测试功能

测试函数返回一个布尔值，您可以在作用域的条件部分中测试该值。测试函数可以分为内置函数和函数库。

另请参见替换函数。

内置测试功能

基本测试函数作为内置函数实现。

缓存（变量名、[设置|添加|子] [瞬态] [超级|短划]、[源变量名]）

这是您通常不需要的内部函数。

此功能在 Qt 5.0 中引入。

配置（配置）

此函数可用于测试放置在 CONFIG 变量中的变量。这与作用域相同，但具有额外的优点，即可以传递第二个参数来测试活动配置。由于值的顺序在变量中很重要（即，最后一组将被视为互斥值的活动配置），因此可以使用第二个参数来指定要考虑的一组值。例如：CONFIG

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

由于发布被视为活动设置（用于功能分析），因此它将是用于生成构建文件的 CONFIG。在常见情况下，不需要第二个参数，但对于特定的相互排除测试，它是无价的。

包含（变量名、值）

如果变量包含值 ;则成功。否则将失败。可以为参数*值*指定正则表达式。variablenamevalue

可以使用作用域检查此函数的返回值。

例如：

```
contains( drivers, network ) {
```

```
    HEADERS += network.h
    SOURCES += network.cpp
}
```

仅当变量包含值 时，才会处理作用域的内容。如果是这种情况，则会将相应的文件添加到源和标头变量中。driversnetwork

计数（变量名、数字）

如果变量包含具有指定值的列表，则成功;否则将失败。variablenamenumber

此函数用于确保仅当变量包含正确数量的值时才处理作用域内的声明。例如：

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

调试（级别，消息）

检查 qmake 是否在指定的调试级别运行。如果是，则返回 true 并打印调试消息。

已定义（名称[，类型]）

测试是否定义了函数或变量。如果省略，则检查所有函数。要仅检查变量或特定类型的函数，请指定 。它可以具有以下值：nametypetype

- test仅检查测试函数
- replace仅检查替换函数
- var仅检查变量

等于（变量名、值）

测试 是否等于字符串 。variablenamevalue

例如：

```
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

```
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)

Starts a loop that iterates over all values in , setting to each value in turn. As a convenience, if is 1..10 then iterate will iterate over the values 1 through 10.`listiteratelist`

For example:

```
LIST = 1 2 3
```

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` `value`

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)

Includes the contents of the file specified by into the current project at the point where it is included. This function succeeds if is included; otherwise it fails. The included file is processed immediately.`filename` `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
```

```
}
```

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)

Loads the feature file () specified by , unless the feature has already been loaded.`.prffeature`

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.

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
```

See also the replace variant of system().

touch(filename, reference_filename)

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

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.

packagesExist(packages)

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:

```
    DEFINES += USE_FANCY_UI
}
```

And then, in the code:

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

prepareRecursiveTarget(target)

Facilitates the creation of project-wide targets similar to the target by preparing a target that iterates through all subdirectories. For example:`install`

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

Subdirs that have or specified in their .CONFIG are excluded from this target:`have_no_defaultno_<target>_target`

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

You must add the prepared target manually to QMAKE_EXTRA_TARGETS:

```
QMAKE_EXTRA_TARGETS += check
```

To make the target global, the code above needs to be included into every subdirs subproject. In addition, to make these targets do anything, non-subdirs subprojects need to include respective code. The easiest way to achieve this is creating a custom feature file. For example:

```
# <project root>/features/mycheck.prf
equals(TEMPLATE, subdirs) {
    prepareRecursiveTarget(check)
} else {
    check.commands = echo hello user
}
QMAKE_EXTRA_TARGETS += check
```

The feature file needs to be injected into each subproject, for example by qmake.conf

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

This function was introduced in Qt 5.0.

qtCompileTest(test)

Builds a test project. If the test passes, true is returned and is added to the CONFIG variable. Otherwise, false is returned.`config_<test>`

To make this function available, you need to load the respective feature file:

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

This also sets the variable QMAKE_CONFIG_TESTS_DIR to the subdirectory of the project's parent directory. It is possible to override this value after loading the feature file.`config.tests`

Inside the tests directory, there has to be one subdirectory per test that contains a simple qmake project. The following code snippet illustrates the .pro file of the project:

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

The following code snippet illustrates the main .cpp file of the project:

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

The following code snippet shows the invocation of the test:

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

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

The test results are automatically cached, which also makes them available to all subprojects. It is therefore recommended to run all configuration tests in the top-level project file.

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

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

© 2022 The Qt Company Ltd. Documentation contributions included herein are the copyrights of their respective owners. The documentation provided herein is licensed under the terms of the GNU Free Documentation License version 1.3 as published by the Free Software Foundation. Qt and respective logos are trademarks of The Qt Company Ltd. in Finland and/or other countries worldwide. All other trademarks are property of their respective owners.

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

Qt DOCUMENTATION

© 2022 The Qt Company

Feedback Sign In