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

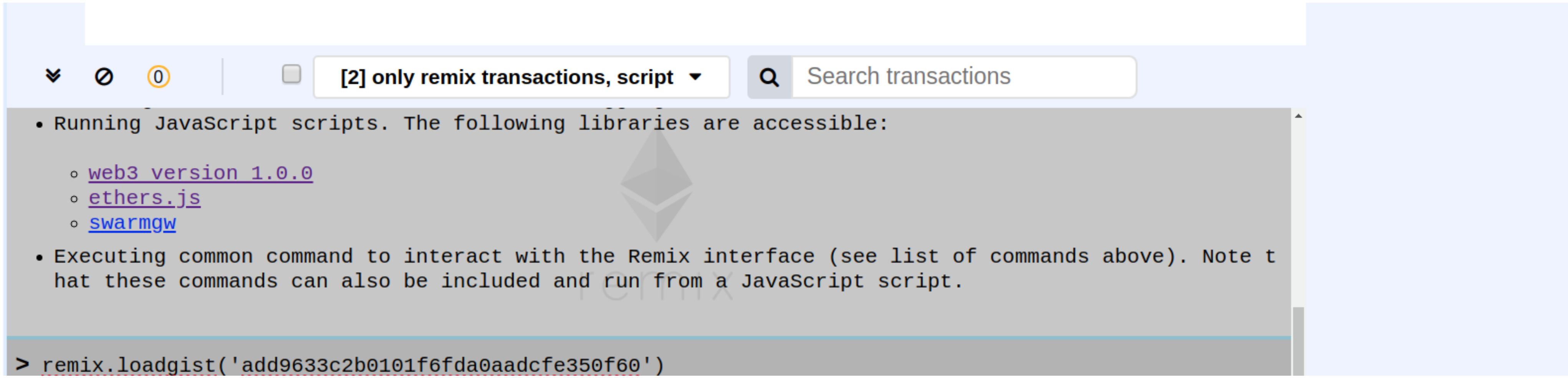
Using Proxy Contract pattern allows to save gas at deployment time.

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:

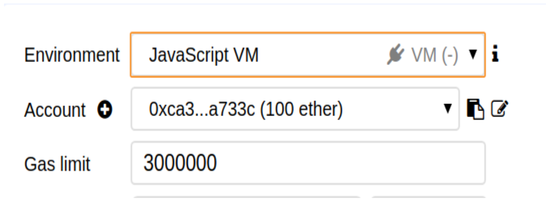
`remix.loadgist('add9633c2b0101f6fda0aadcfe350f60')`



- Running JavaScript scripts. The following libraries are accessible:
    - web3 version 1.0.0
    - ethers.js
    - swarmgw
- Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.

```
> remix.loadgist('add9633c2b0101f6fda0aadcfe350f60')
```

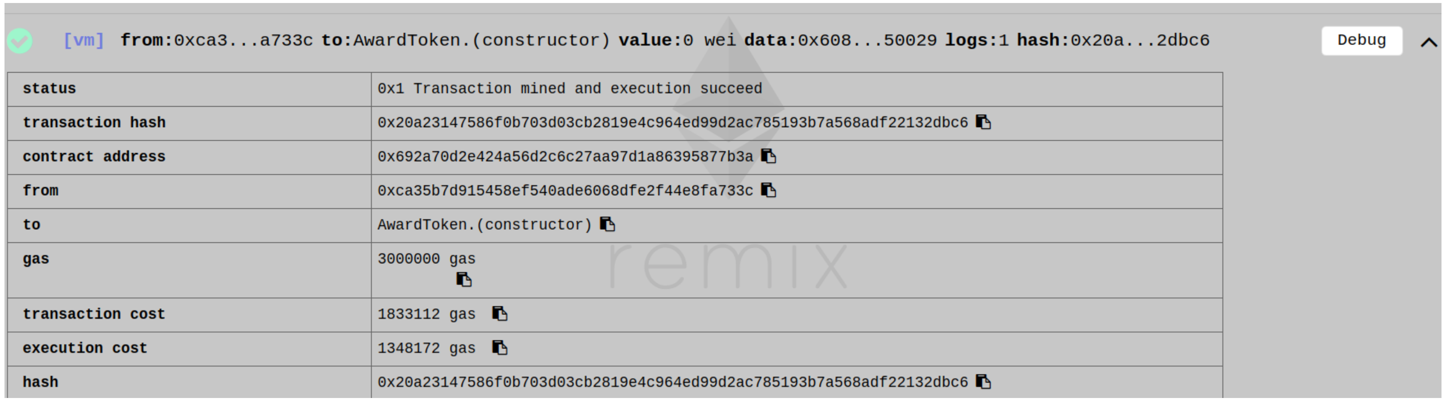
Deploy AwardToken



Environment	JavaScript VM — VM (-)
Account	0xca3...a733c (100 ether)
Gas limit	3000000

