Работа со смарт-контрактом.

Основы работы со смарт-контрактом

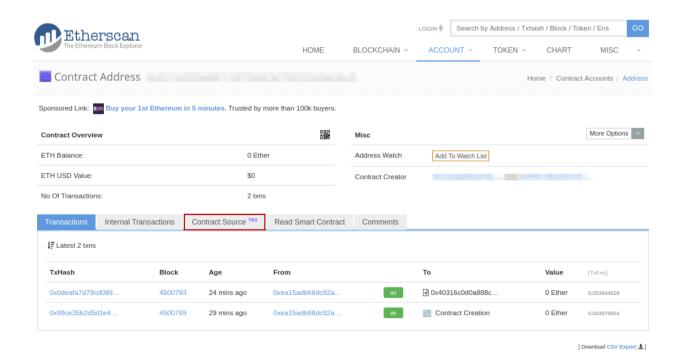
Взаимодействие со смарт-контрактом

Взаимодействие со смарт-контрактом происходит посредством вызова функций контракта с указанием параметров. Для взаимодействия необходимо знать адрес смарт-контракта. А на основе адреса нужно получить ABI-интерфейс. По ABI интерфейсу можно вызвать целевую функцию.

Получение АВІ интерфейса

Для получения интерфейса вам необходимо:

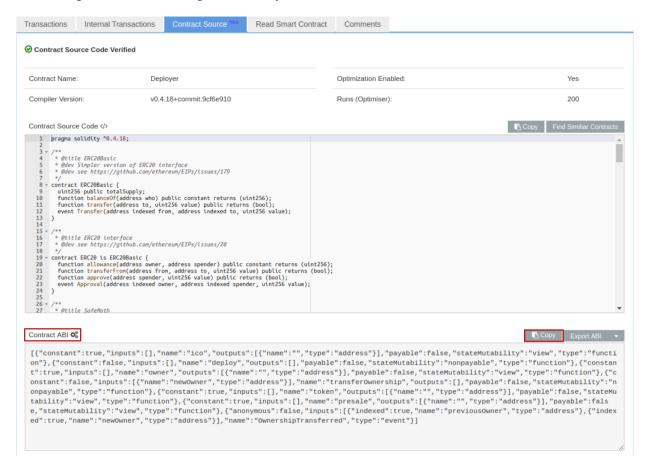
- 1. открыть https://etherscan.io/
- 2. В правом верхнем углу в окне с надписью «search by address» ввести адрес целевого смарт-контракта и нажать «go»
- 3. Затем нажать на вкладку «contract source»:



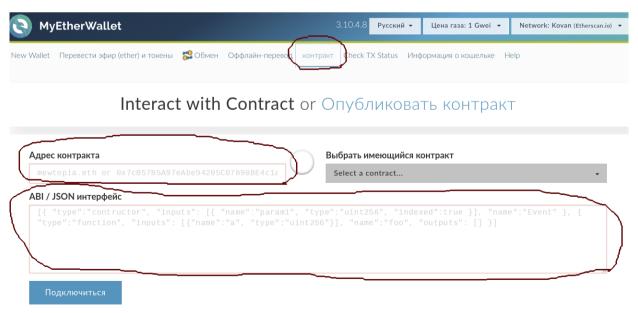
4. Найти поле с надписью «Contract ABI» и нажать на кнопку «сору» справа от надписи Интерфейс ABI контракта теперь скопирован в буффер обмена.

Вызов функции

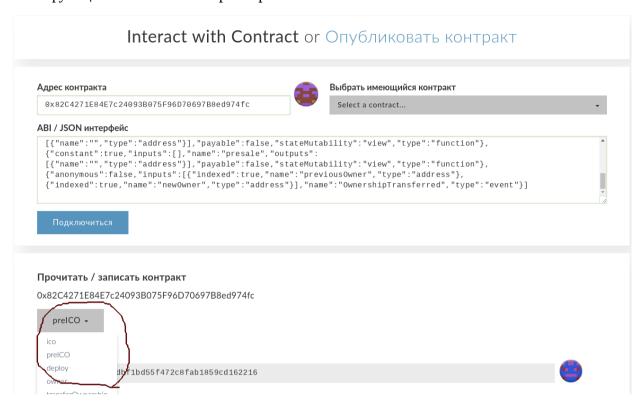
1. Откройте кошелек https://www.myetherwallet.com



