

# Writing Tests

In a project like boo there are a lot of components and areas of functionality that need to be individually tested: the classes in the class library, the parser, the error checking code, the code emitter just to cite a few. The integration among these various components needs to be properly and thoroughly tested as well.

We use [NUnit](#) as our testing framework. If you don't know NUnit yet please take some time to [familiarize yourself with it](#). You'll be glad you did.

The current tests can always be found [here](#).

## Writing tests for the class library

Class library test cases are vanilla NUnit test fixture classes written in boo preferably.

The [Boo.Lang.List fixture](#) is a great example.

## Writing regression tests

Whenever a bug is found a set of test cases should be written to both prove the bug exists in the first place and make sure it will never pop up again after corrected.

A regression test case is simply a boo script that uses the [NUnit](#) API and is placed in the [tests/testcases/regression](#) directory. The script should be named after its related jira issue (which is also to say that **all bugs should be recorded in jira first**).

The following test case was saved as `BOO-46-1.boo` since it is related to the [BOO-46](#) bug.

```

import NUnit.Framework

class Foo:

    _prefix as object

    def constructor(prefix):
        _prefix = prefix

    def call(c as ICallable):
        return c()

    def foo():
        return "$_prefix - foo"

    def run():
        return call(foo)

Assert.AreEqual("bar - foo",
Foo("bar").run())

```

 *the test case just need to be in the `tests/testcases/regression` directory and it will be automatically picked up and run by the build.*

## Writing tests for the parser

Writing tests for the parser involves three simple steps:

- writing a boo module with the code you want to make sure the parser is able to handle correctly
- adding the [roundtrip form](#) of this same code as the module's [documentation string](#)
- saving the module in the [tests/testcases/parser/roundtrip](#) directory

Example:

```
"""
import Gtk from 'gtk-sharp'
import FooBar from 'foo-bar'
"""

import Gtk from "gtk-sharp"
import FooBar from 'foo-bar'
```

**i** the test case just need to be in the `tests/testcases/parsers/roundtrip` directory and it will be automatically picked up and run by the build.

## Writing error checking tests

A error checking test case is a boo module whose docstring contains all the errors that are expected to be reported by the compiler.

The test case must be saved in the [tests/testcases/errors](#) directory with a name in the form `<error code>-<sequential number>.boo`.

```
"""
BCE0005-1.boo(9,11): BCE0005: Unknown
identifier: local.
"""

def foo():
    local = "foo"
    bar()

def bar():
    print(local)
```

**i** the test case just need to be in the `tests/testcases/errors` directory and it will be automatically picked up and run by the build.

## Writing integration tests

Integration test cases test the whole compilation pipeline all the way from the parser to the emitter to the interaction between the generated code and the class libraries.

A integration test case can either use the NUnit API to assert the expected behavior of the generated program or include its expected output as the module's documentation string.

The test case should be named after the feature it tests and placed in the [tests/testcases/integration](#).

Take for instance [one of the integration test cases for the and operator](#):

```
"""
True
True
3
evaluated
False
evaluated
evaluated
True
True
0
"""
def fun(value):
    print('evaluated')
    return value

a = null and true
print(a is null)

b = true and 3
print(b isa int)
print(b)

c = fun(false) and fun(true)
print(c)
```

```
d = fun(true) and fun(null)
print(d is null)
```

```
e = 0 and false
print(e isa int)
print(e)
```

 *the test case just need to be in the tests/testcases/integration directory and it will be automatically picked up and run by the build.*