Qt – Basics

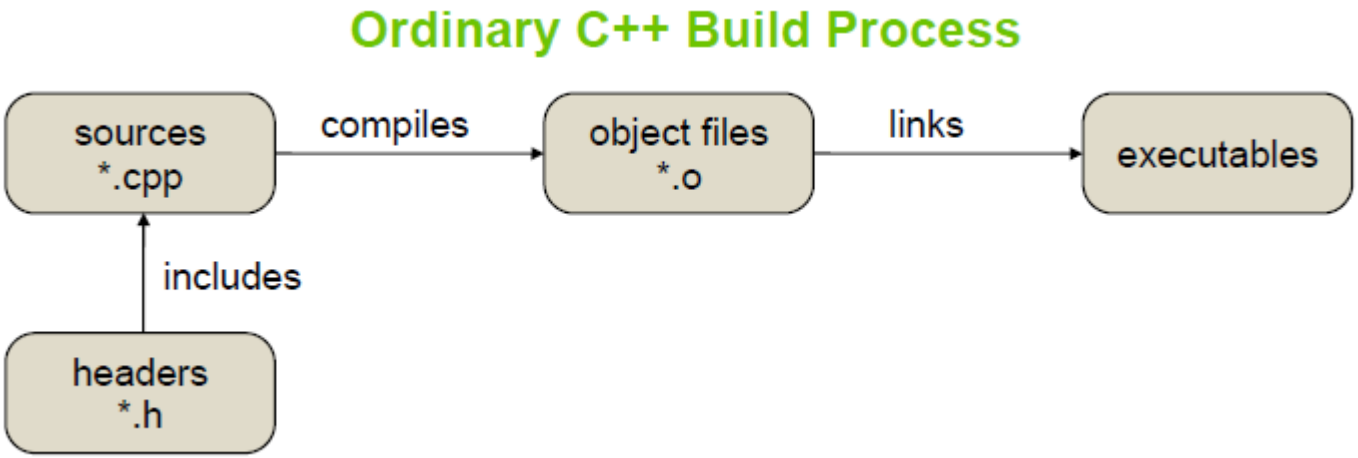
Yih-Chuan Lin

CSIE Windows Programming Class

National Formosa University

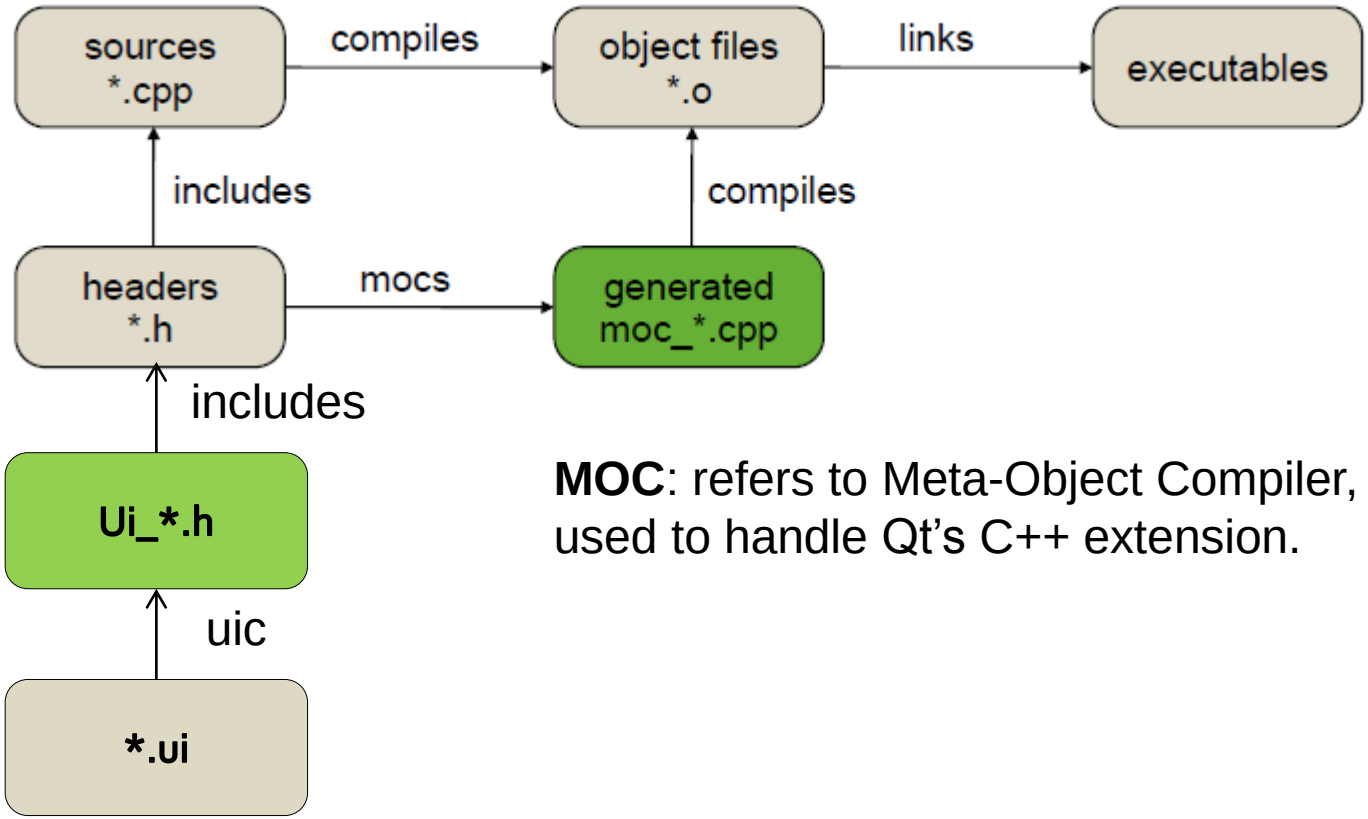
Qt- Program Compiling

Ordinary C/C++ compiling process



Qt- Program Compiling

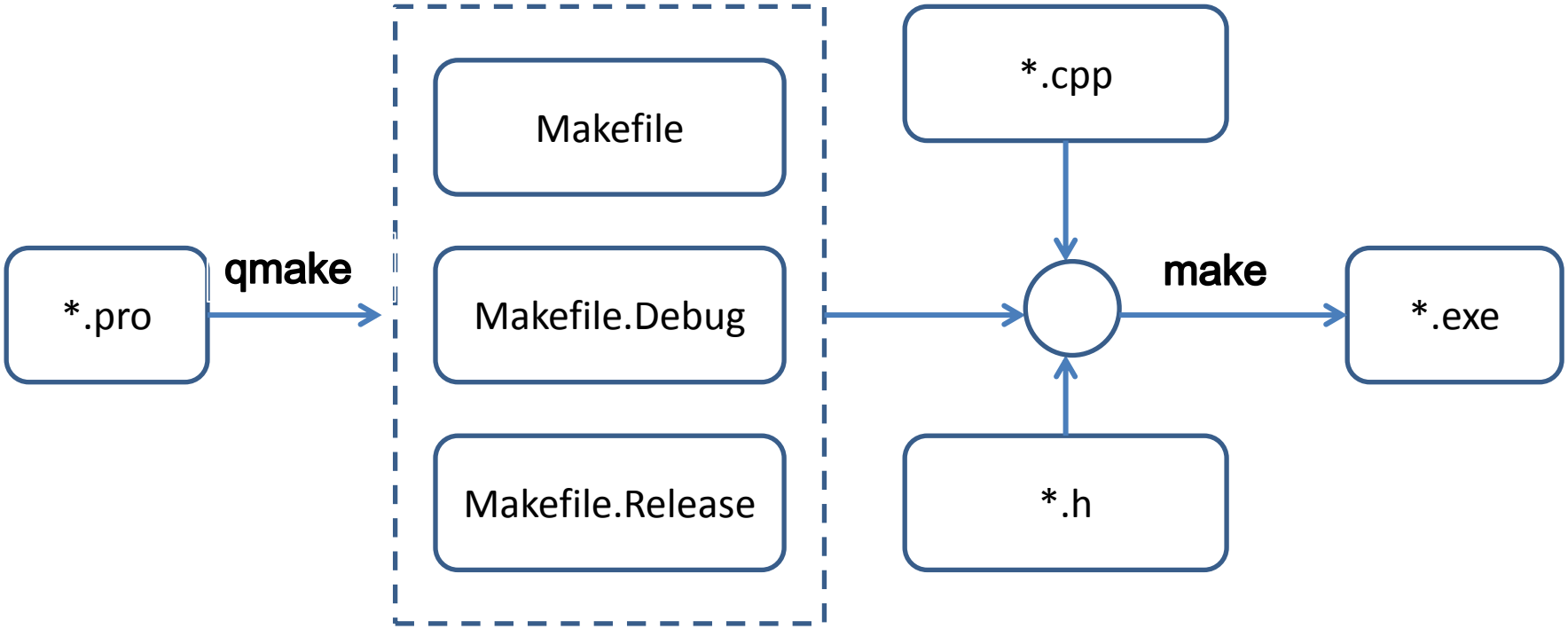
Qt program compiling process



MOC: refers to Meta-Object Compiler, which is used to handle Qt's C++ extension.

Qt- Program Compiling

Qt cross-platform compiling process



- Commands that are needed to build each project.
- Operating System Dependent
- Across different platforms

Qt- Program Compiling

Qt cross-platform compiling process

- The ***qmake*** tool gives you control over the source files used, and allows each of the steps in the C++ compiling process to be described concisely, typically within a single file.

- ***qmake*** expands the information in each project file to a Makefile that executes the necessary commands for compiling and linking.

Qt- Program Compiling

Qt project file (*.pro) contains:

- list of source and header files.
- general configuration information
- any application-specific details
- list of extra libraries to link
- list of extra include paths to use

Code less.
Create more.
Deploy everywhere.

Qt- Program Compiling

Qt project file:

```
1  #-------------------------------------------------
2  #
3  # Project created by QtCreator 2015-03-28T21:50:37
4  #
5  #-------------------------------------------------
6
7  QT       += core
8
9  QT       += gui widgets
10
11 TARGET = HelloWorld
12 #CONFIG   += console
13 #CONFIG   -= app_bundle
14
15 TEMPLATE = app
16
17
18 SOURCES += main.cpp
19
```

Qt- Program Compiling

Qt project file:

Variable	Contents
CONFIG (組態)	General project configuration options.
DESTDIR (目標路徑)	The directory in which the executable or binary file will be placed.
FORMS (介面模版)	A list of UI files to be processed by the user interface compiler (uic).
HEADERS (標頭檔)	A list of filenames of header (.h) files used when building the project.
QT (QT 模組)	A list of Qt modules used in the project.
RESOURCES (資源檔)	A list of resource (.qrc) files to be included in the final project. See the The Qt Resource Systemfor more information about these files.
SOURCES (原始碼)	A list of source code files to be used when building the project.
TEMPLATE (專案樣版)	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.

Qt- Program Compiling

Qt project file:

- If you want to use other libraries in your project in addition to those supplied with Qt, you need to specify them in your project file. For example,
- INCLUDEPATH = c:/msdev/include d:/stl/include
- LIBS += "C:/mylibs/extra libs/extra.lib”
- LIBS += -L/usr/local/lib -lmath

Qt- Program Compiling

Specifying QT variable:

Option	Module Enabled
axcontainer	QAxContainer, which is part of the Active Qt framework
axserver	QAxServer, which is part of the Active Qt framework
concurrent	Qt Concurrent
core (included by default)	Qt Core
dbus	Qt D-Bus
declarative	Qt Quick 1 (deprecated)
designer	Qt Designer
gui (included by default)	Qt GUI
help	Qt Help
multimedia	Qt Multimedia
multimediawidgets	Qt Multimedia Widgets
network	Qt Network

Qt- Program Compiling

Specifying QT variable:

opengl	Qt OpenGL (deprecated)
printsupport	Qt Print Support
qml	Qt QML
qmltest	Qt QML Test
x11extras	Qt X11 Extras
quick	Qt Quick
script	Qt Script (deprecated)
scripttools	Qt Script Tools (deprecated)
sensors	Qt Sensors
serialport	Qt Serial Port
sql	Qt SQL
svg	Qt SVG

Code less.
Create more.
