GHF automation GmbH

# Boost Test

## C++ Unit Test Framework

# Organize Tests

GHF automation GmbH

BOOST_TEST_MODULE
BOOST_TEST_SUITE
BOOST_TEST_CASE

```
#define BOOST_TEST_MODULE SampleTest
//…

BOOST_AUTO_TEST_SUITE(MyTestSuite_1)
    BOOST_AUTO_TEST_SUITE(MyTestSuite_1_1)
        BOOST_AUTO_TEST_CASE(MyTestCase_1)
        {}
        BOOST_AUTO_TEST_CASE(MyTestCase_2)
        {}
    BOOST_AUTO_TEST_SUITE_END()
    BOOST_AUTO_TEST_SUITE(MyTestSuite_1_2)
    //…
    BOOST_AUTO_TEST_SUITE_END()
BOOST_AUTO_TEST_SUITE_END()
```

# Organize Tests

GHF automation GmbH

```
L:\Projects\BoostSamples\bin>BoostSamplesAppUd --list_content
MyTestSuite_1*
    MyTestSuite_1_1*
        MyTestCase_1*
    MyTestSuite_1_2*
        MyTestCase_2*
```

Select tests to execute

| | |
|---|---|
| All | `run_test=*` |
| Branch | `run_test=/MyTestSuite_1/*` |
| Test case | `run_test=/MyTestSuite_1/MyTestSuite_1_1/MyTestCase_1` |

# Test Macros

GHF automation GmbH

- **BOOST_*Level***: validate given predicate
- **BOOST_*Level*_BITWISE_EQUAL**: show differing bits, also with different sizes
- **BOOST_*Level*_CLOSE**: comparison with max percentage deviation
- **BOOST_*Level*_CLOSE_FRACTION**: comparison with max absolute deviation
- **BOOST_*Level*_EQUAL**: check for equal, reporting both values
- **BOOST_*Level*_EQUAL_COLLECTION**: compare elements, showing mismatches
- **BOOST_*Level*_EXCEPTION**: expects exception of given type and settings thrown
- **BOOST_*Level*_GE**: greater or equal
- **BOOST_*Level*_GT**: greater than
- **BOOST_*Level*_LE**: less or equal
- **BOOST_*Level*_LT**: less than
- **BOOST_*Level*_MESSAGE**: validate and show given message in case of failure
- **BOOST_*Level*_NE**: not equal
- **BOOST_*Level*_NO_THROW**: error, if expression throws exception; catches it
- **BOOST_*Level*_PREDICATE**: use given predicate functor
- **BOOST_*Level*_SMALL**: absolute of value must not exceed given limit
- **BOOST_*Level*_THROW**: expects expression of given type
- **BOOST_TEST_*Level***: validate expression, supports optional modifiers

*Level*: WARN, CHECK, REQUIRE

# Severity Levels

GHF automation GmbH

| | |
|---|---|
| *Level* = **WARN** | issue a warning, but do not increment error counter |
| *Level* = **CHECK** | report error, but continue |
| *Level* = **REQUIRE** | report error and throw exception |

# More Macros

GHF automation GmbH

- **BOOST_IS_DEFINED**: check for preprocessor macro
- **BOOST_ERROR**: always report error, no expression to evaluate
- **BOOST_FAIL**: always report error and throw exception, no expression to evaluate
- **BOOST_TEST_CHECKPOINT**: mark position with a text, which is displayed in case of an exception
- **BOOST_TEST_PASSPOINT**: like BOOST_TEST_CHECKPOINT, but without text
- **BOOST_TEST_MESSAGE**: output given message to test log
- **BOOST_AUTO_TEST_CASE_EXPECTED_FAILURES**: number of accepted failures

# Fixtures with Classes

GHF automation GmbH

If test cases require a specific initialization and release procedure, the constructor of the fixture class can perform the initialization and the destructor the cleanup, similar to a local variable declared in the test case routine.

BOOST_GLOBAL_FIXTURE( MyFixClass )
is once created before the first test case, all test cases use the same instance

BOOST_FIXTURE_TEST_SUITE( MySuite, MyFixClass )
new instance of fixture class is created for each test case in the suite, i.e. each test case gets a **different instance** of the fixture

BOOST_FIXTURE_TEST_CASE( MyCase, MyFixClass )
instance of fixture class is created for the test case

# Fixtures with Static Functions

GHF automation GmbH

If there is no fixture class, but a setup and teardown method, use the ‚fixture' decorator.

