Debugging an Example Smart Contract

Note : This tutorial requires Truffle version 4.0 or newer.

A smart contract in Ethereum is just code. Unlike the "paper" contracts that you find elsewhere, this contract needs to make sense in a very precise manner.

(And that's a good thing. Imagine how much clearer real-world contracts would be if they needed to "compile"?)

If our contracts are not coded correctly, our transactions may fail, which can result in the loss of ether (in the form of gas), not to mention wasted time and effort.

Luckily, Truffle (as of version 4) has a built in debugger for stepping through your code. So when something goes wrong, you can find out exactly what it was, and fix it promptly.

In this tutorial, we will migrate a basic contract to a test blockchain, introduce some errors into it, and solve each one through the use of the built-in Truffle debugger.

A basic smart contract¶

One of the most basic, non-trivial, types of smart contract is asimple storage contract . (This example was adapted from thSolidity documentation .)

pragma solidity

^ 0.8.10 ; contract

SimpleStorage

{

uint

myVariable ;

function

set (uint

x)

public

{

myVariable

x;

}

function

get ()

constant

public

returns

(uint)

{

return

myVariable;

} } This contract does two things:

- Allows you to set a variable (myVariable
-) to a particular integer value
- Allows you to query that variable to get the selected value

This isn't a very interesting contract, but that's not the point here. We want to see what happens when things go wrong.

First, let's set up our environment.

Deploying the basic smart contract¶

1. Create a new directory where we will house our contract locally:

mkdir simple-storagecd

simple-storage 1. Create a bare Truffle project:

truffle init This will create directories such ascontracts/ andmigrations/ , and populate them with files we will use when we deploy our contract to the blockchain.

1. Inside thecontracts/
2. directory, create a file calledStore.sol
3. with the following content:

pragma solidity

^ 0.8.10 ; contract

SimpleStorage

{

uint

myVariable ;

function

set (uint

x)

public

{

myVariable

x;

}

function

get ()

constant

public

returns

(uint)

{

return

myVariable;

} } This is the contract that we will be debugging. While the full details of this file are beyond the scope of this tutorial, note that there is a contract namedSimpleStorage that contains a numeric variablemyVariable and two functions:set() andget() . The first function stores a value in that variable and the second queries that value.

1. Inside themigrations/
2. directory, create a file called2_deploy_contracts.js
3. and populate it with the following content:

var

SimpleStorage

=

artifacts . require ("SimpleStorage"); module . exports

=

function

(deployer)

{

deployer . deploy (SimpleStorage); }; This file is the directive that allows us to deploy theSimpleStorage contract to the blockchain.

1. On the terminal, compile the smart contract:

truffle compile 1. Open a second terminal and runtruffle develop 2. to start a development blockchain built directly into Truffle that we can use to test our contract:

truffle develop The console will display a prompttruffle(develop)> . From here, unless otherwise specified, all commands will be typed on this prompt.

1. With the develop console up and running, we can now deploy our contracts to the blockchain by running our migrations:

migrate The response should look something like below, though the specific IDs will differ:

Starting migrations...======================

Network name: 'develop'

Network id: 5777

Block gas limit: 6721975

(0x6691b7)

1_initial_migration.js

Deploying 'Migrations'

---

transaction hash: 0xbaf1963942bd99e949a966c16d204c4786fdbfde096f5fed0ec4c82e7b85aff5

...

total cost: 0 .000497708 ETH

Saving migration to chain.

Saving artifacts

Total cost: 0 .000497708 ETH# 2_deploy_contracts.js Deploying 'SimpleStorage'

---

transaction hash: 0xf4bf0a56cff1e1e5c121a3b1688a0103f12b8c45c4ed99818d160a1e3cc064f1

...

total cost: 0 .000251306 ETH

Saving migration to chain.

Saving artifacts

Total cost: 0 .000251306 ETH# Summary

Total deployments: 2

Final cost: 0 .000749014 ETH

- Fetching solc version list from solc-bin. Attempt

1

- Blocks: 0

Seconds: 0 - Saving migration to chain. - Blocks: 0

Seconds: 0 - Saving migration to chain.

Interacting with the basic smart contract¶

The smart contract is now deployed to a test network viatruffle develop , which launches aconsole againstGanache , a local development blockchain built right into Truffle.

We next want to interact with the smart contract to see how it works when working correctly. We'll interact using thetruffle develop console.

Note : If you're wondering why we didn't need to mine to get the transaction to be secured, the Truffle Develop console already takes care of that for us. If using a different network, you'll need to make sure you mine to get the transaction on the blockchain.

