

---

# **Vreo ICO Solidity Smart Contracts**

***Release 1***

**Sicos et al.**

**Jun 06, 2018**

# CONTENTS

1	VreoToken	1
2	VreoTokenSale	2
3	VreoTokenBounty	8
4	IconiqInterface	9

## VREOTOKEN

```
1  pragma solidity 0.4.24;
2
3  import "../zeppelin/token/ERC20/CappedToken.sol";
4  import "../zeppelin/token/ERC20/PausableToken.sol";
5  import "../zeppelin/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 = "Vreo MTC";
15     string public symbol = "MTC";
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 }
```

## VREOTOKENSALE

```
1  pragma solidity 0.4.24;
2
3  import "../zeppelin/crowdsale/distribution/FinalizableCrowdsale.sol";
4  import "../zeppelin/crowdsale/emission/MintedCrowdsale.sol";
5  import "../VreoTokenBounty.sol";
6  import "../IconiqInterface.sol";
7
8
9  /// @title VreoTokenSale
10 /// @author Autogenerated from a Dia UML diagram
11 contract VreoTokenSale is FinalizableCrowdsale, MintedCrowdsale {
12
13     struct Investment {
14         bool isVerified; // whether or not the investor passed the KYC process
15         uint value;      // invested wei
16         uint amount;     // amount of token quantum the investor wants to purchase
17     }
18
19     uint public constant TOKEN_SHARE_OF_TEAM = 85000000e18; // = 85.000.000 e18
20     uint public constant TOKEN_SHARE_OF_ADVISORS = 58000000e18; // = 58.000.000 e18
21     uint public constant TOKEN_SHARE_OF_LEGALS = 57000000e18; // = 57.000.000 e18
22     uint public constant TOKEN_SHARE_OF_BOUNTY = 50000000e18; // = 50.000.000 e18
23
24     uint public constant TOTAL_TOKEN_CAP_OF_SALE = 450000000e18; // = 450.000.000 e18
25
26     // Extra token percentages
27     uint public constant EXTRA_TOKEN_PCT_IN_ICONIQ_SALE = 20;
28     uint public constant EXTRA_TOKEN_PCT_IN_VREO_PRESALE = 15;
29
30     // Minimum duration of KYC verification before finalization
31     uint public constant MINIMUM_KYC_VERIFICATION_PERIOD = 14 days;
32
33     // Amount of available tokens
34     uint public remainingTokensForSale;
35
36     // Opening and closing times of different sale periods
37     uint public openingTimeOfIconiqSale;
38     uint public closingTimeOfIconiqSale;
39     uint public openingTimeOfVreoPresale;
40     uint public closingTimeOfVreoPresale;
41     uint public openingTimeOfPublicSale;
42     uint public closingTimeOfPublicSale;
43
44     IconiqInterface public iconiq;
45     address public teamAccount;
46     address public advisorsAccount;
47     address public legalsAccount;
48     VreoTokenBounty public bounty;
49 }
```

```

50 mapping(address => Investment) public investments;
51
52 /// @dev Log entry on rate changed
53 /// @param newRate A positive number
54 event RateChanged(uint newRate);
55
56 /// @dev Log entry on investor verified
57 /// @param investor An Ethereum address
58 event InvestorVerified(address investor);
59
60 /// @dev Log entry on investor falsified
61 /// @param investor An Ethereum address
62 event InvestorFalsified(address investor);
63
64 /// @dev Log entry on token delivered
65 /// @param investor An Ethereum address
66 /// @param amount A positive number
67 event TokensDelivered(address investor, uint amount);
68
69 /// @dev Log entry on withdrawn
70 /// @param investor An Ethereum address
71 /// @param value A positive number
72 event Withdrawn(address investor, uint value);
73
74 /// @dev Constructor
75 /// @param _token A VreoToken
76 /// @param _openingTimeOfIconiqSale A positive number
77 /// @param _closingTimeOfIconiqSale A positive number
78 /// @param _openingTimeOfVreoPresale A positive number
79 /// @param _closingTimeOfVreoPresale A positive number
80 /// @param _openingTimeOfPublicSale A positive number
81 /// @param _closingTimeOfPublicSale A positive number
82 /// @param _rate A positive number
83 /// @param _iconiq An IconiqInterface
84 /// @param _teamAccount An Ethereum address
85 /// @param _advisorsAccount An Ethereum address
86 /// @param _legalsAccount An Ethereum address
87 /// @param _bounty A VreoTokenBounty
88 /// @param _wallet An Ethereum address
89 constructor(
90     VreoToken _token,
91     uint _openingTimeOfIconiqSale,
92     uint _closingTimeOfIconiqSale,
93     uint _openingTimeOfVreoPresale,
94     uint _closingTimeOfVreoPresale,
95     uint _openingTimeOfPublicSale,
96     uint _closingTimeOfPublicSale,
97     uint _rate,
98     IconiqInterface _iconiq,
99     address _teamAccount,
100     address _advisorsAccount,
101     address _legalsAccount,
102     VreoTokenBounty _bounty,
103     address _wallet
104 )
105 public
106 Crowdsale(_rate, _wallet, _token)
107 TimedCrowdsale(_openingTimeOfIconiqSale, _closingTimeOfPublicSale)
108 {
109     // Token sanity check
110     require(_token.cap() >= TOTAL_TOKEN_CAP_OF_SALE
111             + TOKEN_SHARE_OF_TEAM
112             + TOKEN_SHARE_OF_ADVISORS

