Shouldly Documentation

Release 2.6.0

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

Nov 06, 2018

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 was 1
```

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

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

Exception

1.2 Numeric

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

Exception

1.3 DateTime(Offset)

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

Exception

1.4 TimeSpan

TimeSpan also has tolerance overloads

Exception

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.

Exception

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.

Exception

1.7 Bools

Exception

CHAPTER 2

ShouldNotBe

`ShouldNotBe` is the inverse of `ShouldBe`.

2.1 Objects

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

Exception

2.2 Numeric

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

2.2.1 Integer

Exception

2.2.2 Long

Exception

2.3 DateTime(Offset)

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

Exception

2.4 TimeSpan

`TimeSpan` also has tolerance overloads

Exception

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

CHAPTER 3

ShouldMatchApproved

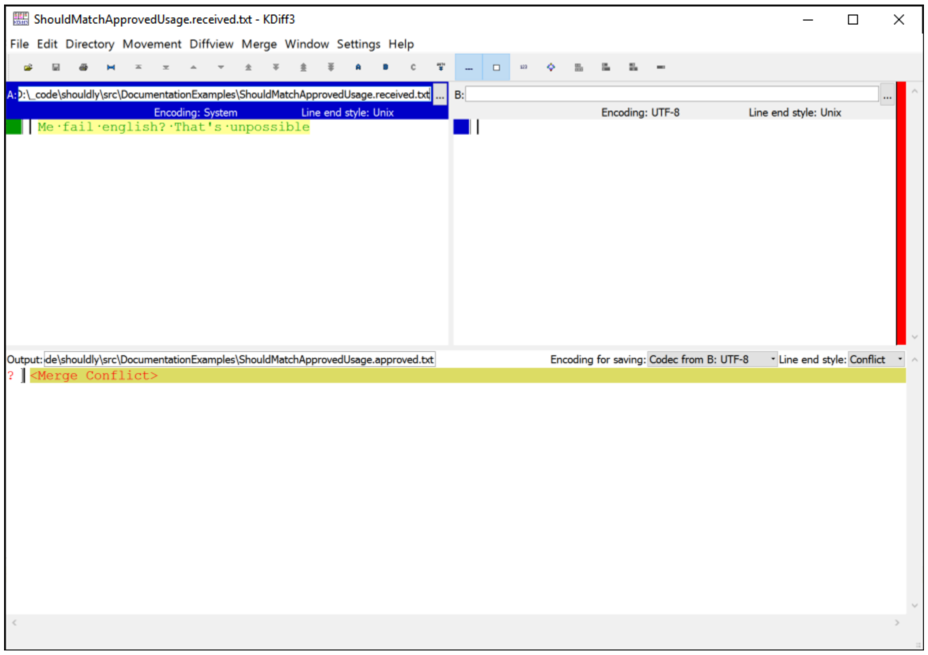
Based on the awesome ApprovalTest.Net, Shouldly has `ShouldMatchApproved()` to do approval based testing. The main goal of Shouldly's approval testing is for it to be simple, intuative and give great error messages.

3.1 Approved File does not exist

When you first run a `ShouldMatchApproved` test, you will be presented with a diff viewer and a failing test.

Exception

Screenshot


ShouldMatchApprovedUsage.received.txt - KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help

A: D:_code\shouldly\src\DocumentationExamples\ShouldMatchApprovedUsage.received.txt
Encoding: System   Line end style: Unix

```
Me·fail·english?·That's·unpossible
```

B:
Encoding: UTF-8   Line end style: Unix

Output: de\shouldly\src\DocumentationExamples\ShouldMatchApprovedUsage.approved.txt
Encoding for saving: Codec from B: UTF-8   Line end style: Conflict

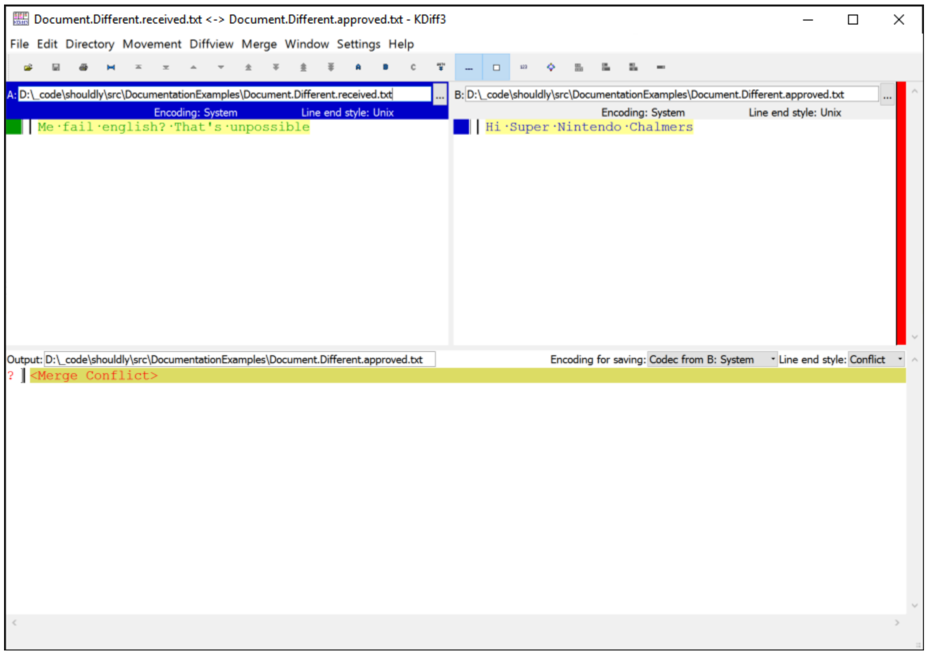
```
? <Merge Conflict>
```

3.2 Approved File does not match received

After you have approved the text, when it changes you get a different experience.

Exception

Screenshot


Document.Different.received.txt <-> Document.Different.approved.txt - KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help

A: D:_code\shouldly\src\DocumentationExamples\Document.Different.received.txt
Encoding: System Line end style: Unix

