ClassGenerator Users Guide

Table of Contents

Introduction

The *ClassGenerator* utility converts the Qt Designer UI forms into standard C++ source and header files. This program was written to simplify the process used by the author to create source files from the Qt Designer UI file for GUI elements. The difference between the output created by this utility and the output produced by Qt version is that the *ClassGenerator* version is more focused on creating traditional C++ source and header files. The downside when using this method is that revisions of the UI forms are difficult to incorporate into existing programs.

In addition to creating the source and header files from UI forms the *ClassGenerator* utility can create or update a project file and create a generic resource file if a resource file does not exist.

Overview

The *ClassGenerator* user interface is shown in illustration 1:

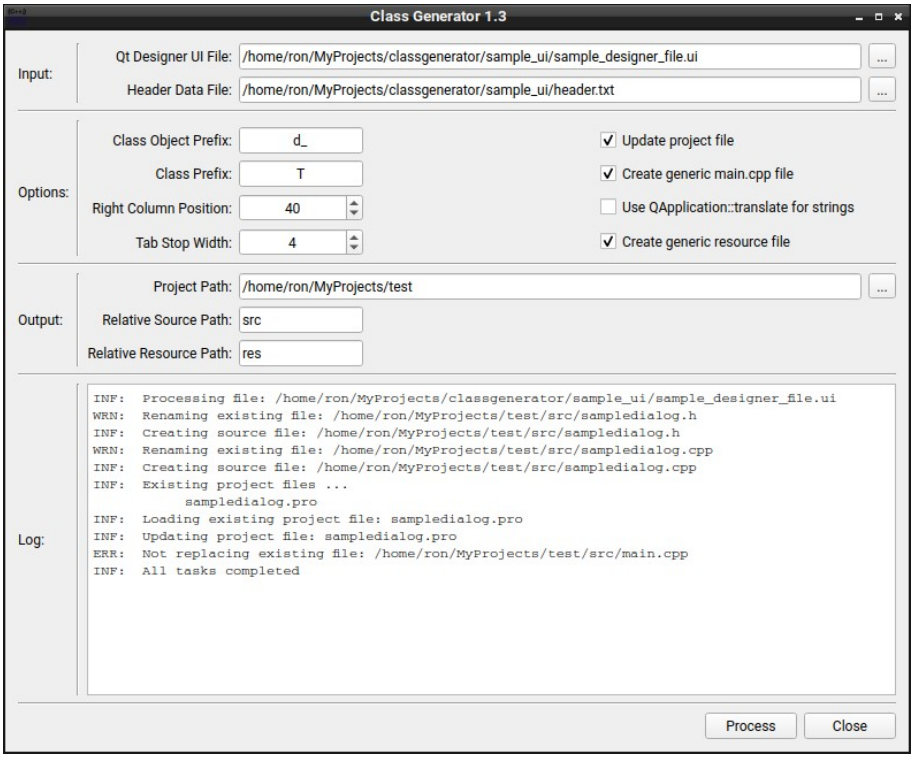

Illustration 1: ClassGenerator user interface

Options:

Input Options	*Description*
Qt Designer UI File	The UI Designer file. This is an XML based file containing all the GUI elements of a dialog, widget, or other control interface.
Header Data File	A file containing the header data that is to appear at the top of all source and header files.

Options	*Description*
Class Object Prefix	Identification text used to decide if an object should be created in the class constructor or in the private data section of the class. Objects found in the designer file that match this prefix will not appear in the constructor but in the private declaration of the class.
Class Prefix	Text added at the beginning of the class name. For example, if the name of a top level Dialog in the Qt Designer is 'MyDialog' and the class prefix is 'T' then the generated source files will have 'TMyDialog' for the class name.
Right Column Position	All variables are created on individual lines. All variables will be placed at this character position from the left margin.
Tab Stop Width	The number of characters representing each tab. Most editors default to 8 characters per tab but some prefer 4 characters instead.
Update project File	The existing project file can be updated with the names of the new source and header file(s) if checked. If there is no existing project file then a new, generic, project file is created.
Create generic main.cpp file	When a new project file is created a new, generic, main.cpp file can also be generated at the same time. The references in this file are always from the current widget. This file is only created when there is no existing project file.
Use QApplication::translate for strings	String constants can be initialized using QStringLiteral() or QApplication::translate() functions. For programs that support multiple languages the translate option should be used.
Create generic resource file	When a new project file is created a new, generic, resource file can also be generated at the same time. This file is only created when there is no existing project file.

Output Options	*Description*
Project Path	Directory where the project file will reside.
Relative Source Path	Subdirectory of the project path where all source files will be created.
Relative Resource Path	Subdirectory of the project path where all resource files will be created.