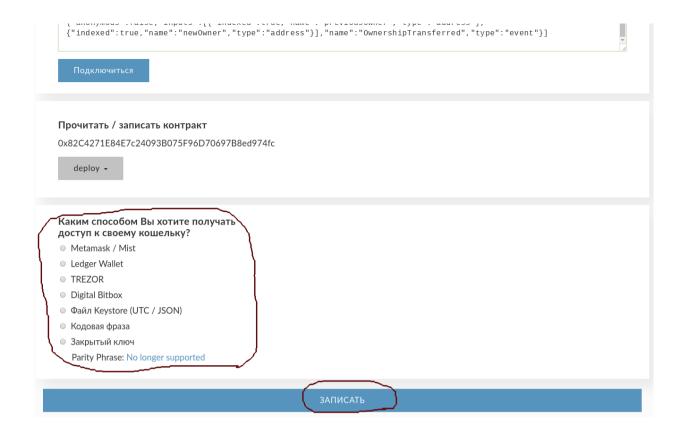
2. Выберите вкладку «контракты» и заполните поля **адрес** и **ABI** (мы его получали в предыдущей части инструкции) и нажмите на «подключиться»



3. После этого в нам станет доступен список функций контракта. Выбираем целевую функцию и заполняем параметры



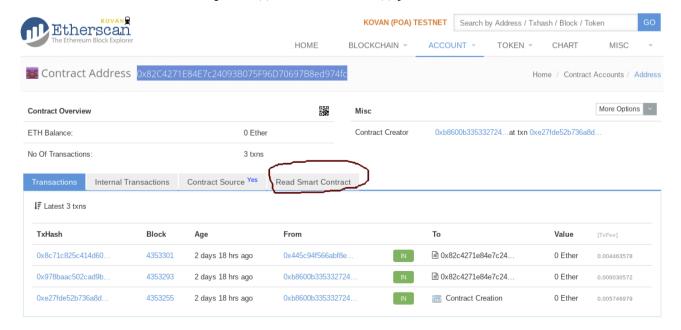
4. После выбора функции и заполнения параметров получите доступ к кошелькувладельцу смарт-контракта удобным вам способом и нажмите на кнопку «записать» и «Да, я уверен, выполнить транзакцию». Внимание — перед тем как нажать «записать» внимательно проверьте введенные данные. Часто формат данных отличается от привычного пользователю вида.



5. После этого внизу появится зеленая панель со ссылкой «Verify transaction». Нажмите на эту ссылку и откроется Etherscan. Ждите пока транзакция не выполнится.

Как посмотреть интерфейс смарт-контракта

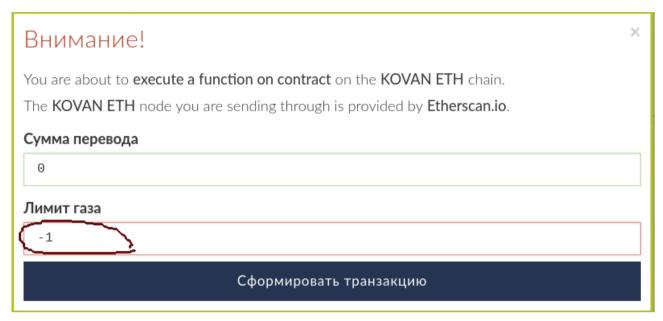
- 1. открыть https://etherscan.io/
- 2. В правом верхнем углу в окне с надписью «search by address» ввести адрес целевого смарт-контракта и нажать «go»
- 3. Появится окно в котором надо нажать на вкладку «Read smart contract»



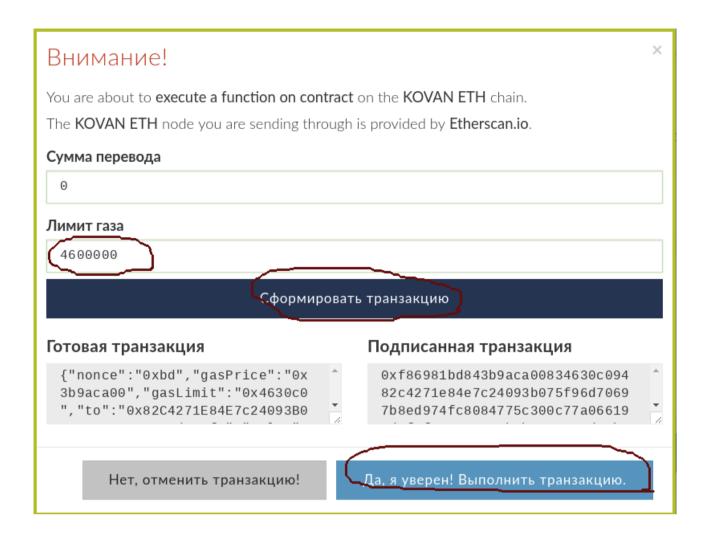
Подготовка к работе — только если контракт заливали самостоятельно!

