
Vreo ICO Solidity Smart Contracts

Release 1

Sicos et al.

Jun 14, 2018

CONTENTS

1	VreoToken	1
2	PostKYCCrowdsale	2
3	VreoTokenSale	5
4	VreoTokenBounty	9

VREOTOKEN

```
1  pragma solidity 0.4.24;
2
3  import "../node_modules/zeppelin-solidity/contracts/token/ERC20/CappedToken.sol";
4  import "../node_modules/zeppelin-solidity/contracts/token/ERC20/PausableToken.sol";
5  import "../node_modules/zeppelin-solidity/contracts/token/ERC20/BurnableToken.sol";
6
7
8  /// @title VreoToken
9  /// @author Autogenerated from a Dia UML diagram
10 contract VreoToken is CappedToken, PausableToken, BurnableToken {
11
12     uint public constant TOTAL_TOKEN_CAP = 700000000e18; // = 700.000.000 e18
13
14     string public name = "MERO Token";
15     string public symbol = "MERO";
16     uint8 public decimals = 18;
17
18     /// @dev Constructor
19     constructor() public CappedToken(TOTAL_TOKEN_CAP) {
20     }
21
22     /// @dev Burn
23     /// @param _value A positive number
24     function burn(uint _value) public whenNotPaused {
25         super.burn(_value);
26     }
27
28 }
```

POSTKYCCROWDSALE

```
1  pragma solidity 0.4.24;
2
3  import "../node_modules/zeppelin-solidity/contracts/ownership/Ownable.sol";
4  import "../node_modules/zeppelin-solidity/contracts/crowdsale/Crowdsale.sol";
5
6
7  /// @title PostKYCCrowdsale
8  /// @author Autogenerated from a Dia UML diagram
9  contract PostKYCCrowdsale is Crowdsale, Ownable {
10
11     struct Investment {
12         bool isVerified;           // whether or not the investor passed the KYC process
13         uint totalWeiInvested;     // invested wei
14         uint pendingTokenAmount;  // amount of token quantum the investor wants to purchase
15     }
16
17     mapping(address => Investment) public investments;
18
19     /// @dev Log entry on investor verified
20     /// @param investor An Ethereum address
21     event InvestorVerified(address investor);
22
23     /// @dev Log entry on tokens delivered
24     /// @param investor An Ethereum address
25     /// @param amount A positive number
26     event TokensDelivered(address investor, uint amount);
27
28     /// @dev Log entry on investment withdrawn
29     /// @param investor An Ethereum address
30     /// @param value A positive number
31     event InvestmentWithdrawn(address investor, uint value);
32
33     /// @dev Verify investors
34     /// @param _investors An Ethereum address
35     function verifyInvestors(address[] _investors) public onlyOwner {
36         for (uint i = 0; i < _investors.length; ++i) {
37             address investor = _investors[i];
38             Investment storage investment = investments[investor];
39
40             if (!investment.isVerified) {
41                 investment.isVerified = true;
42
43                 emit InvestorVerified(investor);
44
45                 uint pendingTokenAmount = investment.pendingTokenAmount;
46
47                 if (pendingTokenAmount > 0) {
48                     investment.pendingTokenAmount = 0;
49                 }
40             }
41         }
42     }
43 }
```

```

50         _forwardFunds(investment.totalWeiInvested);
51         _deliverTokens(investor, pendingTokenAmount);
52
53         emit TokensDelivered(investor, pendingTokenAmount);
54     }
55 }
56 }
57 }
58
59 /// @dev Withdraw investment
60 function withdrawInvestment() public {
61     Investment storage investment = investments[msg.sender];
62
63     require(!investment.isVerified);
64
65     uint totalWeiInvested = investment.totalWeiInvested;
66
67     require(totalWeiInvested > 0);
68
69     investment.totalWeiInvested = 0;
70     investment.pendingTokenAmount = 0;
71
72     msg.sender.transfer(totalWeiInvested);
73
74     emit InvestmentWithdrawn(msg.sender, totalWeiInvested);
75 }
76
77 /// @dev Pre validate purchase
78 /// @param _beneficiary An Ethereum address
79 /// @param _weiAmount A positive number
80 function _preValidatePurchase(address _beneficiary, uint _weiAmount) internal {
81     require(_beneficiary == msg.sender);
82
83     super._preValidatePurchase(_beneficiary, _weiAmount);
84 }
85
86 /// @dev Process purchase
87 /// @param _beneficiary An Ethereum address
88 /// @param _tokenAmount A positive number
89 function _processPurchase(address _beneficiary, uint _tokenAmount) internal {
90     Investment storage investment = investments[msg.sender];
91     investment.totalWeiInvested = investment.totalWeiInvested.add(msg.value);
92
93     if (investment.isVerified) {
94         _deliverTokens(_beneficiary, _tokenAmount);
95
96         emit TokensDelivered(_beneficiary, _tokenAmount);
97     } else {
98         investment.pendingTokenAmount = investment.pendingTokenAmount.add(_tokenAmount);
99     }
100 }
101
102 /// @dev Forward funds
103 function _forwardFunds() internal {
104     // Ensure the investor was verified, i.e. his purchased tokens were delivered,
105     // before forwarding funds.
106     if (investments[msg.sender].isVerified) {
107         super._forwardFunds();
108     }
109 }
110
111 /// @dev Forward funds
112 /// @param _weiAmount A positive number

