Unit testing with Qt Test

How to test C++ and GUI code with Qt

Qt day

VIKING SOFTWARE

In this presentation

- Introduction to Qt Test
  - Setting up a project
  - Unit test class
  - Testing values and conditions
  - Useful macros
- Data driven testing
- GUI testing
  - Testing a QWidget
  - Sending input events to a widget
  - Testing keyboard focus
  - Testing signals
- Qt Creator integration
- Q&A

Let me introduce myself

Professional life

- MSc Computer Science (Naples, IT)
- Game development (London, UK)
- R&D (Barcelona, ES)
- C++/Qt consultant (Barcelona, ES)

After work

- Tech blogging
- Dev projects
- Blender 3D
- Skydiving

SIGBUILD

Introduction to Qt Test

Unit test class

```
#include <QtTest>

class TestMinimal : public QObject        // test case
{
    Q_OBJECT

private slots:
    void testFoo();                       // test 1
    void testBar();                       // test 2
};
```

Unit test class - special members

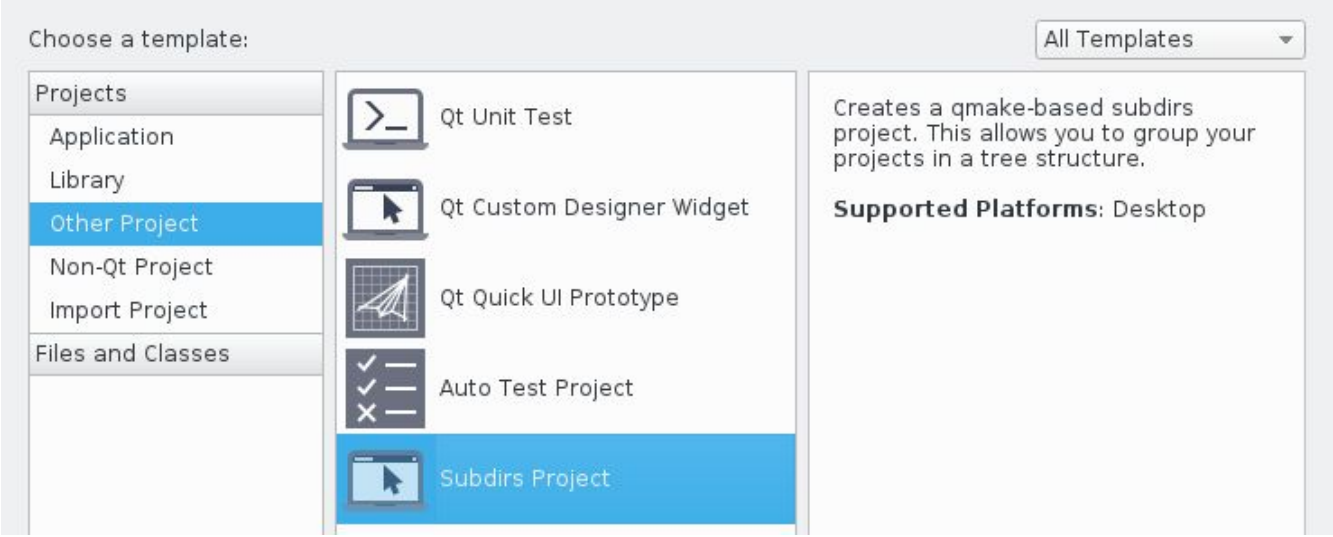
```
class TestFull : public QObject
{
    Q_OBJECT

private slots:
    void initTestCase();

    void init();

    void cleanup();

    void cleanupTestCase();
};
```

```
// example execution flow
TestFull();

    initTestCase();

        init();
            test1();
        cleanup();

        init();
            test2();
        cleanup();

    cleanupTestCase();

~TestFull();
```

Generating main

```
class TestMinimal : public QObject { ... };

// QApplication - full Qt environment
QTEST_MAIN(TestMinimal)

// QCoreApplication - no GUI, but event loop is available
QTEST_GUILESS_MAIN(TestMinimal)

// no Qt application, C++ testing only
QTEST_APPLESS_MAIN(TestMinimal)

#include "TestMinimal.moc"
```

Setting up a project (in Qt Creator)



- Subdirs project
  - Auto Test Project (basic)
  - Qt Unit Test (advanced)

Testing values and conditions

```
void TestCalculator::testConstructor()
{
    // -- default constructor --
    Calculator c1;

    QVERIFY(c1.isValid());

    // -- full constructor --
    const int A = 10;
    const int B = 2;
    Calculator c2(A, B);

    QVERIFY2(c2.getA() == A, "first operand doesn't match");
    QVERIFY2(c2.getB() == B, "second operand doesn't match");
}
```