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



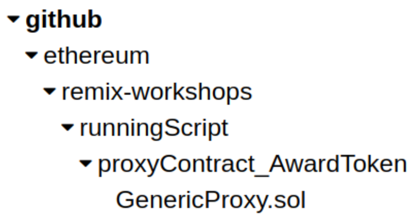
```
[vm] from:0xca3...a733c to:AwardToken.(constructor) value:0 wei data:0x608...50029 logs:1 hash:0x20a...2dbc6
```

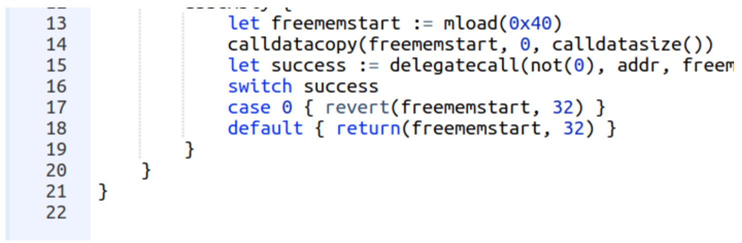
status	0x1 Transaction mined and execution succeed
transaction hash	0x20a23147586f0b703d03cb2819e4c964ed99d2ac785193b7a568adf22132dbc6
contract address	0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	AwardToken.(constructor)
gas	3000000 gas
transaction cost	1833112 gas
execution cost	1348172 gas
hash	0x20a23147586f0b703d03cb2819e4c964ed99d2ac785193b7a568adf22132dbc6

Loading a generic proxy

By running in the console

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





```
      let freememstart := mload(0x40)
      calldatacopy(freememstart, 0, calldatasize())
      let success := delegatecall(not(0), addr, freem
      switch success
      case 0 { revert(freememstart, 32) }
      default { return(freememstart, 32) }
    }
  }
}
```

This solidity contract is not directly usable and we need to add some modifications.

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 more gas saved.

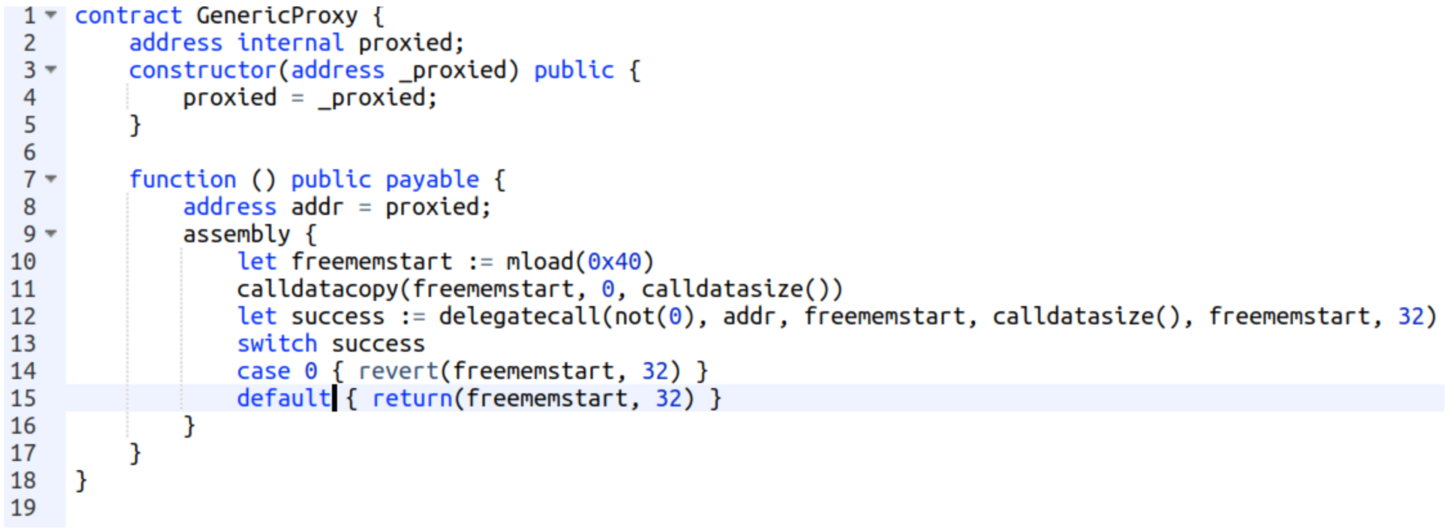
- Instead of implementing all the function of AwardToken, we use the anonymous function to forward the call to the AwardToken code -

Generix Proxy

The forwarding happens line 12. see “delegatecall”.