```

```
113     function _forwardFunds(uint _weiAmount) internal {  
114         wallet.transfer(_weiAmount);  
115     }  
116  
117 }
```

VREOTOKENSALE

```
1 pragma solidity 0.4.24;
2
3 import "../node_modules/zeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol";
4 import "../node_modules/zeppelin-solidity/contracts/crowdsale/distribution/FinalizableCrowdsale.sol
5 ↵";
6 import "../node_modules/zeppelin-solidity/contracts/crowdsale/emission/MintedCrowdsale.sol";
7 import "../PostKYCCrowdsale.sol";
8 import "../VreoToken.sol";
9
10 /// @title VreoTokenSale
11 /// @author Autogenerated from a Dia UML diagram
12 contract VreoTokenSale is PostKYCCrowdsale, FinalizableCrowdsale, MintedCrowdsale {
13
14     // Maximum number of tokens sold in Presale+Iconiq+Vreo sales
15     uint public constant TOTAL_TOKEN_CAP_OF_SALE = 450000000e18; // = 450.000.000 e18
16
17     // Extra tokens minted upon finalization
18     uint public constant TOKEN_SHARE_OF_TEAM = 85000000e18; // = 85.000.000 e18
19     uint public constant TOKEN_SHARE_OF_ADVISORS = 58000000e18; // = 58.000.000 e18
20     uint public constant TOKEN_SHARE_OF_LEGALS = 57000000e18; // = 57.000.000 e18
21     uint public constant TOKEN_SHARE_OF_BOUNTY = 50000000e18; // = 50.000.000 e18
22
23     // Extra token percentages
24     uint public constant BONUS_PCT_IN_ICONIQ_SALE = 30; // TBD
25     uint public constant BONUS_PCT_IN_VREO_SALE_PHASE_1 = 20;
26     uint public constant BONUS_PCT_IN_VREO_SALE_PHASE_2 = 10;
27
28     // Date/time constants
29     uint public constant ICONIQ_SALE_OPENING_TIME = 1530432000; // 2018-07-01 10:00:00 CEST
30     uint public constant ICONIQ_SALE_CLOSING_TIME = 1531598400; // 2018-07-14 22:00:00 CEST
31     uint public constant VREO_SALE_OPENING_TIME = 1532160000; // 2018-07-21 10:00:00 CEST
32     uint public constant VREO_SALE_PHASE_1_END_TIME = 1532462400; // 2018-07-24 22:00:00 CEST
33     uint public constant VREO_SALE_PHASE_2_END_TIME = 1533153600; // 2018-08-01 22:00:00 CEST
34     uint public constant VREO_SALE_CLOSING_TIME = 1534622400; // 2018-08-18 22:00:00 CEST
35     uint public constant KYC_VERIFICATION_END_TIME = 1535832000; // 2018-09-01 22:00:00 CEST
36
37     // Max amount of wei ICONIQ investors can buy per ICONIQ TOKEN_SHARE_OF_TEAM
38     uint public constant ICONIQ_TOKENS_NEEDED_PER_INVESTED_WEI = 500;
39
40
41     ERC20Basic public iconiqToken;
42     address public teamAddress;
43     address public advisorsAddress;
44     address public legalsAddress;
45     address public bountyAddress;
46
47     uint public remainingTokensForSale;
48
```

```
49  /// @dev Log entry on rate changed
50  /// @param newRate A positive number
51  event RateChanged(uint newRate);
52
53  /// @dev Constructor
54  /// @param _token A VreoToken
55  /// @param _rate A positive number
56  /// @param _iconiqToken An IconiqInterface
57  /// @param _teamAddress An Ethereum address
58  /// @param _advisorsAddress An Ethereum address
59  /// @param _legalsAddress An Ethereum address
60  /// @param _bountyAddress A VreoTokenBounty
61  /// @param _wallet An Ethereum address
62  constructor(
63      VreoToken _token,
64      uint _rate,
65      ERC20Basic _iconiqToken,
66      address _teamAddress,
67      address _advisorsAddress,
68      address _legalsAddress,
69      address _bountyAddress,
70      address _wallet
71  )
72  public
73  Crowdsale(_rate, _wallet, _token)
74  TimedCrowdsale(ICONIQ_SALE_OPENING_TIME, VREO_SALE_CLOSING_TIME)
75  {
76      // Token sanity check
77      require(_token.cap() >= TOTAL_TOKEN_CAP_OF_SALE
78          + TOKEN_SHARE_OF_TEAM
79          + TOKEN_SHARE_OF_ADVISORS
