# Техническая документация
# по проекту
# ReturnGasToken (RGT)

Цель создания проекта RGT – возврат затраченного газа за транзакции в сети Ethereum. В прямом смысле вернуть газ за транзакцию невозможно. Даже при удалении большого объема данных сумма gas refund не может превышать 50% от всего затраченного газа. Поэтому используется механизм **токенизации**, то есть перевод затраченного газа в токены. Сама идея достаточно проста, однако, при самой простой реализации возникает вопрос – кто будет покупать эти токены? Ведь смысл проекта в возврате потраченных эфиров. Был придуман и реализован механизм **награды** пользователей с большим количеством токенов. Вознаграждение увеличивается пропорционально количеству имеющихся у пользователя токенов. Если пользователь имеет 100% токенов, его вознаграждение будет больше в 100 раз. Если пользователь имеет 1% всех токенов, вознаграждение будет больше в 2 раза. Это создаст спрос на токены RGT у тех, кто часто выполняет дорогостоящие транзакции.

## Кому это нужно

Проект выгоден всем. Но рассмотрим подробнее.

1. Обычные пользователи могут все транзакции пропускать через проект RGT, имея прибыль фактически ни за что.
2. Администраторы хайпов на смарт-контрактах могут добавить небольшую инструкцию в свой контракт, получая по 40 RGT за каждую транзакцию пользователей (газ будет оплачиваться ими).

## Как работает контракт

Для каждого пользователя создается дочерний RGT-контракт. Газ, затраченный на создание контракта, не токенизируется, чтобы предотвратить злоупотребление. Это действие производится один раз для пользователя во время первой транзакции в проекте RGT. Контракт, по своей сути, является посредником между пользователем и другими пользователями или контрактами. Для упрощения работы пользователей с контрактом RGT был создан удобный интерфейс, взаимодействующий с RGT через MetaMask или Trust Wallet. Токены можно производить не только взаимодействуя с другими адресами или контрактами, но и просто отправляя 0 на адрес контракта RGT. В этом случае будет начислено 40 RGT – стандартная величина, т.к. на логику контракта RGT расходуется порядка 40000 единиц газа, а RGT=0.001 газа.

Разберем работу контракта подробно.

определяем версию компилятора

```
pragma solidity ^0.4.24;
```

стандартный интерфейс (пример есть на ethereum.org)

```
interface tokenRecipient {

    function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData)
external;

}
```

дочерние контракты, создающиеся индивидуально для каждого пользователя

```
contract Interacting {
```

записываем создателя контракта в owner

```
    address private owner = msg.sender;
```

модификатор, разрешающий использование функций только создателем контракта

```
    modifier onlyOwner {

        require(msg.sender == owner);

        _;

    }
```

шлюз, отправляющий Эфиры по адресу _to, это может быть личный кошелек или контракт (transfer не используется, т.к. у него ограничение по газу 2300)

```
    function sendEther(address _to) external payable onlyOwner {

        require(_to.call.value(msg.value)(''));

    }
```

шлюз, вызывающий определенный метод контракта _contract с данными _extraData

```
    function callMethod(address _contract, bytes _extraData) external payable onlyOwner {

        require(_contract.call.value(msg.value)(_extraData));

    }
```

вывод средств с контракта его создателем на личный кошелек _to

```
    function withdrawEther(address _to) external onlyOwner {

        _to.transfer(address(this).balance);

    }
```

разрешаем поступление эфиров на контракт

```
    function () external payable {


    }

}
```

основной контракт

```
contract RGT {

    string public name = 'RGT';

    string public symbol = 'RGT';

    uint8 public decimals = 18;

    uint public k = 10 ** uint(decimals);
```

```
uint public k1000 = k / 1000;
uint public totalSupply = 1000000000 * k;
```

