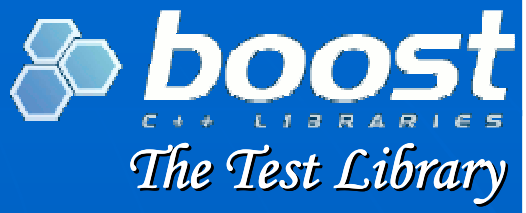
Boost Test Library

The Unit Test Framework Tutorial

Gennadiy Rozental
rogeeff@gmail.com

Introduction



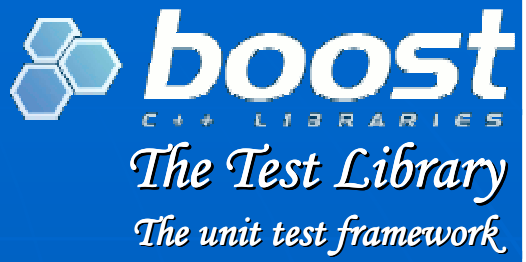
- The Boost Test Library: matched set of components
  - The Execution Monitor
  - The Program Execution Monitor
  - The Minimal Testing Facility
  - The Unit Test Framework

The Test Library

The unit test framework overview

The Unit Test Framework

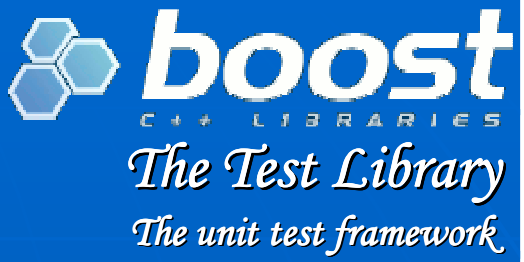
Introduction



- Three questions about unit testing

The Unit Test Framework

Introduction



- Three questions about unit testing
  - **Why?**

The Unit Test Framework

Introduction

boost C++ LIBRARIES
The Test Library
The unit test framework

- Three questions about unit testing
  - Why?
  - **What?**
    - General answer: Test everything that could possibly break
    - Practical answer: check ALL expectations
      - Interfaces expectations
      - "positive" and "negative" functionality
      - Performance and memory expectation
      - Test interactions with external components

The Unit Test Framework

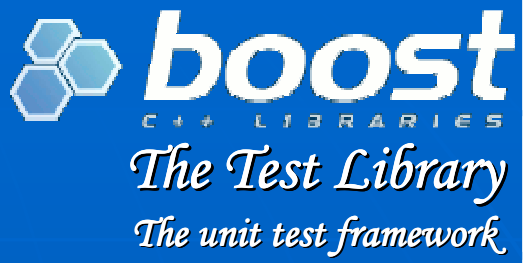
Introduction

boost C++ LIBRARIES
The Test Library
The unit test framework

- Three questions about unit testing
  - Why?
  - What?
    - General answer: Test everything that could possibly break
    - Practical answer: check ALL expectations
      - Interfaces expectations
      - "positive" and "negative" functionality
      - Performance and memory expectation
      - Test interactions with external components
  - **How?**

The Unit Test Framework

Overview



- Simplify writing test cases by using various testing tools.
- Organize test cases into a test tree.
- Relieve you from messy error detection, reporting duties and framework runtime parameters processing

Jump start example 1

boost C++ LIBRARIES
The Test Library
The unit test framework

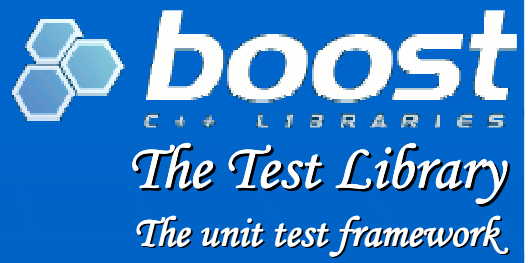
```
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

void free_test_function()
{
    int* p = (int*)123;
    BOOST_CHECK( *p == 0 );
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int, char* [] ) {
    framework::master_test_suite().p_name.value = “example";

    framework::master_test_suite().add( BOOST_TEST_CASE( &free_test_function ) );

    return 0;
}

//____________________________________________________________________________//
```

Jump start example 2

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________________________________________//

BOOST_AUTO_TEST_SUITE( my_suite1 );

BOOST_AUTO_TEST_CASE( my_test1 )
{
    BOOST_CHECK( 2 == 1 );
}

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( my_test2 )
{
    int i = 0;

    BOOST_CHECK_EQUAL( i, 2 );

    BOOST_CHECK_EQUAL( i, 0 );
}

//____________________________________________________________________________//

BOOST_AUTO_TEST_SUITE_END();
```

Usage variants



- The static library variant
- The single-header variant
- The dynamic library variant
- The external test runner variant

Unit testing terms

boost C++ LIBRARIES
The Test Library
The unit test framework

- **Test module** - a single binary that performs the test
  - Single-file
  - Multi-file

Unit testing terms

boost C++ LIBRARIES
The Test Library
The unit test framework

- Test module
  - **Test initialization** – a part of a test module that is responsible for the test preparation

Unit testing terms

boost C++ LIBRARIES
The Test Library
The unit test framework

- Test module
  - Test initialization
  - **Test body** - a part of a test module that actually performs the test

Unit testing terms

boost C++ LIBRARIES
The Test Library
The unit test framework

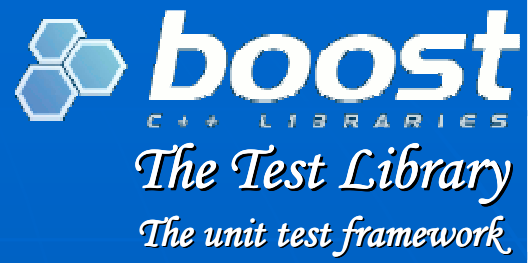
- Test module
  - Test initialization
  - Test body
  - **Test cleanup** - of test module that is responsible for cleanup operations

Unit testing terms

boost C++ LIBRARIES
The Test Library
The unit test framework

- Test module
  - Test initialization
  - Test body
  - Test cleanup
  - **Test runner** - an "executive manager" that runs the show

Test module initialization



- **Initialization tasks**

Test module initialization

- Initialization tasks
- **Initialization function**

Test module initialization

- Initialization tasks
- Initialization function
- **Initialization function specification**
  - Original
  - Alternative

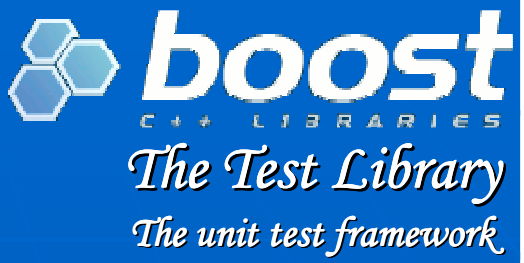
```
boost::unit_test::test_suite*
init_unit_test_suite( int argc, char* argv[] );


bool init_unit_test();
```

Test module initialization

boost C++ LIBRARIES
The Test Library
The unit test framework

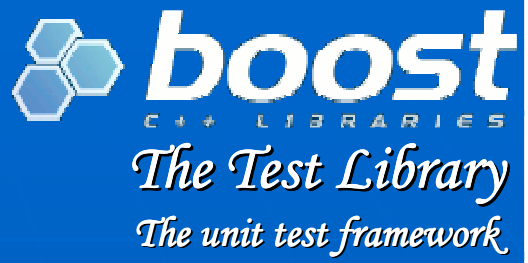
- Initialization tasks
- Initialization function and global fixtures
- Initialization function specification
- **Supplied test runners requirements**

Test module initialization



- Initialization tasks
- Initialization function and global fixtures
- Initialization function specification
- Supplied test runners requirements
- **Automated generation**
  - BOOST_TEST_MAIN/BOOST_TEST_MODULE

Initialization function examples



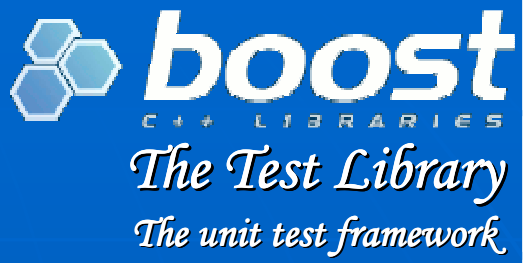

```
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;
…
test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
    framework::master_test_suite().add( BOOST_TEST_CASE( &test_case1 ) );
    return 0;
}
```

```
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;
…
bool
init_unit_test()
{
    framework::master_test_suite().add( BOOST_TEST_CASE( &test_case1 ) );

    return true;
}
```

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE( test_case1 )
{
    …
}
```

Test module initialization

Initialization function



- Result value
- Initialization error
- Command line arguments

The Test Library

Test organization

Test organization

boost C++ LIBRARIES
The Test Library
The unit test framework

- Why not single test function?

```
#include <assert.h>
…
int
main()
{
    …
    assert( condition1 );
    …
    if( condition2 )
      std::cout << “error\n”;
    …
    assert( condition3 )
    …

    return 0;
}
```

Test organization

boost C++ LIBRARIES
The Test Library
The unit test framework

- Why not single test function?
- **Test case** - an independently monitored function within a test module

Test organization

boost C++ LIBRARIES
The Test Library
The unit test framework

- Why not single test function?
- Test case - an independently monitored function within a test module
- **Test suite** - a container for one or more test cases

Test organization

boost C++ LIBRARIES
The Test Library
The unit test framework

- Why not single test function?
- Test case - an independently monitored function within a test module
- Test case construction design
- Test suite - a container for one or more test cases
- **Test unit** - a collective name when referred to either test suite or test case

Test organization

boost C++ LIBRARIES
The Test Library
The unit test framework

- Why not single test function?
- Test case - an independently monitored function within a test module
- Test case construction design
- Test suite - a container for one or more test cases
- Test unit - a collective name when referred to either test suite or test case
- **Test tree** - a hierarchical structure of test units

Test case creation

boost C++ LIBRARIES
The Test Library
The unit test framework

- Test case can be created based on any function with matching signature

Test case creation

boost C++ LIBRARIES
The Test Library
The unit test framework

- Test case can be created based on any function with matching signature
- Supported test case signatures
  - Nullary function

Test case creation

boost C++ LIBRARIES
The Test Library
The unit test framework

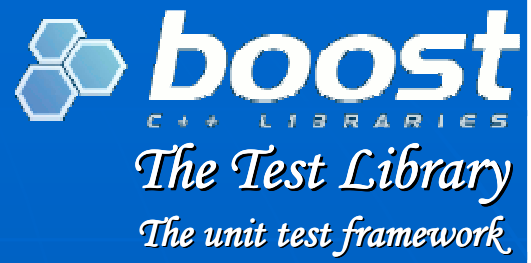
- Test case can be created based on any function with matching signature
- Supported test case signatures
  - Nullary function
  - Unary function

Test case creation

boost C++ LIBRARIES
The Test Library
The unit test framework

- Test case can be created based on any function with matching signature
- Supported test case signatures
  - Nullary function
  - Unary function
  - Nullary function template

Nullary function based test case



- Macro BOOST_TEST_CASE

BOOST_TEST_CASE(test_function)

Free function example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

void free_test_function()
{
    BOOST_CHECK( true /* test assertion */ );
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     framework::master_test_suite().
        add( BOOST_TEST_CASE( &free_test_function ) );

     return 0;
}

//____________________________________________________________________________//
```

Free function example

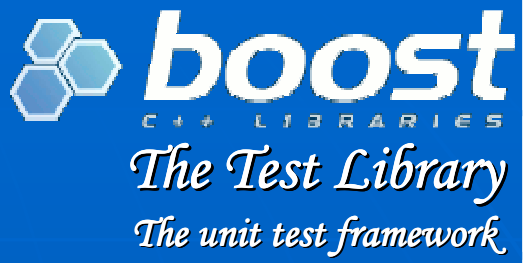
boost C++ LIBRARIES
The Test Library
The unit test framework

```
> example
Running 1 test case...

