ORS Token Sale

Release 1

SICOS

May 06, 2018

CONTENTS

CHAPTER

ONE

ORSTOKEN

```
pragma solidity 0.4.23;

import "../zeppelin-solidity/contracts/token/ERC20/CappedToken.sol";
import "../zeppelin-solidity/contracts/token/ERC20/PausableToken.sol";
import "../zeppelin-solidity/contracts/token/ERC20/StandardBurnableToken.sol";


/// @title ORSToken
/// @author Autogenerated from a Dia UML diagram
contract ORSToken is CappedToken, StandardBurnableToken, PausableToken {

    string public name = "ORS Token";
    string public symbol = "ORS";
    uint8 public decimals = 18;

    /// @dev Constructor
    /// @param _cap A positive number
    constructor(uint _cap) public CappedToken(_cap) {
        pause();
    }

}
```

CHAPTER

TWO

ORSTOKENSALE

```
pragma solidity 0.4.23;

import "./ORSToken.sol";
import "./KYCBase.sol";
import "../eidoo-icoengine/contracts/ICOEngineInterface.sol";
import "../zeppelin-solidity/contracts/math/SafeMath.sol";
import "../zeppelin-solidity/contracts/ownership/Ownable.sol";


/// @title ORSTokenSale
/// @author Autogenerated from a Dia UML diagram
contract ORSTokenSale is KYCBase, ICOEngineInterface, Ownable {

    using SafeMath for uint;

    // Maximum token amounts of each pool
    uint constant public PRESALE_CAP = 250000000e18;                 // 250,000,000 e18
    uint constant public MAINSALE_CAP = 500000000e18 - PRESALE_CAP;  // 250,000,000 e18
    uint constant public BONUS_CAP = 64460000e18;                    //  64,460,000 e18

    // Granted token shares that will be minted upon finalization
    uint constant public TEAM_SHARE = 83333333e18;                   //  83,333,333 e18
    uint constant public ADVISORS_SHARE = 58333333e18;               //  58,333,333 e18
    uint constant public COMPANY_SHARE = 127206667e18;               // 127,206,667 e18

    // Remaining token amounts of each pool
    uint public presaleRemaining = PRESALE_CAP;
    uint public mainsaleRemaining = MAINSALE_CAP;
    uint public bonusRemaining = BONUS_CAP;

    // Beneficiaries of granted token shares
    address public teamWallet;
    address public advisorsWallet;
    address public companyWallet;

    ORSToken public token;

    // Integral token units (10^-18 tokens) per wei
    uint public rate;

    // Mainsale period
    uint public openingTime;
    uint public closingTime;

    // Ethereum address where invested funds will be transferred to
    address public wallet;

    // Purchases signed via Eidoo's platform will receive bonus tokens
    address public eidooSigner;
```

```
50
51      bool public isFinalized = false;
52
53      /// @dev Log entry on rate changed
54      /// @param newRate A positive number
55      event RateChanged(uint newRate);
56
57      /// @dev Log entry on token purchased
58      /// @param buyer An Ethereum address
59      /// @param value A positive number
60      /// @param tokens A positive number
61      event TokenPurchased(address indexed buyer, uint value, uint tokens);
62
63      /// @dev Log entry on buyer refunded
64      /// @param buyer An Ethereum address
65      /// @param value A positive number
66      event BuyerRefunded(address indexed buyer, uint value);
67
68      /// @dev Log entry on finalized
69      event Finalized();
70
71      /// @dev Constructor
72      /// @param _token An ORSToken
73      /// @param _rate A positive number
74      /// @param _openingTime A positive number
75      /// @param _closingTime A positive number
76      /// @param _wallet An Ethereum address
77      /// @param _teamWallet An Ethereum address
78      /// @param _companyWallet An Ethereum address
79      /// @param _advisorsWallet An Ethereum address
80      /// @param _kycSigners A list where each entry is an Ethereum address
81      constructor(
82          ORSToken _token,
83          uint _rate,
84          uint _openingTime,
85          uint _closingTime,
86          address _wallet,
87          address _teamWallet,
88          address _advisorsWallet,
89          address _companyWallet,
90          address[] _kycSigners
91      )
92          public
93          KYCBase(_kycSigners)
94      {
95          require(_token != address(0x0));
96          require(_token.cap() == PRESALE_CAP + MAINSALE_CAP + BONUS_CAP + TEAM_SHARE + ADVISORS_
    ↪SHARE + COMPANY_SHARE);
97          require(_rate > 0);
98          require(_openingTime > now && _closingTime > _openingTime);
99          require(_wallet != address(0x0));
100         require(_teamWallet != address(0x0) && _companyWallet != address(0x0) && _advisorsWallet !=␣
    ↪address(0x0));
101         require(_kycSigners.length >= 2);
102
103         token = _token;
104         rate = _rate;
105         openingTime = _openingTime;
106         closingTime = _closingTime;
107         wallet = _wallet;
108         teamWallet = _teamWallet;
109         advisorsWallet = _advisorsWallet;
110         companyWallet = _companyWallet;
```

```
111
112         eidooSigner = _kycSigners[0];
113     }
114
115     /// @dev Set rate
116     /// @param newRate A positive number
117     function setRate(uint newRate) public onlyOwner {
118         require(newRate > 0);
119
120         if (newRate != rate) {
121             rate = newRate;
122
123             emit RateChanged(newRate);
124         }
125     }
126
127     /// @dev Distribute presale
128     /// @param investors A list where each entry is an Ethereum address
129     /// @param tokens A list where each entry is a positive number
130     /// @param bonuses A list where each entry is a positive number
131     function distributePresale(address[] investors, uint[] tokens, uint[] bonuses) public onlyOwner
↪{
