Proxy Contract

This walktrough tutorials covers the basics of using a proxy contract.
Proxy contract allows to save gas at deployment time.

Introduction

This document has been heavily inspired from

https://blog.gnosis.pm/solidity-delegateproxy-contracts-e09957d0f201

Please check it out ;)

Rationale

Storing contract code at creation time can cost up to:

200 * max_byte_code_length gas

200 * 24576 = 4915200

4915200 * 10 gwei = 49152000 gwei = 0.049152 ether = 9 EUR

see https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md for more infos on max_byte_code_length.

Proxy Contract

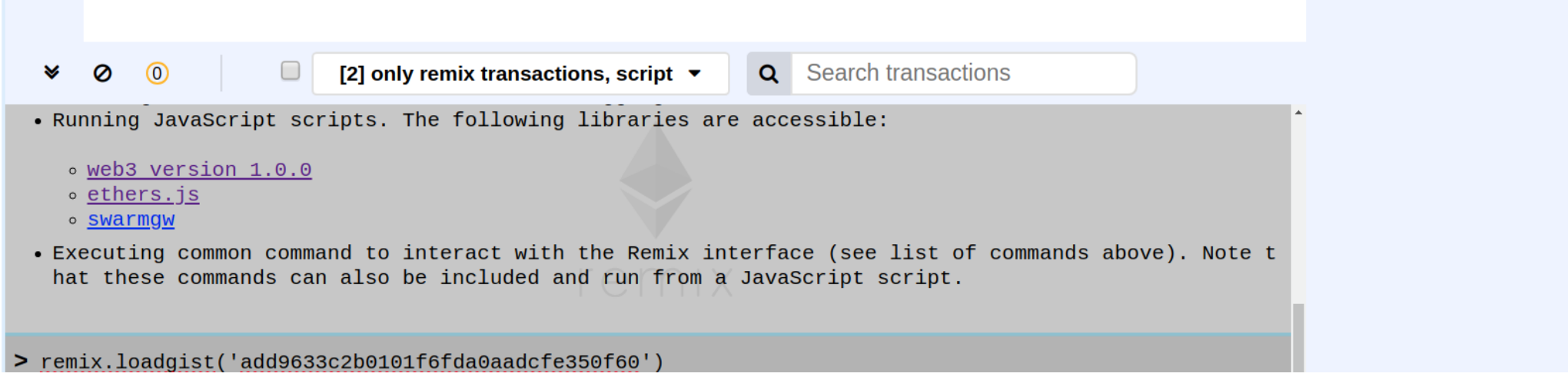
Using Proxy Contract pattern allows to save gas at deployment time.

Useful when a lot of instances of the same contract require to be deployed.

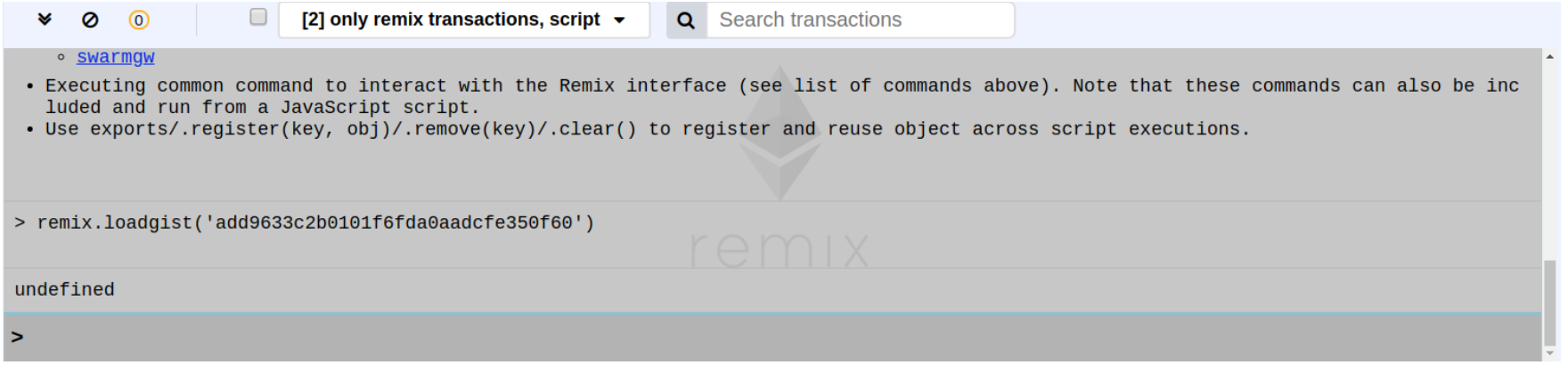
Let’s Start to Proxy the AwardToken

Load the AwardToken code by running the command:

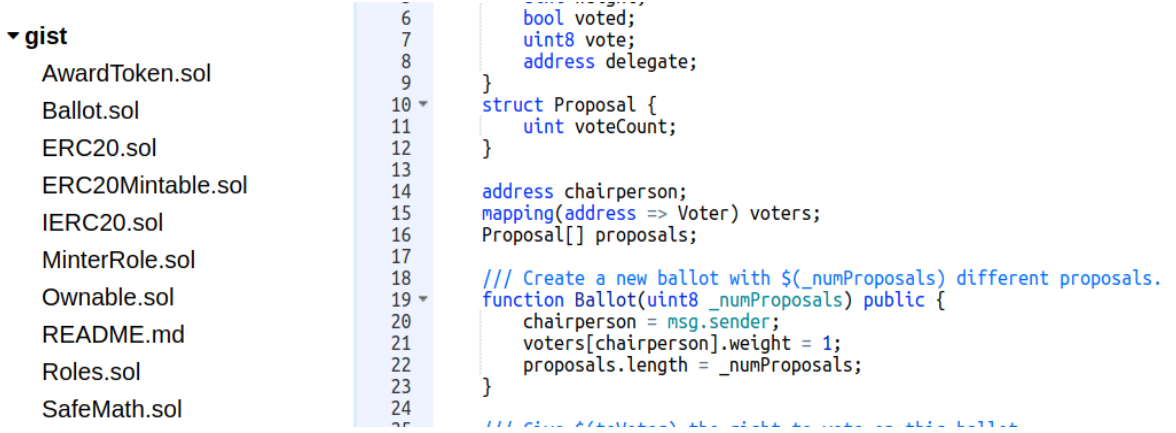
remix.loadgist('add9633c2b0101f6fda0aadcfe350f60')