*** No errors detected
```

```
#include <boost/test/unit
using namespace boost::un

//____________________

void free_test_function()
{
    BOOST_CHECK( true /* test assertion */ );
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     framework::master_test_suite().
        add( BOOST_TEST_CASE( &free_test_function ) );

     return 0;
}

//____________________________________________________________________________//
```

Method of a class example

using global class instance



This example is using the single-header variant of the Unit Test Framework.

```
#include <boost/test/included/unit_test.hpp>
#include <boost/bind.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

class test_class {
public:
    void test_method()
    {
        BOOST_CHECK( false /* test assertion */ );
    }
} tester;

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &test_class::test_method, &tester ) ));

     return 0;
}

//____________________________________________________________________________//
```

Method of a class example

using global class instance

boost C++ LIBRARIES
The Test Library
The unit test framework

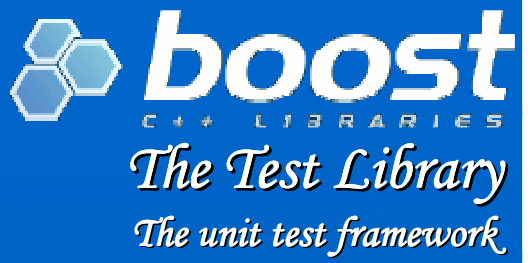
This example is using the single-header variant of the Unit Test Fra

```
> example
Running 1 test case...
test.cpp(11): error in "boost::bind( &test_class::test_method, &tester )": check
      false failed

*** 1 failure detected in test suite "Master Test Suite"
```

```
#include <boost/test
#include <boost/bind
using namespace boos

//____________________

class test_class {
public:
    void test_method()
    {
        BOOST_CHECK( false /* test assertion */ );
    }
} tester;

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &test_class::test_method, &tester ) ));

     return 0;
}

//____________________________________________________________________________//
```

Method of a class example

using shared class instance




```
#include <boost/test/unit_test.hpp>
#include <boost/bind.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

class test_class {
public:
    void test_method1()
    {
        BOOST_CHECK( true /* test assertion */ );
    }
    void test_method2()
    {
        BOOST_CHECK( false /* test assertion */ );
    }
};

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     boost::shared_ptr<test_class> tester( new test_class );

     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &test_class::test_method1, tester )));
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &test_class::test_method2, tester )));
     return 0;
}
```

Method of a class example

using shared class instance

boost C++ LIBRARIES

The Test Library

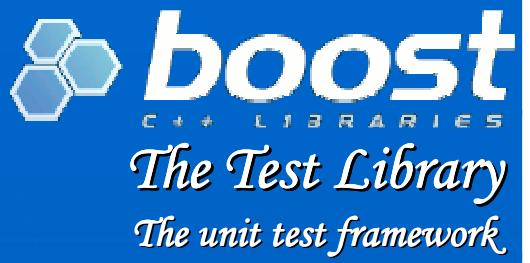
The unit test framework

```
#include <boost/test
#include <boost/bind
using namespace boos

//____________________

class test_class {
public:
    void test_method
    {
        BOOST_CHECK( true /* test assertion */ );
    }
    void test_method2()
    {
        BOOST_CHECK( false /* test assertion */ );
    }
};

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     boost::shared_ptr<test_class> tester( new test_class );

     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &test_class::test_method1, tester )));
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &test_class::test_method2, tester )));
     return 0;
}
```

```
> example
Running 2 test cases...
test.cpp(15): error in "boost::bind( &test_class::test_method2, tester )": check
      false failed

*** 1 failure detected in test suite "Master Test Suite"
```

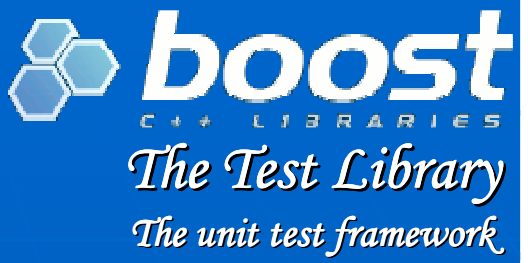
Method of a class example

free function stack class instance




```
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

void free_test_function()
{
    test_class inst;

    inst.test_method();
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     framework::master_test_suite().add( BOOST_TEST_CASE( &free_test_function ));

     return 0;
}

//____________________________________________________________________________//
```

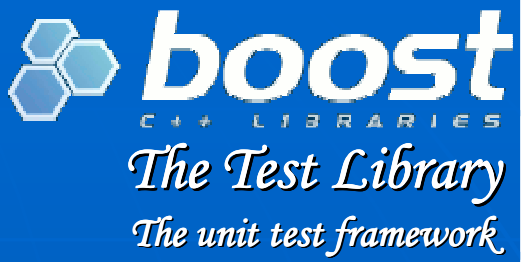
Functions with bound arguments example



This example is using the dynamic library variant of the Unit Test Framework.

```
#define BOOST_TEST_DYN_LINK
#include <boost/test/unit_test.hpp>
#include <boost/bind.hpp>
using namespace boost::unit_test;
void free_test_function( int i, int j )
{
    BOOST_CHECK( true /* test assertion */ );
}

bool
init_function() {
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &free_test_function, 1, 1 ) ) );
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &free_test_function, 1, 2 ) ) );
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &free_test_function, 2, 1 ) ) );

     return true;
}

int
main( int argc, char* argv[] )
{
    return ::boost::unit_test::unit_test_main( &init_function, argc, argv );
}
```

Functions with bound arguments example

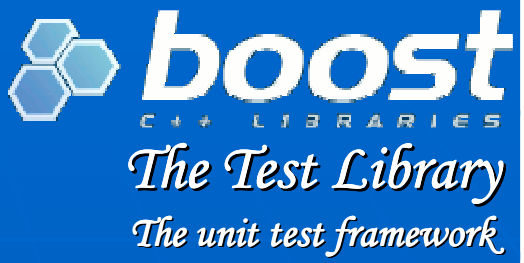


This example is using the dynamic library variant of the Unit Test Framework

```
> example
Running 3 test cases...

*** No errors detected
```

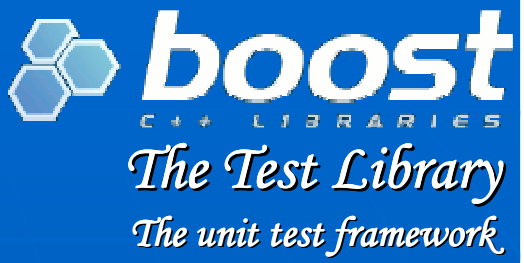
```
#define BOOST_TEST_
#include <boost/tes
#include <boost/bin
using namespace boo
void free_test_func
{
    BOOST_CHECK( true /* test assertion */ );
}

bool
init_function() {
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &free_test_function, 1, 1 ) ) );
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &free_test_function, 1, 2 ) ) );
     framework::master_test_suite().
        add( BOOST_TEST_CASE( boost::bind( &free_test_function, 2, 1 ) ) );

     return true;
}

int
main( int argc, char* argv[] )
{
    return ::boost::unit_test::unit_test_main( &init_function, argc, argv );
}
```

Automated registration



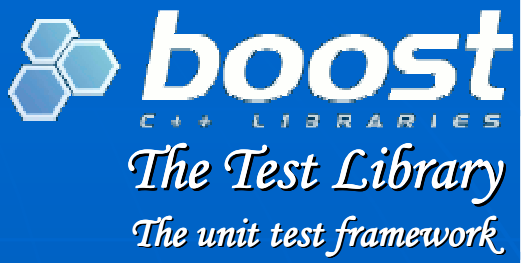
- Manual registration problem: implementation/registration points are remote
  - Forgotten test cases
  - Tedious work

Automated registration



- Manual registration problem: implementation/registration points are remote
  - Forgotten test cases
  - Tedious work
- Automated (in place) registration design and it's limitations

Nullary function based test case with automated registration



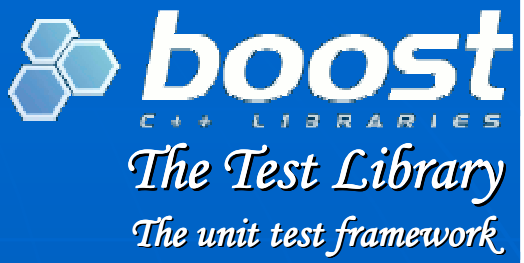
- Macro BOOST_AUTO_TEST_CASE

  BOOST_AUTO_TEST_CASE(test_case_name)

- Empty test module initialization function

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( free_test_function )
{
    BOOST_CHECK( true /* test assertion */ );
}

//____________________________________________________________________________//
```

Nullary function based test case with automated registration



- Macro BOOST_AUTO_TEST_CASE

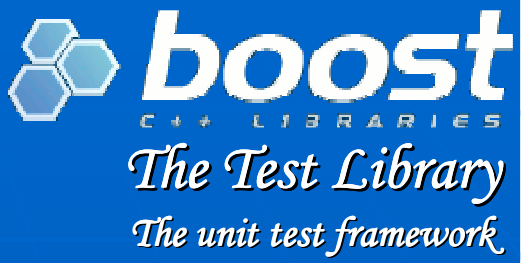
  BOOST_

- Empty

```
> example --log_level=test_suite
Running 1 test case...
Entering test suite "example"
Entering test case "free_test_function"
Leaving test case "free_test_function"; testing time: 16ms
Leaving test suite "example"

*** No errors detected
```

```
#define BOOST_TEST_M
#include <boost/test

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( free_test_function )
{
    BOOST_CHECK( true /* test assertion */ );
}

//____________________________________________________________________________//
```

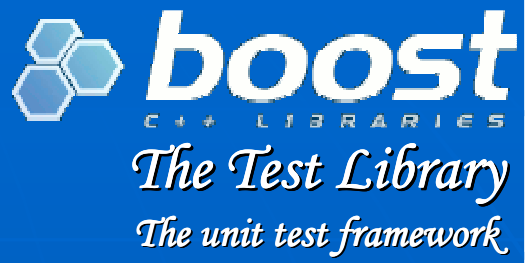
Unary function based test case



Some tests are required to be repeated for a series of different input parameters.

```
void single_test( int i )
{
    BOOST_CHECK( /* test assertion */ );
}

//____________________________________________________________________________//

void combined_test()
{
    int params[] = { 1, 2, 3, 4, 5 };

    std::for_each( params, params+5, &single_test );
}

//____________________________________________________________________________//
```

Unary function based test case

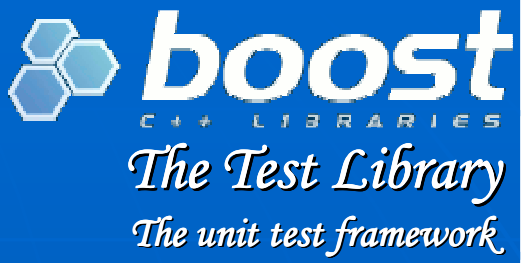


- Macro BOOST_PARAM_TEST_CASE

```
BOOST_PARAM_TEST_CASE(
    test_function,
    params_begin,
    params_end )
```

