Unpack the sample package using a linux command shell:

```
martin@martin:~/Desktop$ tar -xpvzf simple_testsuite.tar.gz
simple_testsuite/
simple_testsuite/src-test/
simple_testsuite/src/
simple_testsuite/srcext/
simple_testsuite/rpc/
simple_testsuite/doc/
simple_testsuite/bin/
simple_testsuite/conf/
simple_testsuite/make/
simple_testsuite/obj/
simple_testsuite/srcext/simple_string_util/
simple_testsuite/src-test/test_simple_string_util.cpp
simple_testsuite/src/simple_testsuite.cpp
simple_testsuite/src/testrunner.cpp
simple_testsuite/src/simple_testsuite.h
simple_testsuite/make/Makefile.bak
simple_testsuite/make/Makefile
simple_testsuite/srcext/simple_string_util/simple_string_util.cpp
simple_testsuite/srcext/simple_string_util/simple_string_util.h
```

Build the sample on your machine (requires g++/gcc or clang++ or intel compiler):

```
martin@martin:~/Desktop$ cd simple_testsuite/make/
martin@martin:~/Desktop/simple_testsuite/make$ make build
rm -f ../bin/*
rm -f ../obj/*
rm -f ../src/*.bck
../src/testrunner.cpp
../src/simple_testsuite.cpp
../src-test/test_simple_string_util.cpp
../srcext/simple_string_util/simple_string_util.cpp
g++  -c  -I../srcext/simple_string_util -I../src -I../srcext/simple_string_util -I../src -O1  -Warray-bounds  -W -Wall -Wcomments -Wsign-compare -Wformat=2 -
sign-conversion  -Wshadow -Wunused -Wextra -ansi -pedantic -Wno-long-long -Wno-write-strings -fstack-protector-all -fshow-column -fno-strict-aliasing  -o ../
bj/testrunner.o ../src/testrunner.cpp
g++  -c  -I../srcext/simple_string_util -I../src -I../srcext/simple_string_util -I../src -O1  -Warray-bounds  -W -Wall -Wcomments -Wsign-compare -Wformat=2 -
sign-conversion  -Wshadow -Wunused -Wextra -ansi -pedantic -Wno-long-long -Wno-write-strings -fstack-protector-all -fshow-column -fno-strict-aliasing  -o ../
bj/simple_testsuite.o ../src/simple_testsuite.cpp
g++  -c  -I../srcext/simple_string_util -I../src -I../srcext/simple_string_util -I../src -O1  -Warray-bounds  -W -Wall -Wcomments -Wsign-compare -Wformat=2 -
sign-conversion  -Wshadow -Wunused -Wextra -ansi -pedantic -Wno-long-long -Wno-write-strings -fstack-protector-all -fshow-column -fno-strict-aliasing  -o ../
bj/simple_string_util.o ../srcext/simple_string_util/simple_string_util.cpp
g++  -c  -I../srcext/simple_string_util -I../src -I../srcext/simple_string_util -I../src -O1  -Warray-bounds  -W -Wall -Wcomments -Wsign-compare -Wformat=2 -
sign-conversion  -Wshadow -Wunused -Wextra -ansi -pedantic -Wno-long-long -Wno-write-strings -fstack-protector-all -fshow-column -fno-strict-aliasing  -o ../
bj/test_simple_string_util.o ../src-test/test_simple_string_util.cpp
g++   -o ../bin/testrunner ../obj/testrunner.o ../obj/simple_testsuite.o ../obj/simple_string_util.o ../obj/test_simple_string_util.o -lpthread -O3 -Wl
martin@martin:~/Desktop/simple_testsuite/make$
```

Now the testsuite with a sample is successfull compiled on your machine. You can run the tests by executing the testrunner (in the /bin folder, see next page)

Executing the testrunner:

