创建项目文件

项目文件包含 qmake 构建应用程序、库或插件所需的所有信息。通常，您可以使用一系列声明来指定项目中的资源，但对简单编程构造的支持使您能够为不同的平台和环境描述不同的生成过程。

项目文件元素

qmake使用的项目文件格式可用于支持简单和相当复杂的构建系统。简单项目文件使用简单的声明性样式，定义标准变量以指示项目中使用的源文件和头文件。复杂项目可以使用控制流结构来微调生成过程。

以下各节介绍项目文件中使用的不同类型的元素。

变量

在项目文件中，变量用于保存字符串列表。在最简单的项目中，这些变量会通知 qmake 要使用的配置选项，或者提供要在构建过程中使用的文件名和路径。

qmake 在每个项目文件中查找某些变量，并使用这些变量的内容来确定它应该写入 Makefile 的内容。例如，标头和 SOURCES 变量中的值列表用于告诉 qmake 与项目文件位于同一目录中的标头文件和源文件。

变量还可以在内部用于存储临时值列表，并且可以使用新值覆盖或扩展现有值列表。

以下代码段说明了如何将值列表分配给变量：

```
HEADERS = mainwindow.h paintwidget.h
```

变量中的值列表按以下方式扩展：

```
SOURCES = main.cpp mainwindow.cpp \
          paintwidget.cpp
CONFIG += console
```

注意： 第一个赋值仅包括与变量在同一行上指定的值。第二个赋值使用反斜杠 (\) 将变量中的值拆分为多行。HEADERSSOURCES

配置变量是 qmake 在生成生成文件时使用的另一个特殊变量。常规配置中对此进行了讨论。在上面的代码段

变量	内容
配置	常规项目配置选项。
德斯特迪尔	将放置可执行文件或二进制文件的目录。
形式	用户界面编译器 （uic） 要处理的 UI 文件的列表。
头	生成项目时使用的标头 （.h） 文件的文件名列表。
.QT	项目中使用的 Qt 模块的列表。
资源	要包含在最终项目中的资源 （.qrc） 文件的列表。有关这些文件的更多信息，请参阅Qt资源系统。
来源	生成项目时要使用的源代码文件的列表。
模板	要用于项目的模板。这决定了构建过程的输出是应用程序、库还是插件。

可以通过在变量名称前面加上 来读取变量的内容。这可用于将一个变量的内容分配给另一个变量：$$

```
TEMP_SOURCES = $$SOURCES
```

该运算符广泛用于对字符串和值列表进行操作的内置函数。有关详细信息，请参阅 qmake 语言。$$

空白

通常，空格分隔变量赋值中的值。要指定包含空格的值，必须用双引号将这些值括起来：

```
DEST = "Program Files"
```

带引号的文本被视为变量所持有的值列表中的单个项目。类似的方法用于处理包含空格的路径，特别是在为 Windows 平台定义包含路径和 LIBS 变量时：

```
win32:INCLUDEPATH += "C:/mylibs/extra headers"
unix:INCLUDEPATH += "/home/user/extra headers"
```

Comments

You can add comments to project files. Comments begin with the character and continue to the end of the same line. For example:#

```
# Comments usually start at the beginning of a line, but they
# can also follow other content on the same line.
```

Built-in Functions and Control Flow

qmake provides a number of built-in functions to enable the contents of variables to be processed. The most commonly used function in simple project files is the include() function which takes a filename as an argument. The contents of the given file are included in the project file at the place where the function is used. The function is most commonly used to include other project files:`includeinclude`

```
include(other.pro)
```

Support for conditional structures is made available via scopes that behave like statements in programming languages:`if`

```
win32 {
    SOURCES += paintwidget_win.cpp
}
```

The assignments inside the braces are only made if the condition is true. In this case, the CONFIG option must be set. This happens automatically on Windows. The opening brace must stand on the same line as the condition.`win32`

More complex operations on variables that would usually require loops are provided by built-in functions such as find(), unique(), and count(). These functions, and many others are provided to manipulate strings and paths, support user input, and call external tools. For more information about using the functions, see qmake Language. For lists of all functions and their descriptions, see Replace Functions and Test Functions.