- Parameter value is stored along
- Require header parameterized_test.hpp

Unary function based test case




```
#include <boost/test/unit_test.hpp>
#include <boost/test/parameterized_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

void free_test_function( int i )
{
    BOOST_CHECK( i < 4 /* test assertion */ );
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     int params[] = { 1, 2, 3, 4, 5 };

     framework::master_test_suite().
        add( BOOST_PARAM_TEST_CASE( &free_test_function, params, params+5 ) );

     return 0;
}

//____________________________________________________________________________//
```

Unary function based test case

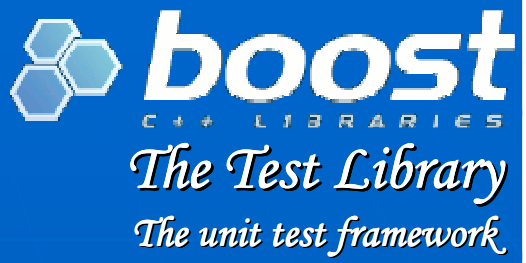
The Test Library

The unit test framework

```
#include <boost/test
#include <boost/test
using namespace boos

//____________________

void free_test_funct
{
    BOOST_CHECK( i <
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     int params[] = { 1, 2, 3, 4, 5 };

     framework::master_test_suite().
        add( BOOST_PARAM_TEST_CASE( &free_test_function, params, params+5 ) );

     return 0;
}

//____________________________________________________________________________//
```

```
> example
Running 5 test cases...
test.cpp(9): error in "free_test_function": check i < 4 failed
test.cpp(9): error in "free_test_function": check i < 4 failed

*** 2 failures detected in test suite "Master Test Suite"
```

Unary function based test case

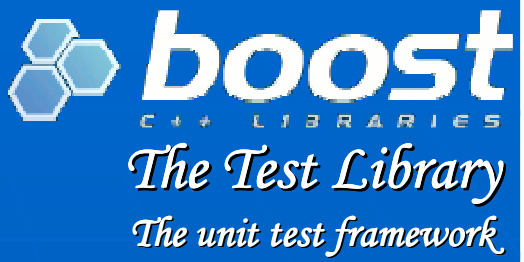


This example is using the alternative initialization function specification and global class instance.

```
#define BOOST_TEST_ALTERNATIVE_INIT_API
#include <boost/test/unit_test.hpp>
#include <boost/test/floating_point_comparison.hpp>
#include <boost/test/parameterized_test.hpp>
#include <boost/bind.hpp>
using namespace boost::unit_test;
using namespace boost;

//____________________________________________________//

class test_class {
public:
    void test_method( double const& d )
    {
        BOOST_CHECK_CLOSE( d * 100,
                           (double)(int)(d*100),
                           0.01 );
    }
} tester;

//____________________________________________________//
```

```
bool init_unit_test()
{
     double params[] = {
         1., 1.1, 1.01, 1.001, 1.0001 };

     callback1<double> tm =
       bind( &test_class::test_method, &tester, _1);

     framework::master_test_suite().
        add( BOOST_PARAM_TEST_CASE(
                 tm, params, params+5 ) );

     return true;
}

//____________________________________________________//
```

Unary function based test case



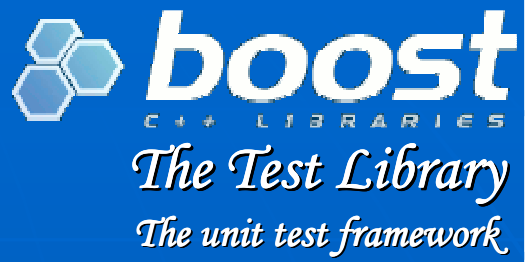
This example
function speci

```
> example
Running 5 test cases...
test.cpp(14): error in "tm": difference between d * 100{100.1} and
      (double)(int)(d*100){100} exceeds 0.01%
test.cpp(14): error in "tm": difference between d * 100{100.01} and
      (double)(int)(d*100){100} exceeds 0.01%

*** 2 failures detected in test suite "Master Test Suite"
```

```
#define BOOST_TEST_ALTE
#include <boost/test/un
#include <boost/test/fl
#include <boost/test/pa
#include <boost/bind.hp
using namespace boost::
using namespace boost;

//____________________________________________//

class test_class {
public:
    void test_method( double const& d )
    {
        BOOST_CHECK_CLOSE( d * 100,
                           (double)(int)(d*100),
                           0.01 );
    }
} tester;

//____________________________________________//
```

```
    framework::master_test_suite().
       add( BOOST_PARAM_TEST_CASE(
                tm, params, params+5 ) );

    return true;
}

//____________________________________________//
```

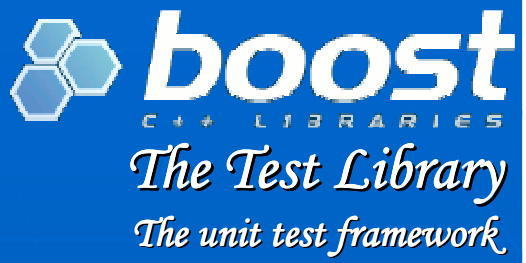
Test case template



To test a template based component it's frequently necessary to perform the same set of checks for a component instantiated with different template parameters.

```
template <typename T>
void single_test()
{
    BOOST_CHECK( /* test assertion */ );
}

//____________________________________________________________________________//

void combined_test()
{
    single_test<int>();
    single_test<float>();
    single_test<unsigned char>();
}

//____________________________________________________________________________//
```

Test case template
Manually registered



- Macro BOOST_TEST_CASE_TEMPLATE_FUNCTION

```
BOOST_TEST_CASE_TEMPLATE_FUNCTION(
    test_case_name,
    type_name )
```

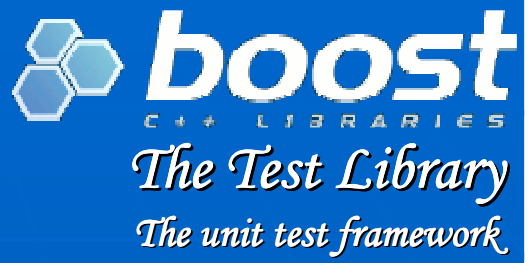
- Macro BOOST_TEST_CASE_TEMPLATE

```
BOOST_TEST_CASE_TEMPLATE(
    test_case_name,
    collection_of_types )
```

- Require header test_case_template.hpp

Test case template

Manually registered




```
#include <boost/test/unit_test.hpp>
#include <boost/test/test_case_template.hpp>
#include <boost/mpl/list.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

BOOST_TEST_CASE_TEMPLATE_FUNCTION( my_test, T )
{
     BOOST_CHECK_EQUAL( sizeof(T), 4 );
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     typedef boost::mpl::list<int,long,unsigned char> test_types;

     framework::master_test_suite().
        add( BOOST_TEST_CASE_TEMPLATE( my_test, test_types ) );

     return 0;
}

//____________________________________________________________________________//
```

Test case template

Manually registered

boost C++ LIBRARIES

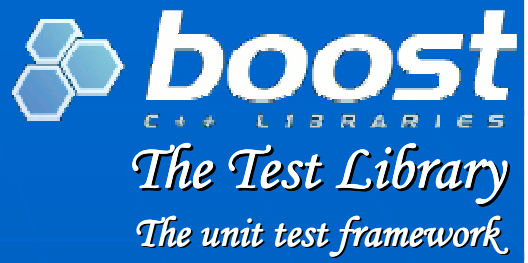
The Test Library

The unit test framework

```
> example
Running 3 test cases...
test.cpp(10): error in "my_test<unsigned char>": check sizeof(T) == 4 failed [1 != 4]

*** 1 failure detected in test suite "Master Test Suite"
```

```
#include <boos
#include <boos
#include <boos
using namespac

//____________

BOOST_TEST_CAS
{
     BOOST_CHECK_EQUAL( sizeof(T), 4 );
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     typedef boost::mpl::list<int,long,unsigned char> test_types;

     framework::master_test_suite().
        add( BOOST_TEST_CASE_TEMPLATE( my_test, test_types ) );

     return 0;
}

//____________________________________________________________________________//
```

Test case template
Automated registration



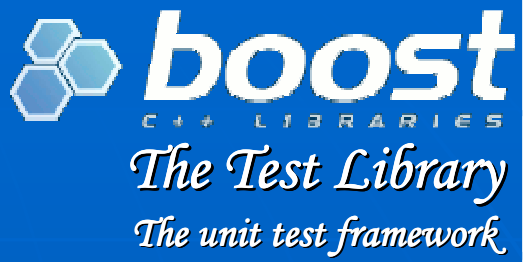
- Macro BOOST_AUTO_TEST_CASE_TEMPLATE

```
BOOST_AUTO_TEST_CASE_TEMPLATE(
    test_case_name,
    type_name,
    type_collection )
```

- Require header test_case_template.hpp

Test case template

Automated registration




```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>
#include <boost/test/test_case_template.hpp>
#include <boost/mpl/list.hpp>

//____________________________________________________________________________//

typedef boost::mpl::list<int,long,unsigned char> test_types;

BOOST_AUTO_TEST_CASE_TEMPLATE( my_test, T, test_types )
{
     BOOST_CHECK_EQUAL( sizeof(T), (unsigned)4 )
}

//____________________________________________________________________________//
```

Test case template

Automated registration

boost C++ LIBRARIES

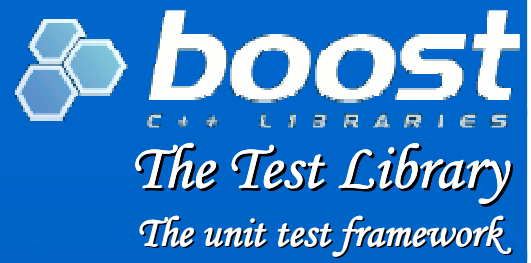
The Test Library

The unit test framework

```
#define BOOST_
#include <boos
#include <boos
#include <boos

//_____________

typedef boost:

BOOST_AUTO_TES
{
     BOOST_CHECK_EQUAL( sizeof(T), (unsigned)4 )
}

//____________________________________________________________________________//
```

```
> example
Running 3 test cases...
test.cpp(12): error in "my_test<unsigned char>": check sizeof(T) == 4 failed [1 != 4]

*** 1 failure detected in test suite "example"
```

Test Suite



- **Branch of test tree**

Test Suite

boost C++ LIBRARIES
The Test Library
The unit test framework

- Branch of test tree
- **Why?**
  - To group test cases per subsystems of the unit being tested
  - To share test case setup/cleanup code
  - To run selected group of test cases only
  - To see test report split by groups of test cases
  - To skip groups of test cases based on the result of another test unit in a test tree

Test Suite

boost C++ LIBRARIES
The Test Library
The unit test framework

- Branch of test tree
- Why?
- **Master test suite**

Test Suite

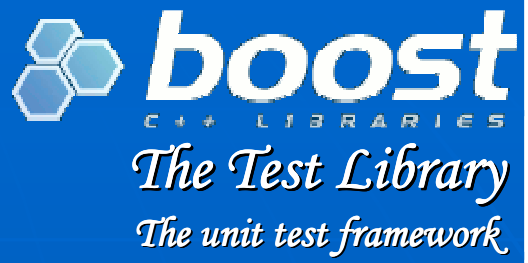
boost C++ LIBRARIES
The Test Library
The unit test framework