Log	*Description*
<Progress Log>	A record of each step is displayed in this section of the utility. All error codes, warnings, and information steps are shown in this log window.

Log	*Description*
Process	Process the input file and perform all requested tasks.
Close	Close the *ClassGenerator* utility.

The input for the UI file, the header data, and the project path are drag and drop aware. Dragging the file or directory into any of these input fields will have the content automatically set.

ClassGenerator Example

The following is an example of a Qt Designer UI file processed by the *ClassGenerator* utility.

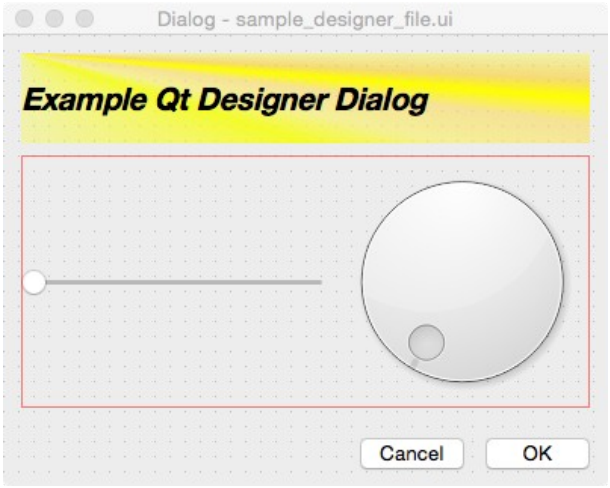

Illustration 2: Example Qt Designer dialog as displayed by the Qt Designer

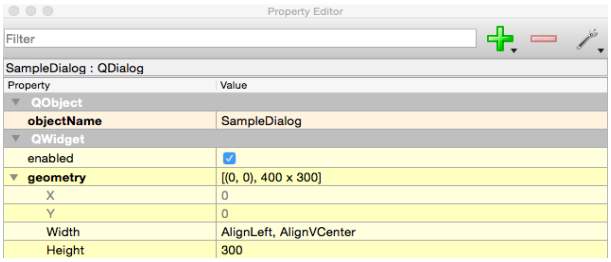

Illustration 3: Property view of example UI file top level widget.

In the example shown in illustration 2 a dialog is created with a various control elements. Illustration 3 shows the name of the top level object set to *SampleDialog* and is a subclass of *QDialog*. The UI file is saved with the name *sample_designer_file.ui*.

Project Output File

With the option *Update project file* checked if an existing project file is found at the project path then this file is updated with the new source files. In this case the project path was empty so a new project file was created.

The name of the existing project file is not changed. When a new project file is created the name of the top level widget from the Qt Designer file is used.

Example of a generated project file:

```
CONFIG +=           release \
                    warn_on \
                    qt \
                    thread \
                    c++11
```

```
TEMPLATE =          app
DESTDIR =           bin

CONFIG(debug, debug|release) {
     TARGET =       sampledialogd
} else {
     TARGET =       sampledialog
}

OBJECTS_DIR =       build/obj
MOC_DIR =           build/moc

QT +=               core \
                    gui

greaterThan(QT_MAJOR_VERSION, 4): {
QT +=               widgets
}

SOURCES +=          src/main.cpp \
                    src/sampledialog.cpp

HEADERS +=          src/sampledialog.h

RESOURCES +=        res/sampledialog.qrc

macx{

# ICON =            res/icons.icns
}

linux-g++{

isEmpty(PREFIX) {
PREFIX =            /usr/lib/sampledialog
}
target.path =       $$PREFIX
target.files =      bin/sampledialog
INSTALLS +=         target

}

win32 {
# RC_FILE =         res/resource.rc
}
```

Some of the options are added but commented out. For example, the Win32:RC_FILE entry requires a resource file otherwise it will generate an error. When the resource file is created this line can be un-commented.

Source Output Files

The source files (*.cpp and *.h) are created when the Qt Designer UI file is processed. All sources files are placed in the *Relative Source Path* subfolder relative to the current location of the project. An example of the contents of the source files are shown below:

Header File:

```
////////////////////////////////////////////////////////////////////
```

```
//
//      HEADER FILE FROM INPUT
//
////////////////////////////////////////////////////////////////////////

#ifndef SAMPLEDIALOGHEADERFILE
#define SAMPLEDIALOGHEADERFILE

#include <QDialog>

class QDial;
class QDialogButtonBox;
class QSlider;

class TSampleDialog : public QDialog
{
Q_OBJECT

public:

// CREATORS
     TSampleDialog(const QWidget *parent=0, Qt::WindowFlags flags=Qt::WindowFlags());
     ~TSampleDialog(void);

// ACCESSORS

// MANIPULATORS

signals:

private slots:

private:
     QDial                                  *d_dial;
     QDialogButtonBox                       *d_button_box;
     QSlider                                *d_slider_widget;

// NOT IMPLEMENTED
     TSampleDialog(const TSampleDialog&);
     TSampleDialog& operator=(const TSampleDialog&);
};

#endif
```

