Platform Notes

Many cross-platform projects can be handled by the basic qmake configuration features. However, on some platforms, it is sometimes useful, or even necessary, to take advantage of platform-specific features. qmake knows about many of these features, which can be accessed via specific variables that only take effect on the platforms where they are relevant.

macOS, iOS, tvOS, and watchOS

Features specific to these platforms include support for creating universal binaries, frameworks and bundles.

Source and Binary Packages

The version of qmake supplied in source packages is configured slightly differently to that supplied in binary packages in that it uses a different feature specification. Where the source package typically uses the `macx-g++` specification, the binary package is typically configured to use the `macx-xcode` specification.

Users of each package can override this configuration by invoking qmake with the `-spec` option (see Running qmake for more information). For example, to use qmake from a binary package to create a Makefile in a project directory, invoke the following command:

```
qmake -spec macx-g++
```

Using Frameworks

qmake is able to automatically generate build rules for linking against frameworks in the standard framework directory on macOS, located at `/Library/Frameworks/`.

Directories other than the standard framework directory need to be specified to the build system, and this is achieved by appending linker options to the LIBS variable, as shown in the following example:

```
LIBS += -F/path/to/framework/directory/
```

The framework itself is linked in by appending the `-framework` options and the name of the framework to the LIBS variable:

Creating Frameworks

Any given library project can be configured so that the resulting library file is placed in a framework, ready for deployment. To do this, set up the project to use the `lib` template and add the `lib_bundle` option to the CONFIG variable:

```
TEMPLATE = lib
CONFIG += lib_bundle
```

The data associated with the library is specified using the QMAKE_BUNDLE_DATA variable. This holds items that will be installed with a library bundle, and is often used to specify a collection of header files, as in the following example:

```
FRAMEWORK_HEADERS.version = Versions
FRAMEWORK_HEADERS.files = path/to/header_one.h path/to/header_two.h
FRAMEWORK_HEADERS.path = Headers
QMAKE_BUNDLE_DATA += FRAMEWORK_HEADERS
```

You use the FRAMEWORK_HEADERS variable to specify the headers required by a particular framework. Appending it to the QMAKE_BUNDLE_DATA variable ensures that information about these headers is added to the collection of resources that will be installed with the library bundle. Also, the framework name and version are specified by the QMAKE_FRAMEWORK_BUNDLE_NAME and QMAKE_FRAMEWORK_VERSION variables. By default, the values used for these variables are obtained from the TARGET and VERSION variables.

See Qt for macOS - Deployment for more information about deploying applications and libraries.

Creating and Moving Xcode Projects

On macOS, developers can generate Xcode project files from an existing qmake project file. For example:

```
qmake -spec macx-xcode project.pro
```

Note: If a project is later moved on the disk, qmake must be run again to process the project file and create a new Xcode project file.

Supporting Two Build Targets Simultaneously

Implementing this is currently not feasible, because the Xcode concept of Active Build Configurations is conceptually different from the qmake idea of build targets.

The Xcode Active Build Configurations settings are for modifying Xcode configurations, compiler flags and similar build options. Unlike Visual Studio, Xcode does not allow for the selection of specific library files based on whether debug or release build configurations are selected. The qmake debug and release settings control which library files are linked to the executable.

Furthermore, the selected *Active Build Configuration* is stored in a .pbxuser file, which is generated by Xcode on first load, not created by qmake.

Windows

Features specific to this platform include support for Windows resource files (provided or auto-generated), creating Visual Studio project files, and handling manifest files when deploying Qt applications developed using Visual Studio 2005, or later.

Adding Windows Resource Files

This section describes how to handle a Windows resource file with qmake to have it linked to an application executable (EXE) or dynamic link library (DLL). qmake can optionally auto-generate a suitably filled Windows resource file.

A linked Windows resource file may contain many elements that can be accessed by its EXE or DLL. However, the Qt resource system should be used for accessing linked-in resources in a platform-independent way. But some standard elements of the linked Windows resource file are accessed by Windows itself. For example, in Windows explorer the version tab of the file properties is filled by resource elements. In addition, the program icon of the EXE is read from these elements. So it is good practice for a Qt created Windows EXE or DLL to use both techniques at the same time: link platform-independent resources via the Qt resource system and add Windows specific resources via a Windows resource file.

Typically, a resource-definition script (.rc file) is compiled to a Windows resource file. Within the Microsoft toolchain, the RC tool generates a .res file, which can be linked with the Microsoft linker to an EXE or DLL. The MinGW toolchain uses the windres tool to generate an .o file that can be linked with the MinGW linker to an EXE or DLL.

The optional auto-generation of a suitably filled .rc file by qmake is triggered by setting at least one of the system variables VERSION and RC_ICONS. The generated .rc file is automatically compiled and linked. Elements that are added to the .rc file are defined by the system variables QMAKE_TARGET_COMPANY, QMAKE_TARGET_DESCRIPTION, QMAKE_TARGET_COPYRIGHT, QMAKE_TARGET_PRODUCT, QMAKE_TARGET_ORIGINAL_FILENAME, QMAKE_TARGET_INTERNALNAME, QMAKE_TARGET_COMMENTS, QMAKE_TARGET_TRADEMARKS, QMAKE_MANIFEST, RC_CODEPAGE, RC_ICONS, RC_LANG and VERSION.

If these elements are not sufficient, qmake has the two system variables RC_FILE and RES_FILE that point directly to an externally created .rc or .res file. By setting one of these variables, the specified file is linked to the EXE or DLL.

Note: The generation of the .rc file by qmake is blocked, if RC_FILE or RES_FILE is set. In this case, no further changes are made to the given .rc file or the .res or .o file by qmake; the variables pertaining to .rc file generation have no effect.

Creating Visual Studio Project Files

This section describes how to import an existing qmake project into Visual Studio. qmake is able to take a project file and create a Visual Studio project that contains all the necessary information required by the development environment. This is achieved by setting the qmake project template to either `vcapp` (for application projects) or `vclib` (for library projects).

This can also be set using a command line option, for example:

```
qmake -tp vc
```

typing:

```
qmake -tp vc -r
```

Each time you update the project file, you need to run qmake to generate an updated Visual Studio project.

> **Note:** If you are using the Visual Studio Add-in, select **Qt** > **Import from .pro file** to import `.pro` files.

Visual Studio Manifest Files

When deploying Qt applications built using Visual Studio 2005, or later, make sure that the manifest file that was created when the application was linked is handled correctly. This is handled automatically for projects that generate DLLs.

Removing manifest embedding for application executables can be done with the following assignment to the CONFIG variable:

```
CONFIG -= embed_manifest_exe
```

Also, the manifest embedding for DLLs can be removed with the following assignment to the `CONFIG` variable:

```
CONFIG -= embed_manifest_dll
```

This is discussed in more detail in the deployment guide for Windows.

< Running qmake qmake Language >

Qt DOCUMENTATION

Investors
Newsroom
Careers
Office Locations

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