Адрес смарт-контракта конфигуратора:

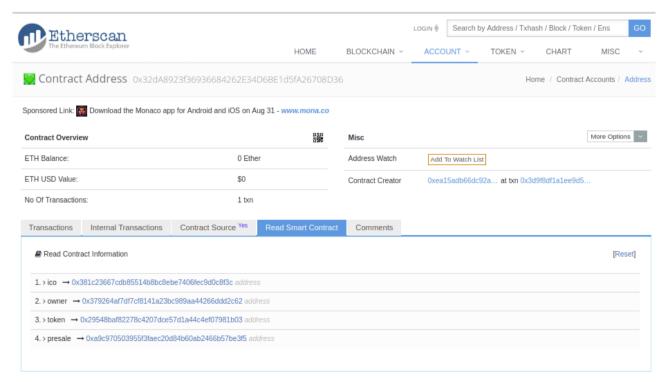
Для начала работы со смарт-контрактами их необходимо проинициализировать. Для этого нужно у контракта-конфигуратора вызывать функцию deploy. Во время ее вызова MyEtherWalet может не определить количество газа, необходимое для вызова функции. И поэтому поставит «-1».



Исправьте число на 4600000 и нажмите «Записать транзакцию» а затем «Да, я уверен! Выполнить транзакцию».



После выполнения транзакции (вам нужно дождаться ее выполнения). Вы можете открыть интерфейс смарт-контракта и вы увидите 4 поля <u>с ненулевыми</u> адресами:



Важные для нас адреса смарт-контрактов:

- 1. presale адрес Presale контракта именно на него инвесторы должны закидывать эфир во время presale-распродажи. Также все даты распродажи, а также цена и функция завершения распродажи вызывается по этому адресу.
- 2. Ісо адрес ICO контракта именно на него инвесторы должны закидывать эфир во время ICO-распродажи. Также все даты распродажи, а также цена и функция завершения распродажи вызывается по этому адресу.
- 3. Token адрес смарт-контракта токена. Этот адрес пользователи будут добавлять себе в кошелек для отображения токена.

Конфигурация

Адреса кошельков:

Presale	Сбор	0x4bB656423f5476FeC4AA729aB7B4EE0fc4d0B314
	Баунти	0xcACBE5d8Fb017407907026804Fe8BE64B08511f4
	Разработчики	0xa20C62282bEC52F9dA240dB8cFFc5B2fc8586652
	Консультанты	0xD3D85a495c7E25eAd39793F959d04ACcDf87e01b
	Владелец	0x379264aF7df7CF8141a23bC989aa44266DDD2c62
	Разработчики смарт-контрактов	0xEA15Adb66DC92a4BbCcC8Bf32fd25E2e86a2A770
ICO	Сбор	0x65954fb8f45b40c9A60dffF3c8f4F39839Bf3596
	Банути	0x6b9f45A54cDe417640f7D49D13451D7e2e9b8918
	Разработчики	0x55A9E5b55F067078E045c72088C3888Bbcd9a64b
	Консультанты	0x3e11Ff0BDd160C1D85cdf04e012eA9286ae1A964
	Владелец	0x379264aF7df7CF8141a23bC989aa44266DDD2c62

Адреса контрактов:

Контракт	Адрес
Токен	0x48Ca21691aba904a97a2f6807d5433B9B7AE3197
Presale	0x86b14014D74981eBE02AF2981F0fc9D0bA73C2A2
ICO	0x548BE6D7065be1870baC64435166871c6E82eA13

Ссылки:

Контракт	Адрес
Токен	https://etherscan.io/token/0x48Ca21691aba904a97a2f6807d5433B9B7AE3197
Presale	https://etherscan.io/address/0x86b14014D74981eBE02AF2981F0fc9D0bA73C2A2
ICO	https://etherscan.io/address/0x548BE6D7065be1870baC64435166871c6E82eA13

Функция finishMinting — должна вызываться у каждой распродажи после того, как закончился срок распродажи

Рекомендованные значения для отправки транзакции инвестора

Обычно кошельки сами способны определить количество газа и цену. Но не всегда. Поэтому вводятся рекомендованные значения:

- 1. Меньше чем 0.1 эфира контракт не принимает.
- 2. Количество газа 250 000

pragma solidity \0.4.18;

3. Цена газа — не менее 21 Gwei. Транзакции с меньшим количеством газа идти будут дольше. Так транзакции с ценой газа в 1 Gwei при большой нагрузке на сеть могут идти несколько недель.