Source File:

```
////////////////////////////////////////////////////////////////////////
//
//      HEADER FILE FROM INPUT
//
////////////////////////////////////////////////////////////////////////

#include <QApplication>
#include <QDial>
#include <QDialogButtonBox>
#include <QFont>
#include <QHBoxLayout>
#include <QLabel>
#include <QSlider>
#include <QVBoxLayout>
```

```
#include "sampledialog.h"

TSampleDialog::TSampleDialog(
    const QWidget                           *parent,
    const Qt::WindowFlags                   flags)
:QDialog(const_cast<QWidget*>(parent),flags)
{
    QFont                                   font;
    QHBoxLayout                             *horizontalLayout;
    QLabel                                  *label;
    QVBoxLayout                             *verticalLayout;

    font.setPointSize(20);
    font.setWeight(75);
    font.setItalic(true);
    font.setBold(true);

    this->resize(400,300);

    verticalLayout = new QVBoxLayout(this);

    label = new QLabel(this);
    label->setMaximumSize(16777215,60);
    label->setFont(font);
    label->setStyleSheet(QStringLiteral("background-color:      qconicalgradient(cx:0,
cy:0, angle:135, stop:0 rgba(255, 255, 0, 69), stop:0.375 rgba(255, 255, 0, 69),
stop:0.423533 rgba(251, 255, 0, 145), stop:0.45 rgba(247, 255, 0, 208), stop:0.477581
rgba(255, 244, 71, 130), stop:0.518717 rgba(255, 218, 71, 130), stop:0.55 rgba(255,
255, 0, 255), stop:0.57754 rgba(255, 203, 0, 130), stop:0.625 rgba(255, 255, 0, 69),
stop:1 rgba(255, 255, 0, 69));"));
    verticalLayout->addWidget(label);

    horizontalLayout = new QHBoxLayout();

    d_slider_widget = new QSlider(this);
    d_slider_widget->setMaximumSize(200,16777215);
    d_slider_widget->setMaximum(100);
    d_slider_widget->setOrientation(Qt::Horizontal);
    horizontalLayout->addWidget(d_slider_widget);

    d_dial = new QDial(this);
    d_dial->setMaximum(100);
    horizontalLayout->addWidget(d_dial);
    verticalLayout->addLayout(horizontalLayout);

    d_button_box = new QDialogButtonBox(this);
    d_button_box->setOrientation(Qt::Horizontal);
    d_button_box->setStandardButtons(QDialogButtonBox::Cancel|QDialogButtonBox::Ok);
    verticalLayout->addWidget(d_button_box);

    this->setWindowTitle(QStringLiteral("Dialog"));
    label->setText(QStringLiteral("Example Qt Designer Dialog"));

    connect(d_button_box,SIGNAL(accepted(void)),this,SLOT(accept(void)));
    connect(d_button_box,SIGNAL(rejected(void)),this,SLOT(reject(void)));
    connect(d_slider_widget,SIGNAL(sliderMoved(int)),d_dial,SLOT(setValue(int)));
}

TSampleDialog::~TSampleDialog(void)
```

```
{
}
```

Effect of Input Options

The generated output is partially defined on the input options. The following is a list of how the different options affected the generated code:

- The output file names are *sampledialog.h* and *sampledialog.cpp*. The file names are always a lower case version of the widget name which is SampleDialog in this case.
- Each source file has the contents of the header data file placed at the top.
- The include guards in the header file are based on the name of the input widget.
- The class name, *TSampleDialog*, is a combined name from the option *Class Prefix* and the name of the widget.
- Class private objects are placed in the header file. Objects that are assigned in this location start with the same text as displayed in the option *Class Object Prefix*. Objects that do not have the same prefix will appear in the constructor instead.
- Forward declarations are added in the header for all class object variables.
- All class signals and slots created in the designer will be automatically declared in the header. All class slots will also be defined in the cpp file (not shown in example).
- The copy and assignment constructors are placed in the private section and left unimplemented.
- All object and function parameters are placed in two columns. The type of variable is placed one tab stop from the left margin and the variable declaration is placed based on the option *Right Column Position*.

Column positions are set using tabs. The actual position of the column will be equal to or the first tab stop position beyond the value of option Right Column Position. The option Tab Stop Width must be set to match the text editor used to display the code in order to have the appearance as intended.

