libAHtest Reference Manual

0.1

Generated by Doxygen 1.4.4

Fri Apr 7 01:18:05 2006

Contents

Chapter 1

libAHtest Directory Hierarchy

1.1 libAHtest Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 2

libAHtest Namespace Index

2.1 libAHtest Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

libAHtest Class Index

3.1 libAHtest Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

libAHtest File Index

4.1 libAHtest File List

Here is a list of all files with brief descriptions:

Chapter 5

libAHtest Directory Documentation

5.1 AH/ Directory Reference

Directories

- directory Test

5.2 AH/Test/ Directory Reference

Files

- file Log.h
- file UnitTest.h

Chapter 6

libAHtest Namespace Documentation

6.1 AH Namespace Reference

Classes

- class TestLog
- class UnitTest

6.2 std Namespace Reference

Chapter 7

libAHtest Class Documentation

7.1 AH::TestLog Class Reference

`#include <Log.h>`

Public Member Functions

- TestLog (ostream &o, const std::string &name, bool v)
- void announceSeries (const std::string &msg)
- void announceTest (const std::string &msg="")
- template<class T> void returned (const T &value)
- template<class T> void failed (const T &expected)
- void passed ()
- int makeReport ()
- unsigned int getErrNum () const
- ~TestLog ()

Protected Member Functions

- bool isVerbose () const

Protected Attributes

- unsigned int errNum

7.1.1 Detailed Description

test logger utility.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 AH::TestLog::TestLog (ostream & *o*, const std::string & *name*, bool *v*) `[inline]`

constructor.

7.1.2.2 AH::TestLog::~TestLog () [inline]

destructor.

7.1.3 Member Function Documentation

7.1.3.1 void AH::TestLog::announceSeries (const std::string & *msg*) [inline]

announce a new test series.

7.1.3.2 void AH::TestLog::announceTest (const std::string & *msg* = "") [inline]

announce a new test.

7.1.3.3 template<class T> void AH::TestLog::failed (const T & *expected*) [inline]

print failure message, with expected value.

7.1.3.4 unsigned int AH::TestLog::getErrNum () const [inline]

return number of detected errors.

7.1.3.5 bool AH::TestLog::isVerbose () const [inline, protected]

7.1.3.6 int AH::TestLog::makeReport () [inline]

report results.

7.1.3.7 void AH::TestLog::passed () [inline]

print passed message.

7.1.3.8 template<class T> void AH::TestLog::returned (const T & *value*) [inline]

print returned value.

7.1.4 Member Data Documentation

7.1.4.1 unsigned int AH::TestLog::errNum [protected]

The documentation for this class was generated from the following file:

- AH/Test/Log.h

7.2 AH::UnitTest Class Reference

`#include <UnitTest.h>`

Public Member Functions

- void setVerbose (bool onoff)
- void setQuiet (bool onoff)
- void setDebugLevel (int level)
- void printHelp (const char *progName)
- virtual void run (const std::vector< std::string > &args)=0
- bool testsFailed () const
- virtual ~UnitTest ()

Static Public Member Functions

- template<class T> static std::string asString (T t)

Protected Member Functions

- UnitTest (const std::string &name, std::ostream &Out=std::cout)
- bool isVerbose () const
- bool isQuiet () const
- int getDebugLevel () const
- virtual void printUserHelp ()
- void announceMessage (const std::string &msg)
- virtual void announceTestSeries (const std::string &series)
- virtual void announceNextTest (const std::string &target)
- virtual void printFailed ()
- template<class T1, class T2> void printFailed (const T1 &got, const T2 &expected)
- virtual void printPassed ()
- template<class T> void printPassed (const T &got)
- virtual void printException (const std::exception &ex)
- virtual void printException (const std::string &msg)
- virtual void printException (const char *msg)
- virtual void printExceptionFailed (const std::string &name)
- template<class A> void checkTrue (const std::string &name, const A &assertion)
- template<class T1, class T2> void checkEqual (const std::string &name, const T1 &got, const T2 &expected)
- template<class T> void checkEqual (const std::string &msg, const char *got, const T &expected)
- template<class T> void checkEqual (const std::string &msg, const T &got, const char *expected)
- void checkEqual (const std::string &msg, const char *got, const char *expected)
- template<class T1, class T2> void checkEqual (const std::string &msg, const T1 &got, const T2 &expected, double eps)

Protected Attributes

- std::ostream & out
- std::string addOptions
- std::string addArguments

7.2.1 Detailed Description

the base class for unit tests.

