An Introduction to Writing Unit Tests (with CFUnit)

Posted At : July 2, 2008 10:00 AM | Posted By : Steve
Related Categories: ColdFusion, Testing

In our last **user's group** meeting, we had a round table discussion about testing. I said that I would put together a simple example of a unit test.

Since I have been using **CFUnit** so far (**MXUnit** looks attractive as well though), that is what I will use for my example. The principles should work the same for any common testing framework.

First I will download CFUnit and install it. This should install a folder "/net/sourceforge/cfunit/". This could either be in the root of the web site or in the CustomTags folder or could be in a folder with an appropriate mapping. The important thing is that components can be called using the path "net.sourceforge.cfunit".

I am going to follow my best understanding of **Test Driven Development** to create a component that does absolute rounding on a number. In short, this means that I will come up with my tests first and then write my code to get them to pass.

The first thing I will create is my test case component. This will start out empty.

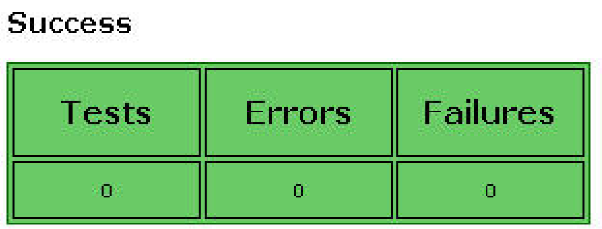
Here is TestAbsoluteRounder.cfc:

```
<cfcomponent displayname="Absolute Rounder Test Case" extends="net.sourceforge.cfunit.framework.TestCase">

</cfcomponent>
```

Next I will create the code to run all of the test methods (any methods that begin with "test") in this component.

```
<cfinvoke component="net.sourceforge.cfunit.framework.TestSuite"
   method="init"
   classes="TestAbsoluteRounder"
   returnvariable="testSuite"
/>

<cfinvoke component="net.sourceforge.cfunit.framework.TestRunner" method="run">
   <cfinvokeargument name="test" value="#testSuite#">
   <cfinvokeargument name="name" value="">
</cfinvoke>
```

This yields the following result:


Success

Tests	Errors	Failures
0	0	0

Now, I decide how to tell (externally) if my component does what I want. So, I decide on some criteria for success:

- The "absround" method will return zero not passed an argument.
- The "absround" method will return zero if the argument is not numeric.
- The number returned by the "absround" method will always be zero or higher.
- The number returned by the "absround" method will always be an integer.
- If the argument has 5 or more as the first digit after the decimal, then the returned value should have a greater absolute value than the argument.
- If the argument has 4 or less as the first digit after the decimal, then the returned value should have a lower absolute value than the argument.
- If the argument is an integer, then the returned value should be the same as the absolute value of the argument.

If the component passes all of these tests then it performs the desired function.

Next I will create a setup method in my test component to call the AbsoluteRounder.cfc component and create the empty AbsoluteRounder.cfc.

setUp:

```
<cffunction name="setUp" access="public" returntype="void" output="no">

   <cfset variables.AbsoluteRounder = CreateObject("component","AbsoluteRounder")>

</cffunction>
```

AbsoluteRounder.cfc:

```
<cfcomponent displayname="Absolute Rounder" output="false">

</cfcomponent>
```

Now I add the first test:

```
<cffunction name="test_shouldNoArgumentReturnZero" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">

   <cfset result = variables.AbsoluteRounder.absround()>

   <cfset assertEquals("Passing no argument does not return zero.",result,0)>

</cffunction>
```

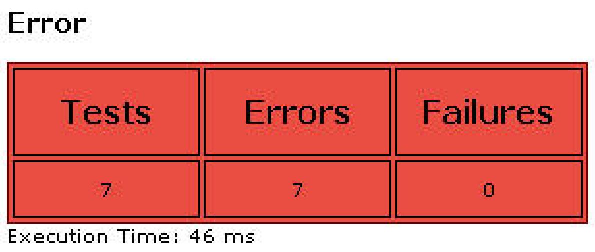
Note the user of the word "should". I got this idea from **Peter Bell** and I have found that it works well. The advantage is that the name of the tests describes the expected result. This naming conventions helps encourage me to write the sorts of tests I should be writing.