- All objects are placed at the top of the constructor and sorted based on type and variable name (in that order).
- All text is initialized at the end of the constructor in one section of the program.
- The option Use *QApplication::translate for strings* will determine if the text is produced using one of these two methods:

```
this->setWindowTitle(QApplication::translate("TSampleDialog","Dialog",0));
```

or

```
this->setWindowTitle(QStringLiteral("Dialog"));
```

Main and Resource Files

This main.cpp is created if the option *Create generic main.cpp file* is checked and only when a new project file is created (not an update of an existing project file). When there is an existing project file likely there is already an existing main.cpp so this is only created if no existing main.cpp file is found.

Main.cpp file:

```
////////////////////////////////////////////////////////////////////
//
//      HEADER FILE FROM INPUT
//
////////////////////////////////////////////////////////////////////

#include <QApplication>
#include <QIcon>
#include <QStyleFactory>

#include "sampledialog.h"

int main(int argc, char *argv[])
{
     int                                        ret_value;

     QApplication app(argc, argv);

#ifndef Q_OS_MAC
     QApplication::setStyle(QStyleFactory::create("fusion"));
#endif

     app.setWindowIcon(QIcon(QStringLiteral(":/icons/sampledialog.png")));

     TSampleDialog *w = new TSampleDialog;

     w->show();
     ret_value = app.exec();

     delete w;
     return ret_value;
}
```

Resource File:

This resource file is created if the option *Create generic resource file* is checked and only when a new project file is created (not an update of an existing project file). When there is an existing project file likely there is already an existing resource file so this is only created if no existing project file is found.

All resource files are created in the subfolder *Relative Resource Path* from the location of the project. The name of the resource file will match the name of the top level widget which would be *sampledialog.qrc* in this case.

```
<!DOCTYPE RCC><RCC version="1.0">
     <qresource prefix="/icons">
          <file>sampledialog.png</file>
     </qresource>
     <qresource prefix="/gui">
     </qresource>
     <qresource prefix="/images">
     </qresource>
</RCC>
```

Compiling Project

Once all project, source, and resource files are in the proper location it is only necessary to compile and run the program:

```
bash-3.2$ qmake
bash-3.2$ make
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/
bin/clang++              -c              -pipe              -O2              -isysroot
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
MacOSX10.11.sdk -mmacosx-version-min=10.7 -Wall -W -fPIE -DQT_NO_DEBUG -DQT_WIDGETS_LIB
-DQT_GUI_LIB -DQT_CORE_LIB -I/usr/lib/qt5/mkspecs/macx-clang -I.
[-- compiler output removed -]
           bin/sampledialog.app/Contents/MacOS/sampledialog           build/obj/main.o
build/obj/sampledialog.o build/obj/qrc_sampledialog.o build/obj/moc_sampledialog.o    -
F/usr/lib/qt5/lib -framework QtWidgets -framework QtGui -framework QtCore -framework
DiskArbitration -framework IOKit -framework OpenGL -framework AGL
bash-3.2$ open bin/sampledialog.app
bash-3.2$
```

Illustration 4: Result of running compiled project.

Existing Projects

When the project file already exists there are some differences in the behavior of the *ClassGenerator* utility:

- The name of the existing project file is not changed.
- The option to generate a resource file is ignored. The existing project is expected to have an existing resource file already.
- The option *Create generic main.cpp file* only works if there is no existing main.cpp file.
- If the option *Update Project File* is checked the new source and header files will be added to the list of sources and headers in the existing project file.

Compilation Errors

In the event that the created source files return a compilation error then, most likely, a property type was not uniquely handled resulted in an invalid source code entry. The functions that deal with this are *Process_Property*() and *Process_Property_Content*() in the code file UI. There should be a corresponding warning in the *ClassGenerator* output log when this happens.

As part of testing every property and attribute combination were run through this utility along with a large number of sample UI files. It is possible some property flags and contents were missed. If something is found then the two *Process_Property*() functions should be updated to handle the missing property type.

Revision History

Date	Version	Changes
Dec 13, 2015	1.0	New Program
Feb 27, 2016	1.1	Bug Fix: Missing ampersand '&' in header copy constructor. Bug Fix: Layout properties not processed.
Oct 15, 2016	1.2	Bug Fix: QDockWidgets not handled properly Bug Fix: Complex QMenu and sub menus not handled properly. Updated debug output to show a more nested view of what is processed along with the generated code.
Nov 13, 2023	1.3	Bug Fix: Wrong character for slot function scope Using Qt::WindowFlags() instead of NULL for flags Removed references to nspool on macOS Added setWindowIcon in main.cpp Changed default icon size to 64x64 Redesign of interface.