Truffle Suite

Archived: This tutorial has been archived and may not work as expected; versions are out of date, methods and workflows may have changed. We leave these up for historical context and for any universally useful information contained. Use at your own risk!

NOTE : This tutorial is written for versions of Solidity prior to v0.4.13. It relies on the deprecated`throw` keyword, now replaced by `revert()`, `require()`, and `assert()`. See Solidity documentation for[error handling](#) for more information.

Truffle 3 brings forth Solidity unit testing, which means one can now test contracts in Solidity itself. This is a boon to contract developers, as there are several reasons why it's useful to have Solidity tests in addition to Truffle's Javascript tests. For us at[Ujo](#) , one of our biggest concerns is testing how contracts interact with each other, rather than just testing their interaction from a web3 perspective, and Solidity tests allow us to do that.

Though Solidity tests can be quite powerful, they do come with some drawbacks. One of those is testing whether or not a function shouldthrow (without breaking the test!), which I'll show you how to do right now.

Diving In[¶](#)

In Solidity, when performing a contract call that produces an error, Solidity automatically rethrows, meaning it will bubble that error up to the caller. However, with araw call, the behavior is different: one can catch the error and decide what to do from there on out. If it throws (fails), it will returnfalse . If it succeeds, it will returntrue . Thus, the following calls will produce a different result.

// Returns false bool

result

=

address . call(bytes4(bytes32 (sha3(" functionThatThrows() ")))); // Returns true bool

result

=

address . call(bytes4(bytes32 (sha3(" functionThatDoesNotThrow() ")))); When calls are made this way, even if a sub-call fails, it won't automatically rethrow.

In order to test for throws, one can either write out each call in the format above (which is cumbersome), or use a proxy contract to wrap the contract call to a raw call and return whether it succeeded or not.

For our purposes, let's look at the example below, similar to a method I saw used by Dapple in a Tester contract. Note that these contracts will likely be placed in separate files within your Truffle project. Here,Thrower is the contract you're testing to see whether or not certain contractsthrow ,ThrowProxy is our helper andTestThrower is our test contract.

The code is as follows:

import

"truffle/Assert.sol" ; // Proxy contract for testing throws contract

ThrowProxy

{

address

public target ;

bytes

data ;

function

ThrowProxy (address

_target)

{

target

_target;

}

//prime the data using the fallback function.

function ()

{

data

msg.data ;

}

function

execute ()

returns

(bool)

{

return

target. call(data);

} } // Contract you're testing contract

Thrower

{

function

doThrow ()

{

throw ;

}

function

doNoThrow ()

{

//

} } // Solidity test contract, meant to test Thrower contract

TestThrower

{

function

testThrow ()

{

Thrower thrower =

new

Thrower();

ThrowProxy throwProxy =

new

ThrowProxy(address (thrower));

//set Thrower as the contract to forward requests to. The target.

//prime the proxy.

Thrower(address (throwProxy)). doThrow();

//execute the call that is supposed to throw.

//r will be false if it threw. r will be true if it didn't.

//make sure you send enough gas for your contract method.

bool

r

=

throwProxy. execute. gas(200000)();

Assert. isFalse(r,

“ Should be false ,

as it should throw ”);

} } Perhaps the most interesting line is the following:

Thrower(address (throwProxy)). doThrow(); This is telling Solidity to call thedoThrow() function at thethrowProxy address. Solidity automatically creates the necessary message data (msg.data) based on this call. Writing it like this, one would assume that atthrowProxy thereis a Thrower contract, but there isn’t. There's a proxy instead. The proxy then receives the msg data, and since it doesn’t have adoThrow() function, the fallback function is triggered in its place.

The fallback function then saves the message data. After that, when executing the proxy, it then forwards the request onwards as a raw call, not a contract call. Since thethrow would use up all the gas, the rest of the tests would legitimately OOG, so we restrict the gas sent through when calling theexecute() method. Note that enough should be sent through so that it doesn’t OOG legitimately and thus miss the actual throw condition.

With this proxy, one can still effectively write the interactions as contracts calling other contracts without having to resort to crafting raw calls manually. (Hooray!)

Gotchas¶

An important caveat here is to recognize the contract caller,msg.sender . If you add a proxy in between, thenmsg.sender will be the proxy, which could break authorization and permissioning algorithms. If your authorization system allows you to change the owner, you can get around this constraint by setting the proxy to be the contract owner. For example:

// Assume our contract under test has a changeOwner() function thrower. changeOwner(address (throwProxy)); // Perform test functions and assertions // ... // Restore previous owner Thrower(address (throwProxy)). changeOwner(address (this)); throwProxy. execute(); It’s also important to know that this only teststhrow 's at this particular level. For instance, if your contract call structure looked like the following:

Test -> Proxy -> ContractToTest -> SomeOtherContract -> AnotherContractThatThrows Then you wouldn’t know where thethrow occurred, since SomeOtherContract and ContractToTest could just rethrow.

It would be prudent to also ensure there isn’t anything faulty in the proxy by creating a second test as a control. This test should be called with the appropriate gas and use the proxy to test a function whereno throw occurs, just to make sure the proxy is setup and working as intended.

Because athrow essentially uses up all gas, one must make doubly sure they catch thethrow andnot a legitimate out-of-gas (OOG) error. As well, take care to manage sending Ether through the proxy (for tests that require it) as that can be difficult as well.

Conclusion¶

Testingthrows is possible from within Solidity tests, but can be cumbersome if you want to write those tests yourself as raw calls. You can get around this by using a proxy contract, which makes many situations a lot easier and makes your tests much easier to write. But keep a look out for potential caveats as Solidity tests with a lot more power but plenty of drawbacks.

Happy developing!

About Simon de la Rouviere

Simon builds decentralised applications for use in the music industry, online communities and the developing world. He has been in the Bitcoin/blockchain space since 2011, developed a decentralised band around a full blown cryptocurrency, and is writing a book on the blockchain. More about Simon can be found onconsensys.net .