Сравнительный обзор FLOSS Testing Frameworks для Embedded C++

Алексей Хлебников

LVEE 2016 Winter

Requirements

Support for:

- Android, iOS, Linux, MacOS X, Windows.
- Custom runner.
- Custom outputter.
- Fixtures.
- Testsuites.

Wishlist

- Easy and pleasant to use.
- Supported and mature.
- Little boilerplating.
- Run only some tests.
- List available tests.

Testing framework list

- Bandit
- Boost.Test
- CATCH
- CppUnit
- CxxTest
- Google Test
- Igloo
- Lest
- TUT
- UnitTest++

How they do it in Ruby

```
class MyTestSuite < Test::Unit::TestCase

  def setup
    @num = 2
  end

  def teardown
    @num = 0
  end

  def test_one_thing
    assert(@num == 2)
  end

  def test_another_thing
    assert_equals(@num * @num, 4)
  end

end
```

CppUnit

```
class MyTestSuite : public CppUnit::TestFixture
{
public:
    void setUp()
    {
        m_num = 2;
    }

    void tearDown()
    {
        m_num = 0;
    }

    void testOneThing()
    {
        CPPUNIT_ASSERT(m_num == 2);
    }

    void testAnotherThing()
    {
        CPPUNIT_ASSERT_EQUAL(m_num * m_num, 4);
    }
    ...
};
```

CppUnit

```
auto* suite = new CppUnit::TestSuite("MyTestSuite");

suite->addTest(
    new CppUnit::TestCaller <MyTestSuite> (
        "testOneThing",
        &MyTestSuite::testOneThing
    )
);

suite->addTest(
    new CppUnit::TestCaller <MyTestSuite> (
        "testAnotherThing",
        &MyTestSuite::testAnotherThing
    )
);

CppUnit::TextUi::TestRunner runner;

runner.addTest(suite);
```

CppUnit

```
class MyTestSuite : public CppUnit::TestFixture
{
    CPPUNIT_TEST_SUITE( MyTestSuite );
    CPPUNIT_TEST( testOneThing );
    CPPUNIT_TEST( testAnotherThing );
    CPPUNIT_TEST_SUITE_END();

public:
    void setUp();
    void tearDown();

    void testOneThing();
    void testAnotherThing();
    ...
};
...
CPPUNIT_TEST_SUITE_REGISTRATION( MyTestSuite );
```

CppUnit

Supports:

- Custom runner.
- Custom outputter by subclassing class TextOutputter and overriding 1 function.
- Fixtures and testsuites.
- Listing available tests.
- Selective running of one particular test.
- Unlike most other frameworks, defining and registering test both using C++-only code and using macros.

CppUnit

Downsides:

- No test autoregistration.
- Very much boilerplating.
- The framework is supported, but not very actively, by LibreOffice team.

Google Test

```
class MyFixtureClass : public testing::Test
{
public:
    void SetUp()
    {
        m_num = 2;
    }

    void TearDown()
    {
        m_num = 0;
    }
    ...
};

TEST_F(MyFixtureClass, testOneThing)
{
    ASSERT_TRUE(m_num == 2);
}

TEST_F(MyFixtureClass, testAnotherThing)
{
    EXPECT_EQ(m_num * m_num, 4);
}
```

Google Test

Supports:

- Custom runner.
- Fixtures and testsuites.
- Test autoregistration.
- Listing available tests.
- Running subset of tests, including and excluding them by path-like wildcards.

Downsides:

- Custom outputter is not supported. The framework uses C file descriptors for output. The best that can be done is redirecting output to the file.

Boost.Test

```
struct MyFixtureStructure {
    MyFixtureStructure()  { m_num = 2; }
    ~MyFixtureStructure() { m_num = 0; }
    ...
};

BOOST_FIXTURE_TEST_SUITE( MyTestSuite, MyFixtureStructure )

    BOOST_AUTO_TEST_CASE( test_one_thing )
    {
        BOOST_REQUIRE(m_num == 2);
    }

    BOOST_AUTO_TEST_CASE( test_another_thing )
    {
        BOOST_CHECK_EQUAL(m_num * m_num, 4);
    }

BOOST_AUTO_TEST_SUITE_END()
```

Boost.Test

Supports:

- Custom runner.
- Custom outputter by subclassing std::ostream.
- Fixtures and testsuites.
- Test autoregistration.
- Listing available tests.
- Running subset of tests, selecting by path-like wildcards and tags.
- Static, dynamic or header-only linking.

Drawbacks:

- Slow compilation, especially as header-only library.