132         require(ended() && !isFinalized);
133         require(tokens.length == investors.length && bonuses.length == investors.length);
134
135         for (uint i = 0; i < investors.length; ++i) {
136             presaleRemaining = presaleRemaining.sub(tokens[i]);
137             bonusRemaining = bonusRemaining.sub(bonuses[i]);
138
139             token.mint(investors[i], tokens[i].add(bonuses[i]));
140         }
141     }
142
143     /// @dev Finalize
144     function finalize() public onlyOwner {
145         require(ended() && !isFinalized);
146         require(presaleRemaining == 0);
147
148         // Distribute granted token shares
149         token.mint(teamWallet, TEAM_SHARE);
150         token.mint(advisorsWallet, ADVISORS_SHARE);
151         token.mint(companyWallet, COMPANY_SHARE);
152
153         // There shouldn't be any remaining presale tokens
154         // Remaining mainsale tokens will be lost (i.e. not minted)
155         // Remaining bonus tokens will be minted for the benefit of company
156         if (bonusRemaining > 0) {
157             token.mint(companyWallet, bonusRemaining);
158             bonusRemaining = 0;
159         }
160
161         // Enable token trade
162         token.finishMinting();
163         token.unpause();
164
165         isFinalized = true;
166
167         emit Finalized();
168     }
169
170     // false if the ico is not started, true if the ico is started and running, true if the ico is␣
↪completed
171     /// @dev Started
```

```
172     /// @return True or false
173     function started() public view returns (bool) {
174         return now > openingTime;
175     }
176 
177     // false if the ico is not started, false if the ico is started and running, true if the ico is␣
    ↪completed
178     /// @dev Ended
179     /// @return True or false
180     function ended() public view returns (bool) {
181         return now > closingTime || mainsaleRemaining < rate;
182     }
183 
184     // time stamp of the starting time of the ico, must return 0 if it depends on the block number
185     /// @dev Start time
186     /// @return A positive number
187     function startTime() public view returns (uint) {
188         return openingTime;
189     }
190 
191     // time stamp of the ending time of the ico, must retrun 0 if it depends on the block number
192     /// @dev End time
193     /// @return A positive number
194     function endTime() public view returns (uint) {
195         return closingTime;
196     }
197 
198     // returns the total number of the tokens available for the sale, must not change when the ico␣
    ↪is started
199     /// @dev Total tokens
200     /// @return A positive number
201     function totalTokens() public view returns (uint) {
202         return MAINSALE_CAP;
203     }
204 
205     // returns the number of the tokens available for the ico. At the moment that the ico starts it␣
    ↪must be
206     // equal to totalTokens(), then it will decrease. It is used to calculate the percentage of␣
    ↪sold tokens as
207     // remainingTokens() / totalTokens()
208     /// @dev Remaining tokens
209     /// @return A positive number
210     function remainingTokens() public view returns (uint) {
211         return mainsaleRemaining;
212     }
213 
214     // return the price as number of tokens released for each ether
215     /// @dev Price
216     /// @return A positive number
217     function price() public view returns (uint) {
218         return rate;
219     }
220 
221     /// @dev Release tokens to
222     /// @param buyer An Ethereum address
223     /// @param signer An Ethereum address
224     /// @return True or false
225     function releaseTokensTo(address buyer, address signer) internal returns (bool) {
226         require(started() && !ended());
227 
228         uint value = msg.value;
229         uint refund = 0;
230 
```

```
231         uint tokens = value.mul(rate);
232         uint bonus = 0;
233 
234         // (Last) buyer whose purchase would exceed available mainsale tokens will be partially␣
↪refunded
235         if (tokens > mainsaleRemaining) {
236             uint valueOfRemaining = mainsaleRemaining.div(rate);
237 
238             refund = value.sub(valueOfRemaining);
239             value = valueOfRemaining;
240             tokens = mainsaleRemaining;
241             // Note:
242             // To be 100% accurate the buyer should receive only a token amount that corresponds to␣
↪valueOfRemaining,
243             // i.e. tokens = valueOfRemaining.mul(rate), because of mainsaleRemaining may not be a␣
↪multiple of rate
244             // (due to regular adaption to the ether/fiat exchange rate).
245             // Nevertheless, we deliver all mainsaleRemaining tokens as the worth of these␣
↪additional n tokens at time
246             // of purchase is less than a wei and the gas costs of a correct solution, i.e.␣
↪calculate value * rate
247             // again, would exceed this by several orders of magnitude.
248         }
249 
250         // Purchases via Eidoo's platform will receive additional 20% bonus tokens
251         if (signer == eidooSigner) {
252             bonus = tokens.div(20);
253         }
254 
255         mainsaleRemaining = mainsaleRemaining.sub(tokens);
256         bonusRemaining = bonusRemaining.sub(bonus);
257 
258         token.mint(buyer, tokens.add(bonus));
259         wallet.transfer(value);
260         if (refund > 0) {
261             buyer.transfer(refund);
262 
263             emit BuyerRefunded(buyer, refund);
264         }
265 
266         emit TokenPurchased(buyer, value, tokens.add(bonus));
267 
268         return true;
269     }
270 
271 }
```