

Proxy Contract

This walkthrough tutorial 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:

$200 * \text{max_byte_code_length}$ gas

$200 * 24576 = 4915200$

$4915200 * 10 \text{ gwei} = 49152000 \text{ gwei} = 0.049152 \text{ ether} = 9 \text{ EUR}$

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

Proxy Contract

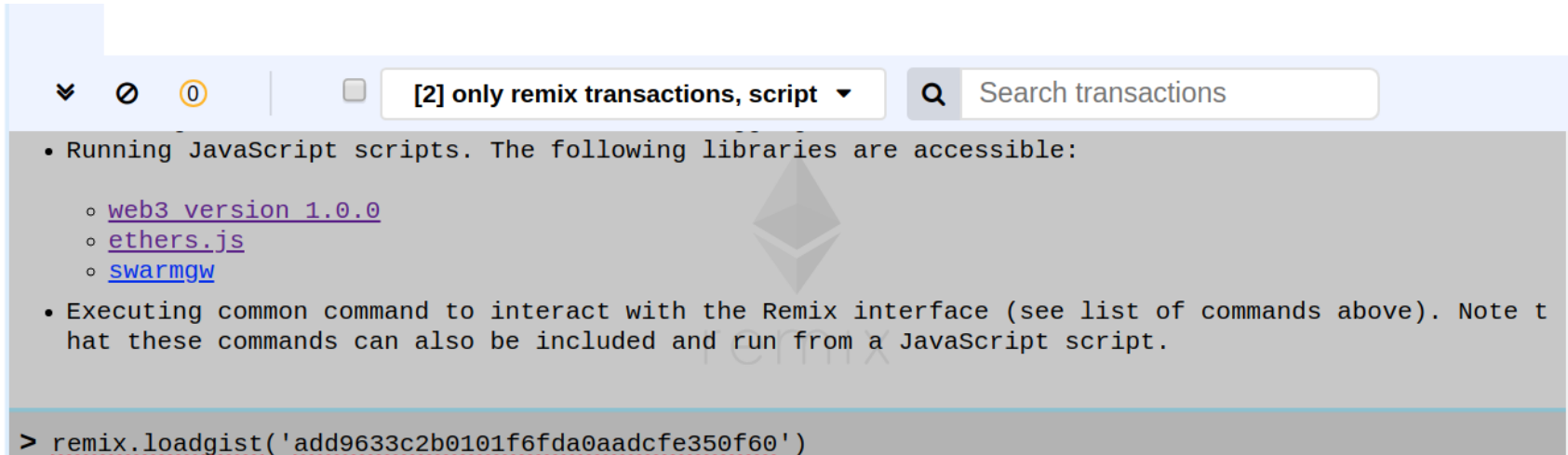
Using Proxy Contract pattern allows to save gas at deployment time.

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('add9633c2b0101f6fda0aadcf350f60')
```



AwardToken

```
▼ ○ ○ [2] only remix transactions, script 🔍 Search transactions
  ◦ 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.
  • Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.

> remix.loadgist('add9633c2b0101f6fda0aadcf350f60')

undefined

>
```

select AwardToken.sol
In the gist explorer

▼ gist

AwardToken.sol
Ballot.sol
ERC20.sol
ERC20Mintable.sol
IERC20.sol
MinterRole.sol
Ownable.sol
README.md
Roles.sol
SafeMath.sol

```
6     bool voted;
7     uint8 vote;
8     address delegate;
9 }
10 struct Proposal {
11     uint voteCount;
12 }
13
14 address chairperson;
15 mapping(address => Voter) voters;
16 Proposal[] proposals;
17
18 /// Create a new ballot with $_numProposals different proposals.
19 function Ballot(uint8 _numProposals) public {
20     chairperson = msg.sender;
21     voters[chairperson].weight = 1;
22     proposals.length = _numProposals;
23 }
24
25 /// Give $delegate the right to vote on this ballot
```

Deploy AwardToken

Environment JavaScript VM 🔥 VM (-) ▼ i

Account 0xca3...a733c (100 ether) 📄 📄

Gas limit 3000000

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

$$1833112 \text{ gas} * 10 \text{ gwei} * 0.0000000001 = 0,01833112 \text{ eth} = 3,7 \text{ euros}$$

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

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

Run in the console:

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

- ▼ github
 - ▼ ethereum
 - ▼ remix-workshops
 - ▼ runningScript
 - ▼ proxyContract_AwardToken
 - GenericProxy.sol

```
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
}  
}  
}  
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) }
```


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.

Deployed Contracts



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



Deploy

Deploy AwardTokenProxy - using the master contract address
it should cost around 924496 gas.

$924496 \text{ gas} * 10 \text{ gwei} * 0.0000000001 = 0,00924496 \text{ ETH} = 1.85 \text{ EUR}$

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.
(in this case we input the address of AwardTokenProxy)

AwardToken

▼

i

Deploy

or

At Address

0x0dcd2f752394c41875e259e00bb44fd505297ca

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>