We call the code of AwardToken (which is stored at “proxied”)

While keeping the same context (msg.sender and storage)

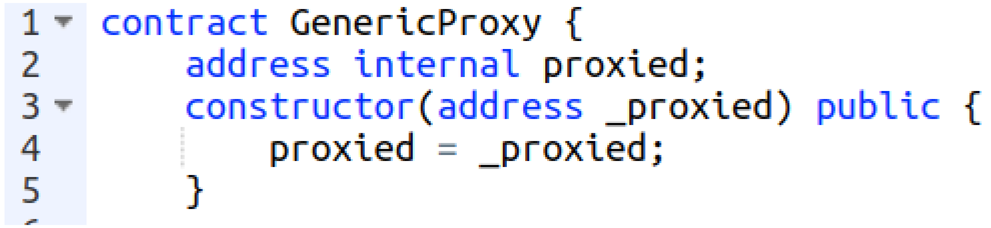


```
 1  contract GenericProxy {
 2      address internal proxied;
 3      constructor(address _proxied) public {
 4          proxied = _proxied;
 5      }
 6
 7      function () public payable {
 8          address addr = proxied;
 9          assembly {
10              let freememstart := mload(0x40)
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  }
19
```

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 can “theorically” use it as it is by deploying this proxy contract with the address of an AwardToken:



```
contract GenericProxy {
    address internal proxied;
    constructor(address _proxied) public {
        proxied = _proxied;
    }
```

Forwarding call will work but unfortunately the Proxy contract is not usable.

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 and an ERC20 like state declaration for the AwardToken.

Storage layout

Forwarding call is a necessary first step, but forwarding call does not change anything on the storage declaration.

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

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

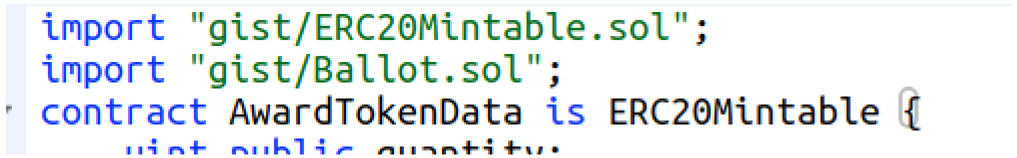
Storage layout

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

Create a file named AwardTokenData.sol, and write a 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 {
```

•

Storage layout

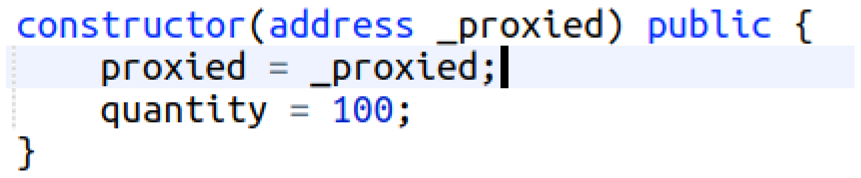
Make the GenericProxy (we can rename it AwardTokenProxy now) and AwardToken inheriting from AwardTokenData.

Initialization

An important issue remain.

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 to the AwardTokenProxy constructor – in our case the assignment of “quantity”

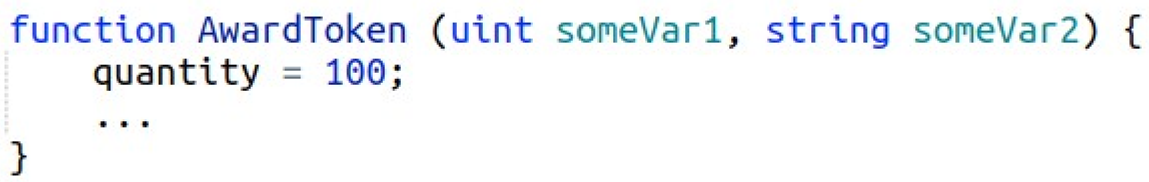