Памятка владельцу контрактов

Владелец контрактов должен после каждой распродажи вызвать функцию finishMinting() у соответствующего контракта распродажи. А именно.

- 1. По окончанию распродажи у контракта presale вызвать *finishMinting()*, адрес контракта: **0x86b14014D74981eBE02AF2981F0fc9D0bA73C2A2**
- 2. По окончанию распродажи у контракта ICO вызвать *finishMinting()*, адрес контракта: **0x548BE6D7065be1870baC64435166871c6E82eA13**
- 3. Для вывода средств с контракта Presale нужно вызвать функцию withdraw.

Перед вызовом убедитесь, что адрес контракта соответствует текущей распродаже!

Расположение адресов в смарт-контракте

```
/**

* @title ERC20Basic

* @dev Simpler version of ERC20 interface

* @dev see https://github.com/ethereum/EIPs/issues/179

*/

contract ERC20Basic {
    uint256 public totalSupply;
    function balanceOf(address who) public constant returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
}

/**

* @title ERC20 interface

* @dev see https://github.com/ethereum/EIPs/issues/20

*/
```

```
contract ERC20 is ERC20Basic {
 function allowance(address owner, address spender) public constant returns (uint256);
 function transferFrom(address from, address to, uint256 value) public returns (bool);
 function approve(address spender, uint256 value) public returns (bool);
 event Approval(address indexed owner, address indexed spender, uint256 value);
/**
* @title SafeMath
* @dev Math operations with safety checks that throw on error
library SafeMath {
 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a * b;
  assert(a == 0 \parallel c / a == b);
  return c;
 function div(uint256 a, uint256 b) internal pure returns (uint256) {
  // assert(b > 0); // Solidity automatically throws when dividing by 0
  uint256 c = a / b;
  // assert(a == b * c + a \% b); // There is no case in which this doesn't hold
  return c:
 }
 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
  assert(b <= a);</pre>
  return a - b;
 }
 function add(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a + b;
  assert(c \ge a);
  return c;
}
* @title Basic token
* @dev Basic version of StandardToken, with no allowances.
contract BasicToken is ERC20Basic {
 using SafeMath for uint256;
 mapping(address => uint256) balances;
 /**
 * @dev transfer token for a specified address
 * @param _to The address to transfer to.
 * @param _value The amount to be transferred.
 function transfer(address _to, uint256 _value) public returns (bool) {
```

```
require( to != address(0));
  require( value <= balances[msg.sender]);
  // SafeMath.sub will throw if there is not enough balance.
  balances[msg.sender] = balances[msg.sender].sub(_value);
  balances[_to] = balances[_to].add(_value);
  Transfer(msg.sender, _to, _value);
  return true;
 * @dev Gets the balance of the specified address.
 * @param owner The address to guery the the balance of.
 * @return An uint256 representing the amount owned by the passed address.
 function balanceOf(address owner) public constant returns (uint256 balance) {
  return balances[_owner];
}
/**
* @title Standard ERC20 token
* @dev Implementation of the basic standard token.
* @dev https://github.com/ethereum/EIPs/issues/20
* @dev Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart contract/FirstBloodToken.sol
*/
contract StandardToken is ERC20, BasicToken {
 mapping (address => mapping (address => uint256)) internal allowed;
 /**
 * @dev Transfer tokens from one address to another
 * @param _from address The address which you want to send tokens from
 * @param to address The address which you want to transfer to
 * @param _value uint256 the amount of tokens to be transferred
 function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
  require(_to != address(0));
  require( value <= balances[ from]);</pre>
  require(_value <= allowed[_from][msg.sender]);</pre>
  balances[_from] = balances[_from].sub(_value);
  balances[_to] = balances[_to].add(_value);
  allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  Transfer(_from, _to, _value);
  return true;
```

/**

* @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.