Testing values and conditions - failures

```
QVERIFY(c1.isValid());
```

```
FAIL!  : TestCalculator::testConstructor() 'c1.isValid()' returned FALSE. ()
   Loc: [../../UnitTests/TestCalculator/TestCalculator.cpp(30)]
```

```
QVERIFY2(c2.getA() == A, "first operand doesn't match");
```

```
FAIL!  : TestCalculator::testConstructor() 'c2.GetA() == A' returned FALSE. (first
operand doesn't match)
   Loc: [../../UnitTests/TestCalculator/TestCalculator.cpp(35)]
```

Comparing values

```
void TestCalculator::testSum()
{
    // sum default
    QCOMPARE(mCalc.Sum(), A0 + B0);
}
```

```
FAIL!  : TestCalculator::testSum() Compared values are not the same
   Actual   (mCalc.Sum()): 1
   Expected (A0 + B0)    : 0
   Loc: [../../UnitTests/TestCalculator/TestCalculator.cpp(58)]
```

Useful macros

```
// fail current test
QFAIL("Fail message...");

// allow to fail next check
QEXPECT_FAIL("DataRowN", "Expected fail message", Continue);
QCOMPARE(a, b);

// skip current test
QSKIP("Skip message...");

// print warning message in logs
QWARN("Warning message");
```

More useful macros

```
// Test and compare values with timeout
QTRY_VERIFY_WITH_TIMEOUT(condition, timeout)

QTRY_VERIFY2_WITH_TIMEOUT(condition, message, timeout)

QTRY_COMPARE_WITH_TIMEOUT(actual, expected, timeout)

// QVERIFY for intercepting exceptions
QVERIFY_EXCEPTION_THROWN(expression, exceptionType)
```

Data driven testing

What, why and when

The concept

- Separate test and data
- Similar to using a database

Advantages

- Avoid repetition and multiple initializations
- Very useful to validate different input/data

Not useful when

- Tests are not focused on data
- Tests are pretty simple

Defining the data

```
void TestCalculator::testDiff_data()
{
    QTest::addColumn<int>("a");                         // col 0
    QTest::addColumn<int>("b");                         // col 1
    QTest::addColumn<int>("res");                       // col 2

    QTest::newRow("all 0") << 0 << 0 << 0;              // row 0
    QTest::newRow("same number") << 10 << 10 << 0;      // row 1
}
```

INDEX	NAME	a	b	result
0	“all 0”	0	0	0
1	“same number”	10	10	0

Writing the test function

```
void TestCalculator::testDiff()
{
    // retrieve data
    QFETCH(int, a);
    QFETCH(int, b);
    QFETCH(int, result);

    // use data
    mCalc.SetValues(a, b);

    // check results
    QCOMPARE(mCalc.Diff(), result);
}
```

GUI testing

Testing a GUI

Ideally use modular configuration

- Container project
  - Application
  - Widgets library (dynamic/shared)
  - Unit tests

Usually deeper access is required

- Provide extra public functions
- Testcase class is friend
- Visitor class is friend and used by multiple testcases
- Children navigation

Example GUI

input

action

output

- QWidget
  - 2x QLineEdit (input)
  - 2x QPushButton (action)
  - 1x QLabel (output)

Sending input events to a QWidget

```
void TestCalculator::TestClear()
{
    // write to input fields
    QTest::keyClicks(mPanel.mInputA, “hello ”);
    QTest::keyClicks(mPanel.mInputB, “world”);

    // click button CONCAT
    QTest::mouseClick(mPanel.mButtonConcat, Qt::LeftButton);
    // click button CLEAR
    QTest::mouseClick(mPanel.mButtonCancel, Qt::LeftButton);

    // check all fields are empty
    QVERIFY2(mPanel.mInputA->text().isEmpty(), "Input A not empty");
    QVERIFY2(mPanel.mInputB->text().isEmpty(), "Input B not empty");
    QVERIFY2(mPanel.mLabelRes->text().isEmpty(), "Result not empty");
}
```

Testing keyboard focus 1/2

```
void TestCalculator::TestFocusUsage()
{
    // IMPORTANT - enables focus and widget events
    QApplication::setActiveWindow(&mPanel);

    // set initial focus to input A
    mPanel.mInputA->setFocus();
    QVERIFY2(mPanel.mInputA->hasFocus(), "Input A doesn't have focus");

    // write input A
    QTest::keyClicks(QApplication::focusWidget(), “hello ”);

    // move focus to input B
    QTest::keyClick(&mPanel, Qt::Key_Tab);
    QVERIFY2(mPanel.mInputB->hasFocus(), "Input B doesn't have focus");

    // write input B
    QTest::keyClicks(QApplication::focusWidget(), ”world”);
```

