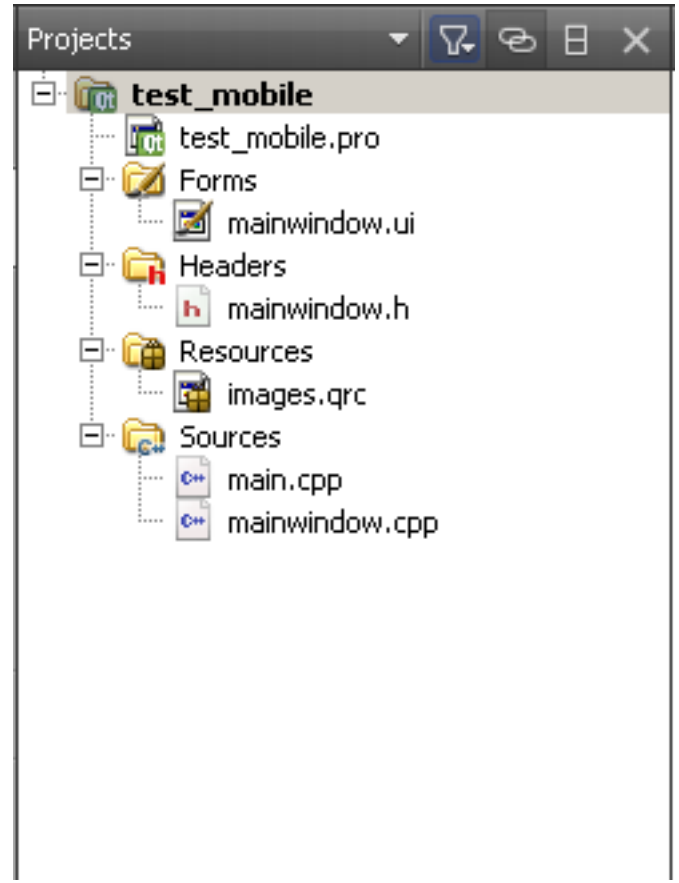
Qt Fundamentals

Qt tools

Basic Project Structure

- Project file (.pro)
- Project source code
  - Header files
  - Source files
- Resources (.qrc)
- UI design files (.ui)

These files are processed by different Qt tools to create the final application.



Basic Project Structure - Project File

The project file, contains the information needed for the *qmake* too to create a platform specific makefile to build our project.

qmake looks for certain variables in our .pro file to figure out what to put in the platform specific makefile

```
QT += gui
TARGET = helloworld
TEMPLATE = app

HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp

# Platform specific files
win32 {
SOURCES += hellowin.cpp
}
unix {
SOURCES += hellounix.cpp
}
symbian {
SOURCES += hellosymbian.cpp
}

RESOURCES += images.qrc
FORMS += coolui.ui
CONFIG += qt debug
```

Note, there are three ways to assign values to variables:
variablename = *value* (assigns and overwrites previous values)
variablename += *value* (appends to previous values)
variablename -= *value* (substracts from previous values)

Basic Project Structure - Project File

- QT
  - Specifies what qt modules our project relies on (core and gui are added per default). If I need networking I would add **network** there
- TARGET
  - Name of the target executable
- TEMPLATE
  - Determines whether we are building an application or libraries.

```
QT += core gui
TARGET = helloworld
TEMPLATE = app

HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp

# Platform specific files
win32 {
SOURCES += hellowin.cpp
}
unix {
SOURCES += hellounix.cpp
}
symbian {
SOURCES += hellosymbian.cpp
}

RESOURCES += images.qrc
FORMS += coolui.ui
CONFIG += qt debug
```

Basic Project Structure - Project File

- HEADERS
  - Contains all the header files use in our project
- SOURCES
  - Source files used in our project

```
QT += core gui
TARGET = helloworld
TEMPLATE = app

HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp

# Platform specific files
win32 {
SOURCES += hellowin.cpp
}
unix {
SOURCES += hellounix.cpp
}
symbian {
SOURCES += hellosymbian.cpp
}

RESOURCES += images.qrc
FORMS += coolui.ui
CONFIG += qt debug
```

