Search



# Test Functions

Test functions return a boolean value that you can test for in the conditional parts of scopes. Test functions can be divided into built-in functions and function libraries.

See also Replace Functions.

## Built-in Test Functions

Basic test functions are implemented as built-in functions.

### cache(variablename, [set|add|sub] [transient] [super|stash], [source variablename])

This is an internal function that you will typically not need.

This function was introduced in Qt 5.0.

### CONFIG(config)

This function can be used to test for variables placed into the CONFIG variable. This is the same as scopes, but has the added advantage that a second parameter can be passed to test for the active config. As the order of values is important in variables (that is, the last one set will be considered the active config for mutually exclusive values) a second parameter can be used to specify a set of values to consider. For example:CONFIG

```
CONFIG = debug
CONFIG += release
CONFIG(release, debug|release):message(Release build!) #will print
CONFIG(debug, debug|release):message(Debug build!) #no print
```

Because release is considered the active setting (for feature parsing) it will be the CONFIG used to generate the build file. In the common case a second parameter is not needed, but for specific mutual exclusive tests it is invaluable.

### contains(variablename, value)

Succeeds if the variable contains the value ; otherwise fails. It is possible to specify a regular expression for

For example:

```
contains( drivers, network ) {
    # drivers contains 'network'
    message( "Configuring for network build..." )
    HEADERS += network.h
    SOURCES += network.cpp
}
```

The contents of the scope are only processed if the variable contains the value . If this is the case, the appropriate files are added to the SOURCES and HEADERS variables.`driversnetwork`

## count(variablename, number)

Succeeds if the variable contains a list with the specified of values; otherwise fails.`variablenamenumber`

This function is used to ensure that declarations inside a scope are only processed if the variable contains the correct number of values. For example:

```
options = $$find(CONFIG, "debug") $$find(CONFIG, "release")
count(options, 2) {
    message(Both release and debug specified.)
}
```

## debug(level, message)

Checks whether qmake runs at the specified debug level. If yes, it returns true and prints a debug message.

## defined(name[, type])

Tests whether the function or variable is defined. If is omitted, checks all functions. To check only variables or particular type of functions, specify . It can have the following values:`nametypetype`

- `test` only checks test functions
- `replace` only checks replace functions
- `var` only checks variables

## equals(variablename, value)

Tests whether equals the string .`variablenamevalue`

For example:

```
TARGET = helloworld
equals(TARGET, "helloworld") {
    message("The target assignment was successful.")
}
```

## error(string)

This function never returns a value. qmake displays as an error message to the user and exits. This function should only be used for unrecoverable errors.`string`

For example:

```
error(An error has occurred in the configuration process.)
```

## eval(string)

Evaluates the contents of the string using qmake syntax rules and returns true. Definitions and assignments can be used in the string to modify the values of existing variables or create new definitions.

For example:

```
eval(TARGET = myapp) {
    message($$TARGET)
}
```

**Note:** Quotation marks can be used to delimit the string, and the return value can be discarded if it is not needed.

## exists(filename)

Tests whether a file with the given exists. If the file exists, the function succeeds; otherwise it fails.`filename`

The argument may contain wildcards. In that case, this function succeeds if any file matches.`filename`

For example:

```
exists( $(QTDIR)/lib/libqt-mt* ) {
      message( "Configuring for multi-threaded Qt..." )
      CONFIG += thread
}
```

**Note:** "/" should be used as a directory separator, regardless of the platform in use.

## export(variablename)

Exports the current value of from the local context of a function to the global context.`variablename`

## for(iterate, list)

For example:

```
LIST = 1 2 3
for(a, LIST):exists(file.$${a}):message(I see a file.$${a}!)
```

Loops can be interrupted with . The statement skips the remainder of the loop's body and continues execution with the next iteration.`break()` `next()`

## greaterThan(variablename, value)

Tests that the value of is greater than . First, this function attempts a numerical comparison. If at least one of the operands fails to convert, this function does a string comparison.`variable` `name` `evalue`

For example:

```
ANSWER = 42
greaterThan(ANSWER, 1) {
    message("The answer might be correct.")
}
```

It is impossible to compare two numbers as strings directly. As a workaround, construct temporary values with a non-numeric prefix and compare these.

For example:

```
VALUE = 123
TMP_VALUE = x$$VALUE
greaterThan(TMP_VALUE, x456): message("Condition may be true.")
```

See also lessThan().

## if(condition)

Evaluates . It is used to group boolean expressions.`condition`

For example:

```
if(linux-g++*|macx-g++*):CONFIG(debug, debug|release) {
    message("We are on Linux or Mac OS, and we are in debug mode.")
}
```

## include(filename)

You can check whether the file was included by using this function as the condition for a scope. For example:

```
include( shared.pri )
OPTIONS = standard custom
!include( options.pri ) {
    message( "No custom build options specified" )
OPTIONS -= custom
}
```

## infile(filename, var, val)

Succeeds if the file (when parsed by qmake itself) contains the variable with a value of ; otherwise fails. If you do not specify , the function tests whether has been assigned in the file.`filenamevarvalvalvar`

## isActiveConfig

This is an alias for the function.`CONFIG`

## isEmpty(variablename)

Succeeds if the variable is empty; otherwise fails. This is the equivalent of .`variablenamecount( variablename, 0 )`

For example:

```
isEmpty( CONFIG ) {
CONFIG += warn_on debug
}
```

## isEqual

This is an alias for the function.`equals`

## lessThan(variablename, value)

Tests that the value of is less than . Works as greaterThan().`variablenamevalue`

For example:

```
ANSWER = 42
lessThan(ANSWER, 1) {
    message("The answer might be wrong.")
}
```

## load(feature)

log(message)

Prints a message on the console. Unlike the function, neither prepends text nor appends a line break.`message`

This function was introduced in Qt 5.0.