- Branch of test tree
- Why?
- Master test suite
- **Test unit registration interface**

```
void
test_suite::add( test_unit*  tc,
                 counter_t   expected_failures = 0,
                 int         timeout = 0 );
```

Test Suite

Manual creation and registration



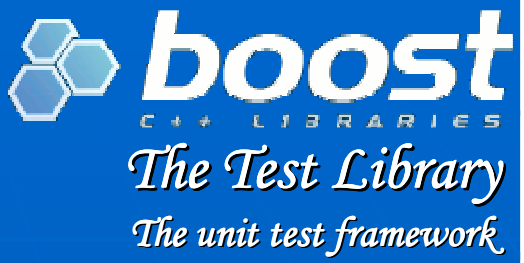
- BOOST_TEST_SUITE

BOOST_TEST_SUITE(test_suite_name)

- Can’t be created on stack

Test Suite

Manual creation and registration



This example is using the single-header variant of the Unit Test Framework.

```
#include <boost/test/included/unit_test.hpp>
using namespace boost::unit_test;

void test_case1() { /* … */ }
void test_case2() { /* … */ }
void test_case3() { /* … */ }
void test_case4() { /* … */ }

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     test_suite* ts1 = BOOST_TEST_SUITE( "test_suite1" );
     ts1->add( BOOST_TEST_CASE( &test_case1 ) );
     ts1->add( BOOST_TEST_CASE( &test_case2 ) );

     test_suite* ts2 = BOOST_TEST_SUITE( "test_suite2" );
     ts2->add( BOOST_TEST_CASE( &test_case3 ) );
     ts2->add( BOOST_TEST_CASE( &test_case4 ) );

     framework::master_test_suite().add( ts1 );
     framework::master_test_suite().add( ts2 );

     return 0;
}
//____________________________________________________________________________//
```

Test Suite

Manual creation and registration

This example is using the single-header variant of the Unit Test Framewor

```
#include <boost/test/included
using namespace boost::unit_t

void test_case1() { /* … */ }
void test_case2() { /* … */ }
void test_case3() { /* … */ }
void test_case4() { /* … */ }

test_suite*
init_unit_test_suite( int arg
{
     test_suite* ts1 = BOOST_
     ts1->add( BOOST_TEST_CAS
     ts1->add( BOOST_TEST_CAS

     test_suite* ts2 = BOOST_
     ts2->add( BOOST_TEST_CAS
     ts2->add( BOOST_TEST_CAS

     framework::master_test_s
     framework::master_test_s

     return 0;
}
//____________________________________________________________________________//
```

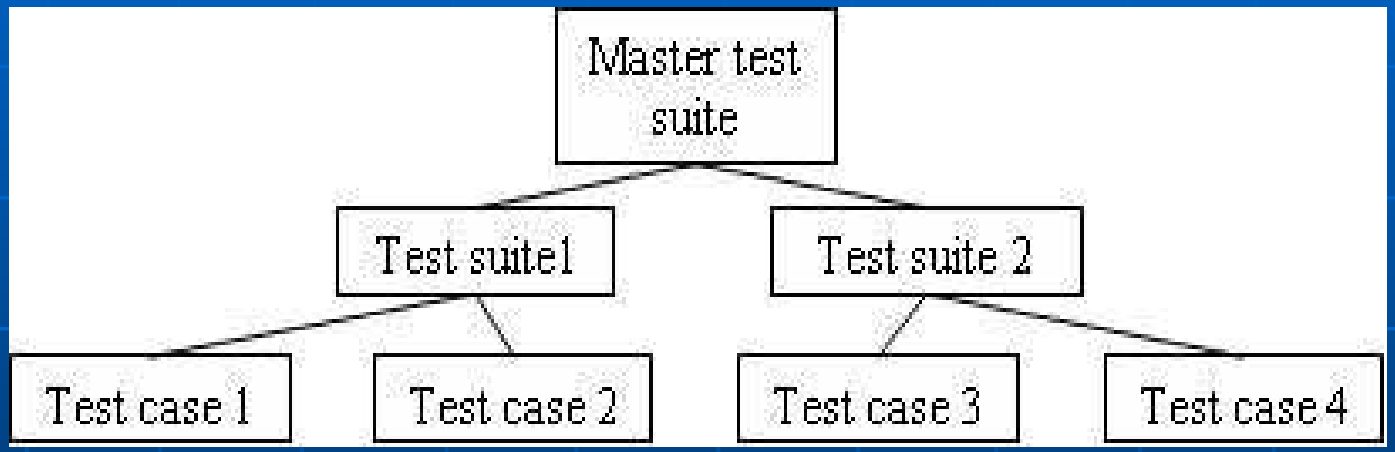
```
> example --log_level=test_suite
Running 4 test cases...
Entering test suite "Master Test Suite"
Entering test suite "test_suite1"
Entering test case "test_case1"
Leaving test case "test_case1"
Entering test case "test_case2"
Leaving test case "test_case2"
Leaving test suite "test_suite1"
Entering test suite "test_suite2"
Entering test case "test_case3"
Leaving test case "test_case3"
Entering test case "test_case4"
Leaving test case "test_case4"
Leaving test suite "test_suite2"
Leaving test suite "Master Test Suite"

*** No errors detected
```

Test Suite

Manual creation and registration

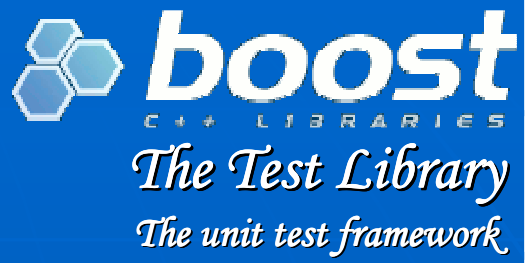
boost C++ LIBRARIES
The Test Library
The unit test framework

Constructed test tree



Test Suite

Automated creation and registration



- BOOST_AUTO_TEST_SUITE

BOOST_AUTO_TEST_SUITE(test_suite_name)

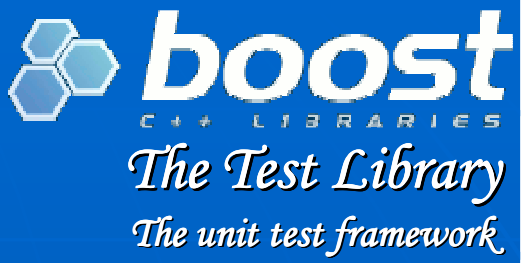
- BOOST_AUTO_TEST_SUITE_END

BOOST_AUTO_TEST_SUITE_END()

- No custom test suites

Test Suite

Automated creation and registration




```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

BOOST_AUTO_TEST_CASE( test_case2 )
{
    BOOST_REQUIRE_EQUAL( 1, 2 );
    BOOST_FAIL( "Should never reach this line" );
}

BOOST_AUTO_TEST_SUITE_END()
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( true );
}

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECK( false );
}

BOOST_AUTO_TEST_SUITE_END()
```

Test Suite

Automated creation and registration

boost C++ LIBRARIES

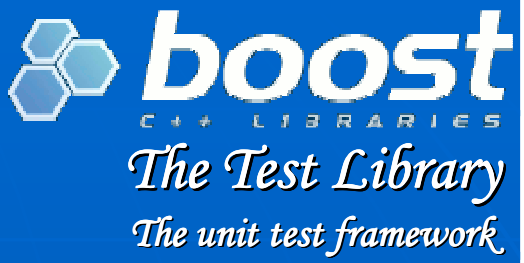
The Test Library

The unit test framework

```
#define BOOST_TEST_M
#include <boost/test

BOOST_AUTO_TEST_SUIT

BOOST_AUTO_TEST_CASE
{
    BOOST_WARN( size
}

BOOST_AUTO_TEST_CASE( test_case2 )
{
    BOOST_REQUIRE_EQUAL( 1, 2 );
    BOOST_FAIL( "Should never reach this line" );
}

BOOST_AUTO_TEST_SUITE_END()
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( true );
}

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECK( false );
}

BOOST_AUTO_TEST_SUITE_END()
```

```
> example
Running 4 test cases...
test.cpp(12): fatal error in "test_case2": critical check 1 == 2 failed [1 != 2]
test.cpp(27): error in "test_case4": check false failed

*** 2 failures detected in test suite "example"
```

Master Test Suite



- Model (temporary)

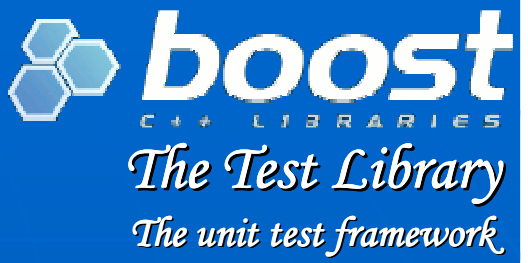
```
namespace boost {
namespace unit_test {

class master_test_suite_t : public test_suite {
public:
    int      argc;
    char**   argv;
};

} // namespace unit_test
} // namespace boost
```

- Access interface

```
namespace boost {
namespace unit_test {
namespace framework {

master_test_suite_t& master_test_suite();

} // namespace framework
} // namespace unit_test
} // namespace boost
```

Master Test Suite

CLA access

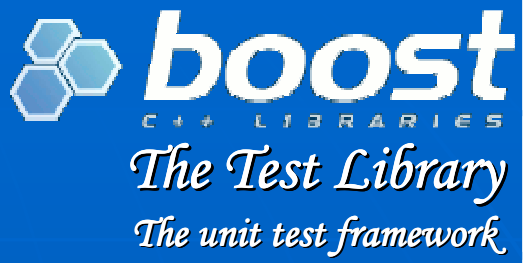



```
#include <boost/test/included/unit_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

void free_test_function()
{
    BOOST_CHECK( true /* test assertion */ );
}

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     if( framework::master_test_suite().argc > 1 )
         return 0;

     framework::master_test_suite().
        add( BOOST_TEST_CASE( &free_test_function ) );

     return 0;
}

//____________________________________________________________________________//
```

- Two references to argc/argv

Master Test Suite

Naming using BOOST_TEST_MODULE



This example is using the dynamic library variant of the Unit Test Framework.

```
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE my master test suite name
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE( free_test_function )
{
    BOOST_CHECK( true /* test assertion */ );
}

//____________________________________________________________________________//
```

Master Test Suite

Naming using BOOST_TEST_MODULE

This example
the Unit Test

```
#define BOOST_TEST_D
#define BOOST_TEST_M
#include <boost/test

BOOST_AUTO_TEST_CASE
{
    BOOST_CHECK( tru
}

//____________________________________________________________________________//
```

```
> example --log_level=test_suite
Running 1 test case...
Entering test suite "my master test suite name"
Entering test case "free_test_function"
Leaving test case "free_test_function"; testing time: 1ms
Leaving test suite "my master test suite name"

*** No errors detected
```

Master Test Suite

Explicit naming

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( free_test_function )
{
    BOOST_CHECK( true /* test assertion */ );
}