Notice we can make comments using the #

Using *scopes* {} we can make conditional structures which are only executed if the specified variable is set. Works pretty much like an if statement.

win32, unix and symbian are automatically specified depending on our target platform

Basic Project Structure - Project File

- RESOURCES
  - resource collection files .qrc (often containing icons or images used in our GUI)
- FORMS
  - UI designer forms .ui
- CONFIG
  - Options and features that the compiler should use

```
QT += core gui
TARGET = helloworld
TEMPLATE = app

HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp

# Platform specific files
win32 {
SOURCES += hellowin.cpp
}
unix {
SOURCES += hellounix.cpp
}
symbian {
SOURCES += hellosymbian.cpp
}

RESOURCES += images.qrc
FORMS += coolui.ui
CONFIG += qt debug
```

Basic Project Structure - Project File

- Other good to know variables:
  - LIBS
    - Specifies 3rd party libraries that we are relying on
  - INCLUDEPATH
    - Specifies include paths of external headers
  - DEFINES
    - Specifies macros to be used when compiling our applications

- Help:
  - http://doc.qt.nokia.com/4.6/qmake-manual.html
  - QtCreator -> Help -> Contents -> QMake Manual

Basic Project Structure - Project File

Symbian specific project options

```
symbian {
    TARGET.CAPABILITIES = LocalServices ReadUserData WriteUserData \
                          NetworkServices UserEnvironment Location ReadDeviceData
}
```

http://doc.qt.nokia.com/4.6/qmake-platform-notes.html#symbian-platform

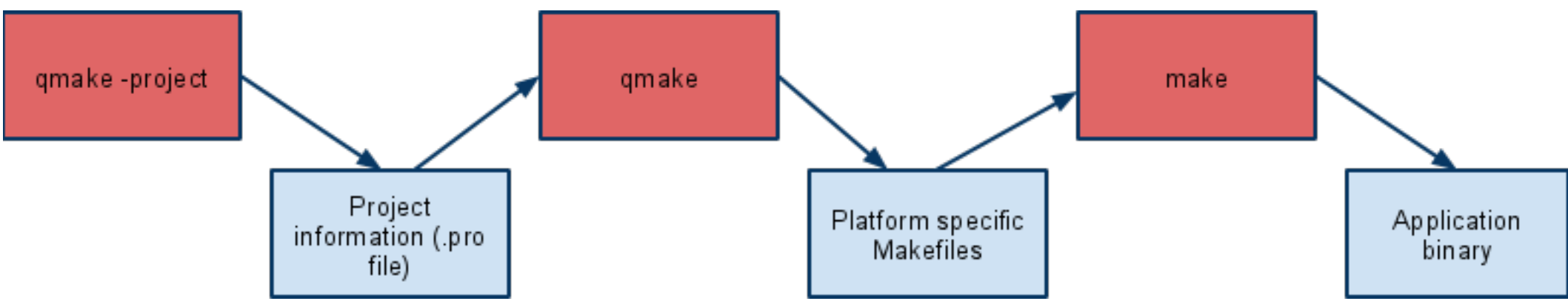
Qt Mobility specific project options:

```
CONFIG += mobility
MOBILITY += systeminfo
```

Qt Creator -> Help -> Contents -> Qt Mobility Project Reference Documentation -> Examples -> Quickstart Example

Building Qt Applications

1. qmake -project
   - Creates the Qt project file (.pro). This can also be created manually, or by using an IDE.
2. qmake
   - Generates platform specific Makefiles based on the contents of the .pro file.
   - Generates the make rules to invoke the **moc** for header files containing the Q_OBJECT macro
3. make
   - Compiles the program for the current platform (probably most familiar to you if you have done Linux programming)
   - Also executes the **moc**, **uic**, and **rcc**