Class UnitTest is the base class for unit tests. It is used as follows:

- You must derive a sub class and define the test functions which are called by the pure virtual function run().
- Also run() has to be implemented.

But both is quite simple, see the following example. Lets assume you have written a function int square(int) which returns the square of its argument, and you want to test it:

```
#include "square.h"
#include <AH/Test/UnitTest.h>
using namespace AH;

// this is your test class:
class Test: public UnitTest
{
    // your test functions, to be called in Test::run():
    void test_valid_args();
    void test_invalid_args();

    // your function to run all your tests:
    virtual void run(const std::vector<std::string>&)
    {
        this->test_valid_args();
        this->test_invalid_args();
    }

 public:
    // your constructor - use cout for all messages
    Test(): UnitTest("function square()", std::cout) {}
};

// this is your single instance of your class Test
UnitTest* theUnitTest =  new Test();

// your test function for testing valid arguments:
void Test::test_valid_args()
{
    this->announceTestSeries("test_valid_arguments");

    // run the tests: we expect no exceptions
    UNIT_TEST_BEGIN
    this->checkEqual("square(1)", square(1), 1);
    this->checkEqual("square(2)", square(2), 4);
    this->checkEqual("square(3)", square(3), 9);
    this->checkEqual("square(-1)", square(-1), 1);
    this->checkEqual("square(-2)", square(-2), 4);
    this->checkEqual("square(-3)", square(-3), 9);
    UNIT_TEST_END
}

// your test function for testing invalid arguments:
void Test::test_invalid_args()
{
    this->announceTestSeries("test_invalid_arguments");

    // run the test: we expect an exception of type SquareNullArg here
    UNIT_TEST_EXCEPTION_BEGIN(SquareNullArg)
    square(0);
    UNIT_TEST_EXCEPTION_END(SquareNullArg)
```

```
}
```

Yes, this is all the code you have to write. No need to write a main() function, or to parse command line arguments, or to write try-catch blocks. All this is done by class UnitTest, the already implemented main() function and the macros.

The following command line options are already handled:

- -q = quiet mode: only two lines message are written at all.
- -v = verbose mode: print all necessary information for bug search.
- -d = increase debug level: print additional debug messages
- -h = help: print help message.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 virtual AH::UnitTest::~UnitTest () [virtual]

destructor.

7.2.2.2 AH::UnitTest::UnitTest (const std::string & *name*, std::ostream & *Out* = std::cout) [protected]

constructor.

Parameters:

name the name of the test target, e.g. "class foo" or "function bar".

Out the output stream for all messages. Defaults to cout if omitted.

7.2.3 Member Function Documentation

7.2.3.1 void AH::UnitTest::announceMessage (const std::string & *msg*) [inline, protected]

print message followed by CR/LF (not in quiet mode though).

Parameters:

msg the message to be printed.

7.2.3.2 virtual void AH::UnitTest::announceNextTest (const std::string & *target*) [inline, protected, virtual]

announce next test (not in quiet mode though).

announceNextTest() prints

- "Test <num>: <target> :" in verbose mode
- "Test <num>: " in normal mode

- nothing in quiet mode.

The test number is always incremented.

You may call announceNextTest() before a call of a function which might throw an exception. In this case call printPassed() behind the function call.

Parameters:

target the name of the test target, e.g. "foo()"

7.2.3.3 virtual void AH::UnitTest::announceTestSeries (const std::string & *series*) [inline, protected, virtual]

announce new test series (not in quiet mode though).

announceTestSeries() prints "— test of 'series' —" followed by CR/LF, but does nothing when in quiet mode.

Parameters:

series the name of the test series

7.2.3.4 template<class T> static std::string AH::UnitTest::asString (T *t*) [inline, static]

utility function: return any type as string.

asString() returns any type as std::string if the operator<<() is implemented for this type.