//____________________________________________________________________________//

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     framework::master_test_suite().p_name.value = "my master test suite name";

     return 0;
}

//____________________________________________________________________________//
```

Master Test Suite

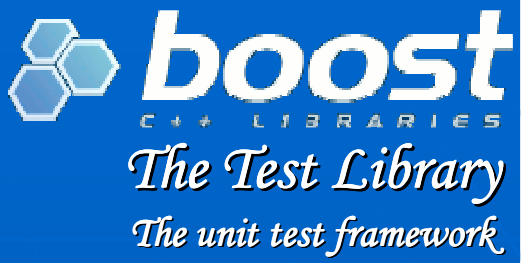
Explicit naming

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#include <boost/test
using namespace boos

//____________________

BOOST_AUTO_TEST_CASE
{
    BOOST_CHECK( tru
}

//____________________

test_suite*
init_unit_test_suite( int argc, char* argv[] )
{
     framework::master_test_suite().p_name.value = "my master test suite name";

     return 0;
}

//____________________________________________________________________________//
```

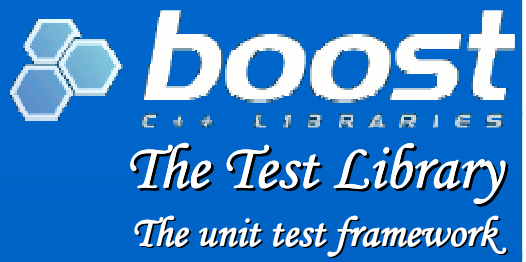
```
> example --log_level=test_suite
Running 1 test case...
Entering test suite "my master test suite name"
Entering test case "free_test_function"
Leaving test case "free_test_function"; testing time: 1ms
Leaving test suite "my master test suite name"

*** No errors detected
```

Expected failures specification



- **Use and misuse**
  - Is not to check for expected functionality failures
  - Per test case - may cause false positives

Expected failures specification

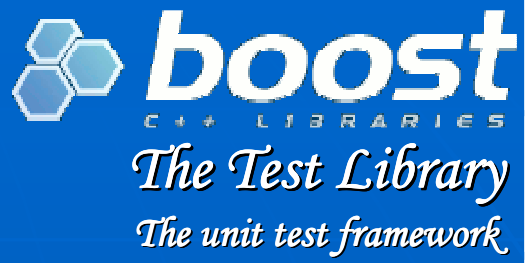


- Use and misuse
- Specified during test tree construction

```
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;

void free_test_function()
{
 // …
}

test_suite*
init_unit_test_suite( int, char* [] ) {
    framework::master_test_suite().
        add( BOOST_TEST_CASE( &free_test_function ), 2 );

    return 0;
}
```

Expected failures specification

usage with automated registration



- BOOST_AUTO_TEST_CASE_EXPECTED_FAILURES

BOOST_AUTO_TEST_CASE_EXPECTED_FAILURES (
    test_case_name,
    number_of_expected_failures)

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE_EXPECTED_FAILURES( my_test1, 1 )

BOOST_AUTO_TEST_CASE( my_test1 )
{
    BOOST_CHECK( 2 == 1 );
}

//____________________________________________________________________________//
```

Expected failures specification

usage with automated registration

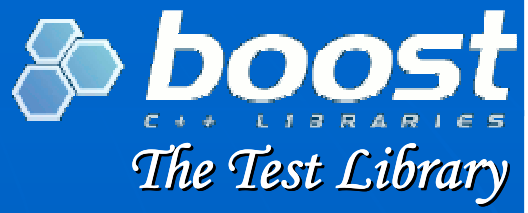
boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_AUTO_TEST_CASE_EXPECTED_FAILURES

BOOST_AUTO_TEST_CASE_EXPECTED_FAILURES (
    test_case_name,
    number_of

```
#define BOOST_TEST_MO
#include <boost/test/

BOOST_AUTO_TEST_CASE_

BOOST_AUTO_TEST_CASE(
{
    BOOST_CHECK( 2 == 1 );
}

//____________________________________________________________________________//
```

```
> example --report_level=short
Running 1 test case...
test.cpp(8): error in "my_test1": check 2 == 1 failed

Test suite "example" passed with:
  1 assertion out of 1 failed
  1 failure expected
  1 test case out of 1 passed
```



Fixtures

Fixtures

boost C++ LIBRARIES
The Test Library
The unit test framework

- Test fixture - collection of:
  - preconditions
  - particular states of tested unites
  - necessary cleanup procedures
- Fixtures and XP
- Practical considerations:
  - Setup/teardown pair
  - Fixture assignment and direct access

Fixtures

Native C++ approach

boost C++ LIBRARIES
The Test Library
The unit test framework

```
struct MyFixture {
     MyFixture() { i = new int; *i = 0; }
     ~MyFixture() { delete i; }

    int* i;
};

BOOST_AUTO_TEST_CASE( test_case1 )
{
    MyFixture f;

    // do something involving f.i
}

BOOST_AUTO_TEST_CASE( test_case2 )
{
    MyFixture f;

    // do something involving f.i
}
```

- Little to do beyond this for manually registration test units

Fixtures

Generic fixture model

boost C++ LIBRARIES
The Test Library
The unit test framework

```
struct <fixture-name> {
     <fixture-name>();  // setup function
     ~<fixture-name>(); // teardown function
};
```

- No teardown error report using exception!

Fixtures

Per test case fixture

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_FIXTURE_TEST_CASE

```
BOOST_FIXTURE_TEST_CASE(
        test_case_name,
        fixture_name)
```

- Use in place of BOOST_AUTO_TEST_CASE
- Direct fixture members access
- Still need to refer to fixture

Fixtures

Per test case fixture

boost C++ LIBRARIES

The Test Library

The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

struct F {
    F() : i( 0 ) { BOOST_TEST_MESSAGE( "setup fixture" ); }
    ~F()         { BOOST_TEST_MESSAGE( "teardown fixture" ); }

    int i;
};

BOOST_FIXTURE_TEST_CASE( test_case1, F )
{
    BOOST_CHECK( i == 1 );
    ++i;
}

BOOST_FIXTURE_TEST_CASE( test_case2, F )
{
    BOOST_CHECK_EQUAL( i, 1 );
}

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( true );
}
```

Fixtures

Per test case fixture

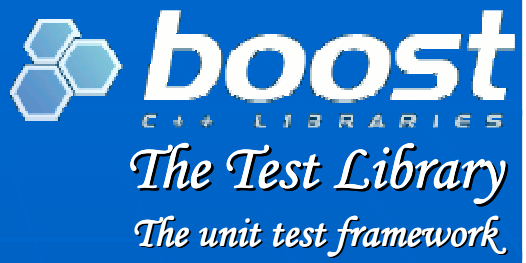
boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

struct F {
    F() : i( 0 ) { BOOST_TEST_MESSAGE( "setup fixture" ); }
    ~F()         { BOOST_TEST_MESSAGE( "teardown fixture" ); }

    int i;
};

BOOST_FIXTURE_TEST_CASE( test_case1, F )
{
    BOOST_CHECK( i == 0 );
    ++i;
}

BOOST_FIXTURE_TEST_CASE
{
    BOOST_CHECK_EQUAL(
}

BOOST_AUTO_TEST_CASE( t
{
    BOOST_CHECK( true )
}
```

```
> example --log_level=message
Running 3 test cases...
setup fixture
teardown fixture
setup fixture
test.cpp(20): error in "test_case2": check i == 1 failed [0 != 1]
teardown fixture

*** 1 failure detected in test suite "example"
```

Fixtures

Test suite level fixture



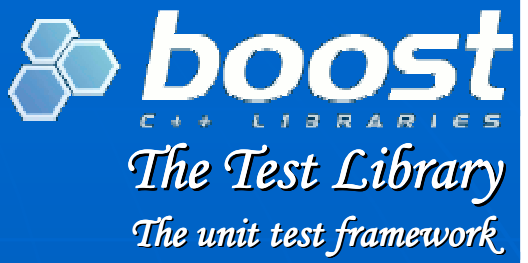
- BOOST_FIXTURE_TEST_SUITE

```
BOOST_FIXTURE_TEST_SUITE(
        suite_name
        fixture_name)
```

- Use in place of BOOST_AUTO_TEST_SUITE
- Direct fixture members access
- No need to refer to fixture with every test case

Fixtures

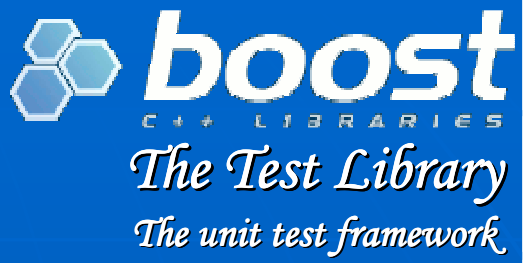
Test suite level fixture




```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

struct F {
    F() : i( 0 ) { BOOST_TEST_MESSAGE( "setup fixture" ); }
    ~F()         { BOOST_TEST_MESSAGE( "teardown fixture" ); }

    int i;
};

//____________________________________________________________________________//

BOOST_FIXTURE_TEST_SUITE( s, F )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_CHECK( i == 1 );
}

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test_case2 )
{
    BOOST_CHECK_EQUAL( 1, 0 );
}

//____________________________________________________________________________//

BOOST_AUTO_TEST_SUITE_END()
```

Fixtures

Global fixture



- Global fixture vs. test module init function
  - No place for cleanup
  - Global fixture is guarded
  - Multiple global fixtures
  - Combines setup/cleanup code
  - Easy switch from local to global fixtures
  - Interactive test runners

Fixtures

Global fixture

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_GLOBAL_FIXTURE

BOOST_GLOBAL_FIXTURE(fixture_name)

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>
#include <iostream>
//____________________________________________________________________________//

struct MyConfig {
    MyConfig()   { std::cout << "global setup\n"; }
    ~MyConfig()  { std::cout << "global teardown\n"; }
};

//____________________________________________________________________________//

BOOST_GLOBAL_FIXTURE( MyConfig );

BOOST_AUTO_TEST_CASE( test_case )
{
    BOOST_CHECK( true );
}

//____________________________________________________________________________//
```

Fixtures

Global fixture

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_GLOBAL_FIXTURE

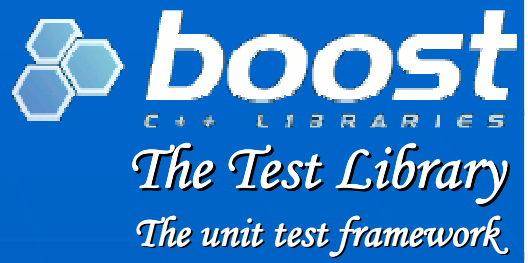
BOOST_GLOB

```
> example
global setup
Running 1 test case...
global teardown

*** No errors detected
```

```
#define BOOST_TEST_MODU
#include <boost/test/un
#include <iostream>
//____________________

struct MyConfig {
    MyConfig()   { std:
    ~MyConfig()  { std::cout << "global teardown\n"; }
};

//____________________________________________________________________________//

BOOST_GLOBAL_FIXTURE( MyConfig );

BOOST_AUTO_TEST_CASE( test_case )
{
    BOOST_CHECK( true );
}

//____________________________________________________________________________//
```