Fixture decorator can be applied to a specific test case or a test suite.

!!! BOOST_FIXTURE_TEST_SUITE() creates a new object of the fixture class for each test case, while BOOST_AUTO_TEST_SUITE() creates an instance of the fixture class when entering it the first time and destroys that instance before leaving the test suite.

```
namespace utf = boost::unit_test;

void init () {}
void cleanup () {}

BOOST_AUTO_TEST_SUITE( MyTS, *utf::fixture(&init, &cleanup) )
BOOST_AUTO_TEST_CASE( MyTestCase )
{
}
BOOST_AUTO_TEST_SUITE_END()
```

# Template Test Cases

GHF automation GmbH

## BOOST_AUTO_TEST_CASE_TEMPLATE()

Calls test case multiple times with different types

```
typedef boost::mpl::list<bool, char, wchar_t, int> templateTypes;

BOOST_TEST_DECORATOR( *utf::label("templ") )
BOOST_AUTO_TEST_CASE_TEMPLATE( MyTC_1, T, templateTypes )
{
  BOOST_TEST_MESSAGE("Param type is '" << typeid(T).name()
    << "' with size of " << sizeof(T) << " bytes\n" );

  BOOST_TEST(true);
}
```

# Data Driven Test Cases I

GHF automation GmbH

Call test case multiple times for each value in the dataset

```
int myValues[] = { 1, 5, 10 };

BOOST_DATA_TEST_CASE( MyTC_1
, utf::data::make(myValues) + utf::data::xrange(20, 100, 10 )
, myVar )
{
  BOOST_TEST_MESSAGE("Param value " << myVar );

  BOOST_TEST(true);
}
```

**utf::data::make(myValues)**: dataset from C-array
**utf::data::xrange(20, 100, 10)**: starting with 20 all values in steps of 10 up to, but excluding 100

# Data Driven Test Cases II

GHF automation GmbH

Call test case multiple times with pairs or more values from given datasets

```
int myValues[] = { 1, 5, 10 };

BOOST_DATA_TEST_CASE( MyTC_2
, utf::data::xrange(3) ^ utf::data::make(myValues)
, myVar1, myVar2 ) // both datasets must have same size
{
  BOOST_TEST_MESSAGE("Param values " << myVar1 << " and " <<
    myVar2 );

  BOOST_TEST(true);
}
```

**utf::data::xrange(3):** values 0, 1 and 2
**utf::data::xrange(3) ^ utf::data::make(myValues):** value pairs (0, 1), (1, 5), (2, 10)

# Data Driven Test Cases III

GHF automation GmbH

Call test case multiple times with combinations of values from given datasets

```
int myValues[] = { 1, 5, 10 };

BOOST_DATA_TEST_CASE(MyTC_2a, utf::data::xrange(2)
* utf::data::xrange(3, 5) * utf::data::make(myValues)
, myVar1, myVar2, myVar3)
{
  BOOST_TEST_MESSAGE("\n\nParam values " << myVar1 << " and " <<
    myVar2 << " and " << myVar3);

  BOOST_TEST(true);
}
```

**utf::data::xrange(2) * utf::data::xrange(3,5) * utf::data::make(myValues):**
values (0,3,1), (0,3, 5), (0,3,10), (0,4,1), (0,4,5), (0,4,10), (1,3,1), (1,3, 5), (1,3,10),
(1,4,1), (1,4,5), (1,4,10)

# Test Decorators

GHF automation GmbH

Add to test suite or test case as parameter after name, e.g.

```
BOOST_AUTO_TEST_CASE(MyTC_1, *utf::label("myLabel") )
```

Or use BOOST_TEST_DECORATOR() macro, e.g.

```
BOOST_TEST_DECORATOR( *utf::label("myLabel") )
```