```
Me·fail·english?·That's·unpossible
```

B: D:_code\shouldly\src\DocumentationExamples\Document.Different.approved.txt
Encoding: System Line end style: Unix

```
Hi·Super·Nintendo·Chalmers
```

Output: D:_code\shouldly\src\DocumentationExamples\Document.Different.approved.txt
Encoding for saving: Codec from B: System Line end style: Conflict

```
? <Merge Conflict>
```

3.3 Options and customisation

While the defaults should work fine, often you need to customise things easily. ApprovalTests is highly configurable but the configuration is not always discoverable. Shouldly wants to make configuration simple and discoverable. This section covers the local customisations availble for a single ShouldMatchApproved call.

3.3.1 Defaults

The first thing to note is that by default **Shouldly ignores line endings**. This saves painful failures on the build server when git checks out the approved files with n rather than rn which the received file has. You can opt out of this behaviour for a single call, or globally. For global defaults see the Configuration section.

3.3.2 Usage

`toVerify.ShouldMatchApproved(configurationBuilder => configurationBuilder.OPTION())` where OPTION can be one of the following methods.

3.3.3 DoNotIgnoreLineEndings

Tells shouldly to use a line ending sensitive comparison.

```
toVerify.ShouldMatchApproved(c => c.DoNotIgnoreLineEndings())
```

3.3.4 WithStringCompareOptions

Sets the string comparison options

```
var options = StringCompareShould.IgnoreCase | StringCompareShould.IgnoreLineEndings;
toVerify.ShouldMatchApproved(c => c.WithStringCompareOptions(options))
```

3.3.5 WithDescriminator

By default the approved and received files are named `${MethodName}.approved.txt`, `WithDescriminator` allows you to descriminate multiple files, useful for data driven tests which can have multiple executions of a single method. For example

`[Fact] void Simpsons() { toVerify.ShouldMatchApproved(c => c.WithDescriminator("Bart")) }`

Will result in a approved file with the name `Simpsons.Bart.approved.txt`

3.3.6 NoDiff

Prevents the diff viewer from opening up. Doing this you can use Shouldly's error messages to verify the changes then run the command in the exception message to approve the changes.

`toVerify.ShouldMatchApproved(c => c.NoDiff())`

3.3.7 WithFileExtension

Override the file exension of the approved/received files. The default is .txt.

`toVerify.ShouldMatchApproved(c => c.WithFileExtension(".cs"))`

3.3.8 SubFolder

Put the approved/received files into a sub-directory

`toVerify.ShouldMatchApproved(c => c.SubFolder("Approvals"))`

3.3.9 UseCallerLocation

By default shouldly will walk the stacktrace to find the first non-shouldly method (not including anonymous methods and compiler generated stuff like the async state machine) and use that method for the approval filename. I.e a test named `MyTest` will result in a received filename of `MyTest.received.txt`.

This setting tells shouldly to walk one more frame, this is really handy when you have created a utility function which calls `ShouldMatchApproved`.

```
[Fact]
public void MyTest()
{
    SomeUtilityMethod("Foo");
}
```

(continues on next page)

(continued from previous page)

```
void SomeUtilityMethod(string toApprove)
{
    toApprove.ShouldMatchApproved(c => c.UseCallerLocation());
}

// -> MyTest.received.txt - without UseCallerLocation() the file would be called␣
↪SomeUtilityMethod.received.txt
```

3.3.10 LocateTestMethodUsingAttribute

If you want to locate your test method using an attribute that is easy too!