Deploy everywhere.

Qt- Program Compiling

Specifying QT variable:

testlib	Qt Test
uitools	Qt UI Tools
webkit	Qt WebKit
webkitwidgets	Qt WebKit Widgets
widgets	Qt Widgets
winextras	Qt Windows Extras
xml	Qt XML (deprecated)
xmlpatterns	Qt XML Patterns

Qt- Program Compiling

Qt project file:

- By default, QT contains both core and gui
- If you want to build a project *without* the Qt GUI module, you need to exclude the gui value
- QT -= gui # Only the core module is used.

Qt- Program Compiling

Specifying the TEMPLATE variable:

Option	Description
app	Creates a Makefile for building applications (the default). See Building an Application for more information.
lib	Creates a Makefile for building libraries. See Building a Library for more information.
subdirs	Creates a Makefile for building targets in subdirectories. The subdirectories are specified using the SUBDIRS variable.
aux	Creates a Makefile for not building anything. Use this if no compiler needs to be invoked to create the target, for instance because your project is written in an interpreted language. **Note:** This template type is only available for Makefile-based generators. In particular, it will not work with the vcxproj and Xcode generators.
vcapp	Windows only. Creates an application project for Visual Studio. See Creating Visual Studio Project Files for more information.
vclib	Windows only. Creates a library project for Visual Studio.

Qt- Program Compiling

Specifying the CONFIG variable:

Option	Description
release	The project is to be built in release mode. If debug is also specified, the last one takes effect.
debug	The project is to be built in debug mode.
debug_and_release	The project is prepared to be built in both debug and release modes.
debug_and_release_target	This option is set by default. If debug_and_release is also set, the debug and release builds end up in separate debug and release directories.
build_all	If debug_and_release is specified, the project is built in both debug and release modes by default.
autogen_precompile_source	Automatically generates a .cpp file that includes the precompiled header file specified in the .pro file.
ordered	When using the subdirs template, this option specifies that the directories listed should be processed in the order in which they are given.

Qt- Program Compiling

Specifying the CONFIG variable:

Option	Description
qt	The target is a Qt application or library and requires the Qt library and header files. The proper include and library paths for the Qt library will automatically be added to the project. This is defined by default.
x11	The target is a X11 application or library. The proper include paths and libraries will automatically be added to the project.
plugin	The target is a plugin (lib only). This enables dll as well.
precompile_header	Enables support for the use of precompiled headers in projects.
shared	The target is a shared object/DLL. The proper include paths, compiler flags and libraries will automatically be added to the project. Note that dll can also be used on all platforms; a shared library file with the appropriate suffix for the target platform (.dll or .so) will be created.
windows	The target is a Win32 window application (app only). The proper include paths, compiler flags and libraries will automatically be added to the project.
console	The target is a Win32 console application (app only). The proper include paths, compiler flags and libraries will automatically be added to the project.