See also message().

## message(string)

Always succeeds, and displays as a general message to the user. Unlike the function, this function allows processing to continue.`stringerror()`

```
message( "This is a message" )
```

The above line causes "This is a message" to be written to the console. The use of quotation marks is optional, but recommended.

> **Note:** By default, messages are written out for each Makefile generated by qmake for a given project. If you want to ensure that messages only appear once for each project, test the variable in conjunction with a scope to filter out messages during builds. For example:`build_pass`

```
!build_pass:message( "This is a message" )
```

## mkpath(dirPath)

Creates the directory path . This function is a wrapper around the QDir::mkpath function.`dirPath`

This function was introduced in Qt 5.0.

## requires(condition)

Evaluates . If the condition is false, qmake skips this project (and its SUBDIRS) when building.`condition`

> **Note:** You can also use the REQUIRES variable for this purpose. However, we recommend using this function, instead.

## system(command)

Executes the given in a secondary shell. Succeeds if the command returns with a zero exit status; otherwise fails. You can check the return value of this function using a scope.`command`

For example:

```
system("ls /bin"): HAS_BIN = TRUE
```

### touch(filename, reference_filename)

Updates the time stamp of to match the time stamp of .`filenamereference_filename`

This function was introduced in Qt 5.0.

### unset(variablename)

Removes from the current context.`variablename`

For example:

```
NARF = zort
unset(NARF)
!defined(NARF, var) {
    message("NARF is not defined.")
}
```

### versionAtLeast(variablename, versionNumber)

Tests that the version number from is greater than or equal to . The version number is considered to be a sequence of non-negative decimal numbers delimited by "."; any non-numerical tail of the string will be ignored. Comparison is performed segment-wise from left to right; if one version is a prefix of the other, it is considered smaller.`variablenameversionNumber`

This function was introduced in Qt 5.10.

### versionAtMost(variablename, versionNumber)

Tests that the version number from is less than or equal to . Works as versionAtLeast().`variablenameversionNumber`

This function was introduced in Qt 5.10.

### warning(string)

Always succeeds, and displays as a warning message to the user.`string`

### write_file(filename, [variablename, [mode]])

Writes the values of to a file with the name , each value on a separate line. If is not specified, creates an empty file. If is and the file already exists, appends to it instead of replacing it.`variablenamefilenamevariablenamemodeappend`

This function was introduced in Qt 5.0.

## Test Function Library

Complex test functions are implemented in a library of .prf files.

Uses the PKGCONFIG mechanism to determine whether or not the given packages exist at the time of project parsing.

This can be useful to optionally enable or disable features. For example:

```
packagesExist(sqlite3 QtNetwork QtDeclarative) {
    DEFINES += USE_FANCY_UI
}
```

然后，在代码中：

```
#ifdef USE_FANCY_UI
    // Use the fancy UI, as we have extra packages available
#endif
```

## 准备回复目标（目标）

通过准备循环访问所有子目录的目标，促进创建与目标类似的项目范围的目标。例如：`install`

```
TEMPLATE = subdirs
SUBDIRS = one two three
prepareRecursiveTarget(check)
```

在其 .CONFIG 中具有或指定的子对象将从此目标中排除：`have_no_defaultno_<target>_target`

```
two.CONFIG += no_check_target
```

必须手动将准备好的目标添加到QMAKE_EXTRA_TARGETS：

```
QMAKE_EXTRA_TARGETS += check
```

要使目标成为全局，需要将上述代码包含在每个子对象子项目中。此外，为了使这些目标执行任何操作，非子对象子项目需要包含相应的代码。实现此目的的最简单方法是创建自定义功能文件。例如：

```
# <project root>/features/mycheck.prf
equals(TEMPLATE, subdirs) {
    prepareRecursiveTarget(check)
```

需要将功能文件注入到每个子项目中，例如通过 .qmake.conf：

```
# <project root>/.qmake.conf
CONFIG += mycheck
```

此功能在 Qt 5.0 中引入。

## qt编译测试（测试）

生成测试项目。如果测试通过，则返回 true 并将其添加到配置变量中。否则，将返回 false。`config_<test>`

要使此功能可用，您需要加载相应的功能文件：

```
# <project root>/project.pro
load(configure)
```

这还会将变量QMAKE_CONFIG_TESTS_DIR设置为项目父目录的子目录。加载功能文件后，可以覆盖此值。`config.tests`

在测试目录中，每个测试必须有一个子目录，其中包含一个简单的 qmake 项目。以下代码段阐释了该项目的 .pro 文件：

```
# <project root>/config.tests/test/test.pro
SOURCES = main.cpp
LIBS += -ltheFeature
# Note that the test project is built without Qt by default.
```

以下代码段阐释了该项目的主.cpp文件：

```
// <project root>/config.tests/test/main.cpp
#include <TheFeature/MainHeader.h>
int main() { return featureFunction(); }
```

以下代码片段显示了测试的调用：

```
# <project root>/project.pro
qtCompileTest(test)
```

If the test project is built successfully, the test passes.

To suppress the re-use of cached results, pass to qmake.`CONFIG+=recheck`

See also load().

This function was introduced in Qt 5.0.

## qtHaveModule(name)

Checks whether the Qt module specified by is present. For a list of possible values, see QT.`name`

This function was introduced in Qt 5.0.1.

‹ Replace Functions

Wiki

Downloads

Marketplace



Feedback Sign In