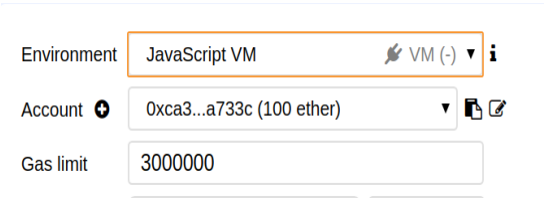
AwardToken



select AwardToken.sol
In the gist explorer

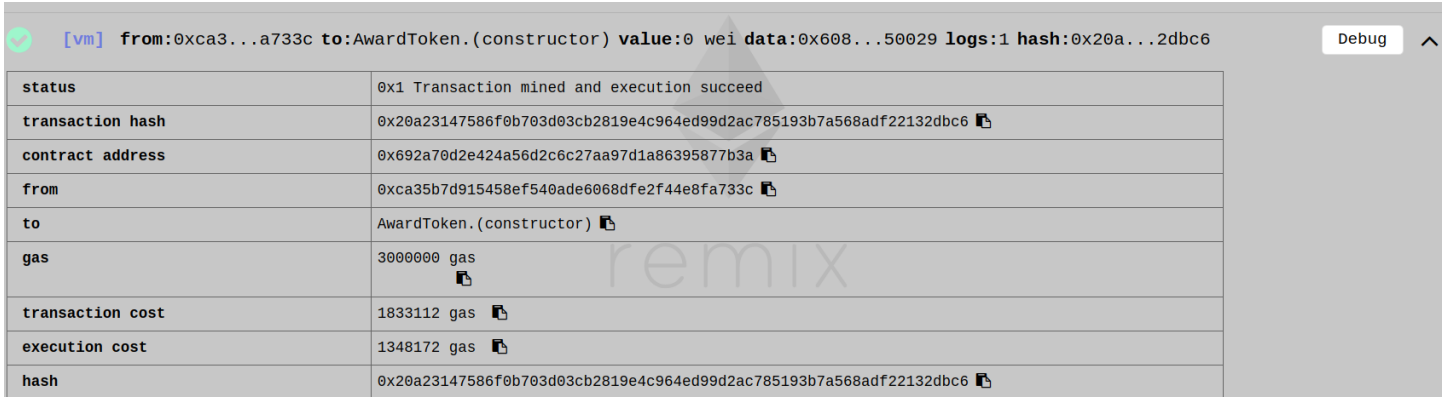


Deploy AwardToken



Deploy the contract on the JavaScript VM. The transaction cost should be around:

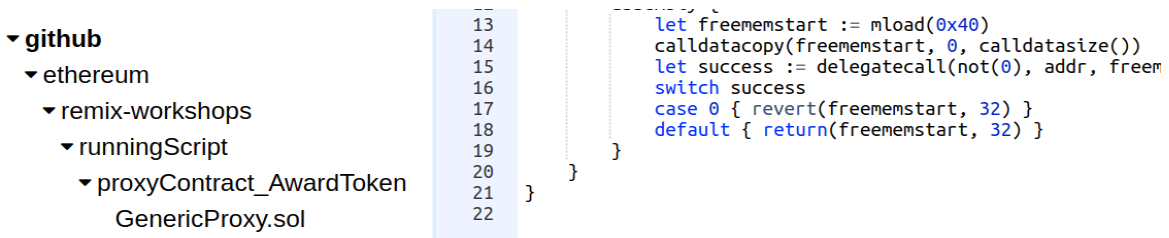
1833112 gas * 10 gwei * 0.000000001 = 0,01833112 eth = 3,7 euros



Loading a generic proxy

Run in the console:

remix.loadurl('https://github.com/ethereum/remix-workshops/proxyContractAwardToken/GenericProxy.sol')



Generic Proxy?

The plan is to deploy the generic proxy and link it to the AwardToken.

We keep the same functionality but with less code to deploy and therefore more gas saved.

- Instead of implementing all the function of AwardToken, we use the anonymous function to forward the call to the AwardToken code -

Generic Proxy

The forwarding happens line 12. see “delegatecall”.

We call the code of AwardToken (which is stored at the address “proxied”) while keeping the same context (msg.sender and storage)

```
1  contract GenericProxy {
2      address internal proxied;
3      constructor(address _proxied) public {
4          proxied = _proxied;
5      }
6
7      function () public payable {
8          address addr = proxied;
9          assembly {
10             let freememstart := mload(0x40)
11             calldatacopy(freememstart, 0, calldatasize())
12             let success := delegatecall(not(0), addr, freememstart, calldatasize(), freememstart, 32)
13             switch success
14             case 0 { revert(freememstart, 32) }
15             default { return(freememstart, 32) }
16         }
17     }
18 }
```

Generic Proxy

Create a new file in the browser explorer (Proxy.sol for instance) and paste the code of the generic proxy to it.

(We are copying it into the browser explorer so that we can edit it.)

We can “theoretically” use it as it is, by deploying this proxy contract with the address of an AwardToken as constructor parameter.

```
1  contract GenericProxy {
2      address internal proxied;
3      constructor(address _proxied) public {
4          proxied = _proxied;
5      }
```

While the “forwarding call” will work, the Proxy contract is **NOT** usable due to a different storage layout between Proxy and Awardtoken contract (see next slide)

Storage layout

The reason is that the state declaration of both contracts (Proxy and AwardToken) are completely different.

We have:

- one state variable of type address for the proxy contract.
- an ERC20 like state declaration for the AwardToken.

They both get “merged” in a single messy state.

Storage layout

While a “forwarding call” is a necessary first step, the “forwarding call” does not change anything on the storage declaration.

We need to tell the Proxy how the AwardToken state looks like ;)

Storage layout

For doing that, the easiest way is to create a data contract (contract representing the state):

Create a file named AwardTokenData.sol, and write an empty contract named AwardTokenData.