```
constructor(address _proxied) public {
    proxied = _proxied;
    quantity = 100;
}
```

Initialization

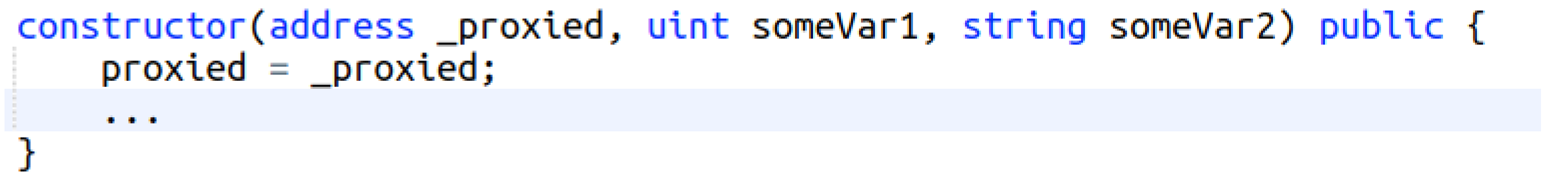
Generally if the master contract has input parameters, the common way is to map variable;

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:

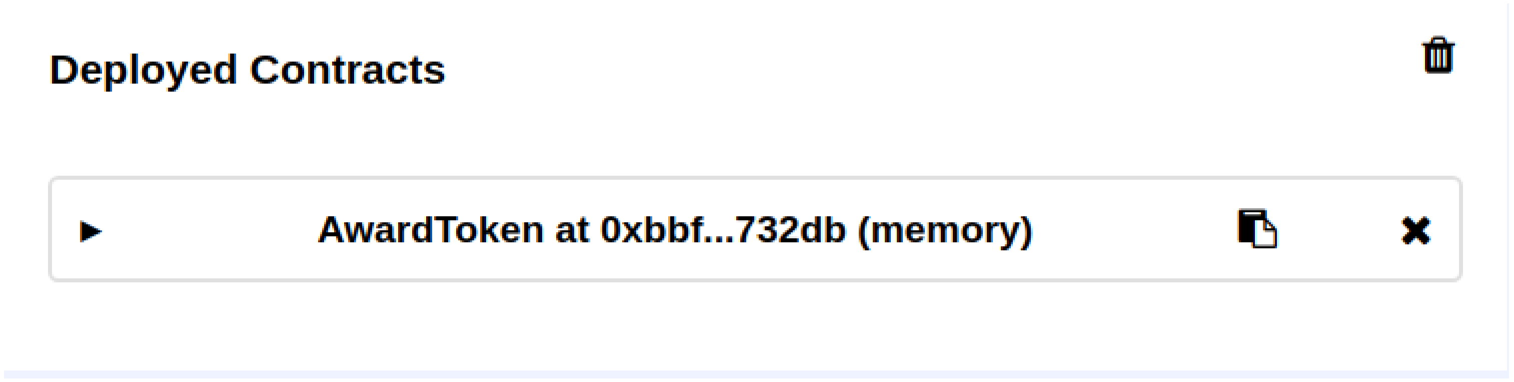