```

```

113         + TOKEN_SHARE_OF_LEGALS
114         + TOKEN_SHARE_OF_BOUNTY);
115
116     // Ensure strict timing order
117     require(now < _openingTimeOfIconiqSale
118         && _openingTimeOfIconiqSale < _closingTimeOfIconiqSale
119         && _closingTimeOfIconiqSale < _openingTimeOfVreoPresale
120         && _openingTimeOfVreoPresale < _closingTimeOfVreoPresale
121         && _closingTimeOfVreoPresale < _openingTimeOfPublicSale
122         && _openingTimeOfPublicSale < _closingTimeOfPublicSale);
123
124     // Sanity check of addresses
125     require(address(_iconiq) != address(0)
126         && _teamAccount != address(0)
127         && _advisorsAccount != address(0)
128         && _legalsAccount != address(0)
129         && address(_bounty) != address(0));
130
131     remainingTokensForSale = TOTAL_TOKEN_CAP_OF_SALE;
132
133     openingTimeOfIconiqSale = _openingTimeOfIconiqSale;
134     openingTimeOfIconiqSale = _closingTimeOfIconiqSale;
135     openingTimeOfVreoPresale = _openingTimeOfVreoPresale;
136     closingTimeOfVreoPresale = _closingTimeOfVreoPresale;
137     openingTimeOfPublicSale = _openingTimeOfPublicSale;
138     closingTimeOfPublicSale = _closingTimeOfPublicSale;
139
140     iconiq = _iconiq;
141     teamAccount = _teamAccount;
142     advisorsAccount = _advisorsAccount;
143     legalsAccount = _legalsAccount;
144     bounty = _bounty;
145 }
146
147 /// @dev Destroy
148 function destroy() public onlyOwner {
149     // TBD
150 }
151
152 /// @dev Set rate
153 /// @param _newRate A positive number
154 function setRate(uint _newRate) public onlyOwner {
155     require(_newRate > 0);
156
157     rate = _newRate;
158
159     emit RateChanged(_newRate);
160 }
161
162 /// @dev Verify investors
163 /// @param _investors A list where each entry is an Ethereum address
164 function verifyInvestors(address[] _investors) public onlyOwner {
165     for (uint i = 0; i < _investors.length; ++i) {
166         address investor = _investors[i];
167         Investment storage investment = investments[investor];
168
169         if (!investment.isVerified) {
170             investment.isVerified = true;
171
172             if (investment.amount > 0) {
173                 fulfillInvestment(investor, investment);
174             }
175         }
176     }
177 }

```

```

176         emit InvestorVerified(investor);
177     }
178 }
179
180
181 /// @dev Falsify investors
182 /// @param _investors A list where each entry is an Ethereum address
183 function falsifyInvestors(address[] _investors) public onlyOwner {
184     for (uint i = 0; i < _investors.length; ++i) {
185         address investor = _investors[i];
186         Investment storage investment = investments[investor];
187
188         if (investment.isVerified) {
189             investment.isVerified = false;
190
191             emit InvestorFalsified(investor);
192         }
193     }
194 }
195
196 /// @dev Withdraw
197 function withdraw() public {
198     require(hasClosed());
199
200     Investment storage investment = investments[msg.sender];
201     investment.amount = 0;
202
203     uint value = investment.value;
204
205     if (value > 0) {
206         investment.value = 0;
207
208         msg.sender.transfer(value);
209
210         emit Withdrawn(msg.sender, value);
211     }
212 }
213
214 function fulfillInvestment(address _investor, Investment _investment) internal {
215     uint value = _investment.value;
216     uint amount = _investment.amount;
217
218     if (amount > remainingTokensForSale) {
219         value = value.mul(remainingTokensForSale).div(amount);
220         amount = remainingTokensForSale;
221     }
222
223     // Dev note: no overflow possible
224
225     _investment.value -= value;
226     _investment.amount -= amount;
227
228     remainingTokensForSale -= amount;
229
230     wallet.transfer(value);
231     _deliverTokens(_investor, amount);
232 }
233
234 /// @dev Pre validate purchase
235 /// @param _beneficiary An Ethereum address
236 /// @param _weiAmount A positive number
237 function _preValidatePurchase(address _beneficiary, uint _weiAmount) internal {
238     require(openingTimeOfIconiqSale <= now && now <= closingTimeOfIconiqSale && iconiq.
    ↪isAllowed(_beneficiary)