Fixtures

Test suite level fixture

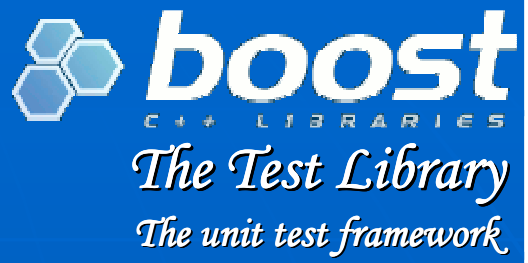


- Is not supported yet

The Test Library

Test tools overview

Test Tools



- Facilitate writing tests
- Macro based, but safe
- Test assertion - a single binary condition
- Test tools levels
  - CHECK
  - REQUIRE
  - WARN

Test Tools

Advanced topics

boost C++ LIBRARIES

The Test Library

The unit test framework

- Beyond bounds of this tutorial
  - Output testing tool
  - BOOST_TEST_DONT_PRINT_LOG_VALUE
  - Logging floating point numbers
  - Floating point comparison algorithms implementation
  - Custom predicate support

Test Tools

Overview

boost C++ LIBRARIES

The Test Library

The unit test framework

- Positive checks
  - BOOST_CHECK
  - BOOST_CHECK_EQUAL
- Negative checks
  - BOOST_CHECK_THROW
  - BOOST_CHECK_EXCEPTION
- Generic BOOST_CHECK_PREDICATE

Test Tools

Examples

boost C++ LIBRARIES

The Test Library

The unit test framework

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test )
{
    int i=2;
    BOOST_WARN( sizeof(int) == sizeof(short) );
    BOOST_CHECK( i == 1 );
    BOOST_REQUIRE( i > 5 );
    BOOST_CHECK( i == 6 ); // will never reach this check
}

//____________________________________________________________________________//
```

Test Tools

Examples

boost C++ LIBRARIES

The Test Library

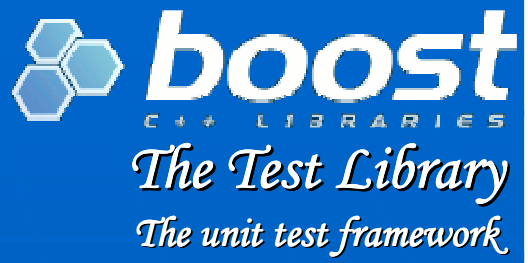
The unit test framework

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test )
{
    int i=2;
    BOOST_WARN( sizeof(int) == sizeof(short) );
    BOOST_CHECK( i == 1 );
    BOOST_REQUIRE( i >
    BOOST_CHECK( i ==
}

//____________________
```

```
> example
Running 1 test case...
test.cpp(10): error in "test": check i == 1 failed
test.cpp(11): fatal error in "test": critical check i > 5 failed

*** 2 failures detected in test suite "Master Test Suite"
```

Test Tools

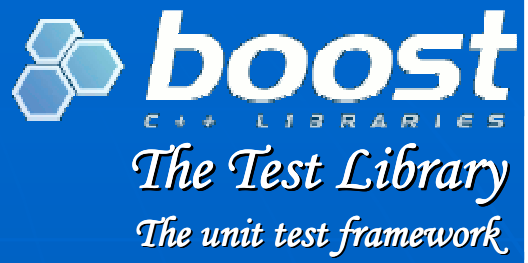
Examples




```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>

class my_exception{};

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test )
{
    int i = 0;
    BOOST_CHECK_THROW( i++, my_exception );
}

//____________________________________________________________________________//
```

Test Tools

Examples

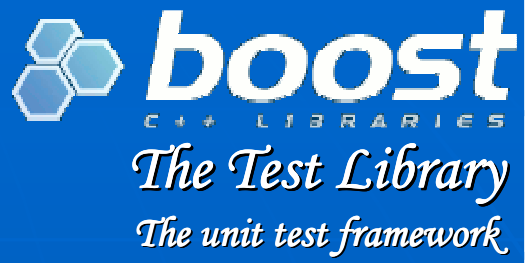



```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>

class my_exception{};

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test )
{
    int i = 0;
    BOOST_CHECK_THROW( i++, my_exception );
}

//____________________________________________________________________________//
```

```
> example
Running 1 test case...
test.cpp(11): error in "test": exception my_exception is expected

*** 1 failure detected in test suite "Master Test Suite"
```

Test Tools

Examples




```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

bool moo( int arg1, int arg2, int mod ) { return ((arg1+arg2) % mod) == 0; }

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test )
{
    int i = 17;
    int j = 15;

    unit_test_log.set_threshold_level( log_warnings );

    BOOST_WARN( moo( 12,i,j ) );
    BOOST_WARN_PREDICATE( moo, (12)(i)(j) );
}

//____________________________________________________________________________//
```

Test Tools

Examples

boost C++ LIBRARIES

The Test Library

The unit test framework

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
using namespace boost::unit_test;

//____________________________________________________________________________//

bool moo( int arg1, int arg2, int mod ) { return ((arg1+arg2) % mod) == 0; }

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test )
{
    int i = 17;
    int j = 15;

    unit_tes

    BOOST_WA
    BOOST_WA
}

//_____________
```

```
> example
Running 1 test case...
test.cpp(18): warning in "test": condition moo( 12,i,j ) is not satisfied
test.cpp(19): warning in "test": condition moo( 12, i, j ) is not satisfied for ( 12, 17, 15 )

*** No errors detected
```

Test Tools

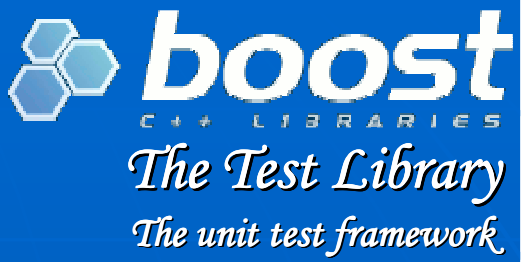
Floating points comparison

boost C++ LIBRARIES
The Test Library
The unit test framework

- For in-depth study refer to:
  - Knuth D.E. *The art of computer programming* (vol II).
  - David Goldberg What Every Computer Scientist Should Know About Floating-Point Arithmetic
  - Kulisch U. Rounding near zero.
  - Philippe Langlois From Rounding Error Estimation to Automatic Correction with Automatic Differentiation
  - Lots of information on William Kahan home page
  - Alberto Squassabia Comparing Floats: How To Determine if Floating Quantities Are Close Enough Once a Tolerance Has Been Reached C++ Report March 2000.
  - Pete Becker The Journeyman's Shop: Trap Handlers, Sticky Bits, and Floating-Point Comparisons C/C++ Users Journal December 2000.

Test Tools

Floating points comparison



- Using a == b

Test Tools

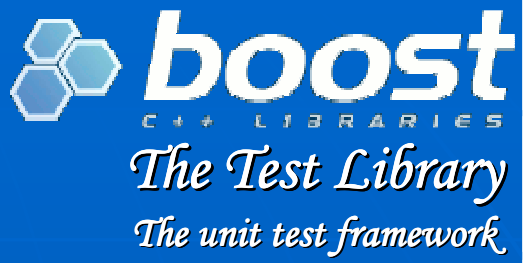
Floating points comparison

boost C++ LIBRARIES
The Test Library
The unit test framework

- Using a == b
- Using |a-b|<=|e|
  - Does 0.1 is acceptable error?
    - Probably yes for a=1e20,b=1e20+1
      - False negative
    - Probably not for a=0.001 b=0.005
      - False positive
    - Probably not for a=0.005 b=-0.005
      - False positive
  - Problem is acceptable error depends on compared values
  - Still may be useful

Test Tools

Floating points comparison



- Using a == b
- Using |a-b|<=|e|
- Using |a-b|/|a|<= |e| (and|or) |a-b|/|b|<= |e|
  - Fix all the problem with absolute comparison
  - But: strong check never true if a or b is zero
    - Use absolute comparisons instead

Test Tools

Floating points comparison: tolerance specification

boost C++ LIBRARIES

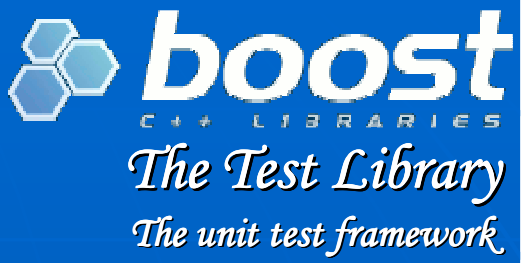
The Test Library

The unit test framework

- Tolerance selection

Test Tools

Floating points comparison: tolerance specification



- Tolerance selection
- Tolerance forms

Test Tools

Floating points comparison: tolerance specification

boost C++ LIBRARIES
The Test Library
The unit test framework

- Tolerance selection
- Tolerance forms:
  - Relative error
    - Strait to the formula
    - In number of epsilons

Test Tools

Floating points comparison: tolerance specification

boost C++ LIBRARIES
The Test Library
The unit test framework

- Tolerance selection
- Tolerance forms:
  - Relative error
  - Percent error
    - Maybe more clear for novice users
    - Relative = percent / 100
    - But it's inconvenient for experts

Test Tools

Floating points comparison: tolerance specification

boost C++ LIBRARIES
The Test Library
The unit test framework

- Tolerance selection
- Tolerance forms:
  - Relative error
  - Percent error
  - Absolute error

Test Tools

Floating points comparison: supplied tools

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_CHECK_CLOSE

BOOST_CHECK_CLOSE(left, right, tolerance)

- Require header floating_point_comparison.hpp

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
#include <boost/test/floating_point_comparison.hpp>

BOOST_AUTO_TEST_CASE( test )
{
    double v1 = 1.23456e-10;
    double v2 = 1.23457e-10;

    BOOST_CHECK_CLOSE( v1, v2, 0.0001 );
    // Absolute value of difference between these two values is 1e-15.
    // They seems to be very close. But we want to checks that these values
    // differ no more then 0.0001% of their value.
    // And this test will fail at tolerance supplied.
}
```

Test Tools

Floating points comparison: supplied tools

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_CHECK_CLOSE

BOOST_CHECK_CLOSE(left, right, tolerance)

- Require header floating_point_comparison.hpp

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
#include <boost/test/floating_point_comparison.hpp>

BOOST_AUTO_TEST_CASE( test )
{
    double v1 = 1.23456e-10;
    double
    BOOST_
    // Abs
    // The
    // dif
    // And
}
```

```
> example
Running 1 test case...
test.cpp(10): error in "test": difference between v1{1.23456e-010} and v2{1.23457e-010}
      exceeds 0.0001%

*** 1 failure detected in test suite "Master Test Suite"
```

Test Tools

Floating points comparison: supplied tools

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_CHECK_CLOSE_FRACTION

BOOST_CHECK_CLOSE_FRACTION(left, right, tolerance)

- Require header floating_point_comparison.hpp

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
#include <boost/test/floating_point_comparison.hpp>

BOOST_AUTO_TEST_CASE( test )
{
    double v1 = 1.23456e28;
    double v2 = 1.23457e28;

    BOOST_CHECK_CLOSE_FRACTION( v1, v2, 1e-5 );
    // Absolute value of difference between these two values is 1e+23.
    // But we are interested only that relative error does not exceed 1e-5
    // And this test will pass.
}
```

Test Tools

Floating points comparison: supplied tools

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_CHECK_CLOSE_FRACTION

BOOST_CHECK_CLOSE_FRACTION(left, right, tolerance)

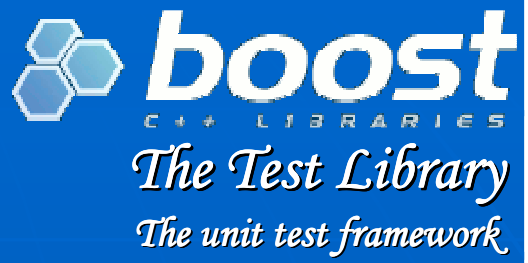
- Require header floating_point_comparison.hpp

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
#include <boost/test/floating_point_comparison.hpp>

BOOST_AUTO_TEST_CASE( test )
{
    double v1 = 1.23456e28;
    double v2 = 1.23457e28;

    BOOST_CHECK_CLOSE_FRA
    // Absolute value of
    // But we are interes
    // And this test will
}
```

```
> example
Running 1 test case...