Parameters:

t the type to be returned as string

7.2.3.5 template<class T1, class T2> void AH::UnitTest::checkEqual (const std::string & *msg*, const T1 & *got*, const T2 & *expected*, double *eps*) [inline, protected]

check the floating point arguments for equality (==) within epsilon range: print fail if not, otherwise passed.

This is a specialized version of checkEqual() for comparison of floating point types within a specified epsilon range.

Parameters:

msg the description of the test

got the result of a test

expected the expected result of a test

eps the size of the epsilon range

7.2.3.6 void AH::UnitTest::checkEqual (const std::string & *msg*, const char * *got*, const char * *expected*) [inline, protected]

check the arguments for equality (==): print fail if not, otherwise passed.

This is just a specialized version of checkEqual() for character pointers. Visual C++ seems to need this.

Parameters:

msg the description of the test

got the result of a test

expected the expected result of a test

7.2.3.7 template<class T> void AH::UnitTest::checkEqual (const std::string & *msg*, const T & *got*, const char * *expected*) [inline, protected]

check the arguments for equality (==): print fail if not, otherwise passed.

This is just a partly specialized version of checkEqual() for character pointers. Visual C++ seems to need this.

Parameters:

msg the description of the test

got the result of a test

expected the expected result of a test

7.2.3.8 template<class T> void AH::UnitTest::checkEqual (const std::string & *msg*, const char * *got*, const T & *expected*) [inline, protected]

check the arguments for equality (==): print fail if not, otherwise passed.

This is just a partly specialized version of checkEqual() for character pointers. Visual C++ seems to need this.

Parameters:

msg the description of the test

got the result of a test

expected the expected result of a test

7.2.3.9 template<class T1, class T2> void AH::UnitTest::checkEqual (const std::string & *name*, const T1 & *got*, const T2 & *expected*) [inline, protected]

check the arguments for equality (==): print fail if not, otherwise passed

checkEqual() checks whether 'got' is equal to 'expected'. If both are equal it

- prints "Test <num>: <msg>: got=1: passed." followed by CR/LF in verbose mode
- prints "Test <num>: passed." followed by CR/LF in normal mode

If they are not equal it

- prints "Test <num>: <msg>: got=<got>: expected=<expected>: failed." followed by CR/LF in verbose mode
- prints "Test <num>: failed." followed by CR/LF in normal mode
- increments the number of failures.

In quiet mode nothing is printed.

Call this function if you want to test two parameters for equality. You can do this in all cases where

- the operator==() is implemented for both types, and
- the operator<<() is implemented for both types.

If these operators are not implemented you must resort to checkTrue().

Parameters:

name the name of the test

got the result of a test

expected the expected result of a test

7.2.3.10 template<class A> void AH::UnitTest::checkTrue (const std::string & *name*, const A & *assertion*) [inline, protected]

check the assertion for true: print fail if not, otherwise passed.

checkTrue() checks the assertion 'A'. If 'A' is true it

- prints "got=1: passed." followed by CR/LF in verbose mode
- prints "passed." followed by CR/LF in normal mode

If 'A' is false it

- prints "got=0: expected=1: failed." followed by CR/LF in verbose mode
- prints "failed." followed by CR/LF in normal mode
- increments the number of failures.

In quiet mode nothing is printed.

Call this function if you want to test a statement as true, but cannot call checkEqual().

Parameters:

name the name of the test

assertion the assertion to be tested for true

7.2.3.11 int AH::UnitTest::getDebugLevel () const [inline, protected]

return debug level.

7.2.3.12 bool AH::UnitTest::isQuiet () const [inline, protected]

return true if quiet mode is enabled.

7.2.3.13 bool AH::UnitTest::isVerbose () const [inline, protected]

return true if verbose mode is enabled.

7.2.3.14 virtual void AH::UnitTest::printException (const char * *msg*) [inline, protected, virtual]

print "Test <num>: caught exception: <msg>: failed. "followed by CR/LF.

printException() prints

- "Test <num>: caught exception <msg>: failed." followed by CR/LF in verbose mode
- "Test <num>: failed." followed by CR/LF in normal mode
- nothing in quiet mode.