So, here is the completed test component:

```
<cfcomponent displayname="Absolute Rounder Test Case" extends="net.sourceforge.cfunit.framework.TestCase">

<cffunction name="setUp" access="public" returntype="void" output="no">

   <cfset variables.AbsoluteRounder = CreateObject("component","AbsoluteRounder")>

</cffunction>

<cffunction name="test_shouldNoArgumentReturnZero" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">

   <cfset result = variables.AbsoluteRounder.absround()>

   <cfset assertEquals("Passing no argument does not return zero.",result,0)>

</cffunction>

<cffunction name="test_shouldNonNumericArgumentReturnZero" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">

   <cfset result = variables.AbsoluteRounder.absround("text")>

   <cfset assertEquals("Passing non-numeric argument does not return zero.",result,0)>

</cffunction>
```

```
<cffunction name="test_shouldResultBePositive" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">
   <cfset var args = "0.001992223,-11992,-11922.3443,234234,5433.433,-332.43">
   <cfset var arg = "">

   <cfloop list="#args#" index="arg">
      <cfset result = variables.AbsoluteRounder.absround(arg)>
      <cfif NOT ( result GTE 0 )>
         <cfset fail("Result is not positive or zero.")>
      </cfif>
   </cfloop>

</cffunction>

<cffunction name="test_shouldResultBeInteger" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">
   <cfset var args = "0.001992223,-11992,-11922.3443,234234,5433.433,-332.43">
   <cfset var arg = "">

   <cfloop list="#args#" index="arg">
      <cfset result = variables.AbsoluteRounder.absround(arg)>
      <cfset assertEquals("Result is not an integer.",result,Int(result))>
   </cfloop>

</cffunction>

<cffunction name="test_shouldHalfOverIncreaseAbsValue" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">
   <cfset var args = "0.50099222,-0.500993,23345.59998,-22334.556">
   <cfset var arg = "">

   <cfloop list="#args#" index="arg">
      <cfset result = variables.AbsoluteRounder.absround(arg)>
      <cfif NOT ( Abs(result) GT arg )>
         <cfset fail("Result not greater than argument for first decimal of 5.")>
      </cfif>
   </cfloop>

</cffunction>

<cffunction name="test_shouldNotHalfOverIncreaseAbsValue" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">
   <cfset var args = "0.40099222,-0.400993,23345.49998,-22334.456,0.009,-0.009">
   <cfset var arg = "">

   <cfloop list="#args#" index="arg">
      <cfset result = variables.AbsoluteRounder.absround(arg)>
      <cfif NOT ( Abs(result) LT Abs(arg) )>
         <cfset fail("Result not less than argument for first decimal less of 5.")>
      </cfif>
   </cfloop>

</cffunction>

<cffunction name="test_shouldIntegerMatchAbsValue" access="public" returntype="any" output="false" hint="">

   <cfset var result = "">
   <cfset var args = "0,23345,-22334">
   <cfset var arg = "">
   <cfloop list="#args#" index="arg">
      <cfset result = variables.AbsoluteRounder.absround(arg)>
      <cfset assertEquals("Absolute values of arg and result should match for integer arguments.",Abs(arg),Abs(result))>
   </cfloop>

</cffunction>

</cfcomponent>
```

The result of this:



Error

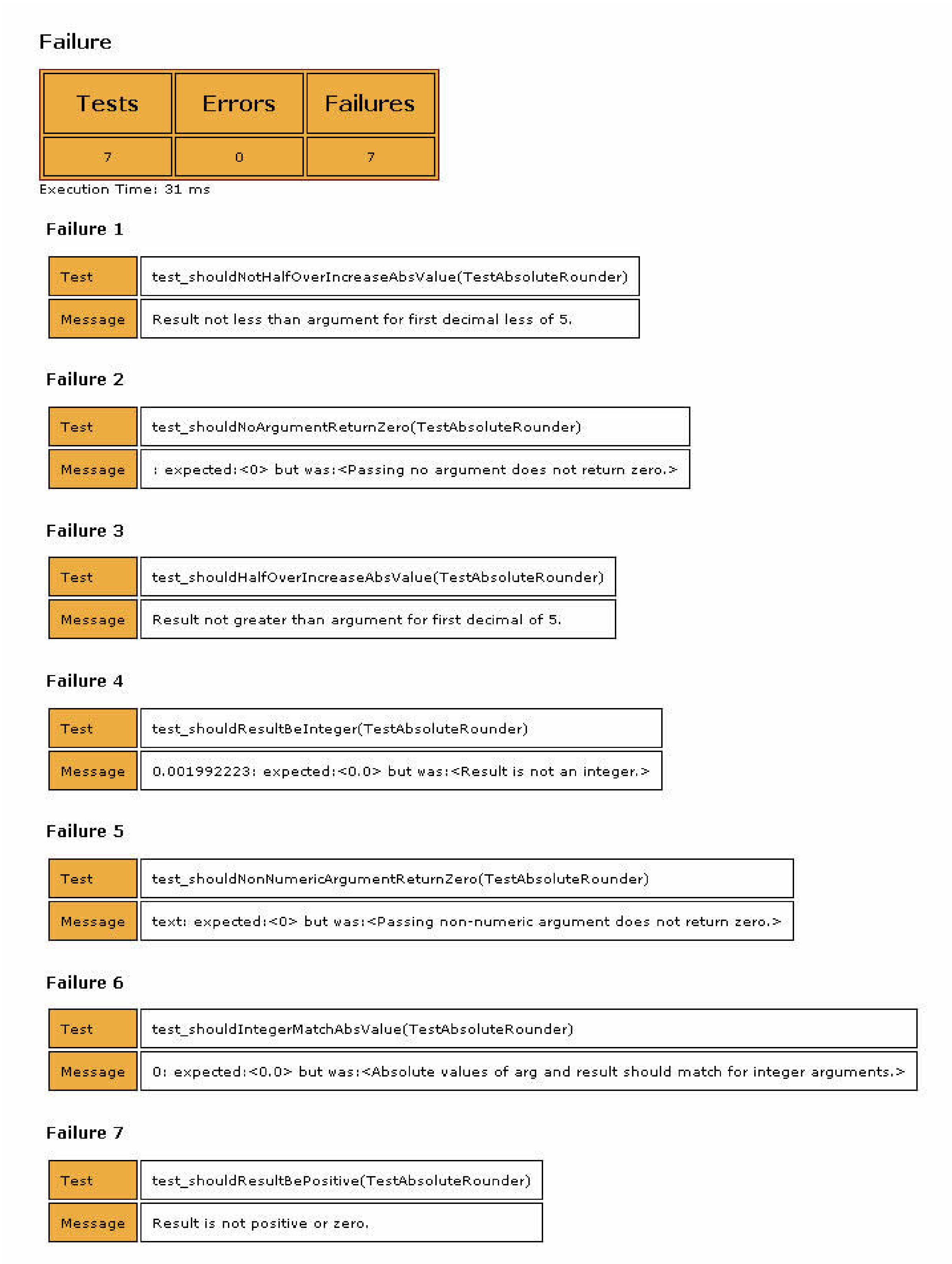
Tests	Errors	Failures
7	7	0

Execution Time: 46 ms

I got errors because I haven't even created the method yet. So, I will create a method:

```
<cffunction name="absround" access="public">
   <cfargument name="number" type="string" default="">

   <cfreturn arguments.number>
</cffunction>
```

Now my tests return a result of this:



Failure

Tests	Errors	Failures
7	0	7

Execution Time: 31 ms

Failure 1

Test	test_shouldNotHalfOverIncreaseAbsValue(TestAbsoluteRounder)
Message	Result not less than argument for first decimal less of 5.

Failure 2

Test	test_shouldNoArgumentReturnZero(TestAbsoluteRounder)
Message	: expected:<0> but was:<Passing no argument does not return zero.>

Failure 3

Test	test_shouldHalfOverIncreaseAbsValue(TestAbsoluteRounder)
Message	Result not greater than argument for first decimal of 5.

Failure 4

Test	test_shouldResultBeInteger(TestAbsoluteRounder)
Message	0.001992223: expected:<0.0> but was:<Result is not an integer.>

Failure 5

Test	test_shouldNonNumericArgumentReturnZero(TestAbsoluteRounder)
Message	text: expected:<0> but was:<Passing non-numeric argument does not return zero.>

Failure 6

Test	test_shouldIntegerMatchAbsValue(TestAbsoluteRounder)
Message	0: expected:<0.0> but was:<Absolute values of arg and result should match for integer arguments.>

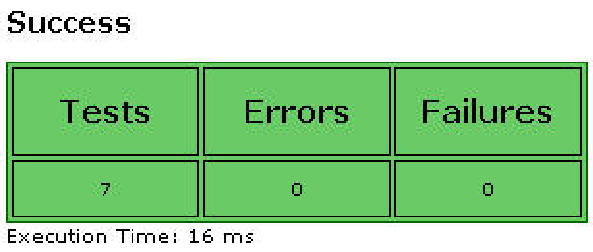
Failure 7

Test	test_shouldResultBePositive(TestAbsoluteRounder)
Message	Result is not positive or zero.

My method exists, so I don't get any errors. I still get failures, however, as it doesn't do what is needed.

Now I will write my method to achieve the desired result:

When I run my tests now, I get success:



Success

Tests	Errors	Failures
7	0	0

Execution Time: 16 ms

I think it is important to comment here that while I think this entry does a good job of showing *how* to write test cases and go through Test Driven Development, it does an awful job in demonstrating *why* you should go through that process. That will have to wait for another day.