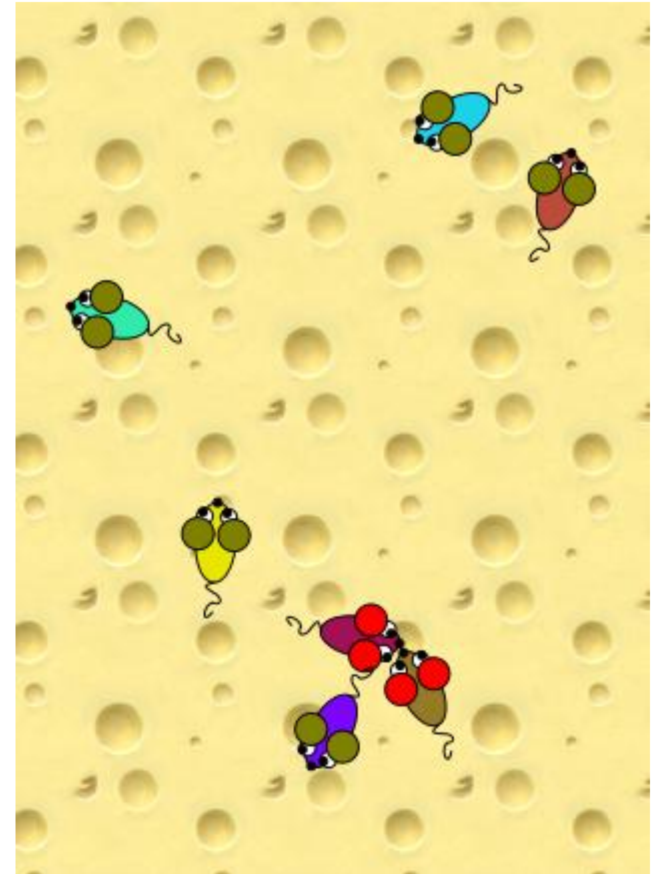






GUI module

- Graphics view
  - Graphics items
  - Graphics widgets
  - Proxy widgets
- Similar concepts, different painting semantics
  - More suitable for mobile devices



GUI module

<symbio>

- Some GUI frameworks that might be interesting, but not discussed in this course
  - Gesture recognition
  - Drag & drop

Network module

<symbio>

- Sockets, including secure ones
  - QTcpSocket, QSslSocket
- Simple HTTP and FTP API’s
  - QNetworkAccessManager

Multimedia modules

<symbio>

- OpenGL for 3D rendering
- OpenVG for 2D rendering
- Svg for processing vector graphics files
- Phonon multimedia framework
  - Not in mobile devices

Scripting module

<symbio>

- Allows Qt objects to used via QtScript code
  - Similar syntax as JavaScript, which is used with web browsers
  - However, environment is not browser (i.e. no DOM tree)

Other modules

- XML
  - SAX and DOM parsers
- XmlPatterns
  - XPath, XQuery, XSLT, schemas
- WebKit browser engine
- SQL for accessing databases

Mobile development

- Mobility API's are not part of standard QT
  - http://doc.qt.nokia.com/qtmobility-1.0/index.html
- Devices integration not yet in good shape

	API Maturity Level	Tier 1 Platforms				
		S60 3rd Edition, Feature Pack 1	S60 3rd Edition, Feature Pack 2	S60 5th Edition	Symbian^3	Maemo 5
Service Framework (in-process)	FINAL					
Messaging	FINAL					
Bearer Management	FINAL					
Publish and Subscribe	FINAL					
Contacts*	FINAL					
Location	FINAL					
Multimedia**	BETA					
System Information	FINAL					
Sensors*	FINAL					
Versit	FINAL					

Future stuff

<symbio>

- Declarative UI programming
  - Part of Qt 4.7 (Qt Quick)
  - QML language

```
Item {
    Rectangle {
        id: myRect
        width: 100
        height: 100
    }
    Rectangle {
        width: myRect.width
        height: 200
    }
}
```