```
constructor(address _proxied, uint someVar1, string someVar2) public {
    proxied = _proxied;
    ...
}
```

Deploy

Now that’s it!

Compile and deploy AwardToken - this is our master contract.



Deployed Contracts

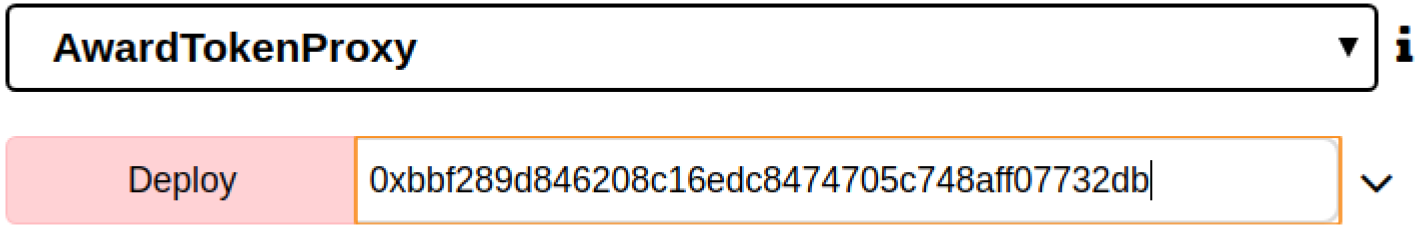
AwardToken at 0xbbf...732db (memory)

Deploy

Deploy AwardTokenProxy - using the master contract address

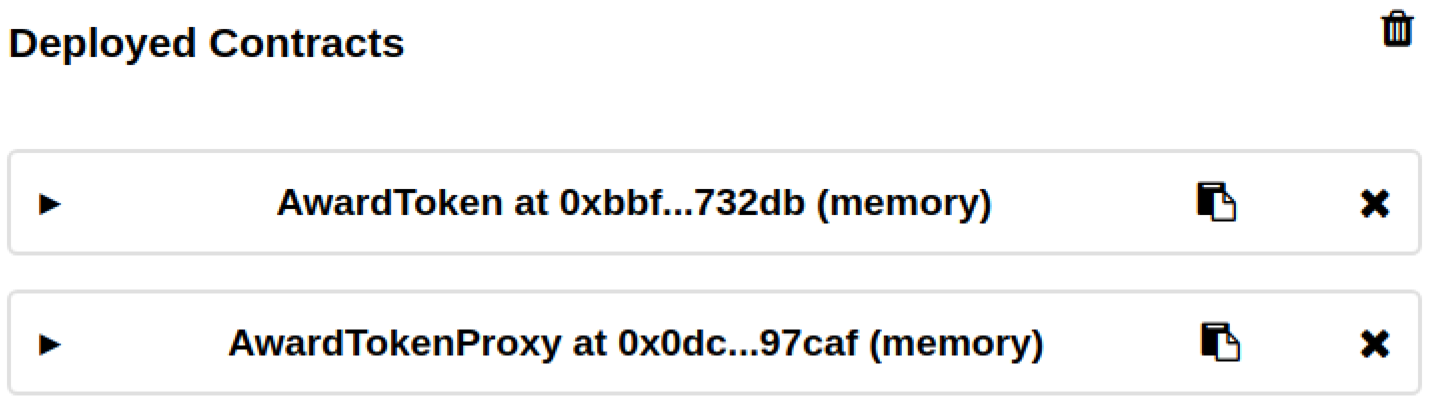
it should cost around 924496 gas.

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


AwardTokenProxy

Deploy 0xbbf289d846208c16edc8474705c748aff07732db

Deploy



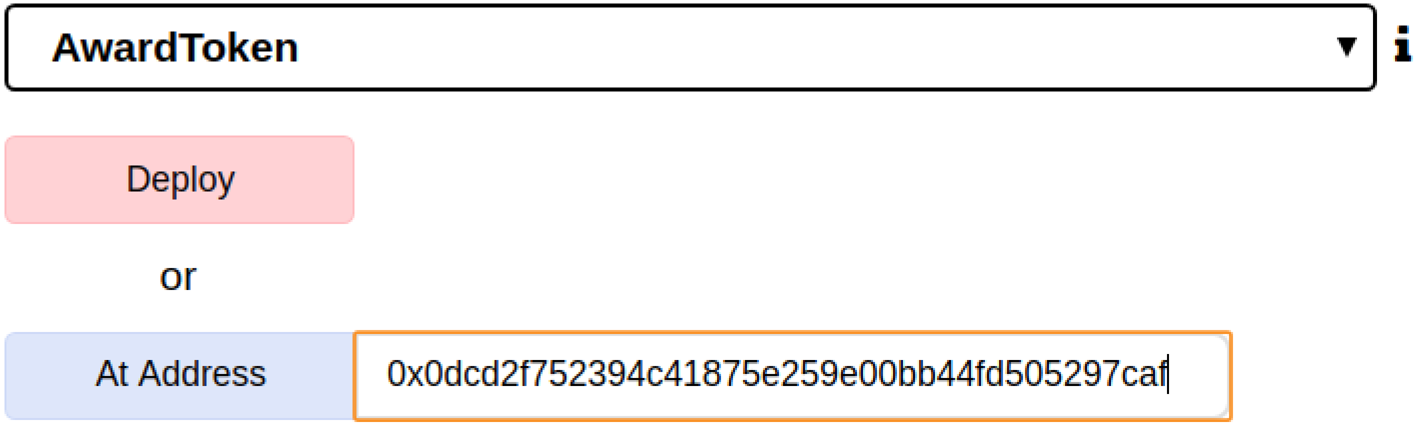
Deployed Contracts