*

* Beware that changing an allowance with this method brings the risk that someone may use both the old

* and the new allowance by unfortunate transaction ordering. One possible solution to mitigate

this

* race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:

```
* https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
* @param spender The address which will spend the funds.
* @param value The amount of tokens to be spent.
function approve(address _spender, uint256 _value) public returns (bool) {
 allowed[msg.sender][ spender] = value;
 Approval(msg.sender, _spender, _value);
 return true;
}
/**
* @dev Function to check the amount of tokens that an owner allowed to a spender.
* @param owner address The address which owns the funds.
* @param spender address The address which will spend the funds.
* @return A uint256 specifying the amount of tokens still available for the spender.
function allowance(address owner, address spender) public constant returns (uint256 remaining)
 return allowed[_owner][_spender];
* approve should be called when allowed[_spender] == 0. To increment
* allowed value is better to use this function to avoid 2 calls (and wait until
* the first transaction is mined)
* From MonolithDAO Token.sol
function increase Approval (address spender, uint added Value) public returns (bool success) {
 allowed[msg.sender][ spender] = allowed[msg.sender][ spender].add( addedValue);
 Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
 return true;
}
function decreaseApproval (address _spender, uint _subtractedValue) public returns (bool success)
 uint oldValue = allowed[msg.sender][ spender];
 if (_subtractedValue > oldValue) {
  allowed[msg.sender][_spender] = 0;
 } else {
  allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
 Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
```

```
return true;
 function () public payable {
  revert();
 }
}
/**
* @title Ownable
* @dev The Ownable contract has an owner address, and provides basic authorization control
* functions, this simplifies the implementation of "user permissions".
contract Ownable {
 address public owner;
 event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
 /**
 * @dev The Ownable constructor sets the original `owner` of the contract to the sender
 * account.
 function Ownable() public {
  owner = msg.sender;
 /**
 * @dev Throws if called by any account other than the owner.
 modifier onlyOwner() {
  require(msg.sender == owner);
  * @dev Allows the current owner to transfer control of the contract to a newOwner.
 * @param newOwner The address to transfer ownership to.
 function transferOwnership(address newOwner) onlyOwner public {
  require(newOwner != address(0));
  OwnershipTransferred(owner, newOwner);
  owner = newOwner;
 }
}
contract LockableChanges is Ownable {
```

```
bool public changesLocked;
 modifier notLocked() {
  require(!changesLocked);
 function lockChanges() public onlyOwner {
  changesLocked = true;
}
* @title Mintable token
* @dev Simple ERC20 Token example, with mintable token creation
* @dev Issue: * https://github.com/OpenZeppelin/zeppelin-solidity/issues/120
* Based on code by TokenMarketNet:
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
*/
contract GENSharesToken is StandardToken, Ownable {
 using SafeMath for uint256;
 event Mint(address indexed to, uint256 amount);
 event MintFinished();
 string public constant name = "GEN Shares";
 string public constant symbol = "GEN";
 uint32 public constant decimals = 18;
 bool public mintingFinished = false;
 address public saleAgent;
 function setSaleAgent(address newSaleAgent) public {
  require(saleAgent == msg.sender || owner == msg.sender);
  saleAgent = newSaleAgent;
 function mint(address _to, uint256 _amount) public returns (bool) {
  require(!mintingFinished);
  require(msg.sender == saleAgent);
  totalSupply = totalSupply.add(_amount);
  balances[_to] = balances[_to].add(_amount);
  Mint(_to, _amount);
  Transfer(address(0), _to, _amount);
  return true;
```

```
}
 function finishMinting() public returns (bool) {