Copy the state (including events) from the AwardToken contract to AwardTokenData contract.

Don’t forget that AwardToken is an ERC20Mintable and will require some import:

```
import "gist/ERC20Mintable.sol";
import "gist/Ballot.sol";
contract AwardTokenData is ERC20Mintable {
    uint public quantity;
```

Storage layout

- The file that we had just created called Proxy.sol - we can now rename to AwardTokenProxy.
- Make sure AwardTokenProxy inherits from AwardTokenData
- With this setup, AwardToken also now inherits from AwardTokenData

```
import "./AwardTokenData.sol";

contract AwardTokenProxy is AwardTokenData {
```

Initialization

An important issue remains.

If we deploy the AwardTokenProxy now, we would not execute the AwardToken constructor. AwardTokenProxy does not inherit AwardToken.

The solution is to simply put the AwardToken constructor code in the AwardTokenProxy constructor – in our case the assignment of “quantity”

```
constructor(address _proxied) public {
    proxied = _proxied;
    quantity = 100;
}
```

Initialization

An important issue remains.

If we deploy the AwardTokenProxy now, we would not execute the AwardToken constructor. AwardTokenProxy does not inherit AwardToken.

The solution is to simply put the AwardToken constructor code in the AwardTokenProxy constructor – in our case the assignment of “quantity”

```
constructor(address _proxied) public {
    proxied = _proxied;
    quantity = 100;
}
```

Initialization

If the master contract has input parameters, you'd need to use the same input parameters in the same order BUT after the address parameter.

e.g if the master contract is

```
function AwardToken (uint someVar1, string someVar2) {
    quantity = 100;
    ...
}
```

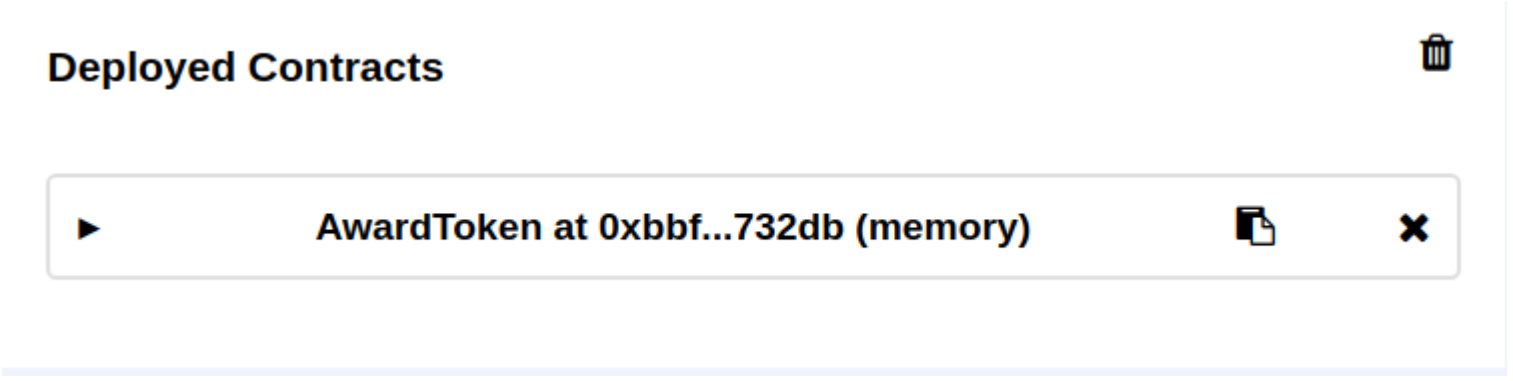
The constructor of the Proxy contract is going to be:

```
constructor(address _proxied, uint someVar1, string someVar2) public {
    proxied = _proxied;
    ...
}
```

Deploy

Now that’s it!

Compile and deploy AwardToken - this is our master contract.