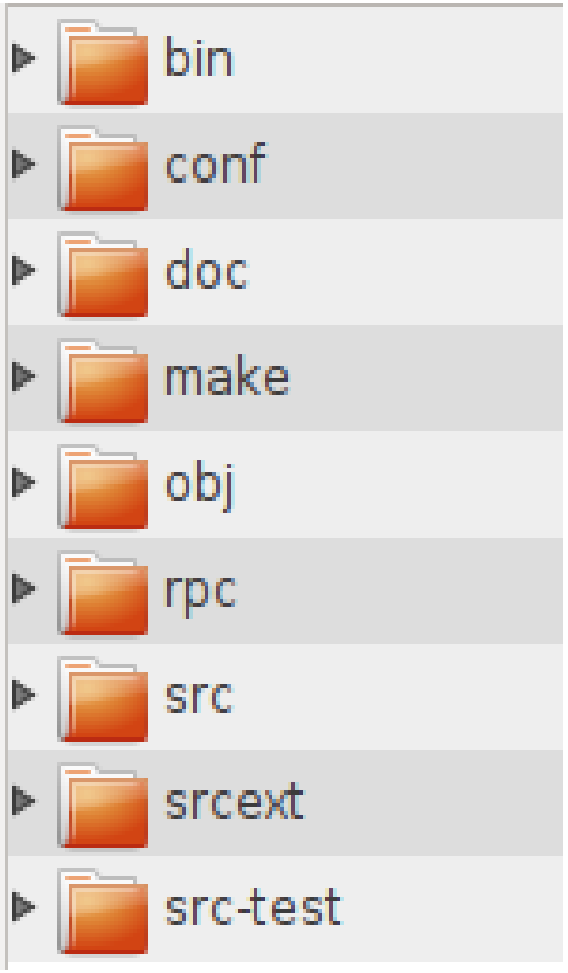
```
martin@martin:~/Desktop/simple_testsuite/make$ ../bin/testrunner
Testsimple_string_util::trim_left
Testsimple_string_util::trim_right
Testsimple_string_util::trim_left_and_right
Testsimple_string_util::strLeftTrim
Testsimple_string_util::strRightTrim
Testsimple_string_util::strRightAndLeftTrim
Testsimple_string_util::strShowCtrlChars
Testsimple_string_util::bHasOnlyAlphabeticChars
Testsimple_string_util::bHasOnlyNumericChars
Testsimple_string_util::strRemoveAllTabs
Testsimple_string_util::strRemoveAllLineFeeds
Testsimple_string_util::strRemoveAllTabsAndLineFeeds
Testsimple_string_util::bStartsNumericChar
Testsimple_string_util::strCloneAndConcat
Testsimple_string_util::strRevert
Testsimple_string_util::strToUpper
Testsimple_string_util::strToLower


Testing Complete
Number of tests: 17
Number of todos: 0
Tests failed: 0
```

Here are all tests listed that have been implemented to show you how it works. The tests reside in */src-test/test_simple_string_util.cpp*

After the run, a short information is presented on how many tests have been performed.

The folder structure of the simple_testsuite

simple_testsuite folder contains subfolders:



Testrunner [executable]

config files [if needed]

Documentation [generated with doxygen]

Linux Makefile

Objects (*.o -files)

Files generated by idl2rpc to test

Source files of the testsuite

Here we import (with svn-externals) the classes we want to test.

The tests, each class gets its own testclass

How to use simple_testsuite

• In folder *srcext/simple_string_util* there we have a class (simple_string_util) we intend to test. The class contains several functions to modify strings.
• We create a file *test_simple_string_util.cpp* and store it in folder *src-test.. This file must contain a class that has the following basic structure:*

```
#include <string>
#include "simple_testsuite.h"
class Testsimple_testsuite : public TestFixture
{
      public:
         Testsimple_testsuite() : TestFixture("Testsimple_testsuite")
         { }
      private:
         void run()
         {
            TEST_CASE(strToLower)
         }

        void strToLower()
        {
            // put here your testcode using ASSERT_EQUALS()
        }
};
REGISTER_TEST(Testsimple_testsuite)
```

Name of the test (can be freely choosen)

Register a test using the TEST_CASE macro

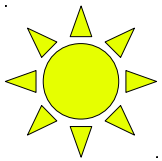
How the ASSERT_EQUALS work?

Lets take a look at the unit-test-function simple_string_util: strToUpper()

```
void strToUpper()
{
    ASSERT_EQUALS("XYZ" , simple_string_util::strToUpper("xyz"));
}
```

Expected result!

Call of a function to test, here strToUpper()



The ASSERT_EQUALS – macro accepts also integers and boolians.!

Take a look at the next page to see what happens if an assertion fails!

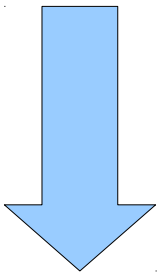
How the asserts work?

If an assertion fails, it will be reported at the end of the run of the testrunner:

```
void strToUpper()
{
    ASSERT_EQUALS("XYZx" , simple_string_util::strToUpper("xyz"));
}
```

Expected result!

Call of a function to test, here strToUpper()



Output of the testrunner:

```
Testing Complete
Number of tests: 17
Number of todos: 0
Tests failed: 1
Assertion failed in ../src-test/test_simple_string_util.cpp at line 175
Expected:
"XYZx"
Actual:
"XYZ"
```

For demonstration:
Expect a wrong result. In other words, the function strToUpper does not behave as expected!

How to create TODO_ASSERTS and whats for?

• **TODO_ASSERTS** can be used to create a test that your testfunction does not support. This can also be used in a case when there is a bug found in a function.

```
void TestCase()
{
    TODO_ASSERT_EQUALS(„expected result" , simple_string_util::FunctionWithABug());
}
```

Expected result!

Call of a function to test, here strToUpper()

Output of the testrunner:

```
Testing Complete
Number of tests: 18
Number of todos: 1
Tests failed: 0
```

You get a message about a new TODO

General information:

- We have tested this testsuite on several Linux platforms such as Debian/RedHat and Ubuntu
- All compiler versions from gcc-2.95 to gcc-4.5 work with the suite. There where also successfull tests with icc (intel compiler and clang++)
- This testsuite is used to test our basic c++-toolset of our software repository in Wettzell. It is contains routines for VLBI/SLR/LLR and system monitoring.
- It will be available to download at our University homepage, soon.
- Please keep us posted if you have suggestions for improvements, or if you find bugs!

Please contact us at:
ettl@fs.wettzell.de or ettl.martin78@googlemail.com
neidhardt@fs.wettzell.de