```

```

239         || openingTimeOfVreoPresale <= now && now <= closingTimeOfVreoPresale
240         || openingTimeOfPublicSale <= now && now <= closingTimeOfPublicSale);
241
242         super._preValidatePurchase(_beneficiary, _weiAmount);
243     }
244
245     /// @dev Post validate purchase
246     /// @param _beneficiary An Ethereum address
247     /// @param _weiAmount A positive number
248     function _postValidatePurchase(address _beneficiary, uint _weiAmount) internal {
249         // Nothing to do here...
250     }
251
252     /// @dev Deliver tokens
253     /// @param _beneficiary An Ethereum address
254     /// @param _tokenAmount A positive number
255     function _deliverTokens(address _beneficiary, uint _tokenAmount) internal {
256         super._deliverTokens(_beneficiary, _tokenAmount);
257
258         emit TokensDelivered(_beneficiary, _tokenAmount);
259     }
260
261     /// @dev Process purchase
262     /// @param _beneficiary An Ethereum address
263     /// @param _tokenAmount A positive number
264     function _processPurchase(address _beneficiary, uint _tokenAmount) internal {
265         Investment storage investment = investments[_beneficiary];
266
267         investment.value = investment.value.add(msg.value);
268         investment.amount = investment.amount.add(_tokenAmount);
269
270         if (investment.isVerified) {
271             fulfillInvestment(_beneficiary, investment);
272         }
273     }
274
275     /// @dev Update purchasing state
276     /// @param _beneficiary An Ethereum address
277     /// @param _weiAmount A positive number
278     function _updatePurchasingState(address _beneficiary, uint _weiAmount) internal {
279         // Nothing to do here...
280     }
281
282     /// @dev Get token amount
283     /// @param _weiAmount A positive number
284     /// @return A positive number
285     function _getTokenAmount(uint _weiAmount) internal view returns (uint) {
286         uint amount = super._getTokenAmount(_weiAmount);
287
288         if (now <= closingTimeOfIconiqSale) {
289             return amount.mul(100 + EXTRA_TOKEN_PCT_IN_ICONIQ_SALE).div(100);
290         }
291
292         if (now <= closingTimeOfVreoPresale) {
293             return amount.mul(100 + EXTRA_TOKEN_PCT_IN_VREO_PRESALE).div(100);
294         }
295
296         return amount;
297     }
298
299     /// @dev Forward funds
300     function _forwardFunds() internal {
301         // Postponed. Nothing to do here...

```



```
302     }
303
304     /// @dev Finalization
305     function finalization() internal {
306         require(now >= closingTimeOfPublicSale + MINIMUM_KYC_VERIFICATION_PERIOD);
307
308         MintableToken(token).mint(teamAccount, TOKEN_SHARE_OF_TEAM);
309         MintableToken(token).mint(advisorsAccount, TOKEN_SHARE_OF_ADVISORS);
310         MintableToken(token).mint(legalsAccount, TOKEN_SHARE_OF_LEGALS);
311         MintableToken(token).mint(bounty, TOKEN_SHARE_OF_BOUNTY);
312     }
313
314 }
```

## VREOTOKENBOUNTY

```
1  pragma solidity 0.4.24;
2
3  import "../zeppelin/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 Log entry on token distributed
14      /// @param recipient An Ethereum address
15      /// @param amount A positive number
16      event TokenDistributed(address recipient, uint amount);
17
18      /// @dev Constructor
19      /// @param _token A VreoToken
20      constructor(VreoToken _token) public {
21          require(address(_token) != address(0));
22
23          token = _token;
24      }
25
26      /// @dev Distribute tokens
27      /// @param _recipients A list where each entry is an Ethereum address
28      /// @param _amounts A list where each entry is a positive number
29      function distributeTokens(address[] _recipients, uint[] _amounts) public onlyOwner {
30          require(_recipients.length == _amounts.length);
31
32          for (uint i = 0; i < _recipients.length; ++i) {
33              require(_amounts[i] <= token.balanceOf(this)); // TODO: superfluous
34
35              token.transfer(_recipients[i], _amounts[i]);
36
37              emit TokenDistributed(_recipients[i], _amounts[i]);
38          }
39      }
40
41  }
```

## ICONIQINTERFACE

```
1  pragma solidity 0.4.24;
2
3
4  /// @title IconiqInterface
5  /// @author Autogenerated from a Dia UML diagram
6  interface IconiqInterface {
7
8      /// @dev Is allowed
9      /// @param _account An Ethereum address
10     /// @return True or false
11     function isAllowed(address _account) external view returns (bool);
12
13 }
```