1. In the console wheretruffle develop
2. is running, run the following command:

SimpleStorage . deployed ()

. then (function

(instance)

{

return

instance . get . call ();

})

. then (function

(value)

{

return

value . toNumber ();

}); This command looks at the SimpleStorage contract, and then calls theget() function as defined inside it. It then returns the output, which is usually rendered as a string, and converts it to a number:

0 This shows us that our variable,myVariable , is set to0 , even though we haven't set this variable to any value (yet). This is becausevariables with integer types are automatically populated with the value of zero in Solidity , unlike other languages where it might beNULL orundefined .

1. Now let's run a transaction on our contract. We'll do this by running theset()
2. function, where we can set our variable value to some other integer. Run the following command:

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . set (4); }); This sets the variable to4 . The output shows some information about the transaction, including the transaction ID (hash), transaction receipt, and any event logs that were triggered during the course of the transaction:

{

tx :

'0x3af6c0644b34cfb60b00d352212da19ba425dd70d9175380cc709cd5020bc06b' ,

receipt :

{

transactionHash :

'0x3af6c0644b34cfb60b00d352212da19ba425dd70d9175380cc709cd5020bc06b' ,

transactionIndex :

0 ,

blockHash :

'0x243abc6a762a89c526256833c38e1ce3fd166dffeaff721f55e31cff89c719d9' ,

blockNumber :

5 ,

from :

'0x8e0128437dc799045b9c24da41eda77f0dea281b' ,

to :

'0x30775260f639d51a837b094cc9f66dc1426f3efb' ,

gasUsed :

41602 ,

cumulativeGasUsed :

41602 ,

contractAddress :

null ,

logs :

[],

status :

true ,

logsBloom :

'0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
,

rawLogs :

[]

},

logs :

[] } Most important to us is the transaction ID (listed here both astx and astransactionHash). We'll need to copy that value when we start to debug.

Note : Your transaction IDs will likely be different from what is listed here.

1. To verify that the variable has changed values, run theget()
2. function again:

SimpleStorage . deployed ()

. then (function

(instance)

{

return

instance . get . call ();

})

. then (function

(value)

{

return

value . toNumber ();

}); The output should look like this:

4

Debugging errors¶

The above shows how the contractshould work. Now, we will introduce some small errors to the contract and redeploy it. We will see how the issues present themselves, and alsouse Truffle's built-in debug feature to fix the issues .

We will look at the following issues:

- Invalid error check
- No error, but a function isn't operating as desired

Issue #1: An invalid error check¶

1. Typeq
2. to exit the debugger.

Issue #2: An invalid error check¶

Smart contracts can use statements likeassert() to ensure that certain conditions are met. These can conflict with the state of the contract in ways that are irreconcilable.

Here we will introduce such a condition, and then see how the debugger can find it.

Introducing the error¶

1. OpenStore.sol
2. again.
3. Replace theset()
4. function with the following:

function

set (uint

x)

public

{

assert(x ==

0);

myVariable

x; } This is the same as the original version, but with anassert() function added, testing to make sure thatx == 0 . This will be fine until we set that value to something else, and then we'll have a problem.

Testing the contract¶

Just as before, we'll reset the contract on the blockchain.

1. In the Truffle Develop console, reset the contract on the blockchain to its initially deployed state:

migrate --reset 1. Now we are ready to test the new transaction. Run the same command as above:

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . set (4); }); You will see an error:

Uncaught Error: Returned error: VM Exception while processing transaction: revert at evalmachine.:0:66 This means that we have a problem on our hands.

1. In the log window, note the transaction ID with that error in the data key:

data: {

'0x51f9cce23b57b15fafb13defc52225b1da2e29c5ce15f40a8ef793d2fff1546b' : {

error: 'revert' , program_counter: 346 , ...

Debugging the issue¶

1. Copy the transaction ID and use it as an argument to thedebug

2. command:

debug 0x51f9cce23b57b15fafb13defc52225b1da2e29c5ce15f40a8ef793d2fff1546b Note : Again, your transaction ID will be different from what is listed here.

Now we are back in the debugger:

Store. sol: 1 :

pragma solidity

^ 0.8.10 ; 2 : 3 :

contract

SimpleStorage

{

^^^^^^^^^^^^^^^^^^^^^^^^ debug(develop: 0x51f9cce2 ...)