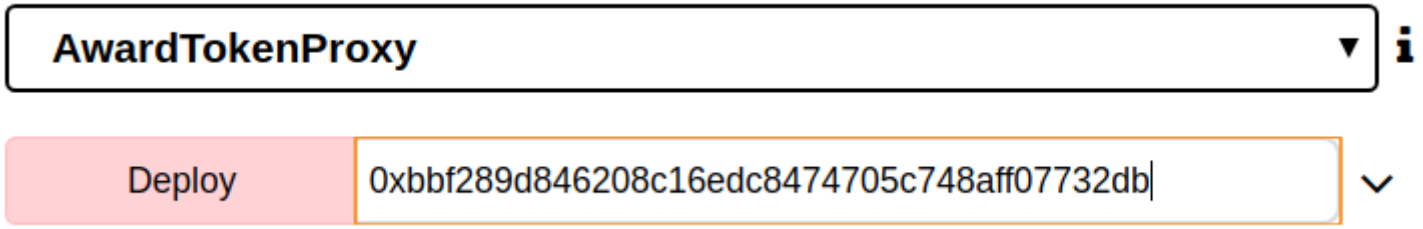


Deploy

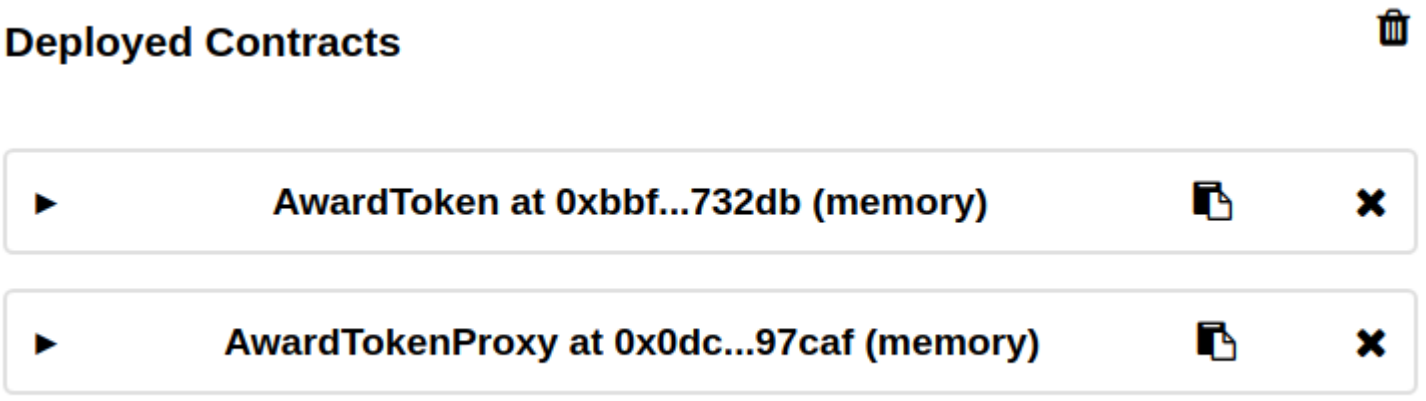
Deploy AwardTokenProxy - using the master contract address

it should cost around 924496 gas.