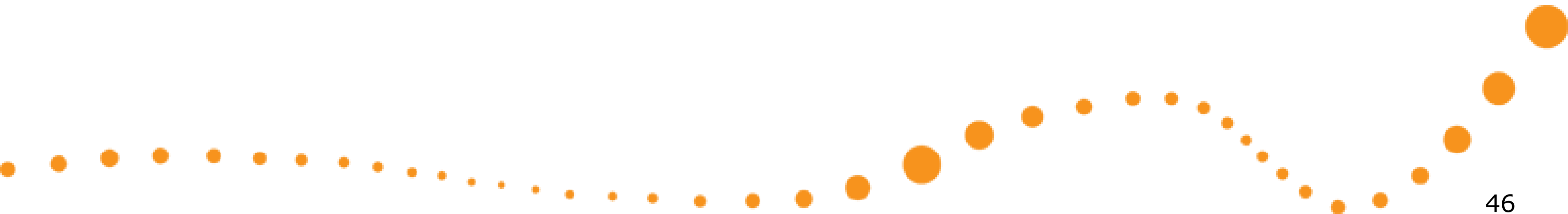
<symbio>

QtCreator overview

QtCreator overview

<symbio>

- This is an interactive part…
  - Build and run configurations
  - Session management
  - Edit, search, navigate, refactor
  - Running & debugging