CxxTest

```
class MyTestSuite : public CxxTest::TestSuite
{
public:
    void setUp()
    {
        m_num = 2;
    }

    void tearDown()
    {
        m_num = 0;
    }

    void testOneThing()
    {
        TS_ASSERT(m_num == 2);
    }

    void testAnotherThing()
    {
        TS_ASSERT_EQUALS(m_num * m_num, 4);
    }
    ...
};
```

CxxTest

Supports:

- Custom runner.
- Custom outputter by subclassing class OutputStream and overriding 3 functions.
- Fixtures and testsuites.
- Test autoregistration by running a Pyhon script.
- Listing available tests.
- Selective running of one particular test or testsuite.

Catch

```
TEST_CASE( "My test suite name", "[my_tag]" ) {

    num = 2;
    REQUIRE( num == 2 );

    SECTION( "increment" ) {
        num++;
        REQUIRE(num == 3);
    }

    SECTION( "decrement" ) {
        num--;
        REQUIRE(num == 1);

        SECTION( "increment after decrement" ) {
            num++;
            REQUIRE(num == 2);
        }
    }
}
```

Catch

```
SCENARIO( "My test suite name", "[my_tag]" ) {

  GIVEN( "a number" ) {
    num = 2;
    REQUIRE( num == 2 );

    WHEN( "increment happens" ) {
      num++;

      THEN( "number becomes bigger" ) {
        REQUIRE(num == 3);
      }
    }

    WHEN( "decrement happens" ) {
      num--;

      THEN( "number becomes smaller" ) {
        REQUIRE(num == 1);
      }
    }
  }
}
```

Catch

Supports:

- Custom runner.
- Custom outputter by subclassing std::ostream.
- Fixtures and testsuites.
- SECTIONS and SCENARIOS!
- Test autoregistration.
- Listing available tests.
- Running subset of tests, selecting by path-like wildcards and tags.

Lest

```
const lest::test specification[] = {
    CASE( "My test suite name", "[my_tag]" ) {
        SETUP ( "setting up a number" ) {
            num = 2;
            EXPECT( num == 2 );

            SECTION( "increment" ) {
                num++;
                EXPECT(num == 3);
            }

            SECTION( "decrement" ) {
                num--;
                EXPECT(num == 1);

                SECTION( "increment after decrement" ) {
                    num++;
                    EXPECT(num == 2);
                }
            }
        }
    }
}
```

Lest

```
const lest::test specification[] = {
    SCENARIO( "My test suite name", "[my_tag]" ) {

        GIVEN( "a number" ) {
            num = 2;
            EXPECT( num == 2 );

            WHEN( "increment happens" ) {
                num++;

                THEN( "number becomes bigger" ) {
                    EXPECT(num == 3);
                }
            }

            WHEN( "decrement happens" ) {
                num--;

                THEN( "number becomes smaller" ) {
                    EXPECT(num == 1);
                }
            }
        }
    }
}
```

Igloo

```
Describe(MyTestSuite) {
    void SetUp() {
        m_num = 2;
    }

    It(is_initialized_correctly) {
        Assert::That(m_num, Equals(2));
    }

    Describe(increment) {
        void SetUp() {
            m_num++;
        }

        It(increased) {
            Assert::That(m_num, Equals(3));
        }
    };

    int m_num;
};
```

Bandit

```
go_bandit([](){
    describe("My test suite", [](){
        int m_num;

        before_each([&](){
            m_num = 2;
        });

        it("is initialized correctly", [&](){
            AssertThat(m_num, Equals(2));
        });

        describe("increment", [&](){

            before_each([&](){
                m_num++;
            });

            it("increased", [&](){
                AssertThat(m_num, Equals(3));
            });
        });
    });
});
```

TUT

```
namespace tut
{
    struct MyFixtureStructure
    {
        MyFixtureStructure()  { m_num = 2; }
        ~MyFixtureStructure() { m_num = 0; }
        ...
    };

    test_group<MyFixtureStructure> my_test_suite;

    template<> template<>
    void test_group<MyFixtureStructure>::object::test<1>()
    {
        ensure("initialized incorrectly", m_num == 2);
    }

    template<> template<>
    void test_group<MyFixtureStructure>::object::test<1>()
    {
        set_test_name("test another thing");
        ensure_equals("it does not compute", m_num * m_num, 4);
    }
}
```

UnitTest++

```
SUITE(MyTestSuite)
{
    class MyFixtureStructure
    {
        MyFixtureStructure()  { m_num = 2; }
        ~MyFixtureStructure() { m_num = 0; }
        ...
    };

    TEST_FIXTURE(MyFixtureStructure, TestOneThing )
    {
        CHECK(m_num == 2);
    }

    TEST_FIXTURE(MyFixtureStructure, TestAnotherThing )
    {
        CHECK_EQUAL(m_num * m_num, 4);
    }
}
```

UnitTest++

Supports:

- Custom runner.
- Custom outputter by subclassing class TestReporter and overriding 4 functions.
- Fixtures and testsuites.
- Test autoregistration.

Downsides:

- No support for listing available tests.
- No support for selective test running.

Спасибо за внимание

Вопросы?