▶ AwardToken at 0xbbf...732db (memory)

▶ AwardTokenProxy at 0x0dc...97caf (memory)

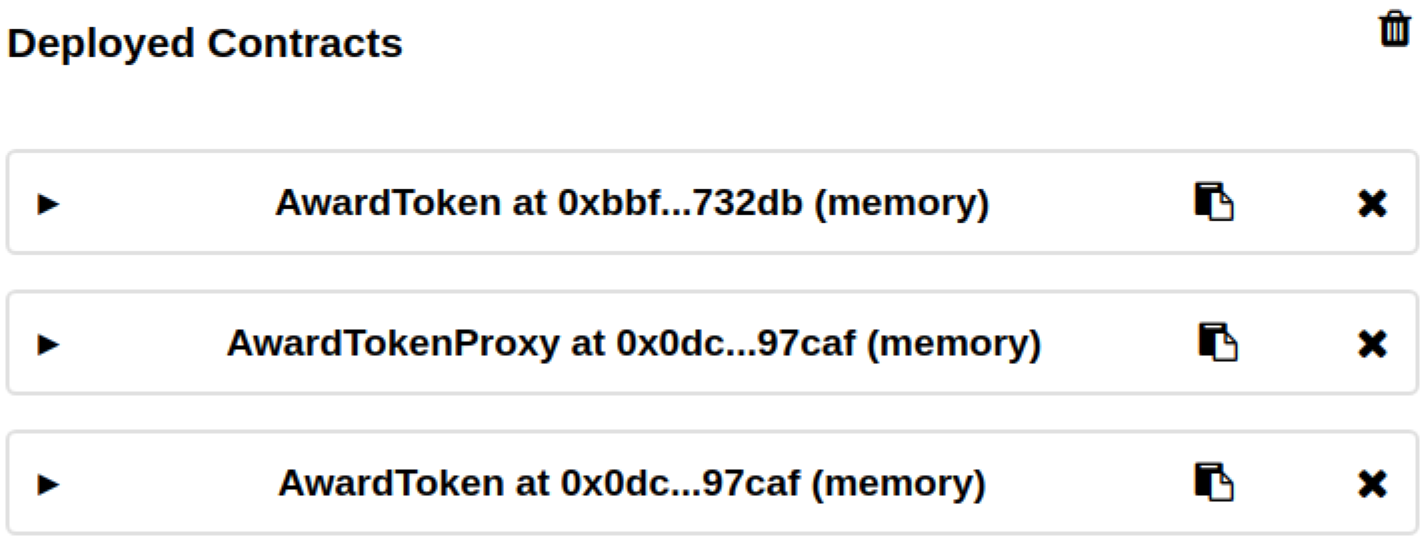
Deploy

Use the “At Address” action to access the proxy as the AwardToken.



0x0dcd2f752394c41875e259e00bb44fd505297caf

Deploy



Deployed Contracts

▶ AwardToken at 0xbbf...732db (memory)

▶ AwardTokenProxy at 0x0dc...97caf (memory)

▶ AwardToken at 0x0dc...97caf (memory)

Deploy

Alternatively, we can use a script to deploy the Proxy:

execute from the console:

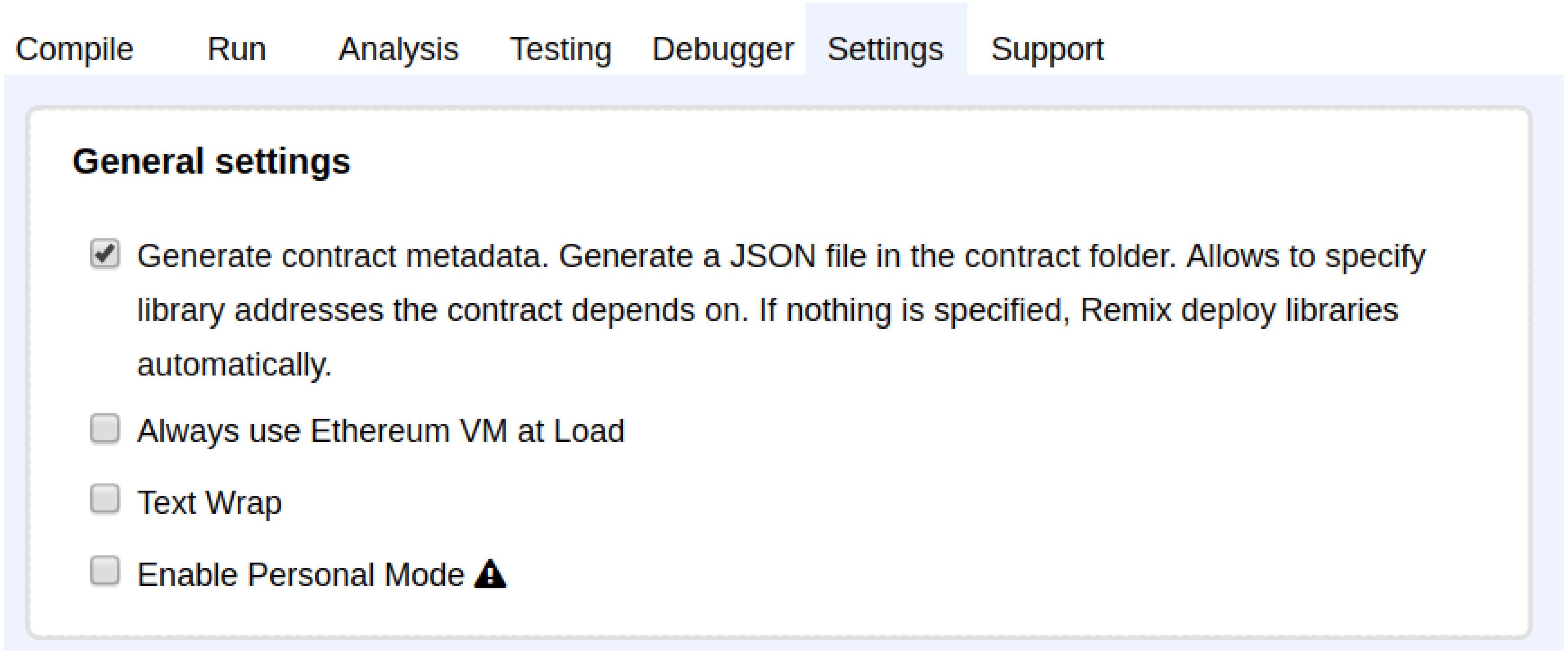
```
remix.loadurl(‘https://github.com/ethereum/remix-workshops/runningScript/proxyContract_AwardToken/global.js')
```