Qt- Program Compiling

Project file example:

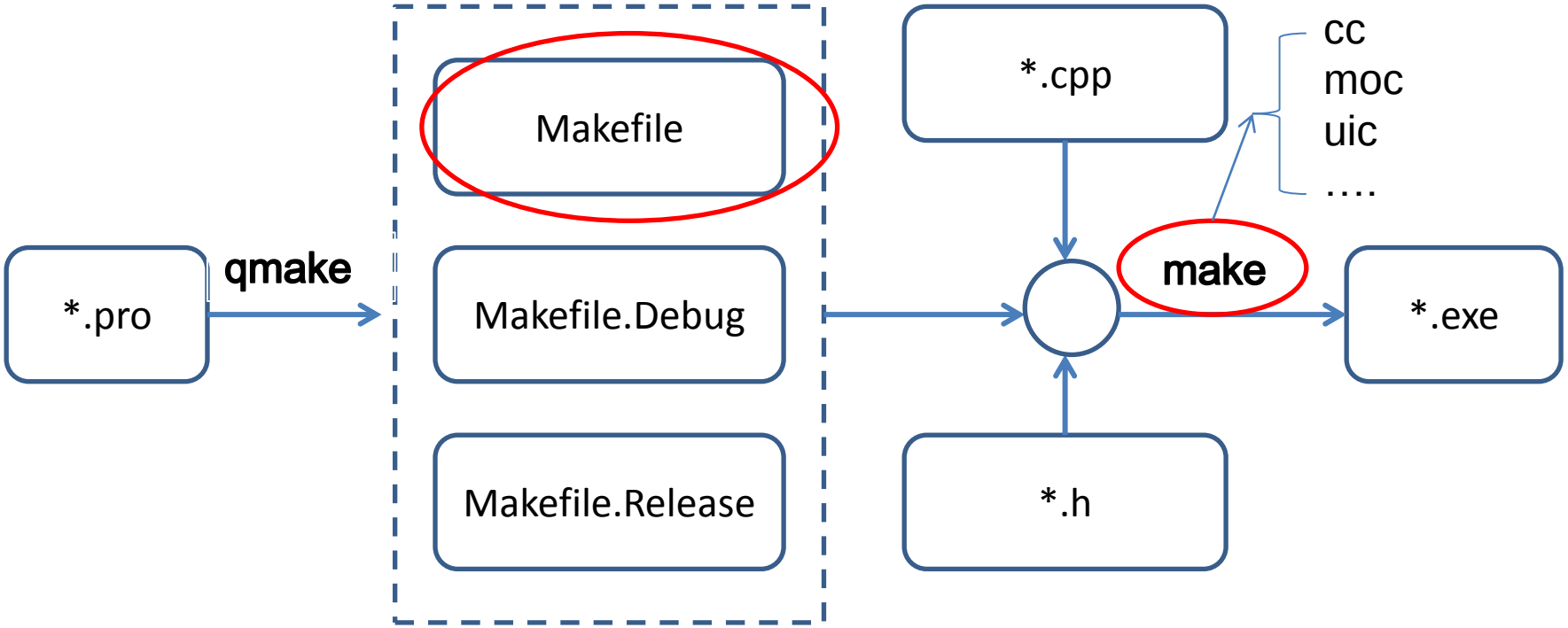
```
TEMPLATE      = app
LANGUAGE      = C++
CONFIG        += console precompile_header
CONFIG        -= app_bundle

# Use Precompiled headers (PCH)
PRECOMPILED_HEADER  = stable.h

HEADERS       =  stable.h \
                 mydialog.h \
                 myobject.h
SOURCES       = main.cpp \
                mydialog.cpp \
                myobject.cpp \
                util.cpp
FORMS         = mydialog.ui
```

Qt- Program Compiling

Qt cross-platform compiling process

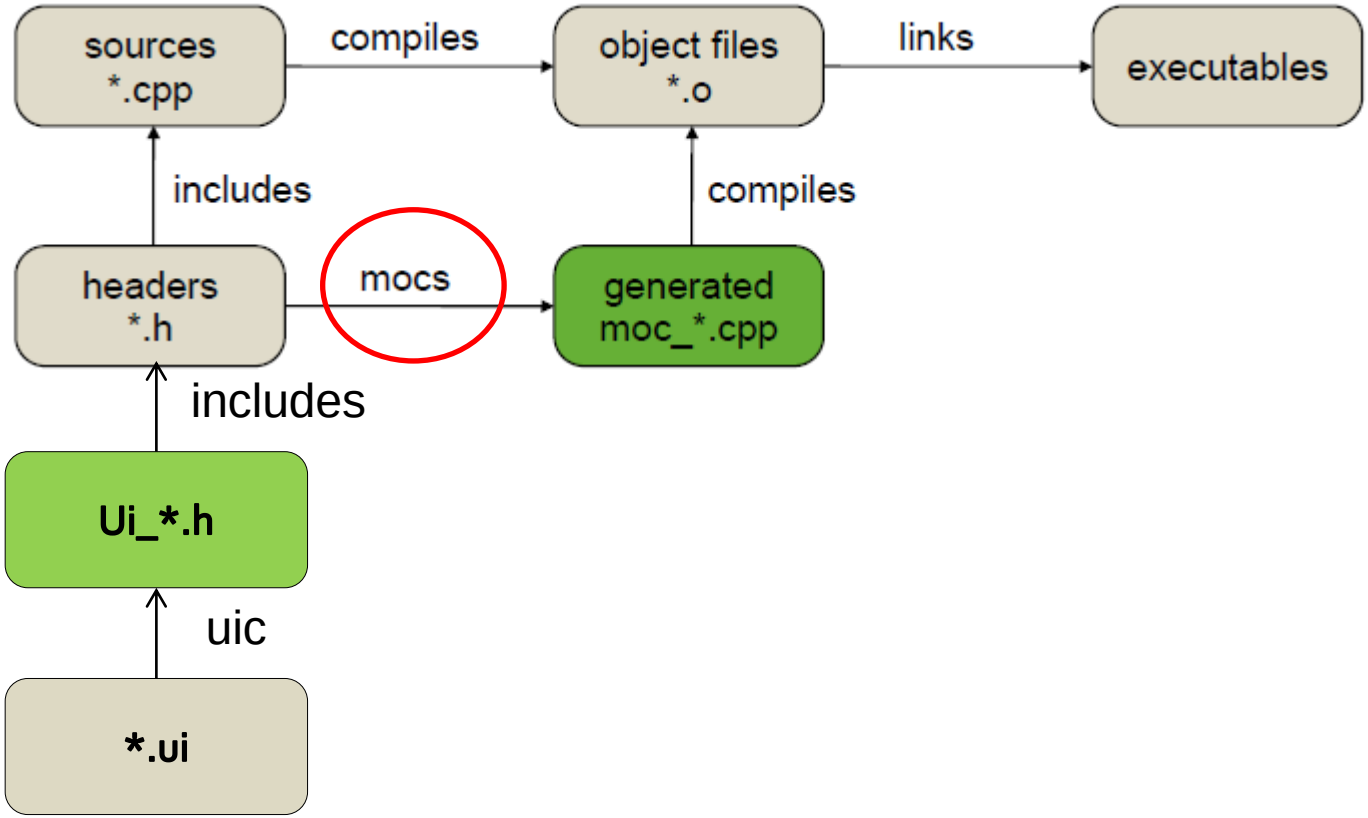


- Commands that are needed to build each project.
- Operating System Dependent
- Across different platforms

Code less.
Create more.
Deploy everywhere.

Qt- Program Compiling

Qt Meta-Object Compiler:



Code less.
Create more.
Deploy everywhere.