Qt Creator does this for

Qt Tools - moc, uic, rcc

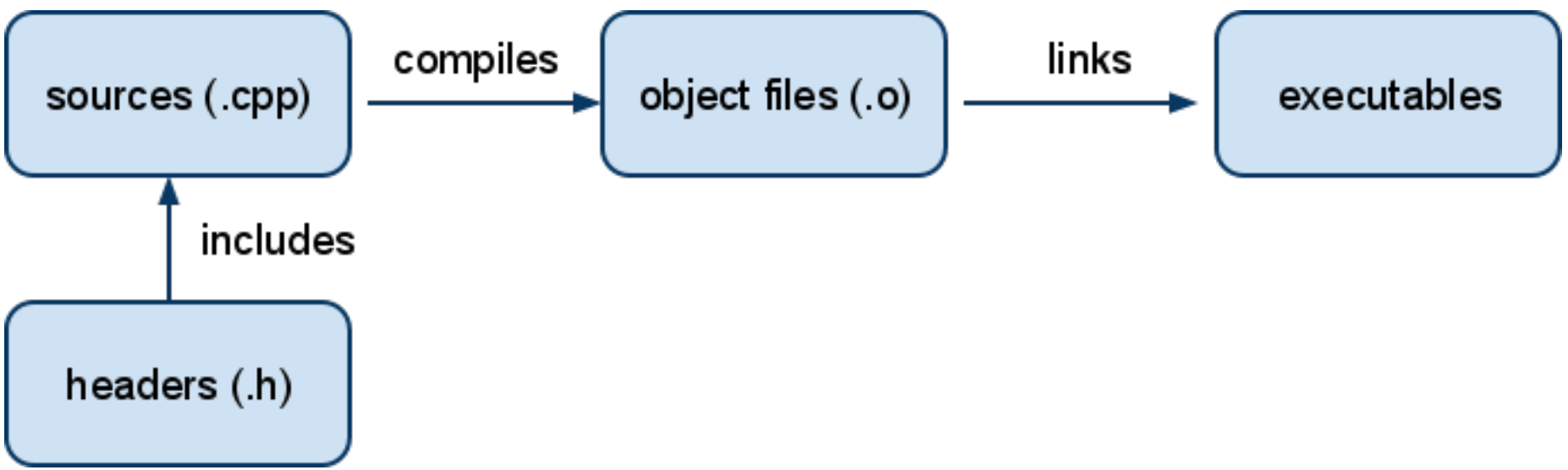
- moc (Meta Object Compiler)
  - From each class header, a specific meta object source filer is generated
- uic (UI Compiler)
  - Generates class header files (containing source code..) from the Qt Designer XML-files.
- rcc (Resource Compiler)
  - Works by generating a C++ source file containing data specified in a Qt resource collection (.qrc) file.

These are executed automatically during the compilation process.

MOC

The meta data needed to provide signals/slots and other Qt functionalities are gathered at compile time by the meta object compiler, moc.