1. PressEnter
2. a few times to step through the code. Eventually, the debugger will halt with an error message:

Store. sol: 5 : 6 :

function

set (uint

x)

public

{ 7 :

assert(x== 0);

^^^^^^^^^^^^ debug(develop: 0x51f9cce2 ...)

Transaction has halted;

cannot advance. It is this last event that is triggering the error. You can see that it is theassert() that is to blame.

Issue #2: A function isn't operating as desired¶

Sometimes, an error isn't a true error, in that it doesn't cause a problem at runtime, but instead is just doing something that you don't intend it to do.

Take for example an event that would run if our variable was odd and another event that would run if our variable was even.If we accidentally swapped this conditional so that the opposite function would run, it wouldn't cause an error; nevertheless, the contract would act unexpectedly.

Once again, we can use the debugger to see where things go wrong.

Introducing the error¶

1. OpenStore.sol
2. again.
3. Replace theset()
4. function with the following:

event

Odd (); event

Even (); function

set (uint

x)

public

{

myVariable

x;

if

(x %

2

==

0)

{

emit Odd();

}

else

{

emit Even();

} } This code introduces two dummy events,Odd() andEven() that are triggered based on a conditional that checks whetherx is divisible by2 .

But notice that we have the results flipped. Ifx is divisible by2 , theOdd() event will run.

Testing the contract¶

Just as before, we'll reset the contract on the blockchain.

1. In the Truffle Develop console, update the contract:

migrate --reset You will see both the compiler output and the migration output.

1. Now we are ready to test the new transaction. Run the same command as above:

SimpleStorage . deployed (). then (function

(instance)

{

return

instance . set (4); }); Note that there is no error here. The response is given as a transaction ID with details:

{

tx :

'0x31d64ba6ed196d12b634b1ea7cbe0612b3dc623ee6d25f0fc59091e1e19dfe08' ,

receipt :

{

transactionHash :

'0x31d64ba6ed196d12b634b1ea7cbe0612b3dc623ee6d25f0fc59091e1e19dfe08' ,

transactionIndex :

0 ,

blockHash :

'0x4ef7b0987604e6ca92382d75d16e746de2415fa482d7cfc85d9183e966d5beaf' ,

blockNumber :

5 ,

from :

'0x8e0128437dc799045b9c24da41eda77f0dea281b' ,

to :

'0x30775260f639d51a837b094cc9f66dc1426f3efb' ,

gasUsed :

42597 ,

cumulativeGasUsed :

42597 ,

contractAddress :

null ,

logs :

[

[Object]

],

status :

true ,

logsBloom :

'0x000...' ,

rawLogs :

[

[Object]

]

},

logs :

[

{

logIndex :

0 ,

transactionIndex :

0 ,

transactionHash :

'0x31d64ba6ed196d12b634b1ea7cbe0612b3dc623ee6d25f0fc59091e1e19dfe08' ,

blockHash :

'0x4ef7b0987604e6ca92382d75d16e746de2415fa482d7cfc85d9183e966d5beaf' ,

blockNumber :

5 ,

address :

'0x30775260F639D51a837b094Cc9f66DC1426f3EFB' ,

type :

'mined' ,

id :

'log_8a20539f' ,

event :

'Odd' ,

args :

[Result]

}

] } But notice the logs of the transaction show the eventOdd . That's wrong, and so our job is to find out why that's being invoked.

Debugging the contract¶

1. Copy that transaction ID and use it as an argument with thedebug
2. command:

debug 0x31d64ba6ed196d12b634b1ea7cbe0612b3dc623ee6d25f0fc59091e1e19dfe08 Note : Again, your transaction ID will be different from what is listed here.

You will enter the debugger as before.

1. PressEnter
2. multiple times to cycle through the steps. Eventually you will see that the conditional leads to theOdd()
3. event:

Store. sol: 9 :

function

set (uint

x)

public

{ 10 :

myVariable

x; 11 :

if

(x %

2

==

0)

{

^^^^^^^^^^^^^^^^ debug(develop: 0x31d64ba6 ...)

n

Store. sol: 10 :

myVariable

x; 11 :

if

(x %

2

==

0)

{ 12 :

emit Odd();

^^^^^ debug(develop: 0x31d64ba6 ...)

The problem is revealed. The conditional is leading to the wrong event.

Conclusion¶

With the ability to debug your contracts directly within Truffle, you have even more power at your hands to make your smart contracts rock-solid and ready to deploy. Make sure to read more about Truffle Develop console and the debugger in the docs. If you have any trouble, please don't hesitate to open an issue onGithub !

Happy debugging!