Some decorators

**label**: used to select test cases with "--run_test=@myLabel"

**description**: text display with "--list_content" command

**disabled/enabled**: disable or enable test cases

**fixture**: provide a static init and cleanup function, not class

**depends_on**: specify prerequisite test case

**enable_if**: enable depending on compilation constant

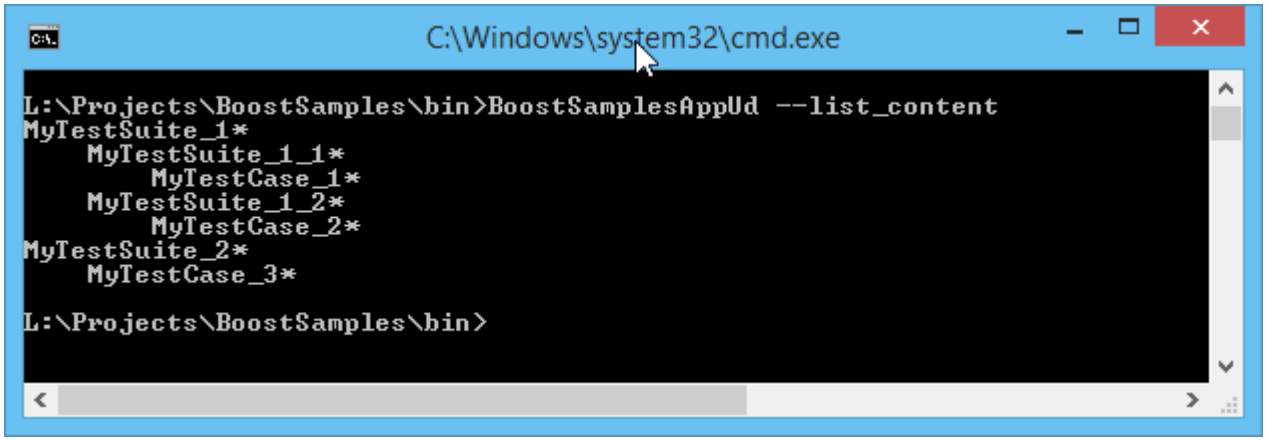
# How to execute tests

GHF automation GmbH

1. Command line
2. Boost Test GUI (https://boosttestui.wordpress.com/boost-test/)
3. Visual Studio Boost Unit Test Adapter (via VS extension manager)