Qt projects

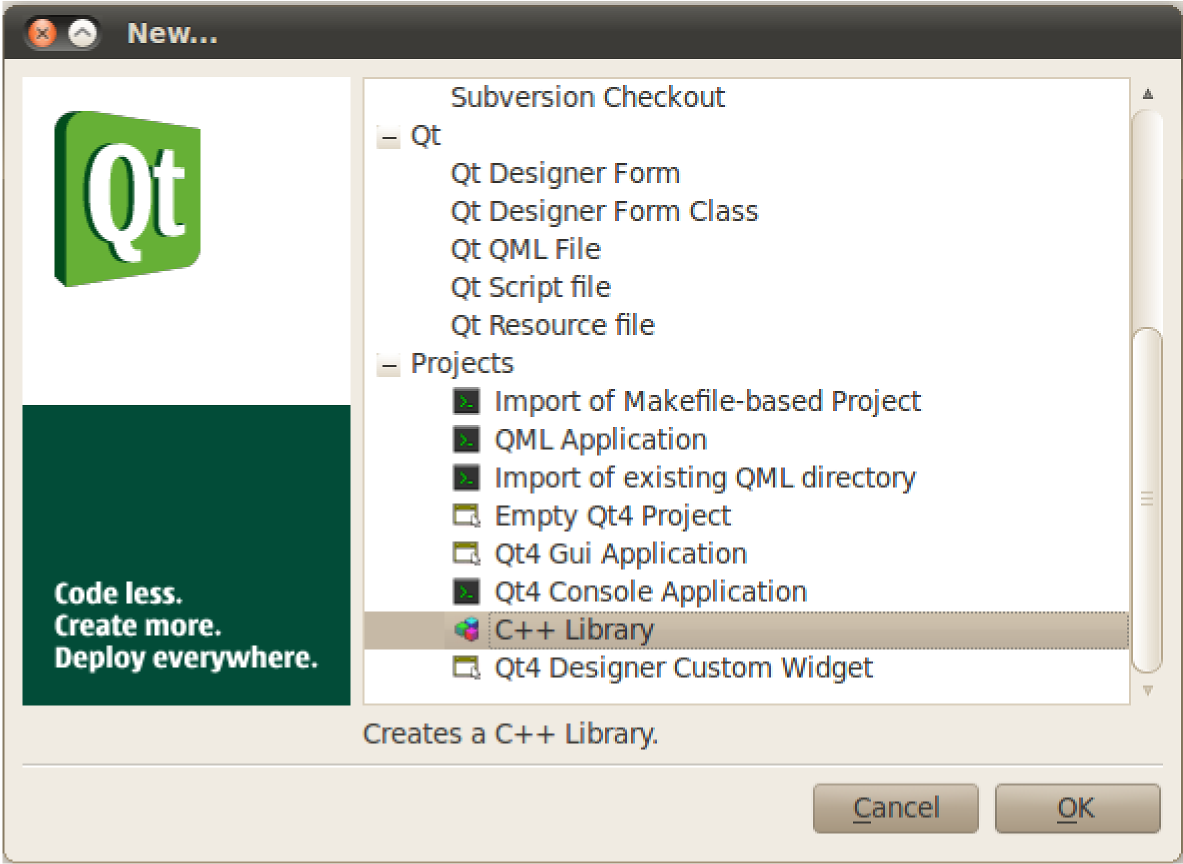
Beyond the basics

Shared libraries

<symbio>

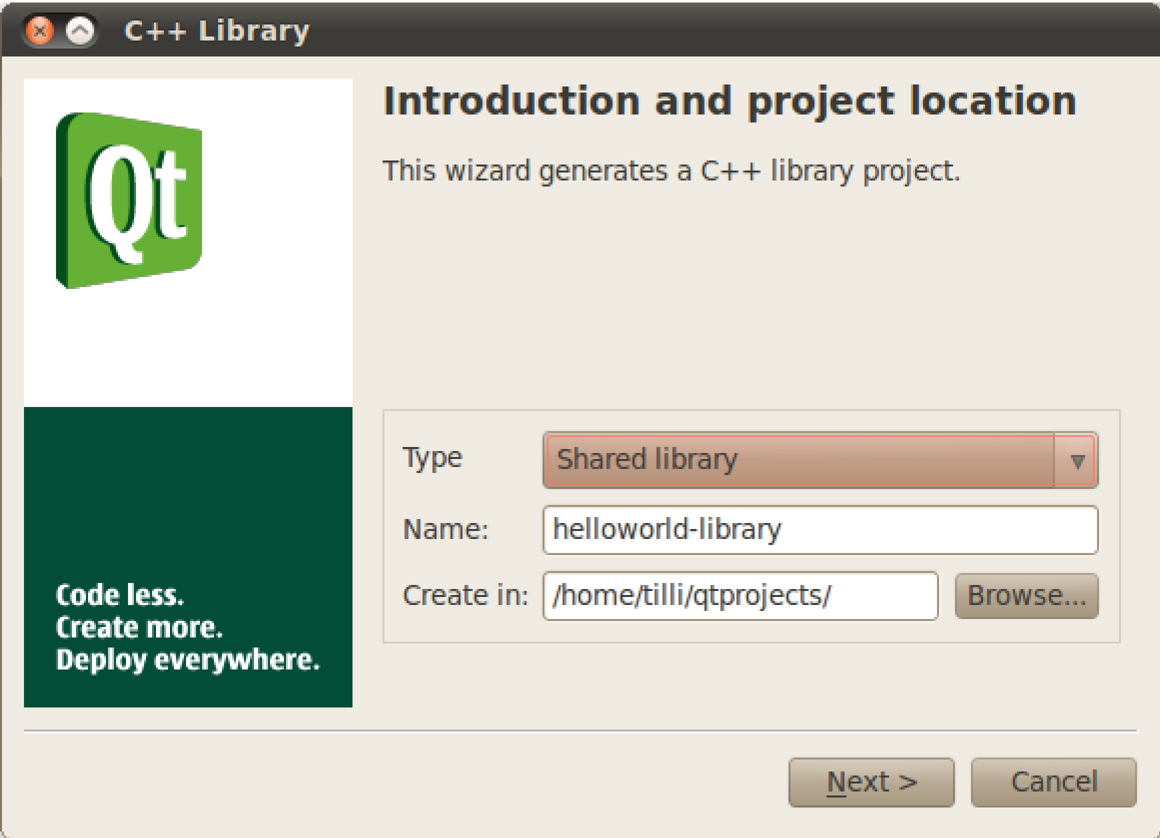
- A shared library contains code that is loaded once and shared by all executables that use it
- Saves resources, which is especially important in mobile devices