80          + TOKEN_SHARE_OF_LEGALS
81          + TOKEN_SHARE_OF_BOUNTY);
82
83      // Sanity check of addresses
84      require(address(_iconiqToken) != address(0)
85          && _teamAddress != address(0)
86          && _advisorsAddress != address(0)
87          && _legalsAddress != address(0)
88          && _bountyAddress != address(0));
89
90      iconiqToken = _iconiqToken;
91      teamAddress = _teamAddress;
92      advisorsAddress = _advisorsAddress;
93      legalsAddress = _legalsAddress;
94      bountyAddress = _bountyAddress;
95
96      remainingTokensForSale = TOTAL_TOKEN_CAP_OF_SALE;
97  }
98
99  /// @dev Distribute presale
100  /// @param _investors A list where each entry is an Ethereum address
101  /// @param _amounts A list where each entry is a positive number
102  function distributePresale(address[] _investors, uint[] _amounts) public onlyOwner {
103      require(_investors.length == _amounts.length);
104
105      uint totalAmount = 0;
106
107      for (uint i = 0; i < _investors.length; ++i) {
108          VreoToken(token).mint(_investors[i], _amounts[i]);
109          totalAmount = totalAmount.add(_amounts[i]);
110      }
111  }
```



```

112     remainingTokensForSale = remainingTokensForSale.sub(totalAmount);
113 }
114
115 /// @dev Set rate
116 /// @param _newRate A positive number
117 function setRate(uint _newRate) public onlyOwner {
118     // A rate change by a magnitude order of ten and above is rather a typo than intention.
119     // If it was indeed desired, several setRate transactions have to be sent.
120     require(rate / 10 < _newRate && _newRate < 10 * rate);
121
122     rate = _newRate;
123
124     emit RateChanged(_newRate);
125 }
126
127 function withdrawInvestment() public {
128     require(hasClosed());
129
130     super.withdrawInvestment();
131 }
132
133 function iconiqSaleOngoing() public view returns (bool) {
134     return ICONIQ_SALE_OPENING_TIME <= now && now <= ICONIQ_SALE_CLOSING_TIME;
135 }
136
137 function vreoSaleOngoing() public view returns (bool) {
138     return VREO_SALE_OPENING_TIME <= now && now <= VREO_SALE_CLOSING_TIME;
139 }
140
141 /// @dev Get maximum possible wei investment while Iconiq sale
142 /// @param _investor An Ethereum address
143 /// @return Maximum allowed wei investment
144 function getIconiqMaxInvestment(address _investor) public view returns (uint) {
145     // Ensure the investor has Iconiq tokens
146     uint iconiqBalance = iconiqToken.balanceOf(_investor);
147     uint prorataLimit = iconiqBalance.div(ICONIQ_TOKENS_NEEDED_PER_INVESTED_WEI);
148
149     // How many additional MEROs the ICONIQ investor can buy
150     return prorataLimit.sub(investments[_investor].totalWeiInvested);
151 }
152
153 /// @dev Pre validate purchase
154 /// @param _beneficiary An Ethereum address
155 /// @param _weiAmount A positive number
156 function _preValidatePurchase(address _beneficiary, uint _weiAmount) internal {
157     super._preValidatePurchase(_beneficiary, _weiAmount);
158
159     require(iconiqSaleOngoing() && getIconiqMaxInvestment(msg.sender) >= _weiAmount ||
160 ↪vreoSaleOngoing());
161 }
162
163 /// @dev Get token amount
164 /// @param _weiAmount A positive number
165 /// @return A positive number
166 function _getTokenAmount(uint _weiAmount) internal view returns (uint) {
167     uint tokenAmount = super._getTokenAmount(_weiAmount);
168
169     if (now <= ICONIQ_SALE_CLOSING_TIME) {
170         return tokenAmount.mul((100 + BONUS_PCT_IN_ICONIQ_SALE) / 100);
171     }
172
173     if (now <= VREO_SALE_PHASE_1_END_TIME) {
174         return tokenAmount.mul((100 + BONUS_PCT_IN_VREO_SALE_PHASE_1) / 100);