*** No errors detected
```

Test Tools

Floating points comparison: supplied tools



- BOOST_CHECK_SMALL

BOOST_CHECK_SMALL(val, tolerance)

- Require header floating_point_comparison.hpp

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
#include <boost/test/floating_point_comparison.hpp>

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( test )
{
    double v = -1.23456e-3;

    BOOST_CHECK_SMALL( v, 0.000001 );
}

//____________________________________________________________________________//
```

Test Tools

Floating points comparison: supplied tools

boost C++ LIBRARIES
The Test Library
The unit test framework

- BOOST_CHECK_SMALL

BOOST_CHECK_SMALL(val, tolerance)

- Require header floating_point_comparison.hpp

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
#include <boost/test/floating_point_comparison.hpp>

//____________________________________________________________________________//

BOOST_AUTO_TEST_CASE( t
{
    double v = -1.23456

    BOOST_CHECK_SMALL(
}

//____________________
```

```
> example
Running 1 test case...
test.cpp(9): error in "test": absolute value of v{-0.00123456} exceeds 1e-06

*** 1 failure detected in test suite "Master Test Suite"
```

The Test Library

Test module output

Test module output

boost C++ LIBRARIES
The Test Library
The unit test framework

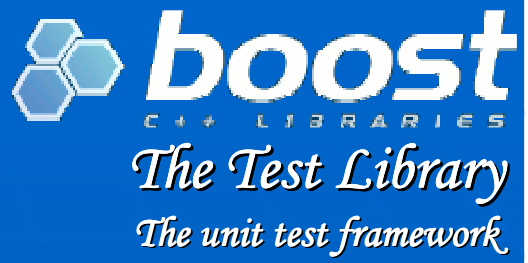
- One of the major assets the UTF
  - All test errors are reported uniformly
  - Detailed information on the source of an error
  - Flexibility in what is shown in the output
  - Flexibility in how output is formatted
  - Separation of the test errors description from the results report summary
    - **Test log** – a record of all events that occur during the testing
    - **Test results report** – produced after the testing is completed

Test log output

boost C++ LIBRARIES
The Test Library
The unit test framework

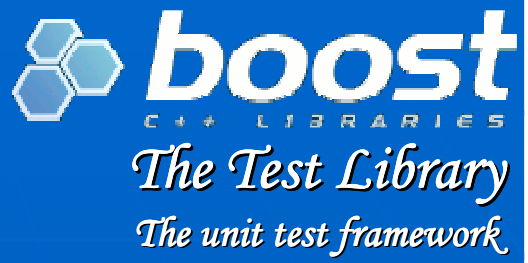
- Log levels
  - Success information messages (all messages)
  - Test tree traversal notifications
  - General information messages
  - Warning messages
  - Non fatal error messages
  - Uncaught C++ exceptions notifications
  - Non-fatal system error
  - Fatal system error
  - No messages

Test log output



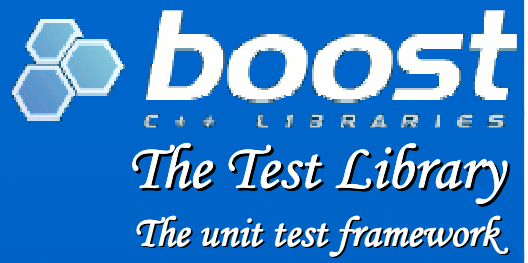
- Log levels
- Two log formats
  - Human readable format
  - XML based format

Test log output



- Log levels
- Two log formats
- Runtime configuration

Test log output



- Log levels
- Two log formats
- Runtime configuration
- Compile-time configuration

```
namespace boost {
namespace unit_test {

unit_test_log()::set_stream( std::ostream& );
unit_test_log()::set_threshold_level( log_level );
unit_test_log()::set_format( output_format );
unit_test_log()::set_formatter( unit_test_log_formatter* );

} // namespace unit_test
} // namespace boost
```

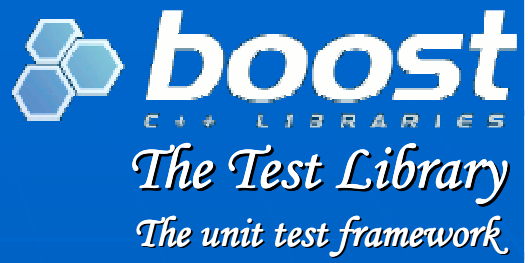
Test log output

boost C++ LIBRARIES
The Test Library
The unit test framework

- Log levels
- Two log formats
- Runtime configuration
- Compile-time configuration
- BOOST_TEST_CHECKPOINT

BOOST_TEST_CHECKPOINT(checkpoint message)

Test log output



- Log levels
- Two log formats
- Runtime configuration
- Compile-time configuration
- BOOST_TEST_CHECKPOINT
- BOOST_TEST_PASSPOINT

BOOST_TEST_PASSPOINT()

Test log output example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case2 )
{
    int i = 0;

    BOOST_MESSAGE( "Value of i=" << i );

    BOOST_REQUIRE_EQUAL( i, 2 );
}

//____________________________________________//

BOOST_AUTO_TEST_SUITE_END()
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECKPOINT( "Entering test area" );
    int* p = 1;

    if( *p != 0 ) return;
}

//____________________________________________//

BOOST_AUTO_TEST_SUITE_END()
```

Test log output example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case2 )
{
    int i = 0;

    BOOST_MESSAGE( "Value of i=" << i );

    BOOST_REQUIRE_EQUAL( i, 2 );
}

//____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECKPOINT( "Entering test area" );
    int* p = 1;

    if( *p != 0 ) return;
}

//____________________________________________//

BOOST_AUTO_TEST_SUITE_END()
```

```
> example --log_level=nothing

*** 4 failures detected in test suite "example"
```

Test log output example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case2 )
{
    int i =

    BOOST_ME

    BOOST_RE
}

//_____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECKPOINT( "Entering test area" );
    int* p = 1;
```

```
> example --log_level=fatal_error
Running 4 test cases...
test.cpp(21): fatal error in "test_case2": critical check i == 2 failed [0 != 2]
unknown location(0): fatal error in "test_case4": memory access violation
      occurred at address 0x00000001, while attempting to  read inaccessible data
test.cpp(41): last checkpoint: Entering test area

*** 4 failures detected in test suite "example"
```

Test log output example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________________________________________//

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//_____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
{
```

```
> example --log_level=cpp_exception
Running 4 test cases...
test.cpp(21): fatal error in "test_case2": critical check i == 2 failed [0 != 2]
unknown location(0): fatal error in "test_case3": unknown type
test.cpp(32): last checkpoint
unknown location(0): fatal error in "test_case4": memory access violation occurred at
      address 0x00000001, while attempting to  read inaccessible data
test.cpp(41): last checkpoint: Entering test area

*** 4 failures detected in test suite "example"
```

Test log output example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
```

```
> example --log_level=error
Running 4 test cases...
test.cpp(21): fatal error in "test_case2": critical check i == 2 failed [0 != 2]
test.cpp(32): error in "test_case3": check false failed
unknown location(0): fatal error in "test_case3": unknown type
test.cpp(32): last checkpoint
unknown location(0): fatal error in "test_case4": memory access violation occurred at
      address 0x00000001, while attempting to  read inaccessible data
test.cpp(41): last checkpoint: Entering test area

*** 4 failures detected in test suite "example"
```

Test log output example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//
```

```
> example --log_level=warning
Running 4 test cases...
test.cpp(10): warning in "test_case1": condition sizeof(int) < 4 is not satisfied
test.cpp(21): fatal error in "test_case2": critical check i == 2 failed [0 != 2]
test.cpp(32): error in "test_case3": check false failed
unknown location(0): fatal error in "test_case3": unknown type
test.cpp(32): last checkpoint
unknown location(0): fatal error in "test_case4": memory access violation occurred at
      address 0x00000001, while attempting to  read inaccessible data
test.cpp(41): last checkpoint: Entering test area

*** 4 failures detected in test suite "example"
```

Test log output example


```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_T
{
    BOOST_WA
}

//_____________

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//_____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
```

```
> example --log_level=message
Running 4 test cases...
test.cpp(10): warning in "test_case1": condition sizeof(int) < 4 is not satisfied
Value of i=0
test.cpp(21): fatal error in "test_case2": critical check i == 2 failed [0 != 2]
test.cpp(32): error in "test_case3": check false failed
unknown location(0): fatal error in "test_case3": unknown type
test.cpp(32): last checkpoint
unknown location(0): fatal error in "test_case4": memory access violation occurred at
      address 0x00000001, while attempting to  read inaccessible data
test.cpp(41): last checkpoint: Entering test area
Test is aborted

*** 4 failures detected in test suite "example"
```

Test log output example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________

BOOST_AUTO_T

BOOST_AUTO_T
{
    BOOST_WA
}

//____________

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )
```

```
> example --log_level=test_suite
Running 4 test cases...
Entering test suite "example"
Entering test suite "test_suite1"
Entering test case "test_case1"
test.cpp(10): warning in "test_case1": condition sizeof(int) < 4 is not satisfied
Leaving test case "test_case1"; testing time: 26ms
Entering test case "test_case2"
Value of i=0
test.cpp(21): fatal error in "test_case2": critical check i == 2 failed [0 != 2]
Leaving test case "test_case2"; testing time: 22ms
Leaving test suite "test_suite1"
Entering test suite "test_suite2"
Entering test case "test_case3"
test.cpp(32): error in "test_case3": check false failed
unknown location(0): fatal error in "test_case3": unknown type
test.cpp(32): last checkpoint
…
```

Test log output example

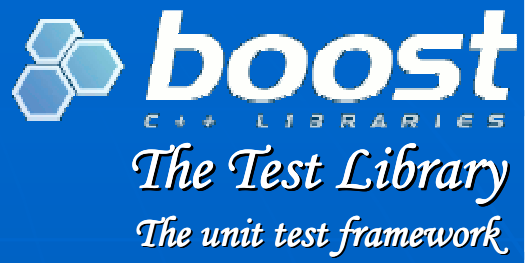
boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WA
}

//____________

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}
```

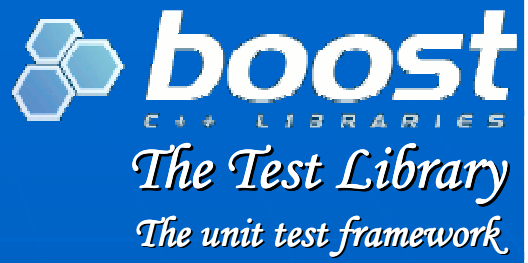
```
> example --log_format=xml
<TestLog>
  <FatalError file="test.cpp" line="21">critical check i == 2 failed [0 != 2]</FatalError>
  <Error file="test.cpp" line="32">check false failed</Error>
  <Exception name="test_case3">unknown type
    <LastCheckpoint file="test.cpp" line="32"></LastCheckpoint>
  </Exception>
  <Exception name="test_case4">memory access violation occurred at address
      0x00000001, while attempting to  read inaccessible data
    <LastCheckpoint file="test.cpp" line="41">Entering test area</LastCheckpoint>
  </Exception>
</TestLog>