Shared libraries

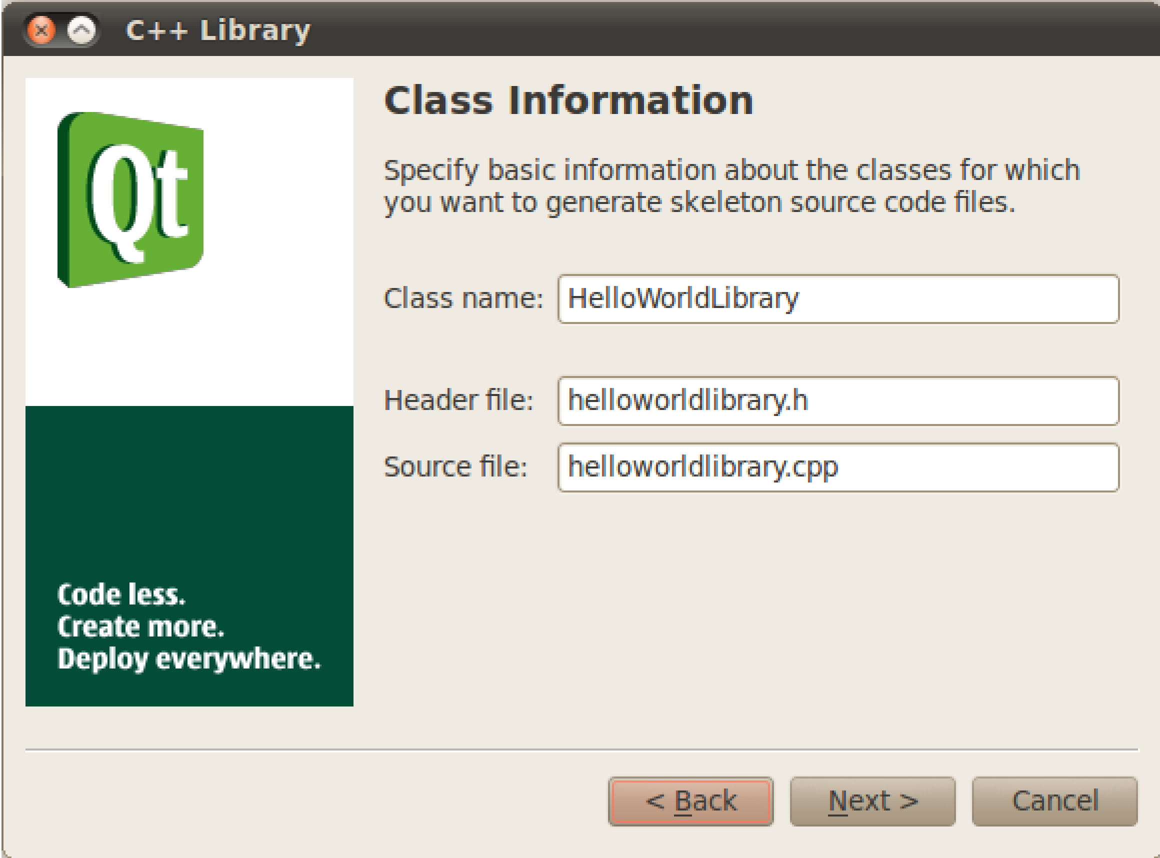


Shared libraries

<symbio>



Shared libraries



Shared libraries

- <name>*_global.h* header file is generated by QtCreator
  - Contains export declarations that are needed in certain platforms (like Windows)
  - Any class that is used from outside the library needs to have the export tag
- The export tag flag is defined in *.pro* file
  - When library is built, it *exports* the classes
  - When someone uses the library, it *imports* them

Shared libraries

<symbio>

```
QT -= gui
TARGET = helloworld-library
TEMPLATE = lib

DEFINES += HELLOWORLDLIBRARY_LIBRARY
```

```
class HELLOWORLDLIBRARYSHARED_EXPORT HelloWorldLibrary {
public:
    HelloWorldLibrary();
};
```

