
Author Name
Title
Optional tagline or other text

Table of Contents

Hello

Coding

The Awesome Book

Hello Friends,

This is an awesome book built by awesome people for even more awesome people. The ideia is to have everything useful at hand. Sometimes, somethings are used just once in a while and we don't remember how it's done. This is a collection of all those usefull things.

This was built using

gitbook. Yeap, you read it right. We are building this book using gitbook.

Solidity

This chapter is about solidity tricks and contracts built within the company and where were they used.

Gas Less

On public mainnet, every transaction has a fee. Not all users have ETH to pay that fee, and it's not very user friendly either.

To remove that step and improve users experience, there are currently some _gas less _tools. But what does this means? Gas Less tools, are tools that can be wired to you current dapp and avoid the need of the final user to pay for a transaction.

By doing so, another account will be paying the transaction, meaning that, most likely, will be the system owner paying it.

To better understand the concept, please have further reading at https://www.reddit.com/r/ethereum/comments/ae71l8/meta_gasless_transactions_explained/.

Possible tools

There are few people building their own tools, but currently there's one well know. https://shipl.co/ is public and available to give a try. The tools simplifies the process of moving to a gas less dapp.

Signatures

StepEscrow.sol

This contract is like an *escrow* contract but divided into steps. For each step, a percentage is taken.

Note: This contract is created using the library *openzeppelin-solidity* without changes.

```
pragma solidity ^0.4.23;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";

contract StepEscrow {
    using SafeMath for uint256;

    address private payer;
    address private payee;
    uint256 private steps;
    uint256 private deposits;

    event Deposited(address indexed payee, uint256 weiAmount);

    /**
    * @dev constructor.
    * @param _payer should be the payer
    * @param _steps how many steps the escrow have
    */
    constructor(address _payer, uint256 _steps) public {
        payer = _payer;
        steps = _steps;
    }

    /**
    * @dev modifier who allows only the payer to do actions
    * @param _who should be the payer
    */
    modifier onlyPayer(address _who) {
        require(_who == payer, "onlyPayer allowed!");
        _;
    }

    /**
    * @dev fallback payable function
    */
    function () public payable {}

    /**
    * @dev deposit method to not put any code inside the fallback method
    * for gas usage reasons
    */
    function deposit() public payable {
```

```
        deposits = msg.value;
    }

    /**
    * @dev return all the ammout deposited
    */
    function getDeposits() public view returns(uint256) {
        return deposits;
    }

    /**
    * @dev set the payee of the escrow contract
    * @param _payee the payee
    */
    function setPayee(address _payee) public {
        payee = _payee;
    }

    /**
    * @dev hit step function
    * @param _who should be the payer
    */
    function hit(address _who) public onlyPayer(_who) returns(bool success) {
        uint256 payment = deposits.div(steps);

        require(payment != 0, "payment can not be 0");
        require(address(this).balance >= payment, "balance should be higher than pa
yment");

        steps = steps.sub(1);
        deposits = deposits.sub(payment);

        payee.transfer(payment);
        success = steps == 0;
    }

    /**
    * @dev cancel escrow, refunding payer
    */
    function cancel() public {
        require(deposits != 0, "deposits can not be 0");
        require(address(this).balance == deposits, "balance should be equal to depo
sits");

        payer.transfer(deposits);
    }
}
```

Used in experimental repository.

EscrowThreeWay.sol

This contract is like an *escrow* contract between two parts, but the difference is that the money is taken for both side and the total amount is sent just to one side.

Note: This contract is created using the library *openzeppelin-solidity* with only one change in *ERC20.sol* which allows using *transferFrom* without any limit.

```
pragma solidity ^0.4.24;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "openzeppelin-solidity/contracts/token/ERC20/ERC20.sol";
import "openzeppelin-solidity/contracts/token/ERC20/SafeERC20.sol";

contract Escrow {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    struct Cases {
        bool isOpen;
        address from;
        address to;
        uint256 amount;
    }

    Cases[] private cases;
    uint256 private totalCases;
    address private escrowWallet;
    // The token being sold
    IERC20 private _token;

    constructor(address _escrowWallet, IERC20 _erc20) public {
        escrowWallet = _escrowWallet;
        _token = _erc20;
        totalCases = 0;
    }

    function escrow(address _to, uint256 _value) public {
        cases.push(Cases(true, msg.sender, _to, _value));
        _token.transferFrom(msg.sender, escrowWallet, _value / 2);
        _token.transferFrom(_to, escrowWallet, _value / 2);
        totalCases = totalCases.add(1);
    }

    function getTotalCases() public view returns(uint256) {
        return totalCases;
    }
```

```
    function revertTo(uint256 _caseId, bool _returnTo) public {
        require(cases[_caseId].isOpen == true, "");
        if(_returnTo == true) {
            _token.transferFrom(escrowWallet, cases[_caseId].to, cases[_caseId].amo
unt);
        } else {
            _token.transferFrom(escrowWallet, cases[_caseId].from, cases[_caseId].a
mount);
        }
        cases[_caseId].isOpen = false;
    }
}
```

Used in https://github.com/HQ20/strea

Miscellaneous

This chapter is about miscellaneous tricks, very useful, but used just a few times.

Parse URL

In NodeJS

```
import url from 'url';

const parts = url.parse(window.location.href, true);
const { id } = parts.query;
```