  require(!mintingFinished);
  require(msg.sender == owner || msg.sender == saleAgent);
  mintingFinished = true;
  MintFinished();
  return true;
}
contract CommonCrowdsale is Ownable, LockableChanges {
 using SafeMath for uint256;
 uint public constant PERCENT_RATE = 100;
 uint public price;
 uint public minInvestedLimit;
 uint public hardcap;
 uint public start;
 uint public end;
 uint public invested;
 uint public minted;
 address public wallet;
 address public bountyTokensWallet;
 address public devTokensWallet;
 address public advisorsTokensWallet;
 uint public bountyTokensPercent;
 uint public devTokensPercent;
 uint public advisorsTokensPercent;
 struct Bonus {
  uint periodInDays;
  uint bonus;
 Bonus[] public bonuses;
```

```
GENSharesToken public token;
 modifier saleIsOn() {
  require(msg.value >= minInvestedLimit && now >= start && now < end && invested <
hardcap);
 function setHardcap(uint newHardcap) public onlyOwner notLocked {
  hardcap = newHardcap;
 function setStart(uint newStart) public onlyOwner notLocked {
  start = newStart;
 function setBountyTokensPercent(uint newBountyTokensPercent) public onlyOwner notLocked {
  bountyTokensPercent = newBountyTokensPercent;
 function setAdvisorsTokensPercent(uint newAdvisorsTokensPercent) public onlyOwner
notLocked {
  advisorsTokensPercent = newAdvisorsTokensPercent;
 function setDevTokensPercent(uint newDevTokensPercent) public onlyOwner notLocked {
  devTokensPercent = newDevTokensPercent;
 function setBountyTokensWallet(address newBountyTokensWallet) public onlyOwner notLocked
  bountyTokensWallet;
 function setAdvisorsTokensWallet(address newAdvisorsTokensWallet) public onlyOwner
notLocked {
  advisorsTokensWallet = newAdvisorsTokensWallet:
 function setDevTokensWallet(address newDevTokensWallet) public onlyOwner notLocked {
  devTokensWallet = newDevTokensWallet;
 function setEnd(uint newEnd) public onlyOwner notLocked {
  require(start < newEnd);</pre>
  end = newEnd;
 function setToken(address newToken) public onlyOwner notLocked {
  token = GENSharesToken(newToken);
 }
```

```
function setWallet(address newWallet) public onlyOwner notLocked {
  wallet = newWallet;
 function setPrice(uint newPrice) public onlyOwner notLocked {
  price = newPrice;
 function setMinInvestedLimit(uint newMinInvestedLimit) public onlyOwner notLocked {
  minInvestedLimit = newMinInvestedLimit;
 function bonusesCount() public constant returns(uint) {
  return bonuses.length;
 function addBonus(uint limit, uint bonus) public onlyOwner notLocked {
  bonuses.push(Bonus(limit, bonus));
 function mintExtendedTokens() internal {
  uint extendedTokensPercent =
bountyTokensPercent.add(devTokensPercent).add(advisorsTokensPercent);
  uint extendedTokens =
minted.mul(extendedTokensPercent).div(PERCENT_RATE.sub(extendedTokensPercent));
  uint summarvTokens = extendedTokens + minted;
  uint bountyTokens = summaryTokens.mul(bountyTokensPercent).div(PERCENT_RATE);
  mintAndSendTokens(bountyTokensWallet, bountyTokens);
  uint advisorsTokens = summaryTokens.mul(advisorsTokensPercent).div(PERCENT_RATE);
  mintAndSendTokens(advisorsTokensWallet, advisorsTokens);
  uint devTokens = summaryTokens.sub(advisorsTokens).sub(bountyTokens);
  mintAndSendTokens(devTokensWallet, devTokens);
 }
 function mintAndSendTokens(address to, uint amount) internal {
  token.mint(to, amount);
  minted = minted.add(amount);
 }
 function calculateAndTransferTokens() internal {
  // update invested value
  invested = invested.add(msg.value);
  // calculate tokens
  uint tokens = msg.value.mul(price).div(1 ether);
  uint bonus = getBonus();
  if(bonus > 0) {
   tokens = tokens.add(tokens.mul(bonus).div(100));
```

```
}
  // transfer tokens
  mintAndSendTokens(msg.sender, tokens);
 function getBonus() public constant returns(uint) {
  uint prevTimeLimit = start;
  for (uint i = 0; i < bonuses.length; i++) {
   Bonus storage bonus = bonuses[i];
   prevTimeLimit += bonus.periodInDays * 1 days;
   if (now < prevTimeLimit)</pre>
    return bonus.bonus;
  return 0;
 function createTokens() public payable;