The number of failures is incremented always. This function need not be called by the user.

Parameters:
msg the message of the exception

7.2.3.15 virtual void AH::UnitTest::printException (const std::string & *msg*) [inline, protected, virtual]

print "Test <num>: caught exception: <msg>: failed. "followed by CR/LF.

printException() prints

- "Test <num>: caught exception <msg>: failed." followed by CR/LF in verbose mode
- "Test <num>: failed." followed by CR/LF in normal mode
- nothing in quiet mode.

The number of failures is incremented always. This function need not be called by the user.

Parameters:
msg the message of the exception

7.2.3.16 virtual void AH::UnitTest::printException (const std::exception & *ex*) [inline, protected, virtual]

print "Test <num>: caught exception <exception>: failed. "followed by CR/LF.

printException() prints

- "Test <num>: caught exception <exception.what()>: failed." followed by CR/LF in verbose mode
- "Test <num>: failed." followed by CR/LF in normal mode
- nothing in quiet mode.

The number of failures is incremented always. This function need not be called by the user.

Parameters:
ex any exception derived from std::exception

7.2.3.17 virtual void AH::UnitTest::printExceptionFailed (const std::string & *name*) [inline, protected, virtual]

print "missing exception <exception>: failed. "followed by CR/LF.

printExceptionFailed() prints

- "missing exception <name>: failed." followed by CR/LF in verbose mode
- "failed." followed by CR/LF in normal mode
- nothing in quiet mode.

The number of failures is incremented always. This function need not be called by the user.

Parameters:

name the name of the exception

7.2.3.18 template<class T1, class T2> void AH::UnitTest::printFailed (const T1 & *got*, const T2 & *expected*) [inline, protected]

print "got=<got>, expected=<expected>. failed." followed by CR/LF.

printFailed() prints

- "got=<got>, expected=<expected>. failed." followed by CR/LF in verbose mode
- "failed." followed by CR/LF in normal mode
- nothing in quiet mode.

The number of failures is incremented always. Usually this function need not be called by the user.

Parameters:

got the result of a test

expected the expected result of a test

7.2.3.19 virtual void AH::UnitTest::printFailed () [inline, protected, virtual]

print "failed" followed by CR/LF (not in quiet mode though).

printFailed() prints "failed." followed by CR/LF, but not when in quiet mode. The number of failures is incremented always.

Usually this function need not be called by the user.

7.2.3.20 void AH::UnitTest::printHelp (const char * *progName*)

print help message - called by main().

7.2.3.21 template<class T> void AH::UnitTest::printPassed (const T & *got*) [inline, protected]

print "got=<got>. passed." followed by CR/LF.

printPassed() prints

- "got=<got>: passed." followed by CR/LF in verbose mode
- "passed." followed by CR/LF in normal mode
- nothing in quiet mode.

Usually this function need not be called by the user.

Parameters:
got the result of a test

7.2.3.22 virtual void AH::UnitTest::printPassed () [inline, protected, virtual]

print "passed." followed by CR/LF.

printPassed() prints "passed." followed by CR/LF, but not when in quiet mode.

You may call printPassed() behind a call of a function which might throw an exception. In this case call announceNextTest() before the function call.

7.2.3.23 virtual void AH::UnitTest::printUserHelp () [inline, protected, virtual]

print user defined help message.

printUserHelp() is called by printHelp() when the test program is called with option -h. Override it in your Test class if you want to add more help lines.

7.2.3.24 virtual void AH::UnitTest::run (const std::vector< std::string > & *args*) [pure virtual]

run the tests.

7.2.3.25 void AH::UnitTest::setDebugLevel (int *level*) [inline]

set debug mode to specified level - called by main().

7.2.3.26 void AH::UnitTest::setQuiet (bool *onoff*) [inline]

set quite mode on or off - called by main().

7.2.3.27 void AH::UnitTest::setVerbose (bool *onoff*) [inline]

set verbose mode on or off - called by main().