*** 4 failures detected in test suite "example"
```

Test results report



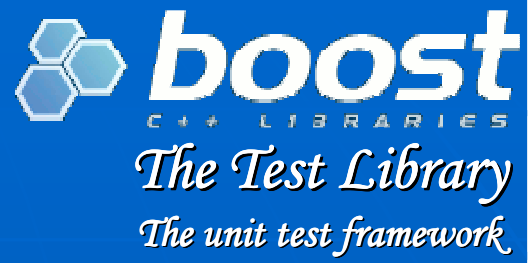
- Report levels
  - No report
  - Confirmation report
  - Short report
  - Detailed report

Test results report



- Report levels
- Report formats
  - Human readable format
  - XML based format

Test results report



- Report levels
- Report formats
- Runtime configuration

Test results report

boost C++ LIBRARIES
The Test Library
The unit test framework

- Report levels
- Report formats
- Runtime configuration
- Compile-time configuration

```
namespace boost {
namespace unit_test {
namespace results_reporter {

void    set_level( report_level );
void    set_stream( std::ostream& );
void    set_format( output_format );
void    set_format( results_reporter::format* );

} // namespace results_reporter
} // namespace unit_test
} // namespace boost
```

Test results report example

boost C++ LIBRARIES

The Test Library

The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case2 )
{
    int i = 0;

    BOOST_MESSAGE( "Value of i=" << i );

    BOOST_REQUIRE_EQUAL( i, 2 );
}

//____________________________________________//

BOOST_AUTO_TEST_SUITE_END()
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECKPOINT( "Entering test area" );
    int* p = 1;

    if( *p != 0 ) return;
}

//____________________________________________//

BOOST_AUTO_TEST_SUITE_END()
```

Test results report example

boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case2 )
{
    int i = 0;

    BOOST_MESSAGE( "Value of i=" << i );

    BOOST_REQUIRE_EQUAL( i, 2 );
}

//____________________________________________//

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECKPOINT( "Entering test area" );
    int* p = 1;

    if( *p != 0 ) return;
}

//____________________________________________//

BOOST_AUTO_TEST_SUITE_END()
```

```
> example --log_level=nothing --report_level=no
```

Test results report example

boost C++ LIBRARIES

The Test Library

The unit test framework

```
#define BOOST_TEST_MODULE example
#include <boost/test/unit_test.hpp>

//____________________________________________//

BOOST_AUTO_TEST_SUITE( test_suite1 )

BOOST_AUTO_TEST_CASE( test_case1 )
{
    BOOST_WARN( sizeof(int) < 4 );
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case2 )
{
    int i = 0;

    BOOST_ME

    BOOST_RE
}

//_____________

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )

BOOST_AUTO_TEST_CASE( test_case3 )
{
    BOOST_CHECK( false );

    throw 1;
}

//____________________________________________//

BOOST_AUTO_TEST_CASE( test_case4 )
{
    BOOST_CHECKPOINT( "Entering test area" );
    int* p = 1;

    if( *p != 0 ) return;
```

```
> example --log_level=nothing --report_level=short

Test suite "example" aborted with:
  4 assertions out of 4 failed
  1 test case out of 4 passed
  3 test cases out of 4 failed
  3 test cases out of 4 aborted
```

Test results report example

boost C++ LIBRARIES

The Test Library

The unit test framework

```
#define BOOS
#include <bo

//_____

BOOST_AUTO_T

BOOST_AUTO_T
{
    BOOST_WA
}

//_____

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//_____

BOOST_AUTO_T
```

```
> example --log_level=nothing --report_level=no

Test suite "example" aborted with:
  4 assertions out of 4 failed
  1 test case out of 4 passed
  3 test cases out of 4 failed
  3 test cases out of 4 aborted

  Test suite "test_suite1" failed with:
    1 assertion out of 1 failed
    1 test case out of 2 passed
    1 test case out of 2 failed
    1 test case out of 2 aborted

    Test case "test_case1" passed

    Test case "test_case2" aborted with:
      1 assertion out of 1 failed

    ...
```

Test results report example

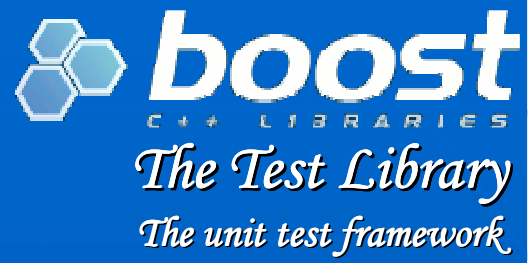
boost C++ LIBRARIES
The Test Library
The unit test framework

```
#define BOOST_TEST_MODULE example
#include <bo

//_____

BOOST_AUTO_T

BOOST_AUTO_T
{
    BOOST_WA
}

//_____

BOOST_AUTO_T
{
    int i =

    BOOST_ME

    BOOST_RE
}

//_____

BOOST_AUTO_T
```

```
BOOST_AUTO_TEST_SUITE( test_suite2 )
```

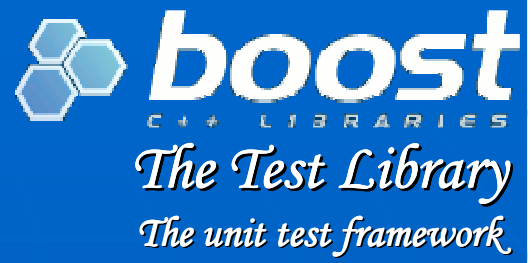
```
> example --log_level=nothing --report_format=xml
<TestResult>
  <TestSuite name="example"
             result="aborted"
             assertions_passed="0"
             assertions_failed="4"
             expected_failures="0"
             test_cases_passed="1"
             test_cases_failed="3"
             test_cases_skipped="0“
             test_cases_aborted="3">
  </TestSuite>
</TestResult>
```

Runtime configuration



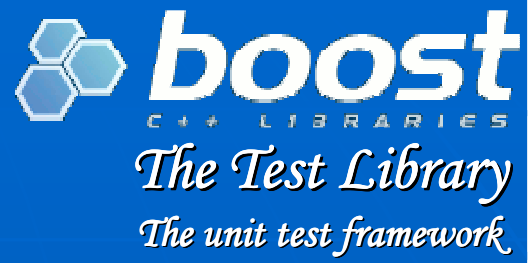
- Using command line parameters

Runtime configuration



- Using command line parameters
- Using environment variable

Runtime configuration



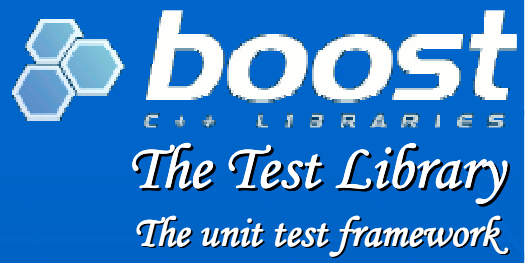
- Using command line parameters
- Using environment variable
- Future rework

Runtime parameters

boost C++ LIBRARIES
The Test Library
The unit test framework

- log_level
- result_code
- report_level
- save_pattern
- build_info
- show_progress
- catch_system_errors
- report_format
- log_format
- output_format
- detect_memory_leaks
- random
- break_exec_path

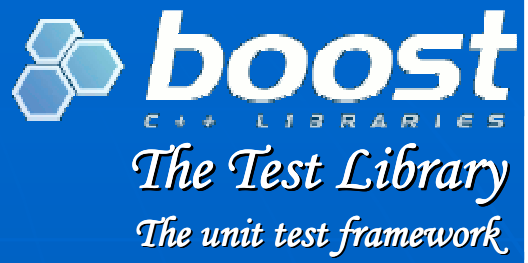
Interaction based testing



- Checks that a particular interaction occurs
  - Particular functions are invoked
- Do not "test against implementation"
- Useful in boundary/collaboration scenarios
  - Data base access
  - Hardware driver
  - Remote service provider
- Mock objects

Interaction based testing

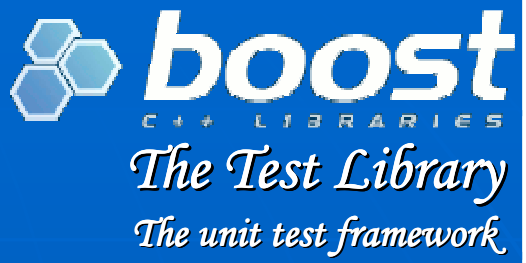
Expectation collection



- "Record phase invocation"
- List of named functions
  - Require reflection
- Both require repetition to write the test

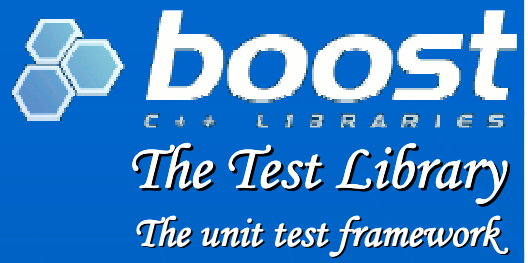
Interaction based testing

Expectation collection



- "Record phase invocation"
- List of named functions
- Both require repetition to write the test
- Logged expectations
  - Logging/Testing mode
  - Pattern

Logged expectation testing



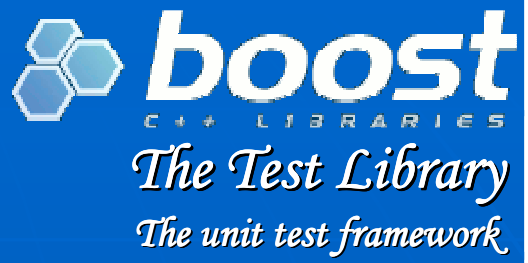
- Write custom mocks
- BOOST_TEST_LOGGED_EXPECTATIONS
- Example: logged_exp_example.cpp

Exception safety testing

boost C++ LIBRARIES
The Test Library
The unit test framework

- Special case of interaction testing
- Use to test template based algorithms
- Original idea by David A.
- Test procedure:
  - Write mock class that may fail at any point and satisfy algorithm interface requirement
  - Run algorithm under test instantiated with mock class within BOOST_TEST_EXCEPTION_SAFETY based test case

Exception safety testing



- Example1: est_example1.cpp
- Example2: est_example2.cpp

Future plans

boost C++ LIBRARIES
The Test Library
The unit test framework

- Major documentation update
- Runtime parameters subsystem update
- Thread safety support
- Performance testing tools
- “diff like” matching
- Unicode support
- BOOST_CHECK_DIFF family
- Run test by name
- Proper FP output

Future plans

boost C++ LIBRARIES
The Test Library
The unit test framework

- New debug interfaces:

```
namespace boost {
namespace debug {

bool        under_debugger();
void        debugger_break();
std::string set_debugger( unit_test::const_string dbg_id,
                          dbg_starter s = dbg_starter() );
bool        attach_debugger( bool break_or_continue = true );
void        detect_memory_leaks( bool on_off );
void        break_memory_alloc( long mem_alloc_order_num );

void        dump_stack();
vector<…> current_stack();

} // namespace debug
} // namespace boost
```