```
// XUnit
"testAttributes".ShouldMatchApproved(b => b.LocateTestMethodUsingAttribute
↪<FactAttribute>());
// NUnit
"testAttributes".ShouldMatchApproved(b => b.LocateTestMethodUsingAttribute
↪<TestAttribute>());
```

3.3.11 WithScrubber

Scrubbers allow you to remove dynamic content, such as the current date

`toVerify.ShouldMatchApproved(c => c.WithScrubber(s => Regex.Replace(s, "\d{1,2}/\d{1,2}/\d{2,4}", "<date>"))`

Will turn `Today is 01/01/2016` into `Today is <date>` in the received file.

3.4 Configuration

Because this feature is quite new shouldly doesn't have many Diff tools or know all the places it shouldn't open the diff tool. The global configuration of Shouldly is very easy to change and extend. If you do add a difftool or a should not open difftool strategy then please submit a pull request so everyone gets the benefits!

3.4.1 Changing default options

All of the instance based configuration can be changed globally through `ShouldlyConfiguration.ShouldMatchApprovedDefaults`. For example to make the default behaviour be line ending sensitive you can just run this before any tests execute `ShouldlyConfiguration.ShouldMatchApprovedDefaults.DoNotIgnoreLineEndings()`

3.4.2 Adding a difftool

So Shouldly doesn't support your favorite difftool yet. No worries, it's easy to add your own.

```
var diffomatic3000 = new DiffTool(
  "Diffomatic3000",
  @"diffomatic3000\diffomatic3000.exe",
```

(continues on next page)

(continued from previous page)

```
  (received, approved, approvedExists) => $"\{received}\" \"{approved}\"")
ShouldlyConfiguration.DiffTools.RegisterDiffTool(diffomatic3000);
```

This will discover diffomatic3000.exe if it's in your PATH or if it exists in any Program Files directory under diffomatic3000diffomatic3000.exe

If you do this, please submit a PR to add it to the `KnownDiffTools`, you can also test how it works by running the Shouldly.TestsTestDiffTools project!

3.4.3 Adding a do not launch difftool strategy

We don't really want to be opening difftools in nCrunch, or on the build server and a number of other scenarios. So `ShouldlyConfiguration.DiffTools.KnownDoNotLaunchStrategies` allows you to add in scenarios which Shouldly doesn't know about yet. Once again, please submit PR's if you need to do this :)

Currently the only strategy is to check for environmental variables, but you can implement `IShouldNotLaunchDiffTool` to implement any logic you want. Assuming it's just an environmental variable:

`ShouldlyConfiguration.DiffTools.AddDoNotLaunchStrategy(new DoNotLaunchWhenEnvVariableIsPresent("NCRUNCH"));`

3.4.4 Setting Diff tool priority

Shouldly launches the first found difftool, if you want to give priority to another difftool you can do that.

```
ShouldlyConfiguration.DiffTools.SetDiffToolPriorities(
  KnownDiffTools.Instance.BeyondCompare4,
  KnownDiffTools.Instance.KDiff3);
```

The priority tools will be checked before falling back to the entire known difftool list.

CHAPTER 4

ShouldBeTrue/False

`ShouldBeTrue` and `ShouldBeFalse` work on boolean values.

4.1 ShouldBeTrue

Exception

4.2 ShouldBeFalse

Exception

CHAPTER 5

ShouldBeNull/NotBeNull

`ShouldBeNull` and `ShouldNotBeNull` allow you to check whether or not a type's reference is null.

5.1 ShouldBeNull

Exception

5.2 ShouldNotBeNull

Exception

CHAPTER 6

ShouldHaveFlag/NotHaveFlag

`ShouldHaveFlag` allows you to assert whether an object is an enum and has a flag specified.

Conversely `ShouldNotHaveFlag` allows you to assert the opposite; that an object is an enum but does not have a flag specified.

6.1 ShouldHaveFlag

Exception

6.2 ShouldNotHaveFlag

Exception

CHAPTER 7

Example Classes

The classes used in these samples are:

```
using System;

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

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

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

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

    public class Person
    {
        public Person()
        {
        }

        public Person(string name)
        {
```

(continues on next page)

(continued from previous page)

```
            Name = name ?? throw new ArgumentNullException(nameof(name));
        }

        public string Name { get; set; }
        public int Salary { get; set; }


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

CHAPTER 8

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.

8.1 Style Guidelines

Please review the following style guides:

- Sphinx Style Guide
- ASP.NET Docs Style Guide

CHAPTER 9

Configuration

Shouldly has a few configuration options:

9.1 DefaultFloatingPointTolerance

Allows specifying a floating point tolerance for all assertions

Default value: 0.0d

9.2 DefaultTaskTimeout

`Should.Throw(Func<Task>)` blocks, the timeout is a safeguard for deadlocks.

Shouldly runs the lambda without a synchronisation context, but deadlocks are still possible. Use `Should.ThrowAsync` to be safe then await the returned task to prevent possible deadlocks.

Default value: 10 seconds

9.3 CompareAsObjectTypes

Types which also are IEnumerable of themselves.

An example is `Newtonsoft.Json.Linq.JToken` which looks like this `class JToken : IEnumerable<JToken>`.

Default value: Newtonsoft.Json.Linq.JToken

CHAPTER 10

Indices and tables

- genindex
- modindex
- search