

NAME

exercisix — lightweight unit-test framework for C++

DESCRIPTION

Exercisix is a unit-test framework for C++, designed for test-aided development. It's main features are:

- test discovery
- textual description as primary test identifier
- terse test tools syntax
- informative error messages
- emacs-friendly error message format
- easy build: framework is source-based, no linking is required

TEST LAYOUT

Each test suite resides in its own translation unit, which is compiled into a separate executable file. Framework provides its own `main()` function. A test suite file must include header `<exercisix.hh>` and may have one or more test cases.

A test case is defined as a word **TEST**, followed by parenthesis, containing a text string literal within them. Then goes code block in curly braces, which contains test case body. A test case body is a sequence of statements, similar to function body. A test case body should contain an expectation statement, which uses one of `EXPECT_*` test tools, described below.

A minimal test suite is specified like this:

```
#include <exercisix.hh>

TEST( "one should be equal to one" )
{
    EXPECT( 1, ==, 1 );
}
```

Each test case is a function with unique name `testN()`, where `N` is a number of the line the macro **TEST** were called. See section 'COMMAND-LINE OPTIONS' below for details of how to match test case with the name of it's function.

ORDER OF EXECUTION

Test cases are executed in sequence, in the same order as they are defined in the source file. If during execution of a test case a test tool reports a failure, then test suite execution stops after the test case finishes and error message is printed. If no failures were reported, then the name of the source file containing test suite definition is printed, followed by the word "SUCCESS".

TEST TOOLS

Four test tools are provided: `EXPECT()`, `EXPECT_NO_EXCEPTIONS()`, `EXPECT_EXCEPTION()`, and `EXERCISIX_FAIL`.

The first, `EXPECT()` have the following form:

```
EXPECT( expr1, op, expr2 );
```

Where `expr1` and `expr2` are any valid C++ expressions. The `op` is any C++ operator, defined for types which expressions `expr1` and `expr2` evaluate to. The tool evaluates the expression

```
(expr1) op (expr2)
```

, casts it to **bool**, then registers failure if result value is **false** and does nothing if result value is **true**.

The second test tool, `EXPECT_NO_EXCEPTIONS` has the following form:

```
EXPECT_NO_EXCEPTIONS( expr );
```

The *expr* is any valid C++ expression. The tool evaluates the expression *expr*, then does nothing if expression evaluates without throwing an exception and registers failure if an exception was thrown during evaluation of the expression *expr*.

The third test tool, `EXPECT_EXCEPTION` has the following form:

```
EXPECT_EXCEPTION( expr, type );
```

Where *expr* is any valid C++ expression and *type* is any valid C++ type. The tool evaluates the expression *expr*, then does nothing if an exception of type *type* was thrown during evaluation of expression *expr*, registers a failure if thrown expression was of a different type or no exceptions were thrown at all.

The last `EXERCISIX_FAIL` test tool is provided for creating custom complex assertions. It is used as a complete statement:

```
EXERCISIX_FAIL;
```

If the line on which this tool is placed is reached, framework registers generic failure.

ERROR MESSAGES

If during execution of a test case a failure is reported, the framework prints source file name, line number (where a test tool that reported a failure was used), and failed test description. After that comes test-tool specific information as described below:

The `EXPECT()` test tool prints operator, texts of both expressions and their values. If expression value is not printable, framework prints expression text only and explicitly states that value can not be printed. The framework can print any value that may be used in `operator<<` for `std::ostream`.

The `EXPECT_NO_EXCEPTIONS()` test tool prints actual type of exception that was thrown. If type of exception can not be determined, the framework explicitly states that exception of unknown type was thrown.

The `EXPECT_EXCEPTION()` test tool prints actual type of exception that was thrown. If type of exception can not be determined, the framework explicitly states that exception of unknown type was thrown. The expected type of exception is also printed.

If during execution of a test method an exception is thrown from outside one of `EXPECT_*` test tools, process is stopped and error message "unexpected exception" containing the type of exception (if it can be determined) is printed. The source file name and a line, which contains `TEST` macro for the test method are also printed.

The framework will detect the type of any exception which is derived from `std::exception`.

EXAMPLES

```
#include <exercisix.hh>
```

```
#include <vector>
```

```
#include <stdexcept>
```

```
using std::vector;
```

```
TEST( "one equals to one" )
{
    EXPECT( 1, ==, 1 );
}
```

```

TEST( "two is less than three" )
{
    EXPECT( 2, <, 3 );
}

TEST( "at() completes successfully if index is within bounds" )
{
    int source[] = { 1, 2, 3 };
    vector <int> v(source, source + 3);

    EXPECT_NO_EXCEPTIONS( v.at(1) );
}

TEST( "at() throws exception of type 'std::out_of_range'
      "if index is out of bounds" )
{
    int source[] = { 1, 2, 3 };
    vector <int> v(source, source + 3);

    EXPECT_EXCEPTION( v.at(7), std::out_of_range);
}

void throw_int()
{
    throw 11;
}

TEST( "if function throws integer, it must be greater than 7" )
{
    try {
        throw_int();
    }
    catch(int e)
    {
        if (e < 7)
            EXERCISIX_FAIL;
    }
}

```

SETUP AND TEARDOWN

The framework does not provide specifically any tools for performing **SETUP** or **TEARDOWN** operations before and after execution of a test method. Usual C++ constructs for declaring local variables inside test method body may be used instead. Local variables will be initialized as control reaches their definition and destroyed at the end of the block, thus eliminating the need for specific **SETUP** or **TEARDOWN** tools.

COMMAND-LINE OPTIONS

The executables produced using the framework accept command-line options. The syntax of the command-line is as follows:

```
test-suite-executable [ -l | --list ] [ -v | --verbose ] [ --version ]
```

The available options are:

-l --list

Do not execute test cases, just list their descriptions instead. This produces a list of all test cases in a suite, useful for documentation. For each test case it's ordinal number, test function name and test description are printed. This option can also be used to match test cases with names of functions that represent these test cases. It's helpful for debugging, since framework obscures the actual structure of test code. This lets you set a breakpoint on the test case.

-v --verbose

Print each test case description before actually executing it. If test case reports no failures, the word "SUCCESS" is printed at the same line after test case is finished.

--version

Print version of the framework. No test cases are executed.

COMPATIBILITY

The framework is believed to be compliant with *ISO/IEC 14882:2003* .

LICENSE

The "**exercisix**" framework is distributed under the terms of BSD license.

LICENSE TEXT

Copyright (c) 2009, 2011 Alexander Churanov. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY ALEXANDER CHURANOV "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ALEXANDER CHURANOV BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SEE ALSO

c++(1)

AUTHORS

Alexander Churanov <exercisix@alexanderchuranov.com>

WEBSITE

<http://alexanderchuranov.com/software/exercisix/>