Cheatography

# Mocker to Sinon Conversion Cheat Sheet

by ajaska via cheatography.com/145268/cs/31268/

## Stub/Spy Equivalents

| | |
|---|---|
| `t.mock er.i ns tal lNe wSp y(Foo, " bar ")` | `const barSpy = t.sand box.sp y(Foo, " bar ")` |
| `t.mock er.i ns tal lNe wMo ck(Foo, " bar ")` | `const barStub = t.sand box.st ub(Foo, " bar ")` |

## Behavior Equivalents

| | |
|---|---|
| `.andRe tur n(foo)` | `.retur ns(foo)` |
| If different behavior for different inputs (rare): | |
| `.invok e(f oo).an dRe tur n(bar)` | `.withA rgs (fo o).r et urn s(bar)` |
| `.andTh row (foo)` | `.throw s(foo)` |
| `.andSt ub( fooFn)` | `.calls Fak e(f ooFn)` |
| Stub a property (not a method) (runWithReplacedGlobals) | `.value (foo)` |

## Verification Equivalents (do you need it?)

| **`t.mocker`** ... | **`t.sand box.assert`** ... |
|---|---|
| `.on().i nv oke()` | No verification necessary! |
| `.on().i nv oke (arg1, arg2)` | `.alway sCa lle dWi thE xac tly (fo oStub, arg1, arg2)` |
| `.expec t().in voke()` | `.calle dOn ce( foo Stub)` |
| `.expec t().in vok e(arg1, arg2)` | `.calle dOn ceW ith Exa ctl y(f ooStub, arg1, arg2)` |
| `.never()` | `.notCa lle d(f ooSpy)` |
| `.once()` | Gives `.on()` the behavior of `.expect()` |

## Matcher Equivalents

| | |
|---|---|
| `Matche r.h asK eyV alues({ foo: bar })` | `Sinon.m atch({ foo: bar })` |
| Necessary for most StateObjects (may surface as "Maximum call stack size exceeded"): | |
| `Matche r.h asK eyV alues({ task: expect ed_task })` | `Sinon.m atch({ task: Sinon.m at ch.s am e(e xpe cte d_ta sk) })` |
| You can also consider matching on e.g. task.global_id | |
| `Matche r.h asE xac tKe yVa lues({ foo: bar })` | `{ foo: bar }` |
| `Matche r.any` | `Sinon.m at ch.any` |
| `_` | `Sinon.m at ch.any` |
| `Matche r.h asS ubs tri ng( " foo ")` | `Sinon.m at ch( " foo ")` |
| `Matche r.i sIn sta nce Of(Foo)` | `Sinon.m at ch.i ns tan ceO f(Foo)` |
| `Matche r.m atc hes Reg ex( /foo/)` | `Sinon.m at ch( /foo/)` |
| `Matche r.i sDe fined()` | `Sinon.m at ch.d efined` |
| `Matche r.i sEq ual (foo, bar)` | `Sinon.m at ch.s am e(foo, bar)` |
| `Matche r.i sSt ring()` | `Sinon.m at ch.s tring` |
| `Matche r.i sAr ray()` | `Sinon.m at ch.a rray` |