Project Templates

The TEMPLATE variable is used to define the type of project that will be built. If this is not declared in the project file, qmake assumes that an application should be built, and will generate an appropriate Makefile (or equivalent file) for the purpose.

The following table summarizes the types of projects available and describes the files that qmake will generate for each of them:

Template	qmake Output
app (default)	Makefile to build an application.
lib	Makefile to build a library.
aux	Makefile to build nothing. 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.
Template	qmake Output

vcapp	Visual Studio Project file to build an application.
vclib	Visual Studio Project file to build a library.
vcsubdirs	Visual Studio Solution file to build projects in sub-directories.

See Building Common Project Types for advice on writing project files for projects that use the and templates.`applib`

When the template is used, qmake generates a Makefile to examine each specified subdirectory, process any project file it finds there, and run the platform's tool on the newly-created Makefile. The variable is used to contain a list of all the subdirectories to be processed.`subdirs` `makeSUBDIRS`

General Configuration

The CONFIG variable specifies the options and features that the project should be configured with.

The project can be built in *release* mode or *debug* mode, or both. If debug and release are both specified, the last one takes effect. If you specify the option to build both the debug and release versions of a project, the Makefile that qmake generates includes a rule that builds both versions. This can be invoked in the following way:`debug_and_release`

```
make all
```

Adding the option to the variable makes this rule the default when building the project.`build_all` `CONFIG`

> **Note:** Each of the options specified in the variable can also be used as a scope condition. You can test for the presence of certain configuration options by using the built-in CONFIG() function. For example, the following lines show the function as the condition in a scope to test whether only the option is in use:`CONFIG` `opengl`

```
CONFIG(opengl) {
    message(Building with OpenGL support.)
} else {
    message(OpenGL support is not available.)
}
```

This enables different configurations to be defined for and builds. For more information, see Using Scopes.`release` `debug`

The following options define the type of project to be built.

> **Note:** Some of these options only take effect when used on the relevant platform.

Option	Description

x11	The project is an X11 application or library. This value is not needed if the target uses Qt.

The application and library project templates provide you with more specialized configuration options to fine tune the build process. The options are explained in detail in Building Common Project Types.

For example, if your application uses the Qt library and you want to build it in mode, your project file will contain the following line:debug

```
CONFIG += qt debug
```

Note: You must use "+=", not "=", or qmake will not be able to use Qt's configuration to determine the settings needed for your project.

Declaring Qt Libraries

If the CONFIG variable contains the value, qmake's support for Qt applications is enabled. This makes it possible to fine-tune which of the Qt modules are used by your application. This is achieved with the QT variable which can be used to declare the required extension modules. For example, we can enable the XML and network modules in the following way:qt

```
QT += network xml
```

Note: includes the and modules by default, so the above declaration *adds* the network and XML modules to this default list. The following assignment *omits* the default modules, and will lead to errors when the application's source code is being compiled:QTcoregui

```
QT = network xml # This will omit the core and gui modules.
```

If you want to build a project *without* the module, you need to exclude it with the "-=" operator. By default, contains both and , so the following line will result in a minimal Qt project being built:guiQTcoregui

```
QT -= gui # Only the core module is used.
```

For a list of Qt modules that you can add to the variable, see QT.QT

Configuration Features

append the feature name (the stem of the feature filename) to the variable.CONFIG

For example, qmake can configure the build process to take advantage of external libraries that are supported by pkg-config, such as the D-Bus and ogg libraries, with the following lines:

```
CONFIG += link_pkgconfig
PKGCONFIG += ogg dbus-1
```

For more information about adding features, see Adding New Configuration Features.

Declaring Other Libraries

If you are using other libraries in your project in addition to those supplied with Qt, you need to specify them in your project file.

The paths that qmake searches for libraries and the specific libraries to link against can be added to the list of values in the LIBS variable. You can specify the paths to the libraries or use the Unix-style notation for specifying libraries and paths.

For example, the following lines show how a library can be specified:

```
LIBS += -L/usr/local/lib -lmath
```

The paths containing header files can also be specified in a similar way using the INCLUDEPATH variable.

For example, to add several paths to be searched for header files:

```
INCLUDEPATH = c:/msdev/include d:/stl/include
```

‹ Getting Started with qmake

Building Common Project Types ›

Qt DOCUMENTATION

The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads
Marketplace

Feedback Sign In