7.2.3.28 bool AH::UnitTest::testsFailed () const [inline]

return true if any of the tests failed - called by main().

7.2.4 Member Data Documentation

7.2.4.1 std::string AH::UnitTest::addArguments [protected]

additional command line arguments.

7.2.4.2 std::string AH::UnitTest::addOptions [protected]

additional command line options.

7.2.4.3 std::ostream& AH::UnitTest::out [protected]

output stream for all messages.

The documentation for this class was generated from the following file:

- AH/Test/UnitTest.h

Chapter 8

libAHtest File Documentation

8.1 AH/Test/Log.h File Reference

```
#include <AH/Config/Platform.h>
#include <string>
#include <iostream>
```

Namespaces

- namespace AH
- namespace std

Classes

- class AH::TestLog

8.2 AH/Test/UnitTest.h File Reference

```
#include <AH/Config/Platform.h>
#include <vector>
#include <string>
#include <iostream>
#include <sstream>
#include <exception>
#include <cmath>
```

Namespaces

- namespace AH

Classes

- class AH::UnitTest

Defines

- #define UNIT_TEST_STANDARD_CATCHES
- #define UNIT_TEST_PRINT_FILE_POSITION
- #define UNIT_TEST_BEGIN
- #define UNIT_TEST_END
- #define UNIT_TEST_EXCEPTION_BEGIN(ExpectedException)
- #define UNIT_TEST_EXCEPTION_END(ExpectedException)

Functions

- int main (int argc, const char *argv[ ])

Variables

- UnitTest * theUnitTest

8.2.1 Define Documentation

8.2.1.1 #define UNIT_TEST_BEGIN

Value:

```
try { \
        UNIT_TEST_PRINT_FILE_POSITION
```

macro: mark begin of normal test.

Call this macro before normal tests where no exceptions are expected to be thrown. It opens a new block, so all instances and variables created below are destroyed with the macro UNIT_TEST_END.

8.2.1.2 #define UNIT_TEST_END

Value:

```
} \
    UNIT_TEST_STANDARD_CATCHES
```

macro: mark end of normal test.

Call this macro behind normal tests where no exceptions are expected to be thrown. It closes the block opened with the macro UNIT_TEST_BEGIN, and destroys all instances and variables created behind UNIT_TEST_BEGIN above.

8.2.1.3 #define UNIT_TEST_EXCEPTION_BEGIN(ExpectedException)

Value:

```
try { \
        std::string unitTestExceptionName = #ExpectedException; \
        UNIT_TEST_PRINT_FILE_POSITION \
        this->announceNextTest("expect throw " + unitTestExceptionName);
```

macro: mark begin of test for expected exception.

Call this macro before tests where a specified exception is expected to be thrown. It opens a new block, so all instances and variables created below are destroyed with the macro UNIT_TEST_EXCEPTION_-END.

Parameters:

ExpectedException the expected exception

8.2.1.4 #define UNIT_TEST_EXCEPTION_END(ExpectedException)

Value:

```
this->printExceptionFailed(unitTestExceptionName); \
    } \
    catch(const ExpectedException&) { this->printPassed(#ExpectedException); } \
    UNIT_TEST_STANDARD_CATCHES
```

macro: mark end of test for expected exception.

Call this macro behind tests where a specified exceptions is expected to be thrown. It closes the block opened with the macro UNIT_TEST_EXCEPTION_BEGIN, and destroys all instances and variables created behind UNIT_TEST_EXCEPTION_BEGIN above.

Parameters:

ExpectedException the expected exception

8.2.1.5 #define UNIT_TEST_PRINT_FILE_POSITION

Value:

```
if (this->getDebugLevel()) \
        this->out << "at " << __FILE__ << " line " << __LINE__ << std::endl;
```

8.2.1.6 #define UNIT_TEST_STANDARD_CATCHES

Value:

```
catch(const std::exception& ex) { this->printException(ex); } \
    catch(...) { this->printException("unknown exception"); }
```

8.2.2 Function Documentation

8.2.2.1 int main (int *argc*, const char * *argv*[ ])

8.2.3 Variable Documentation

8.2.3.1 UnitTest* theUnitTest

Index