Testing keyboard focus 2/2

```
    // move focus to button CONCAT
    QTest::keyClick(&mPanel, Qt::Key_Tab);
    QVERIFY2(mPanel.mButtonConcat->hasFocus(), "Button CONCAT doesn't have focus");

    // press button CONCAT
    QTest::keyClick(QApplication::focusWidget(), Qt::Key_Space);
    QCOMPARE(mPanel.mLabelRes->text(), ”hello world”);

    // move focus to button CANCEL
    QTest::keyClick(&mPanel, Qt::Key_Tab);
    QVERIFY2(mPanel.mButtonCancel->hasFocus(), "Button CANCEL doesn't have focus");

    // press button CANCEL
    QTest::keyClick(QApplication::focusWidget(), Qt::Key_Space);
    QVERIFY2(mPanel.mInputA->text().isEmpty(), "Cancel didn't work on input A");
    QVERIFY2(mPanel.mInputB->text().isEmpty(), "Cancel didn't work on input B");
    QVERIFY2(mPanel.mLabelRes->text().isEmpty(), "Cancel didn't work on res label");
}
```

Testing signals 1/2

```
void PanelConcat::ConcatData() { emit DataAvailable(mLabelRes->text()); }
void PanelConcat::CancelData() { emit DataCleared(); }

void TestPanelConcat::TestSignals()
{
    // set input
    mPanel.mInputA->setText("hello ");
    mPanel.mInputB->setText("world");

    // create spy objects - QList on steroids
    QSignalSpy spy1(&mPanel, &PanelConcat::DataAvailable);
    QSignalSpy spy2(&mPanel, &PanelConcat::DataCleared);

    // click button CONCAT and check signals received
    QTest::mouseClick(mPanel.mButtonConcat, Qt::LeftButton);

    QCOMPARE(spy1.count(), 1);
    QCOMPARE(spy2.count(), 0);
```

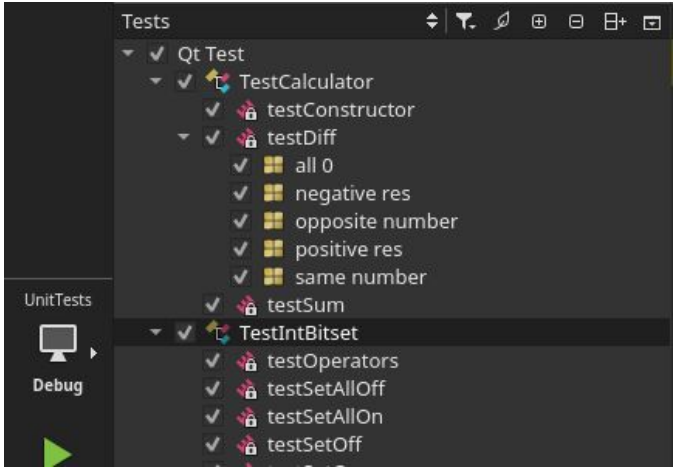
Testing signals 2/2

```
    // check parameter of the signal
    QList<QVariant> args = spy1.takeFirst();
    QCOMPARE(args.at(0).toString(), ”hello world”);

    // click button CANCEL
    QTest::mouseClick(mPanel.mButtonCancel, Qt::LeftButton);

    QCOMPARE(spy1.count(), 0);
    QCOMPARE(spy2.count(), 1);

    // check parameter of the signal
    args = spy2.takeFirst();
    QVERIFY2(args.empty(), "DataCleared signal has parameters now?!?");
}
```

Qt Creator integration

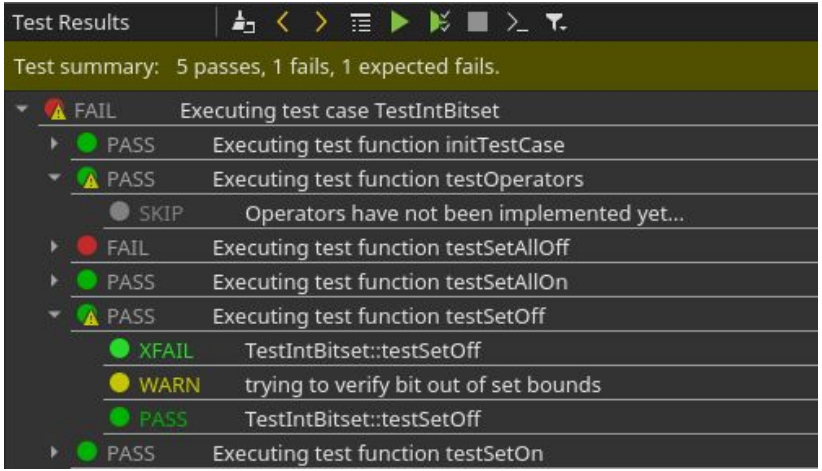
Qt Creator integration

Tests panel



- Enable/disable tests
- Run one or more tests
- Fine level of detail

Test Results panel



- See results clearly
- Jump to tests with 1 click
- Filter messages

The end...

Questions & Answers (& References)

Extended articles: http://blog.davidecoppola.com/category/qt/

Example code: https://github.com/vivaladav/BitsOfBytes

QTest namespace: https://doc.qt.io/qt-5/qtest.html

@vivaladav_3d
@vivaladav_it

in/davidecoppola/

@vivaladav