Projects

- helloworld
- helloworld-console
- helloworld-library
  - helloworld-library.pro
  - Headers
    - helloworld-library_global.h
    - helloworldlibrary.h
  - Sources
    - helloworldlibrary.cpp
- helloworld-static

helloworld-library_global.h — <No Symbols>

```
#ifndef HELLOWORLDLIBRARY_GLOBAL_H
#define HELLOWORLDLIBRARY_GLOBAL_H

#include <QtCore/qglobal.h>

#if defined(HELLOWORLDLIBRARY_LIBRARY)
#  define HELLOWORLDLIBRARYSHARED_EXPORT Q_DECL_EXPORT
#else
#  define HELLOWORLDLIBRARYSHARED_EXPORT Q_DECL_IMPORT
#endif

#endif // HELLOWORLDLIBRARY_GLOBAL_H
```

Shared libraries

- Exported classes define the *public API* of the library and thus the headers are needed by other libraries / executables
- In addition to headers, the library itself needs to be exported
    - Other libraries / executables need to be linked against it
    - The library needs to be present when an executable that uses it is run

Public headers

<symbio>

- Project file variables
  - Project files support user-defined variables
    - For example FOO = 5
  - Variables can be referenced with $$<name>
    - For example $$FOO would be replaced with 5
- Public headers can be separated from private headers with help of a variable

```
PUBLIC_HEADERS += helloworldlibrary.h \
    helloworld-library_global.h

HEADERS += $$PUBLIC_HEADERS \
    hwlibprivate.h
```

Exporting from project

<symbio>

- Project contents are exported with help of makefiles
- Run *make install* in project directory
  - Files and paths need to be specified first

Exporting from project

- INSTALLS directive is used to specify what and where to install
  - var.path specifies where to install
    - Path is relative to project directory
  - var.files specify what to install
    - *target.files* is pre-defined to contain project binaries

```
TARGET = helloworld-library

PUBLIC_HEADERS += helloworldlibrary.h \
    helloworld-library_global.h

public_headers.path = ../inc
public_headers.files = $$PUBLIC_HEADERS
target.path = ../bin
INSTALLS += target \
    public_headers
```

Name	Size	Type
bin	4 items	folder
libhelloworld-library.so	9.9 KB	Link to shared library
libhelloworld-library.so.1	9.9 KB	Link to shared library
libhelloworld-library.so.1.0	9.9 KB	Link to shared library
libhelloworld-library.so.1.0.0	9.9 KB	shared library
inc	2 items	folder
helloworldlibrary.h	244 bytes	C header
helloworld-library_global.h	300 bytes	C header

Using exported libraries

<symbio>

- To use the library, a project needs to find the exported data
  - INCLUDEPATH for headers
  - LIBS for libraries
    - -L<path>
    - -l<library-name>

```
INCLUDEPATH += ../inc
LIBS += -L../bin -lhelloworld-library
```

Managing larger projects

<symbio>

- Larger projects usually consists of multiple shared libraries and executables that share a common configuration
- Building each library separately would be tedious
- Solutions
  - Project include files (*.pri*)
  - Projects with subdirs TEMPLATE

Project include files

<symbio>

- Definitions common to multiple projects should be put into *.pri* file
  - For example, all projects binaries should be installed into *bin* and headers into *inc*

helloworld.pro

```
TARGET = helloworld
TEMPLATE = app
SOURCES += main.cpp\
        hellowidget.cpp
HEADERS  += hellowidget.h
FORMS    += hellowidget.ui
include(../helloworld.pri)
```

helloworld-library.pro

```
QT -= gui
TARGET = helloworld-library
TEMPLATE = lib
DEFINES += HELLOWORLDLIBRARY_LIBRARY
SOURCES += helloworldlibrary.cpp \
    hwlibprivate.cpp
PUBLIC_HEADERS += helloworldlibrary.h \
    helloworld-library_global.h
HEADERS += $$PUBLIC_HEADERS \
    hwlibprivate.h
include(../helloworld.pri)
```

helloworld.pri

```
public_headers.path = $$PWD/inc
public_headers.files = $$PUBLIC_HEADERS

target.path = $$PWD/bin
INSTALLS += target public_headers
```