 function() external payable {
  createTokens();
 function retrieveTokens(address anotherToken) public onlyOwner {
  ERC20 alienToken = ERC20(anotherToken);
  alienToken.transfer(wallet, token.balanceOf(this));
 }
}
contract Presale is CommonCrowdsale {
 uint public devLimit;
 uint public softcap;
 bool public refundOn;
 bool public softcapAchieved;
 bool public devWithdrawn;
 address public devWallet;
 address public nextSaleAgent;
 mapping (address => uint) public balances;
 function setNextSaleAgent(address newNextSaleAgent) public onlyOwner notLocked {
  nextSaleAgent = newNextSaleAgent;
```

```
function setSoftcap(uint newSoftcap) public onlyOwner notLocked {
 softcap = newSoftcap;
function setDevWallet(address newDevWallet) public onlyOwner notLocked {
 devWallet = newDevWallet;
function setDevLimit(uint newDevLimit) public onlyOwner notLocked {
 devLimit = newDevLimit;
function refund() public {
 require(now > start && refundOn && balances[msg.sender] > 0);
 uint value = balances[msg.sender];
 balances[msg.sender] = 0;
 msg.sender.transfer(value);
function createTokens() public payable saleIsOn {
 balances[msg.sender] = balances[msg.sender].add(msg.value);
 calculateAndTransferTokens();
 if(!softcapAchieved && invested >= softcap) {
  softcapAchieved = true;
}
function widthrawDev() public {
 require(softcapAchieved);
 require(devWallet == msg.sender || owner == msg.sender);
 if(!devWithdrawn) {
  devWithdrawn = true;
  devWallet.transfer(devLimit);
function widthraw() public {
 require(softcapAchieved);
 require(owner == msg.sender);
 widthrawDev();
 wallet.transfer(this.balance);
}
function finishMinting() public onlyOwner {
 if(!softcapAchieved) {
  refundOn = true;
  token.finishMinting();
 } else {
  mintExtendedTokens();
  token.setSaleAgent(nextSaleAgent);
}
```

```
}
contract ICO is CommonCrowdsale {
 function finishMinting() public onlyOwner {
  mintExtendedTokens();
  token.finishMinting();
 function createTokens() public payable saleIsOn {
  calculateAndTransferTokens();
  wallet.transfer(msg.value);
}
contract Deployer is Ownable {
 Presale public presale;
 ICO public ico;
 GENSharesToken public token;
 function deploy() public onlyOwner {
  owner = 0x379264aF7df7CF8141a23bC989aa44266DDD2c62; // 1
  token = new GENSharesToken();
  presale = new Presale();
  presale.setToken(token);
  token.setSaleAgent(presale);
  presale.setMinInvestedLimit(4000000000000000000);
  presale.setPrice(25000000000000000000);
  presale.setBountyTokensPercent(4);
  presale.setAdvisorsTokensPercent(2);
  presale.setDevTokensPercent(10):
  presale.setSoftcap(40000000000000000000);
  presale.setHardcap(5000000000000000000000);
  presale.addBonus(7,50);
  presale.addBonus(7,40);
  presale.addBonus(100,35);
  presale.setStart(1511571600);
  presale.setEnd(1514156400);
  presale.setDevLimit(600000000000000000);
  presale.setWallet(0x4bB656423f5476FeC4AA729aB7B4EE0fc4d0B314); // 2
  presale.setBountyTokensWallet(0xcACBE5d8Fb017407907026804Fe8BE64B08511f4); // 3
  presale.setDevTokensWallet(0xa20C62282bEC52F9dA240dB8cFFc5B2fc8586652); // 4
  presale.setAdvisorsTokensWallet(0xD3D85a495c7E25eAd39793F959d04ACcDf87e01b); // 5
  presale.setDevWallet(0xEA15Adb66DC92a4BbCcC8Bf32fd25E2e86a2A770); // 6
```

```
ico = new ICO();
ico.setToken(token);
presale.setNextSaleAgent(ico);
ico.setMinInvestedLimit(100000000000000000);
ico.setPrice(25000000000000000000);
ico.setBountyTokensPercent(4);
ico.setAdvisorsTokensPercent(2);
ico.setDevTokensPercent(10);
ico.setHardcap(20600000000000000000000);
ico.addBonus(7,25);
ico.addBonus(7,10);
ico.setStart(1514163600);
ico.setEnd(1517356800);
ico.setWallet(0x65954fb8f45b40c9A60dffF3c8f4F39839Bf3596); // 7
ico.setBountyTokensWallet(0x6b9f45A54cDe417640f7D49D13451D7e2e9b8918); // 8
ico.setDevTokensWallet(0x55A9E5b55F067078E045c72088C3888Bbcd9a64b); // 9
ico.setAdvisorsTokensWallet(0x3e11Ff0BDd160C1D85cdf04e012eA9286ae1A964); // 10
presale.lockChanges();
ico.lockChanges();
presale.transferOwnership(owner);
ico.transferOwnership(owner);
token.transferOwnership(owner);
```