Qt- Program Compiling

Qt macros、keywords that moc is interested in:

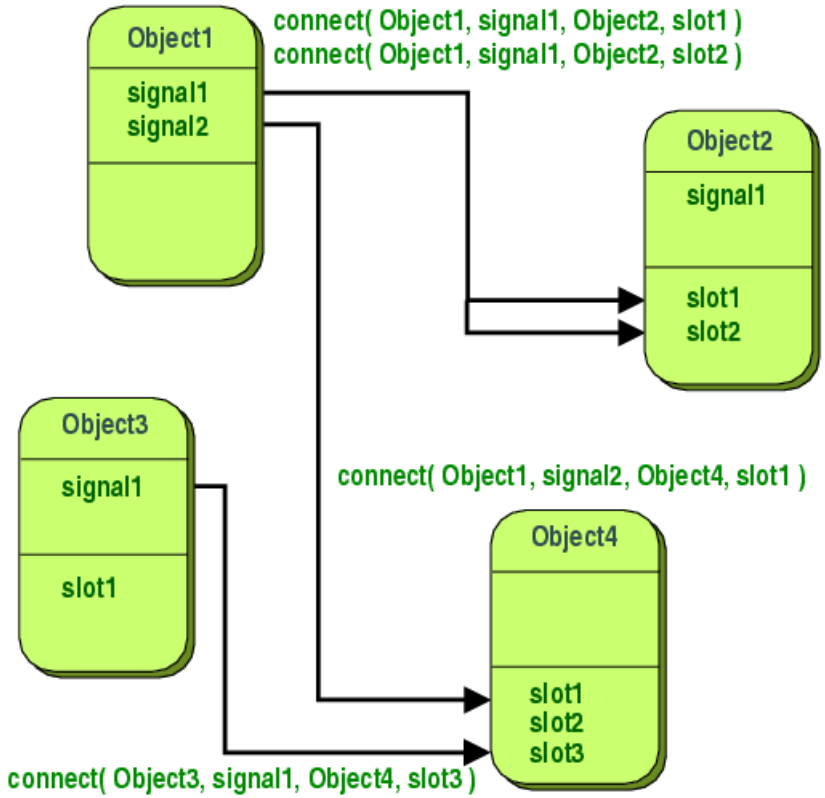
```
1  #ifndef BYTERCONVERTER_H
2  #define BYTERCONVERTER_H
3
4  #include <QDialog>
5  class QLineEdit;
6  class ByterConverter : public QDialog
7  {
8      Q_OBJECT
9
10 public:
11     ByterConverter(QString title,QWidget *parent = 0);
12     ~ByterConverter();
13
14 private:
15     QLineEdit* decEdit;
16     QLineEdit* hexEdit;
17     QLineEdit* binEdit;
18
19 private slots:
20     void decChanged(const QString&);
21     void hexChanged(const QString&);
22     void binChanged(const QString&);
23     void accept();
24 signals:
25     dummySignal();
26 };
```

Code less.
Create more.
Deploy everywhere.

Qt- Signal/Slot

Qt communication between objects:

- Signals and slots are made possible by Qt's meta-object system



Code less.
Create more.
Deploy everywhere.

Qt- Signal/Slot

Qt communication between objects:

- Signals and slots are made possible by Qt's meta-object system

```
class Counter
{
  public:
        Counter() { m_value = 0; }
        int value() const { return m_value; }
        void setValue(int value);
private:
         int m_value;
};
```

C++ class declaration

```
#include <QObject>
class Counter : public QObject
{
  Q_OBJECT
  public:
        Counter() { m_value = 0; }
         int value() const { return m_value; }
  public slots:
         void setValue(int value);
  signals:
          void valueChanged(int newValue);
  private:
          int m_value;
};
```

QObject-based class declaration

Code less.
Create more.
Deploy everywhere.

Qt- Signal/Slots

Qt communication between objects:

- Possible slot function implementation example:

```
void Counter::setValue(int value)
 {
    if (value != m_value)
      {
        m_value = value;
        emit valueChanged(value);
      }
}
```

Code less.
Create more.
Deploy everywhere.

Qt- Signal/Slots

Qt communication between objects:

- Possible slot function implementation example:

```
Counter a, b;
QObject::connect(&a, &Counter::valueChanged,
                 &b, &Counter::setValue);

a.setValue(12); // a.value() == 12, b.value() == 12
b.setValue(48); // a.value() == 12, b.value() == 48
```

Code less.
Create more.
Deploy everywhere.

Qt- Application Object

QApplication/QCoreApplication class:

- QApplication class manages application's GUI control flow and settings.
- QCoreApplication class provides an event loop for Qt applications without GUI

Header:	#include <QApplication>
qmake:	QT += widgets
Inherits:	QGuiApplication.

Header:	#include <QCoreApplication>
qmake:	QT += core
Inherits:	QObject.

Code less.
Create more.
Deploy everywhere.

Qt- Layout Manager

- **QGridLayout class:** construct gridlayout objects

Header:	#include <QGridLayout>
qmake:	QT += widgets
Inherits:	QLayout.

void QGridLayout::addWidget**(QWidget * *widget*, int *row*, int *column*, Qt::Alignment *alignment* = 0)**

Adds the given *widget* to the cell grid at *row*, *column*. The top-left position is (0, 0) by default.

•Qt::Alignment type is simply a typedef for QFlags<Qt::AlignmentFlag>
•QLabel::setAlignment() takes a Qt::Alignment parameter, which means that any combination of Qt::AlignmentFlag values, or 0, is legal:

Qt- Layout Manager

- **QVBoxLayout class:** construct vertical box layout objects

Header:	#include <QVBoxLayout>
qmake:	QT += widgets
Inherits:	QBoxLayout.

void QBoxLayout::addWidget**(QWidget * *widget*, int *stretch* =0,**
Qt::Alignment *alignment* = 0)

Adds *widget* to the end of this box layout, with a stretch factor of *stretch* and alignment *alignment*

label->setAlignment(Qt::AlignLeft | Qt::AlignTop);

Code less.
Create more.
Deploy everywhere.

Qt- Layout Manager

- **QHBoxLayout class:** construct horizontal box layout objects

Header:	#include <QHBoxLayout>
qmake:	QT += widgets
Inherits:	QBoxLayout.

void QBoxLayout::addWidget**(QWidget * *widget*, int *stretch* =0,**
Qt::Alignment *alignment* = 0)

Adds *widget* to the end of this box layout, with a stretch factor of *stretch* and alignment *alignment*

Qt- QString class

- **QString class:** class provides a Unicode character string

Header:	#include <QString>
qmake:	QT += core

int **toInt**(bool * *ok* = 0, int *base* = 10) const
int **toShort**(bool * *ok* = 0, int *base* = 10) const
int **toUInt**(bool * *ok* = 0, int *base* = 10) const
int **toULong**(bool * *ok* = 0, int *base* = 10) const
float **toFloat**(bool * *ok* = 0) const

Qt- QString class

- **QString class:** class provides a Unicode character string

QString QString::arg**(const QString & *a*, int *fieldWidth* = 0, QChar *fillChar* = QLatin1Char(' ')) const**

- Returns a copy of this string with the lowest numbered place marker replaced by string *a.*
- *fieldWidth* specifies the minimum amount of space that argument *a* shall occupy.
- If *a* requires less space than *fieldWidth*, it is padded to *fieldWidth*with character *fillChar*.

```
QString i;              // current file's number
QString total;          // number of files to process
QString fileName;     // current file's name
QString status = QString("Processing file %1 of %2: %3") .arg(i).arg(total).arg(fileName);
```

First, arg(i) replaces %1. Then arg(total) replaces %2. Finally, arg(fileName) replaces %3.

Code less.
Create more.
Deploy everywhere.