Project include files

<symbio>

- Care must be taken with paths when using project include files
  - By default a path is relative to the project *.pro* file, not the included *.pri* file
  - Using $$PWD within include file makes path relative to it

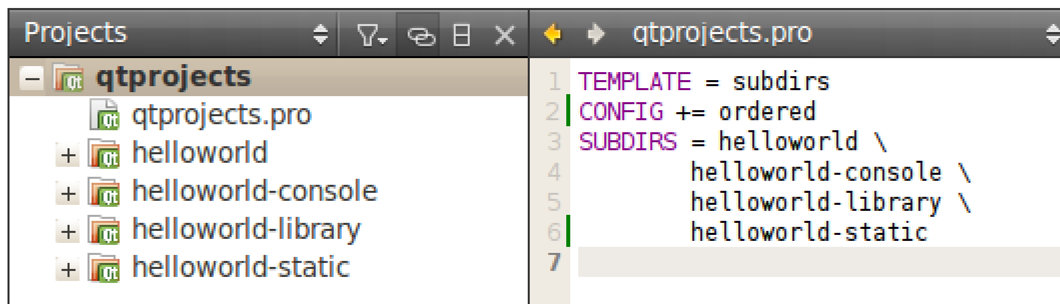
helloworld.pri

```
1 public_headers.path = $$PWD/inc
2 public_headers.files = $$PUBLIC_HEADERS
3
4 target.path = $$PWD/bin
5 INSTALLS += target public_headers
6
```

Root project file

<symbio>

- A project file with *subdirs* TEMPLATE causes all subdirectores to be built with single *qmake && make* command
  - Use *CONFIG += ordered* if build order matters, otherwise *qmake* may do parallel builds in environments with multiple CPU cores



Build issues?

<symbio>

- Note that when building via root project file, *make install* will not be run until all sub-projects have been built
  - Thus, INCLUDEPATH and LIBS cannot use the install target directories
  - INCLUDEPATH is not a problem, but library path might change depending on build configuration and platform
  - Shared library may use DESTDIR to explicitly specify where binaries are put

Build issues?

<symbio>

- Check QMake Manual from QtCreator integrated help
  - Lots of stuff that wasn't covered here

<symbio>

Programming exercise

Project creation

Exercise

<symbio>

- Create four projects:
  - musiclibrary (shared library)
  - musiclibrarymodel (shared library)
  - musiclibraryconsole (console application)
  - musiclibrarygui (QMainWindow gui application)
- Add root project file, which builds all
- Install binaries to *bin* under project root directory

Exercise

<symbio>

- Add some dependencies between libraries and executables
  - musiclibrarymodel need musiclibrary
  - musiclibraryconsole needs musiclibrary
  - musiclibrarygui needs both musiclibrary and musiclibrarymodel

```
INCLUDEPATH += ../inc
LIBS += -L../bin -lhelloworld-library
```

Exercise

Name	Size	Type
bin	10 items	folder
libmusiclibrary.so	57.7 KB	Link to shared library
libmusiclibrary.so.1	57.7 KB	Link to shared library
libmusiclibrary.so.1.0	57.7 KB	Link to shared library
libmusiclibrary.so.1.0.0	57.7 KB	shared library
libmusiclibrarymodel.so	45.7 KB	Link to shared library
libmusiclibrarymodel.so.1	45.7 KB	Link to shared library
libmusiclibrarymodel.so.1.0	45.7 KB	Link to shared library
libmusiclibrarymodel.so.1.0.0	45.7 KB	shared library
musiclibraryconsole	13.6 KB	executable
musiclibrarygui	42.2 KB	executable
musiclibrary	26 items	folder
musiclibraryconsole	4 items	folder
musiclibrarygui	7 items	folder
musiclibrarymodel	14 items	folder
Makefile	11.5 KB	Makefile
qttraining.pro	128 bytes	Qt QMake Profile

<symbio>

SERIOUS ABOUT SOFTWARE