Shouldly Documentation

Release 2.6.0

Dave Newman, Xerxes Battiwalla, Anthony Egerton, Peter van der

January 16, 2016

Contents

How asserting *Should* be

> **Attention:** These docs are in progress! Get involved at Learn more about on GitHub, contributions welcome! First time contributors welcome, we are happy to help you get started.

This is the old *Assert* way:

```
Assert.That(contestant.Points, Is.EqualTo(1337));
```

For your troubles, you get this message, when it fails:

```
Expected 1337 but was 0
```

How it **Should** be:

```
contestant.Points.ShouldBe(1337);
```

Which is just syntax, so far, but check out the message when it fails:

```
contestant.Points should be 1337 but was 0
```

It might be easy to underestimate how useful this is. Another example, side by side:

```
Assert.That(map.IndexOfValue("boo"), Is.EqualTo(2));     // -> Expected 2 but was 1
map.IndexOfValue("boo").ShouldBe(2);                     // -> map.IndexOfValue("boo") should be 2 but
```

Shouldly uses the variables within the *ShouldBe* statement to report on errors, which makes diagnosing easier.

Another example, if you compare two collections:

```
new[] { 1, 2, 3 }.ShouldBe(new[] { 1, 2, 4 });
```

and it fails because they're different, it'll show you the differences between the two collections:

```
    should be
[1, 2, 4]
    but was
[1, 2, 3]
    difference
[1, 2, *3*]
```

Shouldly has plenty of different assertions, have a look under the assertions folder for all the options.

CHAPTER 1

ShouldBe

1.1 Objects

`ShouldBe` works on all types and compares using `.Equals`.

```
var theSimpsonsCat = new Cat() { Name = "Santas little helper" };
theSimpsonsCat.Name.ShouldBe("Snowball 2");
```

Exception:

```
theSimpsonsCat.Name
    should be
"Snowball 2"
    but was
"Santas little helper"
```

1.2 Numeric

`ShouldBe` numeric overloads accept tolerances and has overloads for `float`, `double` and `decimal` types.

```
const decimal pi = (decimal)Math.PI;
pi.ShouldBe(3.24m, 0.01m);
```

Exception:

```
pi
   should be within
0.01
    of
3.24
    but was
3.14159265358979
```

1.3 DateTime(Offset)

DateTime overloads are similar to the numeric overloads and support tolerances.

```
var date = new DateTime(2000, 6, 1);
date.ShouldBe(new DateTime(2000, 6, 1, 1, 0, 1), TimeSpan.FromHours(1));
```

Exception:

```
date
   should be within
01:00:00
    of
1/06/2000 1:00:01 AM
    but was
1/06/2000 12:00:00 AM
```

1.4 TimeSpan

TimeSpan also has tolerance overloads

```
var timeSpan = TimeSpan.FromHours(1);
timeSpan.ShouldBe(timeSpan.Add(TimeSpan.FromHours(1.1d)), TimeSpan.FromHours(1));
```

Exception:

```
timeSpan
   should be within
01:00:00
    of
02:06:00
    but was
01:00:00
```

Want to improve shouldy? We have an open issue at [#303](https://github.com/shouldly/shouldly/issues/303) to improve this error message!

1.5 Enumerables

Enumerable comparison is done on the elements in the enumerable, so you can compare an array to a list and have it pass.

```
var apu = new Person() { Name = "Apu" };
var homer = new Person() { Name = "Homer" };
var skinner = new Person() { Name = "Skinner" };
var barney = new Person() { Name = "Barney" };
var theBeSharps = new List<Person>() { homer, skinner, barney };

theBeSharps.ShouldBe(new[] {apu, homer, skinner, barney});
```

Exception:

```
theBeSharps
    should be
[Apu, Homer, Skinner, Barney]
    but was
[Homer, Skinner, Barney]
    difference
[*Homer*, *Skinner*, *Barney*, *]
```

1.6 Enumerables of Numerics

If you have enumerables of `float`, `decimal` or `double` types then you can use the tolerance overloads, similar to the value extensions.

```
var firstSet = new[] { 1.23m, 2.34m, 3.45001m };
var secondSet = new[] { 1.4301m, 2.34m, 3.45m };
firstSet.ShouldBe(secondSet, 0.1m);
```