стандартный код токена ERC-20

```
// This creates an array with all balances
mapping (address => uint256) public balanceOf;
mapping (address => mapping (address => uint256)) public allowance;
mapping (address => address) public contracts;

// This generates a public event on the blockchain that will notify clients
event Transfer(address indexed from, address indexed to, uint256 value);

// This generates a public event on the blockchain that will notify clients
event Approval(address indexed _owner, address indexed _spender, uint256 _value);

// This notifies clients about the amount burnt
event Burn(address indexed from, uint256 value);

/**
 * Constructor function
 *
 * Initializes contract with initial supply tokens to the creator of the contract
 */
constructor() public {
    balanceOf[msg.sender] = totalSupply;
}

/**
 * Internal transfer, only can be called by this contract
 */
function _transfer(address _from, address _to, uint _value) internal {
    // Prevent transfer to 0x0 address. Use burn() instead
    require(_to != address(0x0));
    // Check if the sender has enough
    require(balanceOf[_from] >= _value);
    // Check for overflows
    require(balanceOf[_to] + _value >= balanceOf[_to]);
    // Save this for an assertion in the future
    uint previousBalances = balanceOf[_from] + balanceOf[_to];
    // Subtract from the sender
    balanceOf[_from] -= _value;
```

```
        // Add the same to the recipient
        balanceOf[_to] += _value;
        emit Transfer(_from, _to, _value);
        // Asserts are used to use static analysis to find bugs in your code. They should never
fail
        assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
    }

    /**
     * Transfer tokens
     *
     * Send `_value` tokens to `_to` from your account
     *
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        _transfer(msg.sender, _to, _value);
        return true;
    }

    /**
     * Transfer tokens from other address
     *
     * Send `_value` tokens to `_to` on behalf of `_from`
     *
     * @param _from The address of the sender
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
{
        require(_value <= allowance[_from][msg.sender]);     // Check allowance
        allowance[_from][msg.sender] -= _value;
        _transfer(_from, _to, _value);
        return true;
    }

    /**
     * Set allowance for other address
     *
     * Allows `_spender` to spend no more than `_value` tokens on your behalf
```

```
     *
     * @param _spender The address authorized to spend
     * @param _value the max amount they can spend
     */
    function approve(address _spender, uint256 _value) public
        returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }

    /**
     * Set allowance for other address and notify
     *
     * Allows `_spender` to spend no more than `_value` tokens on your behalf, and then ping the
contract about it
     *
     * @param _spender The address authorized to spend
     * @param _value the max amount they can spend
     * @param _extraData some extra information to send to the approved contract
     */
    function approveAndCall(address _spender, uint256 _value, bytes memory _extraData)
        public
        returns (bool success) {
        tokenRecipient spender = tokenRecipient(_spender);
        if (approve(_spender, _value)) {
            spender.receiveApproval(msg.sender, _value, address(this), _extraData);
            return true;
        }
    }

    /**
     * Destroy tokens
     *
     * Remove `_value` tokens from the system irreversibly
     *
     * @param _value the amount of money to burn
     */
    function burn(uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);   // Check if the sender has enough
        balanceOf[msg.sender] -= _value;            // Subtract from the sender
```

```
        totalSupply -= _value;                          // Updates totalSupply
        emit Burn(msg.sender, _value);
        return true;
    }

    /**
     * Destroy tokens from other account
     *
     * Remove `_value` tokens from the system irreversibly on behalf of `_from`.
     *
     * @param _from the address of the sender
     * @param _value the amount of money to burn
     */
    function burnFrom(address _from, uint256 _value) public returns (bool success) {
        require(balanceOf[_from] >= _value);                  // Check if the targeted balance is
enough
        require(_value <= allowance[_from][msg.sender]);    // Check allowance
        balanceOf[_from] -= _value;                           // Subtract from the targeted balance
        allowance[_from][msg.sender] -= _value;             // Subtract from the sender's allowance
        totalSupply -= _value;                                // Update totalSupply
        emit Burn(_from, _value);
        return true;
    }
```

выпуск новых токенов

```
    function mint(uint _amount) internal {
        _amount = (_amount + 40000) * k1000 * (1 + balanceOf[msg.sender] * 99 / totalSupply);
        balanceOf[msg.sender] += _amount;
        totalSupply += _amount;
        require(totalSupply >= _amount);
        emit Transfer(address(0), address(this), _amount);
        emit Transfer(address(this), msg.sender, _amount);
    }
```

модификатор, создающий дочерний контракт, если необходимо. Владельцем дочернего контракта будет контракт RGT

```
    modifier createOwnContractIfNeeded {
        if (contracts[msg.sender] == 0x0) {
            contracts[msg.sender] = new Interacting();
        }
        _;
    }
```

отправка эфира с получением вознаграждения

```
    function sendEther(address _to) external payable createOwnContractIfNeeded {

        uint gas = gasleft();

        Interacting(contracts[msg.sender]).sendEther.value(msg.value)(_to);

        mint(gas - gasleft());

    }
```

взаимодействие с другим контрактом с получением вознаграждения

```
    function callMethod(address _contract, bytes _extraData) external payable
createOwnContractIfNeeded {

        uint gas = gasleft();

        Interacting(contracts[msg.sender]).callMethod.value(msg.value)(_contract, _extraData);

        mint(gas - gasleft());

    }
```

вывод средств с дочернего контракта

```
    function withdrawEther() external payable createOwnContractIfNeeded {

        Interacting(contracts[msg.sender]).withdrawEther(msg.sender);

    }
```

майнинг

```
    function () external payable createOwnContractIfNeeded {

        require(msg.value == 0);

        mint(0);

    }

}
```