Qt- QString class

- **QString class:** class provides a Unicode character string

QString QString::arg**(const QString &** ***a1*****, const QString &** ***a2*****) const**

•This is the same as str.arg(a1).arg(a2)

```
QString str;
str = "%1 %2"; str.arg("%1f", "Hello");   // returns "%1f Hello"
str.arg("%1f").arg("Hello");              // returns "Hellof %2"
```

Qt- QString class

- **QString class:** class provides a Unicode character string

QString QString::arg**(int *a*, int *fieldWidth* = 0, int *base* = 10, QChar *fillChar* = QLatin1Char(' ')) const**

•The *a* argument is expressed in base *base*, which is 10 by default and must be between 2 and 36.
•For bases other than 10, *a* is treated as an unsigned integer.

```
QString str;
str = QString("Decimal 63 is %1 in hexadecimal") .arg(63, 0, 16);
// str == "Decimal 63 is 3f in hexadecimal"
QLocale::setDefault(QLocale(QLocale::English, QLocale::UnitedStates));
str = QString("%1 %L2 %L3") .arg(12345) .arg(12345) .arg(12345, 0, 16);
// str == "12345 12,345 3039"
```

Qt- QString class

- **QString class:** class provides a Unicode character string

QString QString::arg**(double *a*, int *fieldWidth* = 0, char *format* = 'g', int *precision* = -1, QChar *fillChar*= QLatin1Char(' ')) const**

•Argument *a* is formatted according to the specified *format* and *precision*.

```
double d = 12.34;
QString str = QString("delta: %1").arg(d, 0, 'E', 3);

// str == "delta: 1.234E+01"
```

Qt- QFile

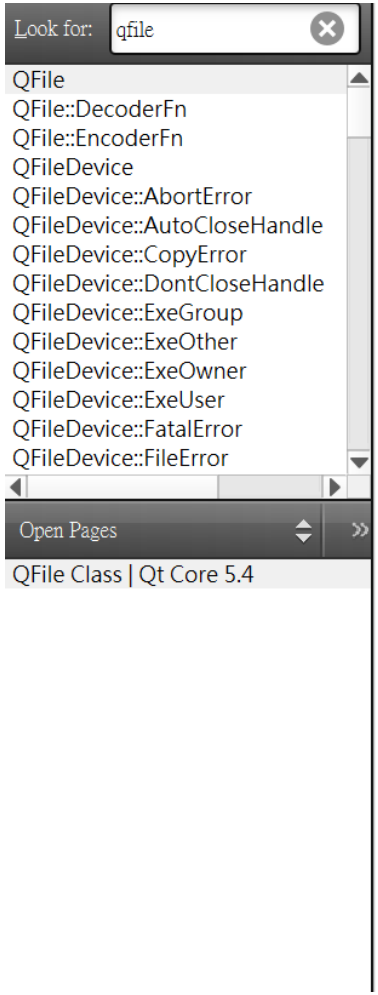
- The QFile class provides an interface for reading from and writing to files.

Header:	#include <QFile>
qmake:	QT += core
Inherits:	QFileDevice.
Inherited By:	QTemporaryFile.

QFile is an I/O device for reading and writing text and binary files. A QFile may be used with a QTextStream or QDataStream.

Code less.
Create more.
Deploy everywhere.

Qt- QFile



Detailed Description

The QFile class provides an interface for reading from and writing to files.

QFile is an I/O device for reading and writing text and binary files and resources. A QFile may be used by itself or, more conveniently, with a QTextStream or QDataStream.

The file name is usually passed in the constructor, but it can be set at any time using setFileName(). QFile expects the file separator to be '/' regardless of operating system. The use of other separators (e.g., '\') is not supported.

You can check for a file's existence using exists(), and remove a file using remove(). (More advanced file system related operations are provided by QFileInfo and QDir.)

The file is opened with open(), closed with close(), and flushed with flush(). Data is usually read and written using QDataStream or QTextStream, but you can also call the QIODevice-inherited functions read(), readLine(), readAll(), write(). QFile also inherits getChar(), putChar(), and ungetChar(), which work one character at a time.

The size of the file is returned by size(). You can get the current file position using pos(), or move to a new file position using seek(). If you've reached the end of the file, atEnd() returns `true`.

Qt- QFile

- The file is opened with open(), closed with close(), and flushed with flush().
- Data is usually read and written using QDataStream or QTextStream.
- You can also call the functions read(), readLine(), readAll(), write().

```
QFile file("in.txt");
 if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    return;
while (!file.atEnd())  {
   QByteArray line = file.readLine();
    process_line(line);
 }
```

```
QFile file("out.txt");
if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
    return;
QTextStream out(&file); out << "The magic number is: " << 49 << "\n";
```

Qt- QTextStream

- The QTextStream class provides a convenient interface for reading and writing text.

Header:	#include <QTextStream>
qmake:	QT += core

```
QFile data("output.txt");
if (data.open(QFile::WriteOnly | QFile::Truncate)) {
    QTextStream out(&data);
    out << "Result: " << qSetFieldWidth(10) << left << 3.14 << 2.7;
    // writes "Result: 3.14 2.7 "
}
```

Qt- QTextStream

- QTextStream also defines several global manipulator functions:

Manipulator	Description
bin	Same as setIntegerBase(2).
oct	Same as setIntegerBase(8).
dec	Same as setIntegerBase(10).
hex	Same as setIntegerBase(16).
showbase	Same as setNumberFlags(numberFlags() | ShowBase).
forcesign	Same as setNumberFlags(numberFlags() | ForceSign).
forcepoint	Same as setNumberFlags(numberFlags() | ForcePoint).
noshowbase	Same as setNumberFlags(numberFlags() & ~ShowBase).
noforcesign	Same as setNumberFlags(numberFlags() & ~ForceSign).
noforcepoint	Same as setNumberFlags(numberFlags() & ~ForcePoint).
uppercasebase	Same as setNumberFlags(numberFlags() | UppercaseBase).
uppercasedigits	Same as setNumberFlags(numberFlags() | UppercaseDigits).
lowercasebase	Same as setNumberFlags(numberFlags() & ~UppercaseBase).

Qt- QTextStream

- QTextStream also defines several global manipulator functions:

lowercasedigits	Same as setNumberFlags(numberFlags() & ~UppercaseDigits).
fixed	Same as setRealNumberNotation(FixedNotation).
scientific	Same as setRealNumberNotation(ScientificNotation).
left	Same as setFieldAlignment(AlignLeft).
right	Same as setFieldAlignment(AlignRight).
center	Same as setFieldAlignment(AlignCenter).
endl	Same as operator<<('\n') and flush().
flush	Same as flush().
reset	Same as reset().
ws	Same as skipWhiteSpace().
bom	Same as setGenerateByteOrderMark(true).

Qt- QDataStream

- The QDataStream class provides serialization of binary data to a QIODevice.

Header:	#include <QDataStream>
qmake:	QT += core

- A data stream is a binary stream of encoded information which is 100% independent of the host computer's operating system, CPU or byte order.
- A data stream cooperates closely with a QIODevice.
- A QIODevice represents an input/output medium one can read data from and write data to.

Qt- QDataStream

- The QDataStream class provides serialization of binary data to a QIODevice.

```
QFile file("file.dat");
file.open(QIODevice::WriteOnly);
QDataStream out(&file); // we will serialize the data into the file
out << QString("the answer is"); // serialize a string
out << (qint32)42; // serialize an integer
```

```
QFile file("file.dat");
file.open(QIODevice::ReadOnly);
QDataStream in(&file); // read the data serialized from the file
QString str;
qint32 a;
in >> str >> a; // extract "the answer is" and 42
```

Qt- QTableWidget

- The QTableWidget class provides an item-based table view with a default model.

