ClassGenerator Users Guide

Table of Contents

Introduction

The ClassGenerator program is a small utility that will convert the Qt Designer UI forms into standard C++ source and header files. The difference between this program and the output produced by Qt version is that the source files are more traditional and not a separate header file.

In addition to creating the source and header files from the Qt Designer forms the ClassGenerator program can either create or update the project (pro) file along with a generic resource file.

This program was written to simplify the processed used by the author to create the various programs based on Qt using the Qt Designer to layout the GUI elements. Although the processes developed by Qt works very well it does not create a traditional set of source and header files that are preferable in some cases.

The ClassGenerator program is cross platform and can be compiled to run on OSX, Linux, and Windows.

Overview

The ClassGenerator user interface appears is described below:

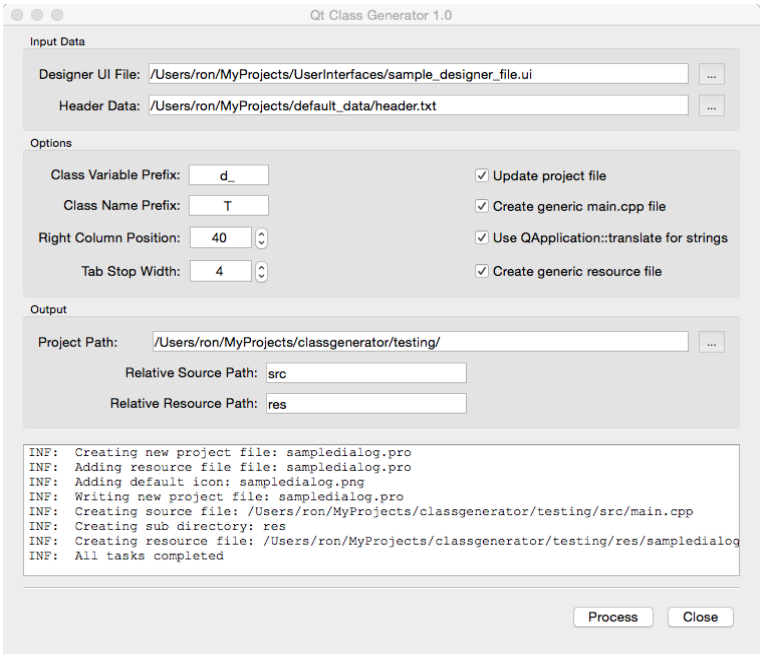


Illustration 1: ClassGenerator user interface

Input Data Option	Description
Designer UI File	The output file from the Qt Designer program. This file contains a description of all the GUI elements that make up a widget.

Input Data Option	Description
Header Data	A file containing the default header data that is to appear at the top of all source files.

Options	Description
Class Variable Prefix	Identification text used to decide if the variable should be created in the class constructor or in the private data section of the class. Variables found in the designer file that match this prefix will not appear in the constructor but in the private declaration of the class.
Class Name Prefix	Text that can be added at the beginning of parent widget class type. For example, if the name of the top level widget in the Qt Designer is 'Dialog' and this prefix is 'T' then the generated source will have 'TDialog' for a 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.

Options	Description
Project Path	Directory where the project file resides.
Relative Source Path	Path relative to the project path where all source files will be created.
Relative Resource Path	Path relative to the project path where all resource files will be created.

Other	Description
<Progress Log>	A record of each step is recorded in this section of the program. All error codes, warnings, and information steps are shown in this log

Other	Description
	window.
Process	Process the input file and perform all requested tasks.
Close	End the program.

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 fields will have the content automatically set.

Example Using the ClassGenerator Program

The best way to describe this is to create an example Qt Designer UI file and show the result of the different options available in the ClassGenerator program.

