Configuring qmake

Properties

qmake has a system for persistent configuration, which allows you to set a property in qmake once, and query it each time qmake is invoked. You can set a property in qmake as follows:

```
qmake -set PROPERTY VALUE
```

The appropriate property and value should be substituted for PROPERTY and VALUE.

You can retrieve this information back from qmake as follows:

```
qmake -query PROPERTY
qmake -query #queries all current PROPERTY/VALUE pairs
```

Note: `qmake -query` lists built-in properties in addition to the properties that you set with `qmake -set PROPERTY VALUE`.

This information will be saved into a QSettings object (meaning it will be stored in different places for different platforms).

The following list summarizes the `built-in` properties:

- QMAKE_SPEC - the shortname of the host `mkspec` that is resolved and stored in the QMAKESPEC variable during a host build
- QMAKE_VERSION - the current version of qmake
- QMAKE_XSPEC - the shortname of the target `mkspec` that is resolved and stored in the QMAKESPEC variable during a target build
- QT_HOST_BINS - location of host executables
- QT_HOST_DATA - location of data for host executables used by qmake
- QT_HOST_LIBS - location of host libraries
- QT_HOST_LIBEXECS - location of executables required by host libraries at runtime

- QT_INSTALL_BINS - location of Qt binaries (tools and applications)
- QT_INSTALL_CONFIGURATION - location for Qt settings. Not applicable on Windows
- QT_INSTALL_DATA - location of general architecture-independent Qt data
- QT_INSTALL_DOCS - location of documentation
- QT_INSTALL_EXAMPLES - location of examples
- QT_INSTALL_HEADERS - location for all header files
- QT_INSTALL_LIBEXECS - location of executables required by libraries at runtime
- QT_INSTALL_LIBS - location of libraries
- QT_INSTALL_PLUGINS - location of Qt plugins
- QT_INSTALL_PREFIX - default prefix for all paths
- QT_INSTALL_QML - location of QML 2.x extensions
- QT_INSTALL_TESTS - location of Qt test cases
- QT_INSTALL_TRANSLATIONS - location of translation information for Qt strings
- QT_SYSROOT - the sysroot used by the target build environment
- QT_VERSION - the Qt version. We recommend that you query Qt module specific version numbers by using $$QT.<module>.version variables instead.

For example, you can query the installation of Qt for this version of qmake with the `QT_INSTALL_PREFIX` property:

```
qmake -query "QT_INSTALL_PREFIX"
```

You can query the values of properties in a project file as follows:

```
QMAKE_VERS = $$[QMAKE_VERSION]
```

QMAKESPEC

qmake requires a platform and compiler description file which contains many default values used to generate appropriate Makefiles. The standard Qt distribution comes with many of these files, located in the `mkspecs` subdirectory of the Qt installation.

The QMAKESPEC environment variable can contain any of the following:

- A complete path to a directory containing a `qmake.conf` file. In this case qmake will open the `qmake.conf` file from within that directory. If the file does not exist, qmake will exit with an error.
- The name of a platform-compiler combination. In this case, qmake will search in the directory specified by the `mkspecs` subdirectory of the data path specified when Qt was compiled (see QLibraryInfo::DataPath).

Note: The QMAKESPEC path will be automatically added to the generated Makefile after the contents of the INCLUDEPATH system variable.

Cache File

The cache file is a special file qmake reads to find settings not specified in the `qmake.conf` file, project files, or at the command line. When qmake is run, it looks for a file called `.qmake.cache` in parent directories of the current directory, unless you specify `-nocache`. If qmake fails to find this file, it will silently ignore this step of processing.

If qmake finds a `.qmake.cache` file then it will process this file first before it processes the project file.

File Extensions

Under normal circumstances qmake will try to use appropriate file extensions for your platform. However, it is sometimes necessary to override the default choices for each platform and explicitly define file extensions for qmake to use. This is achieved by redefining certain built-in variables. For example, the extension used for moc files can be redefined with the following assignment in a project file:

```
QMAKE_EXT_MOC = .mymoc
```

The following variables can be used to redefine common file extensions recognized by qmake:

- QMAKE_EXT_MOC modifies the extension placed on included moc files.
- QMAKE_EXT_UI modifies the extension used for *Qt Designer* UI files (usually in FORMS).
- QMAKE_EXT_PRL modifies the extension placed on library dependency files.
- QMAKE_EXT_LEX changes the suffix used in Lex files (usually in LEXSOURCES).
- QMAKE_EXT_YACC changes the suffix used in Yacc files (usually in YACCSOURCES).
- QMAKE_EXT_OBJ changes the suffix used on generated object files.

All of the above accept just the first value, so you must assign to it just one value that will be used throughout your project file. There are two variables that accept a list of values:

- QMAKE_EXT_CPP causes qmake to interpret all files with these suffixes as C++ source files.
- QMAKE_EXT_H causes qmake to interpret all files with these suffixes as C and C++ header files.

‹ Using Precompiled Headers Reference ›