}

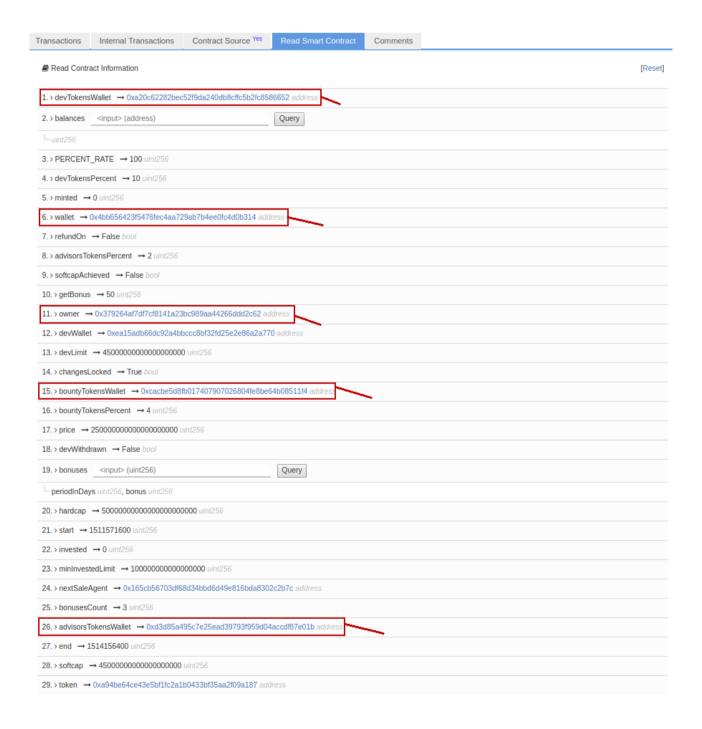
Контракт токена	Владелец	1
Контракт presale	Владелец	1
	Разработчики смарт-контракта	6
	Сбор	2
	Консультанты	5
	Баунти	3
	Разработчики	4
Контракт ICO	Владелец	1
	Сбор	7
	Баунти	8
	Разработчики	9
	Консультанты	10

B etherscan:

Контракт presale

Ссылка в etherscan:

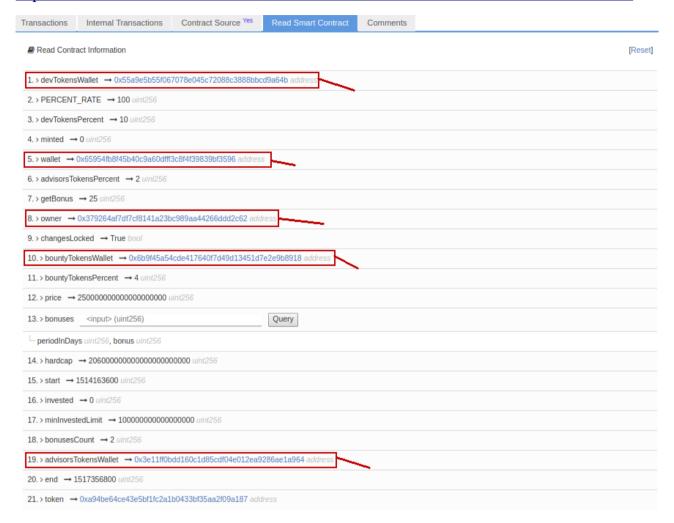
 $\underline{https://etherscan.io/address/0x86b14014D74981eBE02AF2981F0fc9D0bA73C2A2\#readContract}$



Контракт ІСО

Адрес в etherscan

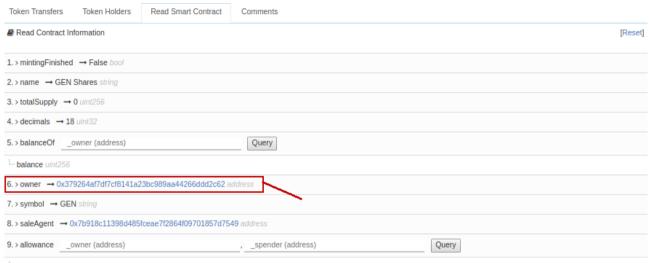
https://etherscan.io/address/0x548BE6D7065be1870baC64435166871c6E82eA13#readContract



Контракт токена

Адрес в etherscan

$\underline{https://etherscan.io/token/0x48Ca21691aba904a97a2f6807d5433B9B7AE3197\#readContract}$



⁻⁻ remaining uint256