Qt Fundamentals: QTestLib

Click to add text

Software Testing

Testing your software is becoming an increasingly important dicipline to master as a developer.

- Many different types of testing strategies exist - what ever you choose some testing is better than no testing.
- Using testing gives you a number of benefits
  - You have a higher degree of confidence in you software
  - You can verify changes does not break existing software (re-factoring)
  - You see regressions faster
  - Spend more time developing new features and less time debugging
- We all make plenty of mistakes testing allows us to catch more before then end up in production code

Software Testing

- Writing testable code is not always easy
- Good concepts for testing
  - Dependency injection
  - Law of Demeter or Principle of Least Knowledge
  - Separating object creation and application logic
    - Using abstract factories
  - Use helper tools
    - Unit testing framework
    - Mock framework
    - Build bot
- Figure out what you need for you project
- If you know a lot about testing you write better code

QTestLib Overview

- The QTestLib is a unit testing framework for Qt applications and libraries (but can also be used for other stuff).
- Features:
  - Light weight - only 6000 lines of code.
  - Rapid testing - very easy and fast to create unit tests and to add new test cases.
  - Data-driven testing allows the same tests to be executed many times with different test data.
  - Basic GUI testing allows keyboard and mouse simulation.
  - Benchmarking support allows profiling critical code.

QTestLib: My First Test

QtCreator has a project wizard to create unit tests

1. Subclass QObject
2. Create a number of **private** slots (these are your test functions)
3. Use the special functions (if necessary)
   - initTestCase()
   - cleanupTestCase()
   - init()
   - cleanup()

1. Run QTest::qExec() to run the tests or use the macro QTEST_MAIN (TestClass)
2. Add QT += testlib to pro file

```
#include <QtCore>
#include <QtTest/QtTest>

class MyFirstTest : public QObject
{
   Q_OBJECT

public:
   MyFirstTest()
   { }
private Q_SLOTS:
   void testCase()
   {
      QVERIFY2(true, "Failure");
   }
};
QTEST_MAIN(MyFirstTest);

#include "myfirsttest.moc"
```

Testing Macros

Typical functions to evaluate expressions

- QVERIFY2 (condition, message)
- QVERIFY (condition)
- QWARN (message)
- QCOMPARE (actual, expected)

```
void TestQString::toUpper()
{
   QString str = "Hello";
   QCOMPARE(str.toUpper(), QString("HELLO"));
}
```

- QBENCHMARK
  - Test the performance of different implementations

```
int size = 1600;

char *a = new char[size];
char *b = new char[size];


QBENCHMARK {
   for(int i = 0; i < length; ++i)
      {
         bufOne[i] ^= bufTwo[i];
      }
}
```

One way to configure your tests

- Create a separate test project
- Use the test .pro file to pull in the .cpp/.h file you need to test