Header:	#include <QTableWidget>
qmake:	QT += widgets
Inherits:	QTableView.

- Table widgets provide standard table display facilities for applications.
- The items in a QTableWidget are provided by QTableWidgetItem.

Qt- QTableWidget

- The QTableWidget class provides an item-based table view with a default model.

```
tableWidget = new QTableWidget(this);
tableWidget->setRowCount(10);
tableWidget->setColumnCount(5);
QTableWidgetItem *newItem = new
QTableWidgetItem(tr("%1").arg( (row+1)*(column+1)));
 tableWidget->setItem(row, column, newItem);
```

Qt- QTableWidget

- The QTableWidget class provides an item-based table view with a default model.

QTableWidgetItem * QTableWidget::takeItem**(int *row*, int *column*)**

QTableWidgetItem * QTableWidget::item**(int *row*, int *column*) const**

void QTableWidget::insertColumn**(int *column*)**

void QTableWidget::insertRow**(int *row*)**

QTableWidgetItem * QTableWidget::item**(int *row*, int *column*) const**

void QTableWidget::itemClicked**(QTableWidgetItem * *item*)** **←signal**

void QTableWidget::setHorizontalHeaderLabels**(const QStringList & *labels*)**

Qt- QTableWidgetItem

- The QTableWidgetItem class provides an item for use with the QTableWidget class.

Header:	#include <QTableWidgetItem>
qmake:	QT += widgets

void QTableWidgetItem::setBackground**(const QBrush & *brush*)**

int QTableWidgetItem::row**() const**

Returns the row of the item in the table. If the item is not in a table, this function will return -1.

int QTableWidgetItem::column**() const**

QString QTableWidgetItem::text**() const**

QTableWidget * QTableWidgetItem::tableWidget**() const**

Qt- QTableWidgetItem

- The QTableWidgetItem class provides an item for use with the QTableWidget class.

QList<QTableWidgetItem *>
QTableWidget::findItems**(const QString & *text*, Qt::MatchFlags *flags*) const**

```
QList<QTableWidgetItem *>  LTempTable =temp->findItems("001",Qt::MatchEndsWith);
cout<<"the matched count:"<<LTempTable.count()<<endl;
foreach(rowPtr, LTempTable)
 {
   rowPtr->setBackground(Qt::red);
  }
```

Qt- QTableWidgetItem

[slot] **void** QTableWidget::**removeRow**(**int** *row*)
→Removes the row *row* and all its items from the table.

[slot] **void** QTableWidget::**scrollToItem**(const **QTableWidgetItem** **item*, **QAbstractItemView::ScrollHint** *hint* = EnsureVisible)

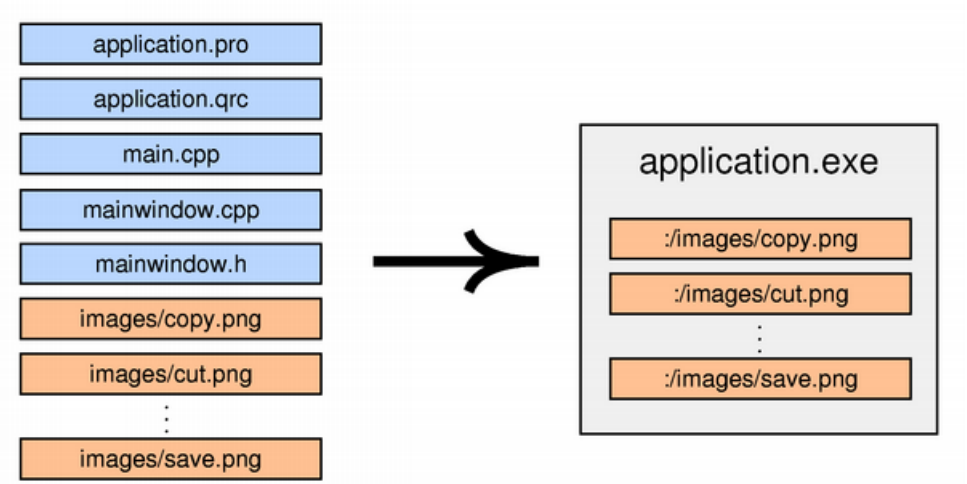
→Scrolls the view if necessary to ensure that the *item* is visible. The *hint* parameter specifies more precisely where the *item* should be located after the operation.

Qt- Resource

- Qt resource system is a platform-independent mechanism for storing binary files in the application's executable

The resources associated with an application are specified in a .qrc file

```
<RCC version="1.0">
<qresource>
<file>images/copy.png</file>
<file>images/cut.png</file>
<file>images/new.png</file>
<file>images/open.png</file>
<file>images/paste.png</file>
<file>images/save.png</file>
</qresource>
</RCC>
```



Qt- Resource

- For a resource to be compiled into the binary the .qrc file must be mentioned in the application's .pro file so that qmake knows about it.

```
RESOURCES = application.qrc
```

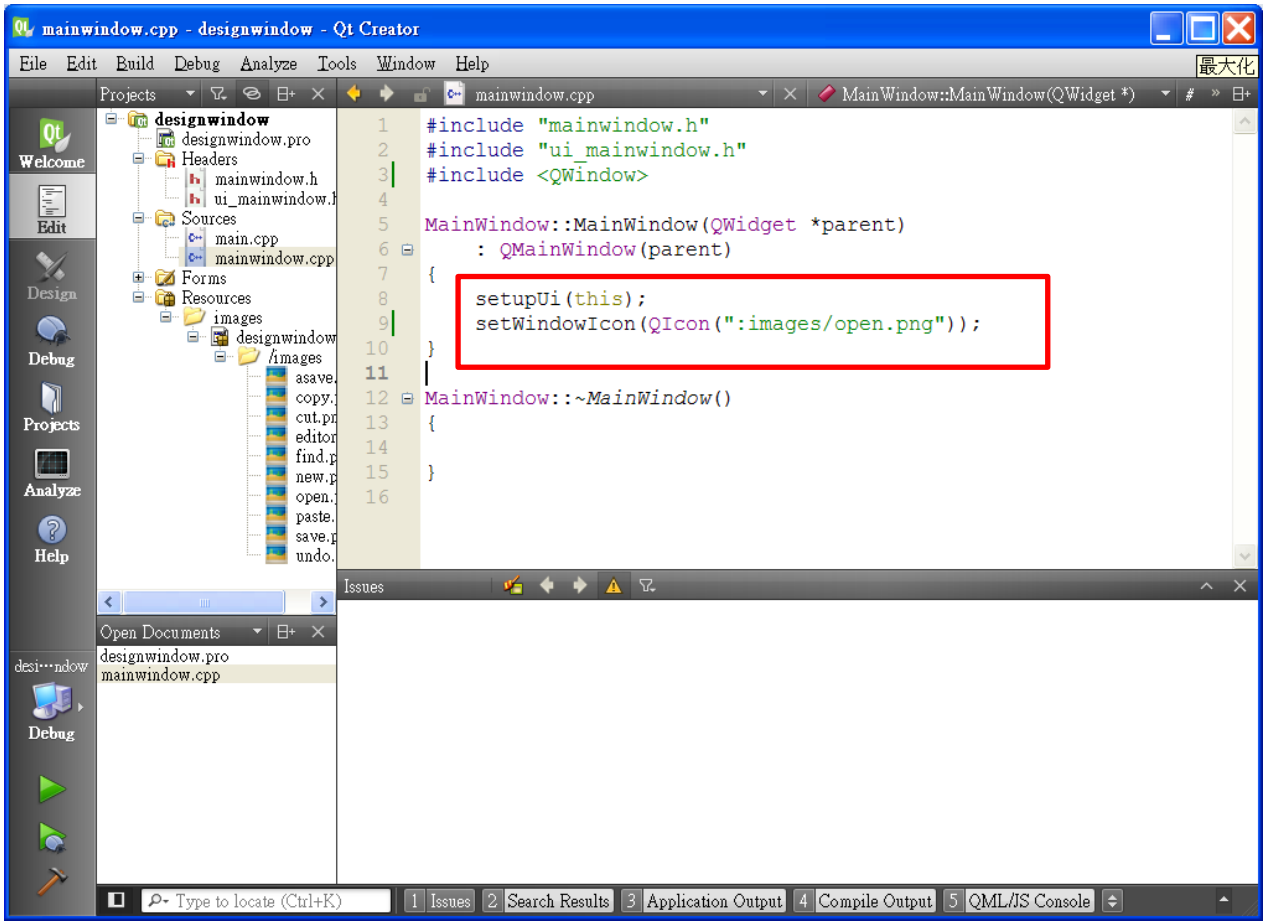
- Using Resources in the Application:
  In the application, resource paths can be used in most places instead of ordinary file system paths.

```
cutAct = new QAction(QIcon(":/images/cut.png"), tr("Cu&t"), this);
```

