Getting Started with qmake

This tutorial teaches you the basics of qmake. The other topics in this manual contain more detailed information about using qmake.

Starting Off Simple

Let's assume that you have just finished a basic implementation of your application, and you have created the following files:

- hello.cpp
- hello.h
- main.cpp

You will find these files in the `examples/qmake/tutorial` directory of the Qt distribution. The only other thing you know about the setup of the application is that it's written in Qt. First, using your favorite plain text editor, create a file called `hello.pro` in `examples/qmake/tutorial`. The first thing you need to do is add the lines that tell qmake about the source and header files that are part of your development project.

We'll add the source files to the project file first. To do this you need to use the SOURCES variable. Just start a new line with SOURCES += and put hello.cpp after it. You should have something like this:

```
SOURCES += hello.cpp
```

We repeat this for each source file in the project, until we end up with the following:

```
SOURCES += hello.cpp
SOURCES += main.cpp
```

If you prefer to use a Make-like syntax, with all the files listed in one go you can use the newline escaping like this:

```
SOURCES = hello.cpp \
          main.cpp
```

Once you have done this, your project file should look something like this:

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

The target name is set automatically. It is the same as the project filename, but with the suffix appropriate for the platform. For example, if the project file is called `hello.pro`, the target will be `hello.exe` on Windows and `hello` on Unix. If you want to use a different name you can set it in the project file:

```
TARGET = helloworld
```

The finished project file should look like this:

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

You can now use qmake to generate a Makefile for your application. On the command line, in your project directory, type the following:

```
qmake -o Makefile hello.pro
```

> **Note:** If you installed Qt via a package manager, the binary may be `qmake6`.

Then type `make` or `nmake` depending on the compiler you use.

For Visual Studio users, qmake can also generate Visual Studio project files. For example:

```
qmake -tp vc hello.pro
```

Making an Application Debuggable

The release version of an application does not contain any debugging symbols or other debugging information. During development, it is useful to produce a debugging version of the application that has the relevant information. This is easily achieved by adding `debug` to the CONFIG variable in the project file.

For example:

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

Use qmake as before to generate a Makefile. You will now obtain useful information about your application when running it in a debugging environment.

Adding Platform-Specific Source Files

After a few hours of coding, you might have made a start on the platform-specific part of your application, and decided to keep the platform-dependent code separate. So you now have two new files to include into your project file: `hellowin.cpp` and `hellounix.cpp`. We cannot just add these to the SOURCES variable since that would place both files in the Makefile. So, what we need to do here is to use a scope which will be processed depending on which platform we are building for.

A simple scope that adds the platform-dependent file for Windows looks like this:

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

When building for Windows, qmake adds `hellowin.cpp` to the list of source files. When building for any other platform, qmake simply ignores it. Now all that is left to be done is to create a scope for the Unix-specific file.

When you have done that, your project file should look something like this:

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
```

Use qmake as before to generate a Makefile.

Stopping qmake If a File Does Not Exist

You may not want to create a Makefile if a certain file does not exist. We can check if a file exists by using the exists() function. We can stop qmake from processing by using the error() function. This works in the same way as scopes do. Simply replace the scope condition with the function. A check for a file called main.cpp looks like this:

```
}
```

The `!` symbol is used to negate the test. That is, `exists( main.cpp )` is true if the file exists, and `!exists( main.cpp )` is true if the file does not exist.

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
unix {
    SOURCES += hellounix.cpp
}
!exists( main.cpp ) {
    error( "No main.cpp file found" )
}
```

Use qmake as before to generate a makefile. If you rename `main.cpp` temporarily, you will see the message and qmake will stop processing.

Checking for More than One Condition

Suppose you use Windows and you want to be able to see statement output with `qDebug()` when you run your application on the command line. To see the output, you must build your application with the appropriate console setting. We can easily put `console` on the `CONFIG` line to include this setting in the Makefile on Windows. However, let's say that we only want to add the `CONFIG` line when we are running on Windows *and* when `debug` is already on the `CONFIG` line. This requires using two nested scopes. First create one scope, then create the other inside it. Put the settings to be processed inside the second scope, like this:

```
win32 {
    debug {
        CONFIG += console
    }
}
```

Nested scopes can be joined together using colons, so the final project file looks like this:

```
CONFIG += debug
HEADERS += hello.h
SOURCES += hello.cpp
SOURCES += main.cpp
win32 {
    SOURCES += hellowin.cpp
}
```

```
}
!exists( main.cpp ) {
    error( "No main.cpp file found" )
}
win32:debug {
    CONFIG += console
}
```

That's it! You have now completed the tutorial for qmake, and are ready to write project files for your development projects.

‹ Overview

Creating Project Files ›