-
  ```
  remix.execute('github/ethereum/remix-workshops/runningScript/proxyContract_AwardToken/global.js')
  ```

That’s a generic JavaScript script which setup all you need for logging the deployment result in a file and deploy a contract

Deploy

For Deploying, we need the abi and bytecode accessible from the script. In Settings tab, check the “Generate contract metadata” option



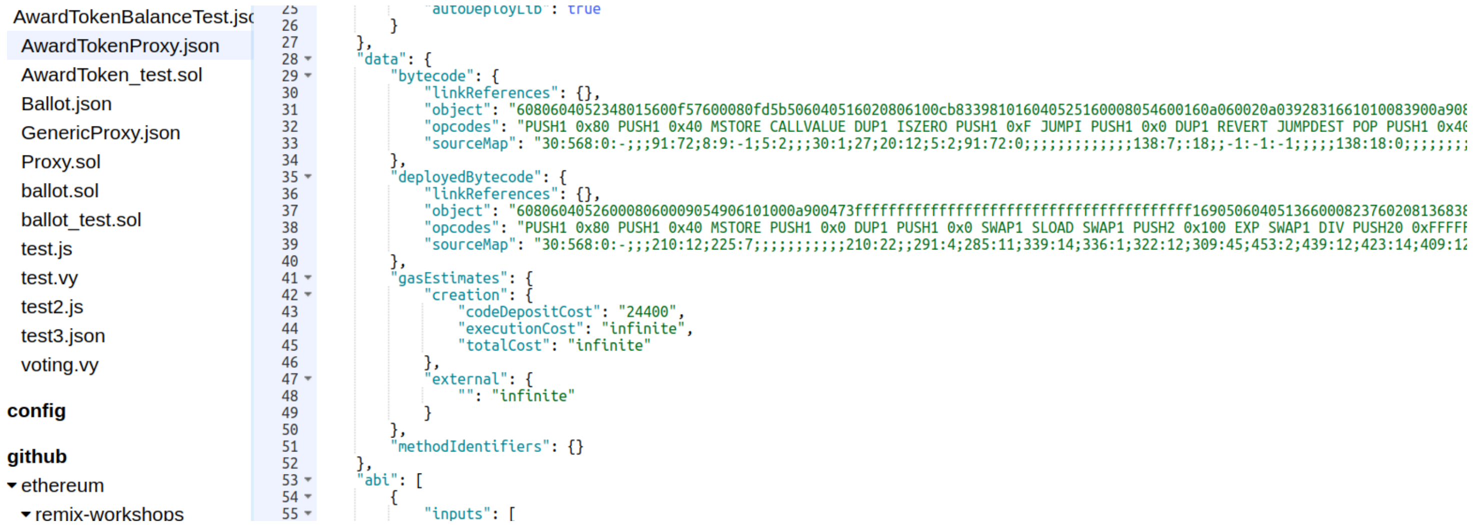
General settings

- [✓] Generate contract metadata. Generate a JSON file in the contract folder. Allows to specify library addresses the contract depends on. If nothing is specified, Remix deploy libraries automatically.
- [ ] Always use Ethereum VM at Load
- [ ] Text Wrap
- [ ] Enable Personal Mode ⚠

Deploy

Recompile the AwardTokenProxy

AwardTokenProxy.json is then created and contains all we need to deploy

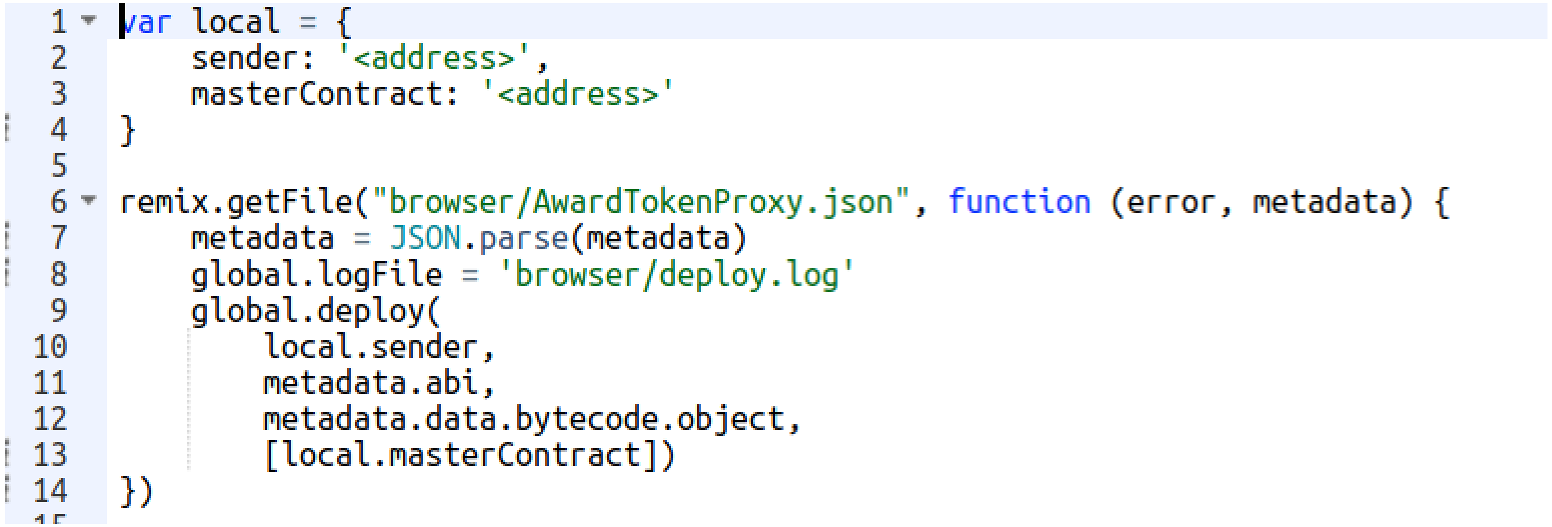


