

---

# **MosquitoNet Documentation**

***Release 0.0.0***

**Simon Bourne**

December 04, 2015

## CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Features . . . . .	1
1.2	Example . . . . .	1
<b>2</b>	<b>Getting Started</b>	<b>4</b>
<b>3</b>	<b>Assertions</b>	<b>5</b>
<b>4</b>	<b>BDD Tests</b>	<b>6</b>
<b>5</b>	<b>Parameterized Tests</b>	<b>7</b>
<b>6</b>	<b>Model Checking</b>	<b>8</b>
<b>7</b>	<b>Customizing Assertions</b>	<b>9</b>
<b>8</b>	<b>Pretty Printing Your Types</b>	<b>10</b>
<b>9</b>	<b>Test Harness Command Line</b>	<b>11</b>
<b>10</b>	<b>Custom Reporters</b>	<b>12</b>
<b>11</b>	<b>Supported Compilers</b>	<b>13</b>

## INTRODUCTION

MosquitoNet is a C++14 unit testing and model checking framework for Linux, OS X and Windows.

### 1.1 Features

- Single header version makes it simple to get started.
- Only one macro, which only adds file and line arguments to a simple function call. This means you're always dealing with core C++ code, so you could generate tests programatically.
- Simple tests or BDD style tests.
- Parameterized tests.
- Model checking. Specify a model and the values each argument can have and MosquitoNet will check every combination of arguments against your model.
- Customizable assertions using natural C++ expressions.
- Tests can continue to run after the first failure.
- Nested test contexts.

### 1.2 Example

This example shows most of the features of MosquitoNet. For a gentler introduction, see [Getting Started](#).

```
#include "MosquitoNet.h"

#include <vector>
#include <set>

using namespace Enhedron::Test;
using std::vector;
using std::set;

// We'll use this later in some parameterized tests.
void checkVectorSize(Check& check, size_t size) {
    vector<int> v(size, 0);
    check(length(VAR(v)) == size);
}

static Suite u("Util",
    given("a very simple test", [] (auto& check) {
```

```

int a = 1;

// VAR is the only macro we need. If the name clashes, undef it
// and use M_ENHEDRON_VAR. Upon failure, this will log "a == 1",
// along with the value of `a`.
check(VAR(a) == 1);
}),

given("an empty set", [] (auto& check) {
    set<int> s;

    // Upon failure, this will log "length(s) == 1", along with
    // the contents of the set.
    check("it is initially empty", length(VAR(s)) == 0u);

    check.when("we add an element", [&] {
        s.insert(1);
        check("the size is 1", length(VAR(s)) == 1u);

        // This test will run twice. The first time, it will run the when
        // block labelled when("we add a different element"), but skip the
        // when block labelled when("we add the same element"). The second
        // time it runs, it will do the inverse. There can be an arbitrary
        // number of when blocks within each block, nested to an arbitrary
        // depth.
        check.when("we add a different element", [&] {
            s.insert(2);
            check("the size is 2", length(VAR(s)) == 2u);
        });

        check.when("we add the same element", [&] {
            s.insert(1);
            check("the size is still 1", length(VAR(s)) == 1u);
        });
    });
}),

// Parameterized tests. There can be any number or type of parameters.
given("a vector of size 0", checkVectorSize, 0),
given("a vector of size 10", checkVectorSize, 10),

// Model checking.
exhaustive(
    choice(0, 10, 20), // These are the 3 values for `initialSize`.
    choice(0, 5, 10, 15, 20, 25) // and these are the 6 for `resizeTo`.
    // This will run the test 3 * 6 = 18 times for every
    // combination of arguments.
).
given("a vector with some elements", [] (
    Check& check,
    size_t initialSize,
    size_t resizeTo
)
{
    vector<int> v(initialSize, 0);
    check("the initial size is correct",
        length(VAR(v)) == initialSize);
}

```

```
        check.when("we resize it", [&] {
            v.resize(resizeTo);
            check("the new size is correct",
                length(VAR(v)) == resizeTo);

            check("the size <= the capacity",
                length(VAR(v)) <= v.capacity());
        });
    },

    context("we can also nest contexts",
        context("to an arbitrary depth",
            given("an empty test to illustrate that tests can go here",
                [] (auto& check) {
            })
        )
    )
);
```

## GETTING STARTED

Download the single header latest release of the single header, *MosquitoNet.h* from [here](#). In the same directory as *MosquitoNet.h*, create a file *Harness.cpp* with this code in it:

```
#include "MosquitoNet.h"

int main(int argc, const char* argv[]) {
    return Enhedron::Test::run(argc, argv);
}
```

Then compile it with `g++` (version 5 or later, but 4.9 will work with `-std=c++1y`):

```
g++ --std=c++14 -o test-harness Harness.cpp
```

Now run `./test-harness` and you should get this output:

```
Totals: 0 tests, 0 checks, 0 fixtures
```

Let's add a simple test. We'll just check the value of a variable. Edit your *TestHarness.cpp* so it contains:

```
#include "MosquitoNet.h"

#include <vector>

using namespace Enhedron::Test;
using std::vector;

static Suite u("Util",
    given("a very simple test", [] (auto& check) {
        int a = 1;

        check(VAR(a) == 1);
    })
);

int main(int argc, const char* argv[]) {
    return run(argc, argv);
}
```

**ASSERTIONS**

**BDD TESTS**



## PARAMETERIZED TESTS

**MODEL CHECKING**

## **CUSTOMIZING ASSERTIONS**

## PRETTY PRINTING YOUR TYPES

## TEST HARNESS COMMAND LINE

## CUSTOM REPORTERS

## **SUPPORTED COMPILERS**

- g++ version 4.9 or later.
- clang version 3.6 or later.
- Microsoft Visual C++ 2015 or later.