Code less.
Create more.
Deploy everywhere.

Qt- Resource

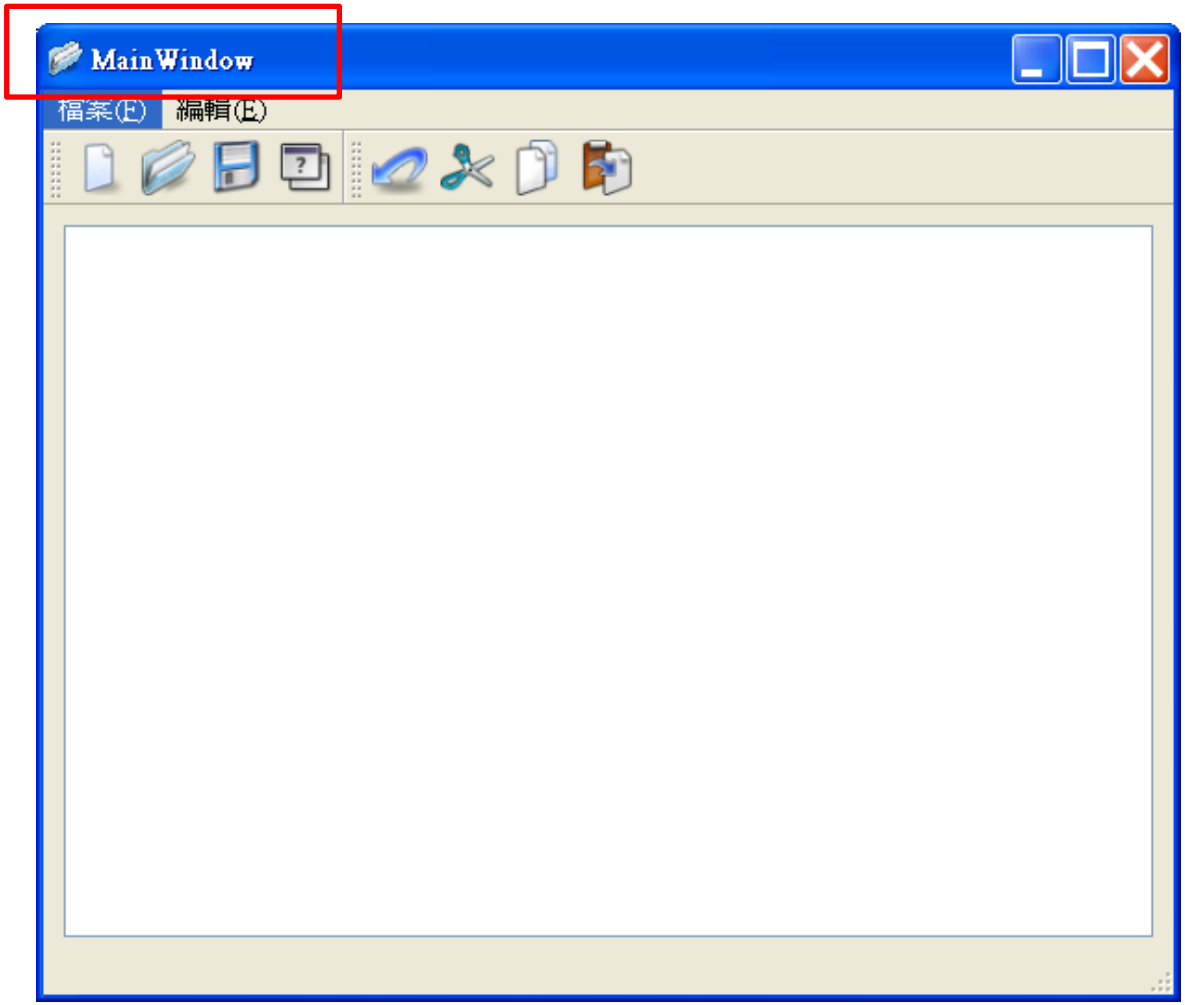
- Set up an icon next to the window title



Code less.
Create more.
Deploy everywhere.