```
        "autoDeployLib": true
    }
},
"data": {
    "bytecode": {
        "linkReferences": {},
        "object": "608060405234801560f57600080fd5b50604051602080610cb83398101604052516000805460016 0a060020a03928316610100839 00a908
        "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH1 0xF JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x40
        "sourceMap": "30:568:0:-;;;91:72;8:9:-1;5:2;;;30:1;27;20:12;5:2;91:72:0;;;;;;;;;;;;;138:7;:18;;-1:-1:-1;;;;;138:18:0;;;;;;;
    },
    "deployedBytecode": {
        "linkReferences": {},
        "object": "6080604052600080600090549061010000a900473ffffffffffffffffffffffffffffffffffffffff169050604051366000823760208136838
        "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x0 DUP1 PUSH1 0x0 SWAP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20 0xFFFFF
        "sourceMap": "30:568:0:-;;;210:12;225:7;;;;;;;;;;;210:22;;291:4;285:11;339:14;336:1;322:12;309:45;453:2;439:12;423:14;409:12
    },
    "gasEstimates": {
        "creation": {
            "codeDepositCost": "24400",
            "executionCost": "infinite",
            "totalCost": "infinite"
        },
        "external": {
            "": "infinite"
        }
    },
    "methodIdentifiers": {}
},
"abi": [
    {
        "inputs": [
```

Deploy

Ultimately, we load the script that will actually deploy

remix.loadurl(‘https://github.com/ethereum/remix-workshops/runningScript/proxyContract_AwardToken/deploy.js’)



```
var local = {
    sender: '<address>',
    masterContract: '<address>'
}

remix.getFile("browser/AwardTokenProxy.json", function (error, metadata) {
    metadata = JSON.parse(metadata)
    global.logFile = 'browser/deploy.log'
    global.deploy(
        local.sender,
        metadata.abi,
        metadata.data.bytecode.object,
        [local.masterContract])
})
```

Deploy

And run remix.execute() first on global.js then on deploy.js

you can check the deployment log in browser/deploy.log

You will find a lot more information in this great blog post:

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