Exception:

```
firstSet
   should be within
0.1
    of
[1.4301, 2.34, 3.45]
    but was
[1.23, 2.34, 3.45001]
    difference
[*1.23*, 2.34, *3.45001*]
```

1.7 Bools

```
protected override void ShouldPass()
{
    true.ShouldBe(true);
}

protected override void ShouldThrowAWobbly()
{
    const bool myValue = false;
    myValue.ShouldBe(true, "Some additional context");
}
```

CHAPTER 2

ShouldNotBe

`ShouldNotBe` is the inverse of `ShouldBe`.

2.1 Objects

`ShouldNotBe` works on all types and compares using `.Equals`.

```
var theSimpsonsCat = new Cat() { Name = "Santas little helper" };
theSimpsonsCat.Name.ShouldNotBe("Santas little helper");
```

Exception:

```
theSimpsonsCat.Name
    should not be
"Santas little helper"
    but was
"Santas little helper"
```

Want to contribute to Shouldly? #304 makes this error message better!

2.2 Numeric

`ShouldNotBe` also allows you to compare numeric values, regardless of their value type.

```
const int one = 1;
one.ShouldNotBe(1)
```

Exception:

```
one should not be 1 but was 1
```

```
const long aLong = 1L;
aLong.ShouldNotBe(1);
```

Exception:

```
aLong should not be 1 but was 1
```

2.3 DateTime(Offset)

`ShouldNotBe` DateTime overloads are similar to the numeric overloads and also support tolerances.

```
var date = new DateTime(2000, 6, 1);
date.ShouldNotBe(new DateTime(2000, 6, 1, 1, 0, 1), TimeSpan.FromHours(1.5));
```

Exception:

```
date
    should not be within
01:30:00
    of
01/06/2000 01:00:01
    but was
01/06/2000 00:00:00
```

2.4 TimeSpan

`TimeSpan` also has tolerance overloads

```
var timeSpan = TimeSpan.FromHours(1);
timeSpan.ShouldNotBe(timeSpan.Add(TimeSpan.FromHours(1.1d)), TimeSpan.FromHours(1.5d));
```

Exception:

```
timeSpan
    should not be within
01:30:00
    of
02:06:00
    but was
01:00:00
```

Want to contribute to Shouldly? #303 makes this error message better!

CHAPTER 3

Example Classes

The classes used in these samples are:

```
namespace Simpsons
{
    public abstract class Pet
    {
        public abstract string Name { get; set; }

        public override string ToString()
        {
            return Name;
        }
    }
}

namespace Simpsons
{
    public class Cat : Pet
    {
        public override string Name { get; set; }
    }
}

namespace Simpsons
{
    public class Dog : Pet
    {
        public override string Name { get; set; }
    }
}

namespace Simpsons
{
    public class Person
    {
        public string Name { get; set; }
        public int Salary { get; set; }


        public override string ToString()
        {
            return Name;
        }
```

```
        }
}
```

CHAPTER 4

Contributing

Once you have cloned Shouldly to your local machine, the following instructions will walk you through installing the tools necessary to build and test the documentation.

1. Download python version 2.7.10 or higher.
2. If you are installing on Windows, add both the Python install directory and the Python scripts directory to your `PATH` environment variable. For example, if you install Python into the `c:\python34` directory, you would add `c:\python34;c:\python34\scripts` to your `PATH` environment variable.
3. Install Sphinx by opening a command prompt and running the following Python command. (Note that this operation might take a few minutes to complete.):

```
pip install sphinx
```

4. By default, when you install Sphinx, it will install the ReadTheDocs custom theme automatically. If you need to update the installed version of this theme, you should run:

```
pip install -U sphinx_rtd_theme
```

5. Run the *make.bat* file using *html* argument to build the stand-alone version of the project in question:

```
make html
```

6. Once make completes, the generated docs will be in the `_build/html` directory. Simply open the `index.html` file in your browser to see the built docs for that project.

4.1 Style Guidelines

Please review the following style guides:

- Sphinx Style Guide
- ASP.NET Docs Style Guide

CHAPTER 5

Indices and tables

- genindex
- modindex
- search