Building Common Project Types

This chapter describes how to set up qmake project files for three common project types that are based on Qt: application, library, and plugin. Although all project types use many of the same variables, each of them uses project-specific variables to customize output files.

Platform-specific variables are not described here. For more information, see Qt for Windows - Deployment and Qt for macOS.

Building an Application

The app template tells qmake to generate a Makefile that will build an application. With this template, the type of application can be specified by adding one of the following options to the CONFIG variable definition:

Option	Description
windows	The application is a Windows GUI application.
console	app template only: the application is a Windows console application.
testcase	The application is an automated test.

When using this template, the following qmake system variables are recognized. You should use these in your .pro file to specify information about your application. For additional platform-dependent system variables, you could have a look at the Platform Notes.

- HEADERS - A list of header files for the application.
- SOURCES - A list of C++ source files for the application.
- FORMS - A list of UI files for the application (created using Qt Designer).
- LEXSOURCES - A list of Lex source files for the application.
- YACCSOURCES - A list of Yacc source files for the application.
- TARGET - Name of the executable for the application. This defaults to the name of the project file. (The extension, if any, is added automatically).
- DESTDIR - The directory in which the target executable is placed.
- DEFINES - A list of any additional pre-processor defines needed for the application.
- INCLUDEPATH - A list of any additional include paths needed for the application.
- DEPENDPATH - The dependency search path for the application.
- VPATH - The search path to find supplied files.

INCLUDEPATHs then you do not need to specify any. qmake will add the necessary default values. An example project file might look like this:

```
TEMPLATE = app
DESTDIR  = c:/helloapp
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
DEFINES += USE_MY_STUFF
CONFIG  += release
```

For items that are single valued, such as the template or the destination directory, we use "="; but for multi-valued items we use "+=" to *add* to the existing items of that type. Using "=" replaces the variable value with the new value. For example, if we write `DEFINES=USE_MY_STUFF`, all other definitions are deleted.

Building a Testcase

A testcase project is an `app` project intended to be run as an automated test. Any `app` may be marked as a testcase by adding the value `testcase` to the `CONFIG` variable.

For testcase projects, qmake will insert a `check` target into the generated Makefile. This target will run the application. The test is considered to pass if it terminates with an exit code equal to zero.

The `check` target automatically recurses through SUBDIRS projects. This means it is possible to issue a `make check` command from within a SUBDIRS project to run an entire test suite.

The execution of the `check` target may be customized by certain Makefile variables. These variables are:

Variable	Description
TESTRUNNER	A command or shell fragment prepended to each test command. An example use-case is a "timeout" script which will terminate a test if it does not complete within a specified time.
TESTARGS	Additional arguments appended to each test command. For example, it may be useful to pass additional arguments to set the output file and format from the test (such as the `-o filename,format` option supported by QTestLib).

Note: The variables must be set while invoking the `make` tool, not in the .pro file. Most `make` tools support the setting of Makefile variables directly on the command-line:

```
# Run tests through test-wrapper and use JUnit XML output format.
# In this example, test-wrapper is a fictional wrapper script which terminates
# a test if it does not complete within the amount of seconds set by "--timeout".
# The "-o result.xml,junitxml" options are interpreted by QTestLib.
make check TESTRUNNER="test-wrapper --timeout 120" TESTARGS="-o result.xml,junitxml"
```

Testcase projects may be further customized with the following `CONFIG` options:

insignificant_test	The exit code of the test will be ignored during `make check`.

Test cases will often be written with QTest or `TestCase`, but it is not a requirement to make use of `CONFIG+=testcase` and `make check`. The only primary requirement is that the test program exit with a zero exit code on success, and a non-zero exit code on failure.

Building a Library

The `lib` template tells qmake to generate a Makefile that will build a library. When using this template, the VERSION variable is supported, in addition to the system variables that the `app` template supports. Use the variables in your .pro file to specify information about the library.

When using the `lib` template, the following options can be added to the CONFIG variable to determine the type of library that is built:

Option	Description
dll	The library is a shared library (dll).
staticlib	The library is a static library.
plugin	The library is a plugin.

The following option can also be defined to provide additional information about the library.

- VERSION - The version number of the target library. For example, 2.3.1.

The target file name for the library is platform-dependent. For example, on X11, macOS, and iOS, the library name will be prefixed by `lib`. On Windows, no prefix is added to the file name.

Building a Plugin

Plugins are built using the `lib` template, as described in the previous section. This tells qmake to generate a Makefile for the project that will build a plugin in a suitable form for each platform, usually in the form of a library. As with ordinary libraries, the VERSION variable is used to specify information about the plugin.

- VERSION - The version number of the target library. For example, 2.3.1.

Building a Qt Designer Plugin

Qt Designer plugins are built using a specific set of configuration settings that depend on the way Qt was configured for your system. For convenience, these settings can be enabled by adding `designer` to the QT variable. For example:

```
QT          += widgets designer
```

See the Qt Designer Examples for more examples of plugin-based projects.

Building and Installing in Debug and Release Modes

hold both debug and release options, only the option that is specified last is applied.

Building in Both Modes

To enable a project to be built in both modes, you must add the debug_and_release option to the CONFIG variable:

```
CONFIG += debug_and_release

CONFIG(debug, debug|release) {
    TARGET = debug_binary
} else {
    TARGET = release_binary
}
```

The scope in the above snippet modifies the build target in each mode to ensure that the resulting targets have different names. Providing different names for targets ensures that one will not overwrite the other.

When qmake processes the project file, it will generate a Makefile rule to allow the project to be built in both modes. This can be invoked in the following way:

```
make all
```

The build_all option can be added to the CONFIG variable in the project file to ensure that the project is built in both modes by default:

```
CONFIG += build_all
```

This allows the Makefile to be processed using the default rule:

```
make
```

Installing in Both Modes

The build_all option also ensures that both versions of the target will be installed when the installation rule is invoked:

```
make install
```

It is possible to customize the names of the build targets depending on the target platform. For example, a library or plugin may be named using a different convention on Windows from the one used on Unix platforms:

```
    mac: TARGET = $$join(TARGET,,,_debug)
    win32: TARGET = $$join(TARGET,,d)
}
```

The default behavior in the above snippet is to modify the name used for the build target when building in debug mode. An `else` clause could be added to the scope to do the same for release mode. Left as it is, the target name remains unmodified.

‹ Creating Project Files

Running qmake ›