# Command Line UI

GHF automation GmbH

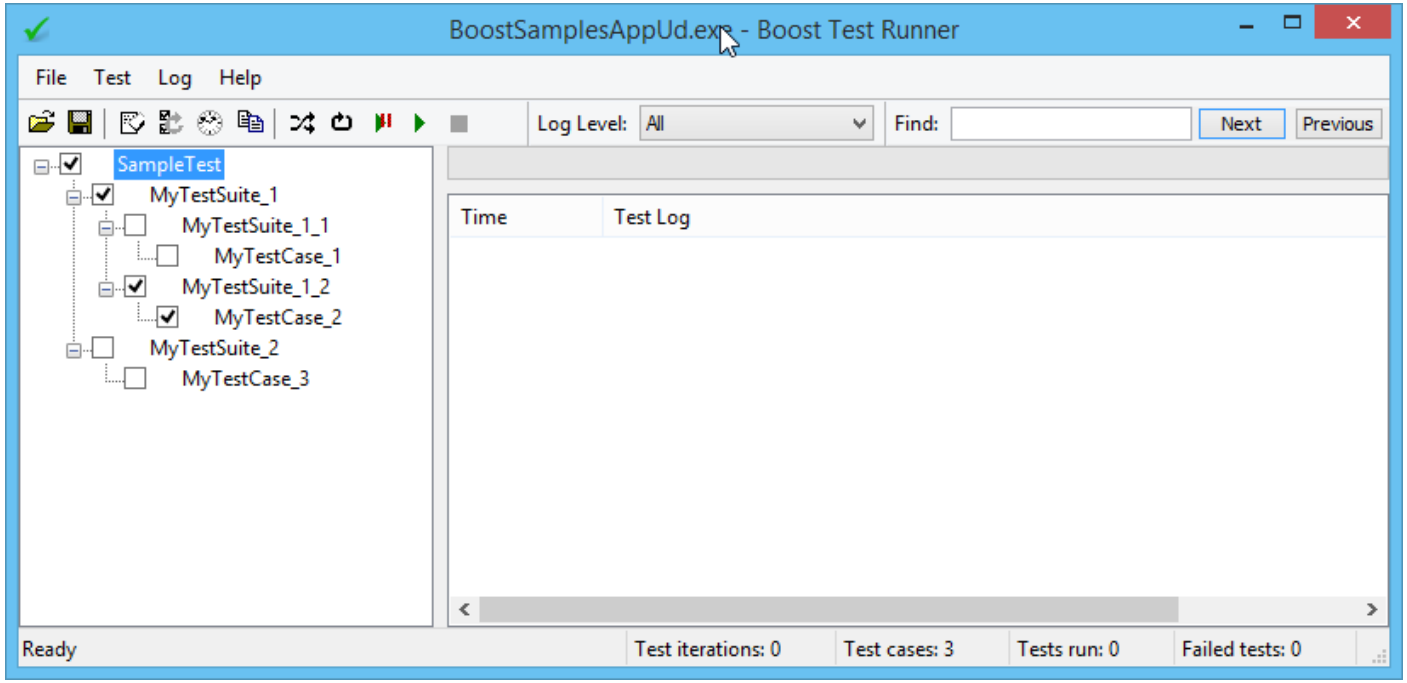



## Important parameters

| | |
|---|---|
| --list_content | List test tree with descriptions |
| --list_labels | List labels for selection |
| --run_test=@*label* | Execute test cases with specified label |
| --help | Show available options |
| --log_level=*level* | Select detail of output (e.g. `all`, `test_suite`, `message`, `error`) |

# Boost Test UI

GHF automation GmbH

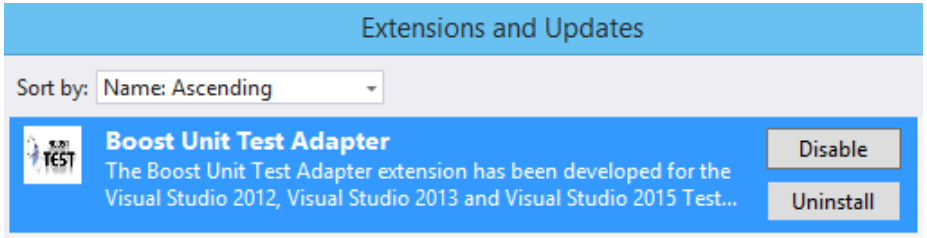



- ✓ Supports also other UTFs, e.g. Google Test
- ✓ Open Source (https://github.com/djeedjay/BoostTestUi)

– Compilation with `unit_test_gui.hpp` instead of `unit_test.hpp`

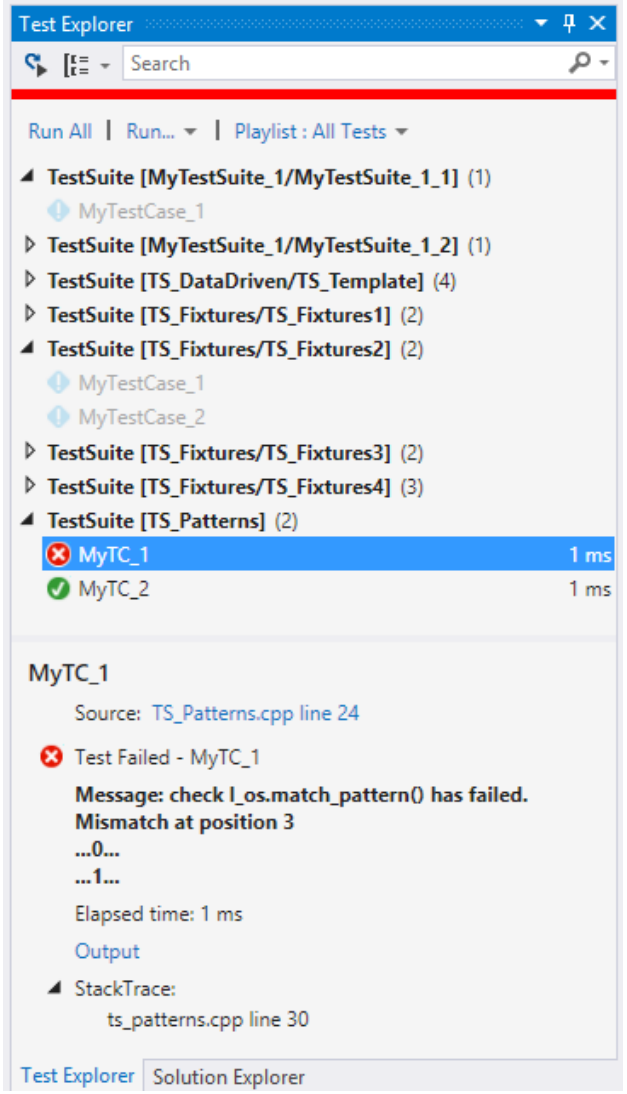
# Visual Studio Unit Test Adapter

GHF automation GmbH




- ✓ Installation as Visual Studio extension
- ✓ Coupling with code coverage
- ✓ Detailed result display
- ✓ Link to source code location

– New data driven test cases not yet supported



# The End

GHF automation GmbH

Thank you for your attention!