```

```
174     }
175
176     if (now <= VREO_SALE_PHASE_2_END_TIME) {
177         return tokenAmount.mul((100 + BONUS_PCT_IN_VREO_SALE_PHASE_2) / 100);
178     }
179
180     return tokenAmount; // No bonus
181 }
182
183 /// @dev Deliver tokens
184 /// @param _beneficiary An Ethereum address
185 /// @param _tokenAmount A positive number
186 function _deliverTokens(address _beneficiary, uint _tokenAmount) internal {
187     remainingTokensForSale = remainingTokensForSale.sub(_tokenAmount);
188
189     super._deliverTokens(_beneficiary, _tokenAmount);
190 }
191
192 /// @dev Finalization
193 function finalization() internal {
194     require(now >= KYC_VERIFICATION_END_TIME);
195
196     VreoToken(token).mint(teamAddress, TOKEN_SHARE_OF_TEAM);
197     VreoToken(token).mint(advisorsAddress, TOKEN_SHARE_OF_ADVISORS);
198     VreoToken(token).mint(legalsAddress, TOKEN_SHARE_OF_LEGALS);
199     VreoToken(token).mint(bountyAddress, TOKEN_SHARE_OF_BOUNTY);
200
201     VreoToken(token).finishMinting();
202     VreoToken(token).unpause();
203
204     super.finalization();
205 }
206
207 }
```

VREOTOKENBOUNTY

```
1  pragma solidity 0.4.24;
2
3  import "../node_modules/zeppelin-solidity/contracts/ownership/Ownable.sol";
4  import "../VreoToken.sol";
5
6
7  /// @title VreoTokenBounty
8  /// @author Autogenerated from a Dia UML diagram
9  contract VreoTokenBounty is Ownable {
10
11      VreoToken public token;
12
13      /// @dev Constructor
14      /// @param _token A VreoToken
15      constructor(VreoToken _token) public {
16          require(address(_token) != address(0));
17
18          token = _token;
19      }
20
21      /// @dev Distribute tokens
22      /// @param _recipients A list where each entry is an Ethereum address
23      /// @param _amounts A list where each entry is a positive number
24      function distributeTokens(address[] _recipients, uint[] _amounts) public onlyOwner {
25          require(_recipients.length == _amounts.length);
26
27          for (uint i = 0; i < _recipients.length; ++i) {
28              token.transfer(_recipients[i], _amounts[i]);
29          }
30      }
31
32  }
```