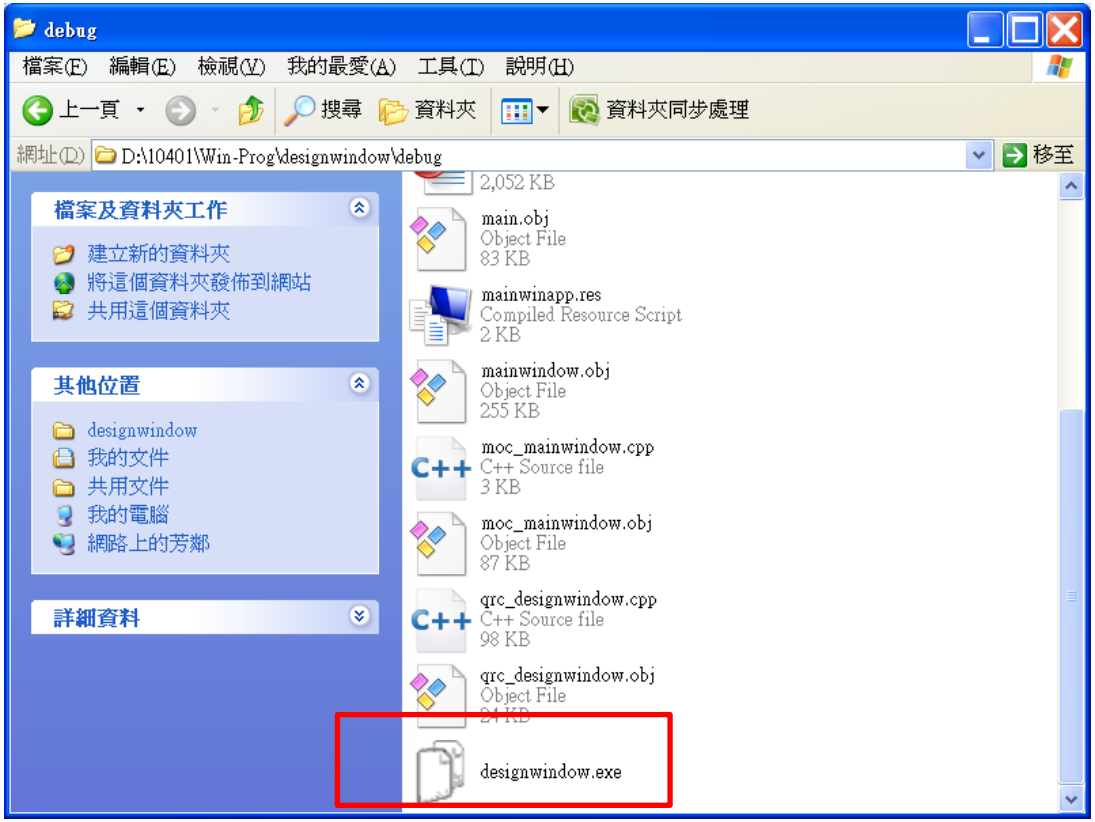
Qt- Resource

- Set up an icon next to the window title



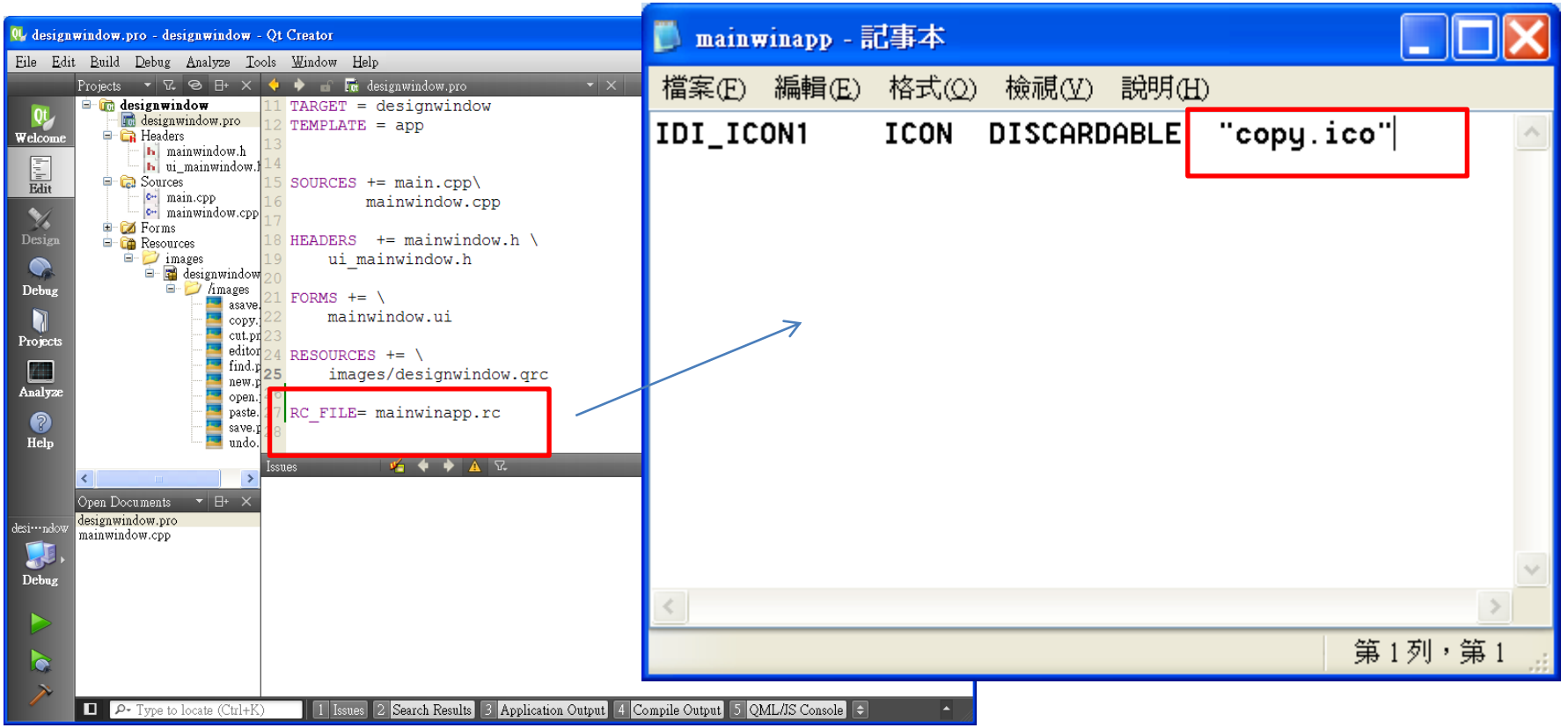
Qt- Resource

- Set up an icon for the application executable file



Qt- Resource

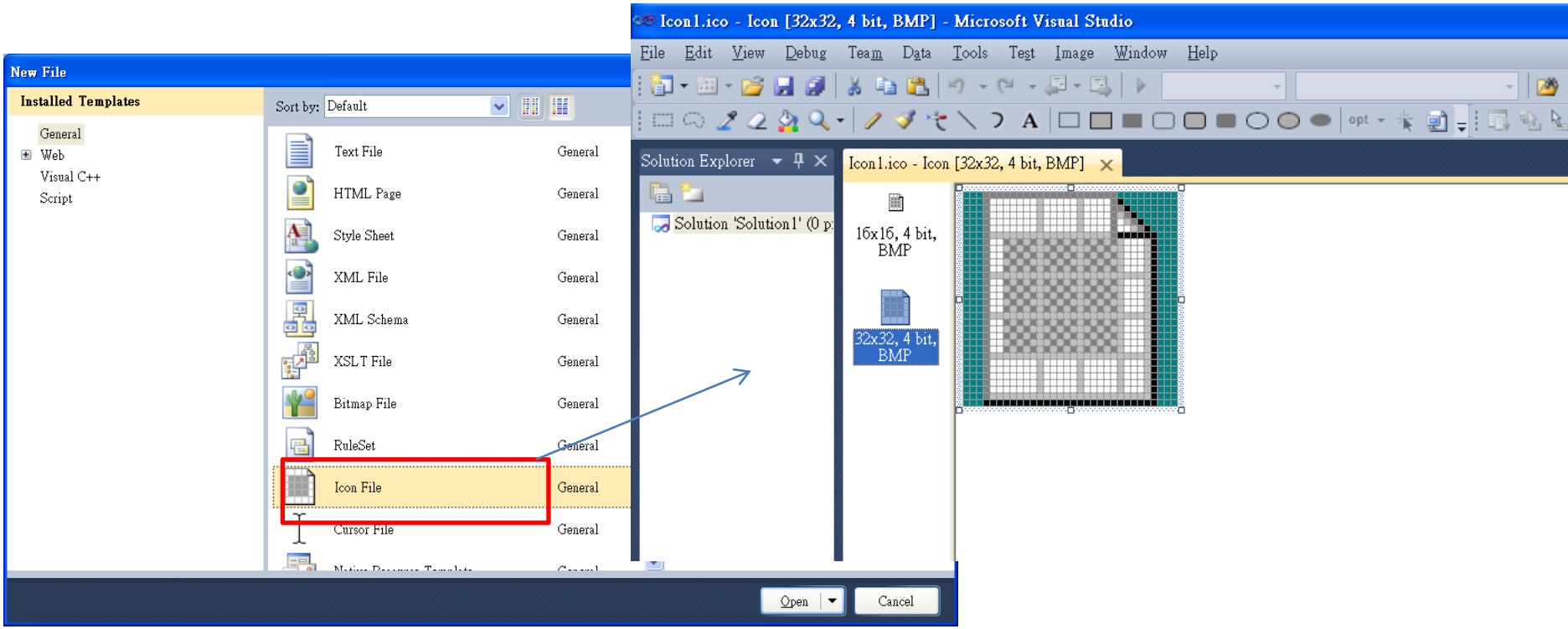
- Set up an icon for the application executable file



Code less.
Create more.
Deploy everywhere.

Qt- Resource

- Set up an icon for the application executable file



Code less.
Create more.
Deploy everywhere.

Qt- Resource

- Set up an icon for the application executable file