924496 gas * 10 gwei * 0.000000001 = 0,00924496 ETH = 1.85 EUR

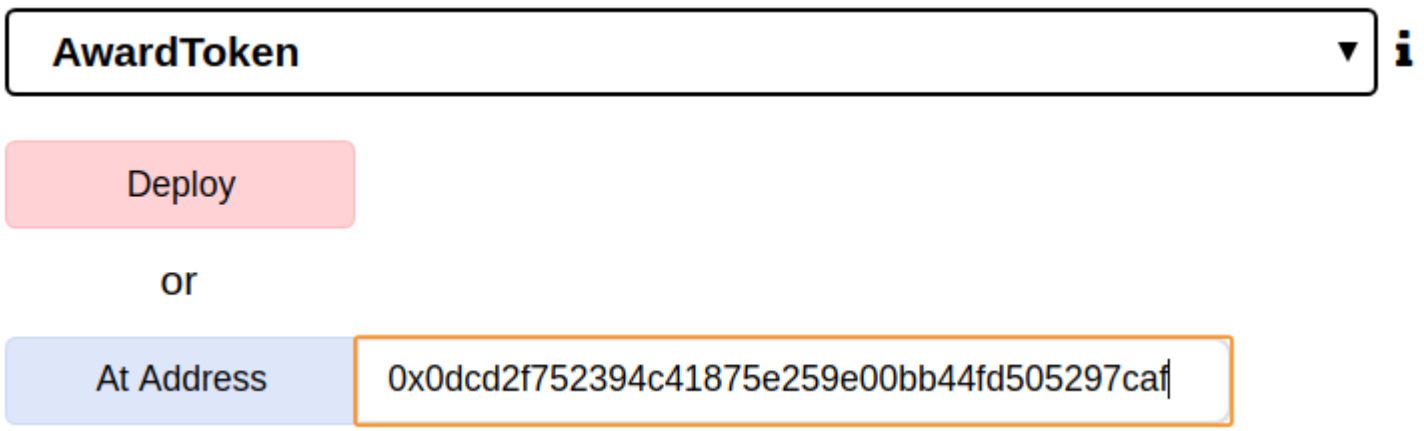


Deploy

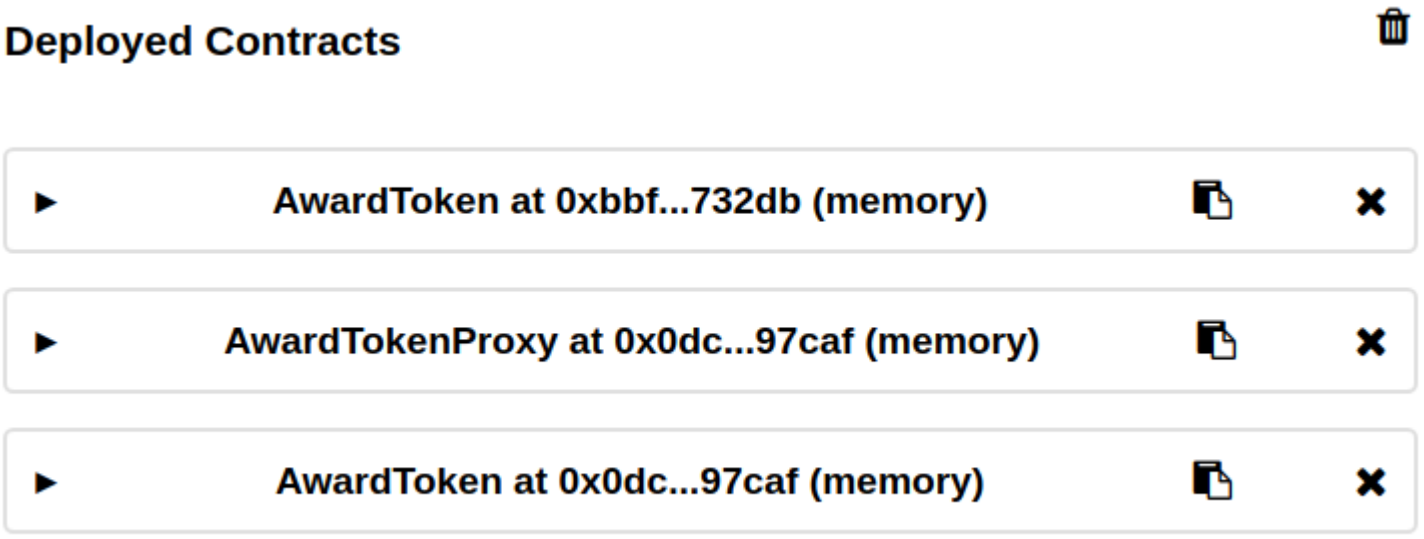

Deployed Contracts
AwardToken at 0xbbf...732db (memory)
AwardTokenProxy at 0x0dc...97caf (memory)

Deploy

Use the “At Address” action to access the proxy as the AwardToken.
(in this case we input the address of AwardTokenProxy)



Deploy


Deployed Contracts
AwardToken at 0xbbf...732db (memory)
AwardTokenProxy at 0x0dc...97caf (memory)
AwardToken at 0x0dc...97caf (memory)

Please be sure to check the following post:

https://blog.gnosis.pm/solidity-delegateproxy-contracts-e09957d0f201

Please be sure to check the following post:

https://blog.gnosis.pm/solidity-delegateproxy-contracts-e09957d0f201