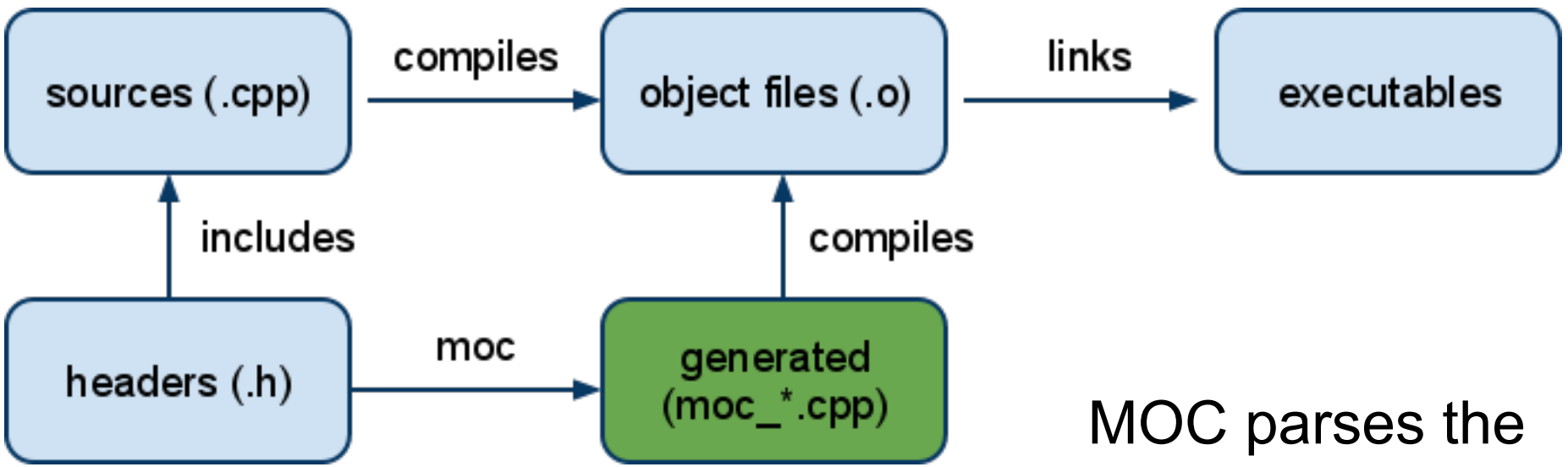
- Standard build process



MOC

The meta data needed to provide signals/slots and other Qt functionalities are gathered at compile time by the meta object compiler, moc.

- Qt build process



MOC parses the application headers.

I18n and Localication

Often it isn't enough to allow users to enter text in their native language; the entire user interface should be translated as well.

Qt makes this easy - lets see how.

- Three tools
  - Command line tool: lupdate
  - GUI tool: Qt Linguist
  - Command line tool: lrelease

I18n and Localication

- Use tr() function in your source code
- Add the TRANSLATIONS entry to the .pro file
- Use lupdate to create translation script files (.ts)
  - Checks tr() functions in the source code
  - Adds entries to the translation sources (XML files)
- Load translation sources in the Qt Linguist tool and provide localized language variants.
- Save the translations into .qm files (binary files)
  - Or create them using lrelease command line tool

Qt Tools - Resource Compiler (RCC)

RCC (Resource Compiler)

- Resources can be compiled into an application binary or used as external binary resource files.
- Information about compiled-in resources specified in XML format (.qrc)

- Compiled-in resources
  - RCC generates C++ source files containing data in Qt resource specification (.qrc) as static C++ arrays
  - Platform independent mechanism for storing binary data.
- External binary resources
  - Compiled manually with rcc switch
  - Must be registered in our Qt application using the QResource API.

Qt Resource Collection Files

- The resources associated with an application are specified in a **resource collection**, a **.qrc** file.
  - XML
  - Lists files on disk
  - Optionally assigns them a resource name that the application must use to access the resource
  - A prefix can be used to logically group resources in the same folder

```
<RCC>
    <qresource prefix="/">
        <file>images/loading.png</file>
        <file>images/down-speed.png</file>
        <file>images/up-speed.png</file>
    </qresource>
</RCC>
```

```
upImage->setPixmap(QPixmap(":/images/up-speed.png")); downImage->setPixmap
(QPixmap(":/images/down-speed.png"));
```

Resource Paths and Localization

- Resources are accessible under the same name as they have in a source tree with a **:/** prefixPath prefix can be changed using qresource tag's prefix attribute

```
<qresource prefix="/myresources">
    <file alias="cut-img.png">images/cut.png</file>
</qresource>
```

- Localization is as easy using qresource tag's lang attribute

```
<qresource>
    <file>cut.jpg</file>
</qresource>

<qresource lang="fr">
    <file alias="cut.jpg">cut_fr.jpg</file>
</qresource>
```

External Binary Resources

- Create a resource data file (extension .rcc)
  - Must use a command line option
- Access the resource in your code using QResource

```
rcc -binary mystuff.qrc -o mystruff.rcc
```

- In the application, this resource would be registered with code like this:

```
QResource::registerResource("/path/to/mystuff.rcc");
```

Compiled-in Resources

- Add resource collection file name into the project file
  - RESOURCES += myApp.qrc
- The qmake tool creates rules to generate qrc_application.cpp file
  - Contains all the data for the resources as static C++ arrays of compressed binary data
- Currently, Qt always stores the data directly into executable