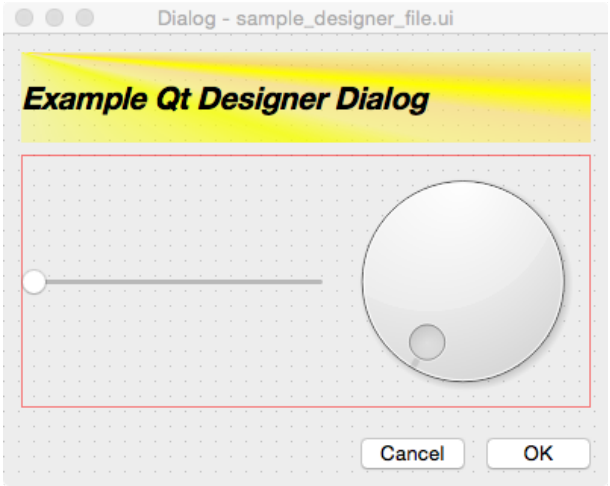


Illustration 2: Example Qt Designer dialog as displayed by the Qt Designer

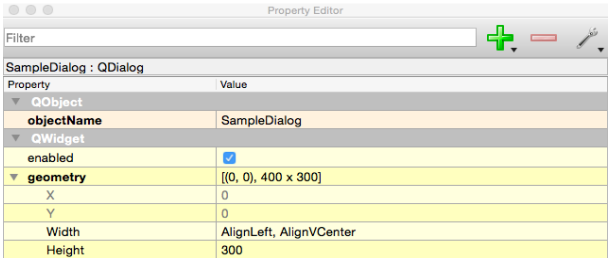
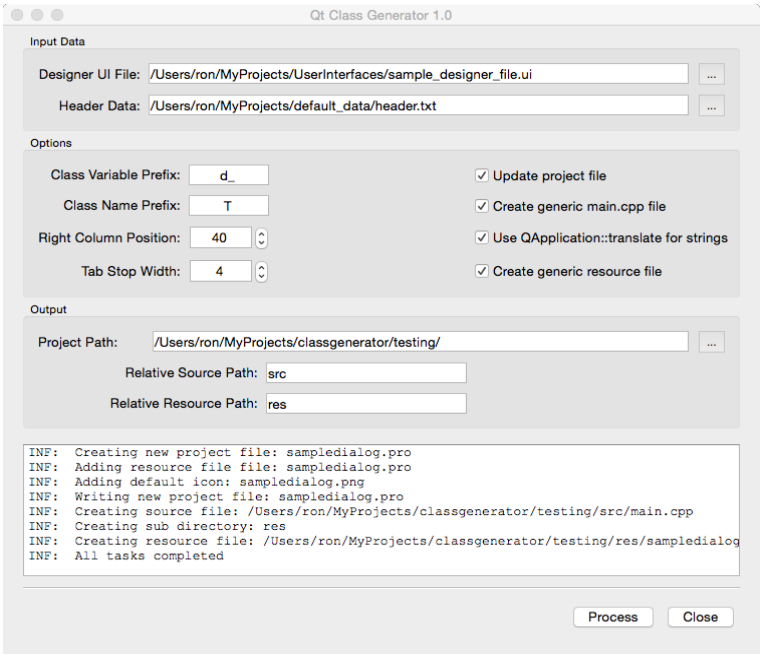


Illustration 3: Property view of example UI file top level widget.

In this example the top level name of the Qt Designer file is '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 (*.PRO) found at the project path location then this file is read and 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.

The generated output project file looks like this:

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

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 will likely be added at some in the future but until the proper resource (RC) file is created this will only generate a compile error.

Source Output Files

The source files (*.cpp and *.h) are always created when the Qt Designer UI file is processed. An example of the contents of these files are shown below in the following section.

All sources files are created in the subfolder 'Relative Source Path' from the location of the project.

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=0);
     ~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(QApplication::translate("TSampleDialog","Dialog",0));
    label->setText(QApplication::translate("TSampleDialog","Example Qt Designer
Dialog",0));

    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 code is partially controlled based 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 input widget name, 'SampleDialog'.
- Each source file has the contents of the input header 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 Name Prefix' and the name of the input widget.
- Class private variables are placed in the header file. Variables that are assigned in this location start with the same text as displayed in the option 'Class Variable Prefix'. Variables that do not have the same prefix appear in the class constructor instead.
- Forward declarations are added in the header for all class private 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 constructors are placed in the private section and left unimplemented. This prevents accidental copying of widgets without properly implementing these functions.
- All variables 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.

- All variables 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 <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

    TSampleDialog *w = new TSampleDialog;
```

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

    delete w;
    return ret_value;
}
```

Resource File:

This resource.qrc 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 resource 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/c
lang++ -c -pipe -O2 -isysroot
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/Mac
OSX10.11.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 program:

- 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 to generate a generic main.cpp will only work 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 well.

As part of testing every property and attribute combination were run through this program 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.