ORS TokenSale Solidity Smart Contracts

Release 2

Sicos et al.

May 10, 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 Sicos et al.
contract ORSToken is CappedToken, StandardBurnableToken, PausableToken {

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

    /// @dev Constructor
    /// @param _cap Maximum number of integral token units; total supply must never exceed this limit
    constructor(uint _cap) public CappedToken(_cap) {
        pause();  // Disable token trade
    }

}
```

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 Sicos et al.
contract ORSTokenSale is KYCBase, ICOEngineInterface, Ownable {

    using SafeMath for uint;

    // Maximum token amounts of each pool
    // Note: BONUS_CAP should be at least 5% of MAINSALE_CAP
    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
```

```
50      address public eidooSigner;
51
52      bool public isFinalized = false;
53
54      /// @dev Log entry on rate changed
55      /// @param newRate New rate in integral token units per wei
56      event RateChanged(uint newRate);
57
58      /// @dev Log entry on token purchased
59      /// @param buyer Ethereum address of token purchaser
60      /// @param value Worth in wei of purchased token amount
61      /// @param tokens Number of integral token units
62      event TokenPurchased(address indexed buyer, uint value, uint tokens);
63
64      /// @dev Log entry on buyer refunded upon token purchase
65      /// @param buyer Ethereum address of token purchaser
66      /// @param value Worth of refund of wei
67      event BuyerRefunded(address indexed buyer, uint value);
68
69      /// @dev Log entry on finalized
70      event Finalized();
71
72      /// @dev Constructor
73      /// @param _token An ORSToken
74      /// @param _rate Rate in integral token units per wei
75      /// @param _openingTime Block (Unix) timestamp of mainsale start time
76      /// @param _closingTime Block (Unix) timestamp of mainsale latest end time
77      /// @param _wallet Ethereum account who will receive sent ether upon token purchase during␣
    ↪mainsale
78      /// @param _teamWallet Ethereum account of team who will receive team share upon finalization
79      /// @param _advisorsWallet Ethereum account of advisors who will receive advisors share upon␣
    ↪finalization
80      /// @param _companyWallet Ethereum account of company who will receive company share upon␣
    ↪finalization
81      /// @param _kycSigners List of KYC signers' Ethereum addresses
82      constructor(
83          ORSToken _token,
84          uint _rate,
85          uint _openingTime,
86          uint _closingTime,
87          address _wallet,
88          address _teamWallet,
89          address _advisorsWallet,
90          address _companyWallet,
91          address[] _kycSigners
92      )
93          public
94          KYCBase(_kycSigners)
95      {
96          require(_token != address(0x0));
97          require(_token.cap() == PRESALE_CAP + MAINSALE_CAP + BONUS_CAP + TEAM_SHARE + ADVISORS_
    ↪SHARE + COMPANY_SHARE);
98          require(_rate > 0);
99          require(_openingTime > now && _closingTime > _openingTime);
100         require(_wallet != address(0x0));
101         require(_teamWallet != address(0x0) && _companyWallet != address(0x0) && _advisorsWallet !=␣
    ↪address(0x0));
102         require(_kycSigners.length >= 2);
103
104         token = _token;
105         rate = _rate;
106         openingTime = _openingTime;
107         closingTime = _closingTime;
```

```
108         wallet = _wallet;
109         teamWallet = _teamWallet;
110         advisorsWallet = _advisorsWallet;
111         companyWallet = _companyWallet;
112 
113         eidooSigner = _kycSigners[0];
114     }
115 
116     /// @dev Set rate, i.e. adjust to changes of fiat/ether exchange rates
117     /// @param newRate Rate in integral token units per wei
118     function setRate(uint newRate) public onlyOwner {
119         require(newRate > 0);
120 
121         if (newRate != rate) {
122             rate = newRate;
123 
124             emit RateChanged(newRate);
125         }
126     }
127 
128     /// @dev Distribute presold tokens and bonus tokens to investors
129     /// @param investors List of investors' Ethereum addresses
130     /// @param tokens List of integral token amounts each investors will receive
131     function distributePresale(address[] investors, uint[] tokens) public onlyOwner {
132         require(!isFinalized);
133         require(tokens.length == investors.length);
134 
135         for (uint i = 0; i < investors.length; ++i) {
136             presaleRemaining = presaleRemaining.sub(tokens[i]);
137 
138             token.mint(investors[i], tokens[i]);
139         }
140     }
141 
142     /// @dev Finalize, i.e. end token minting phase and enable token trading
143     function finalize() public onlyOwner {
144         require(ended() && !isFinalized);
145         require(presaleRemaining == 0);
146 
147         // Distribute granted token shares
148         token.mint(teamWallet, TEAM_SHARE);
149         token.mint(advisorsWallet, ADVISORS_SHARE);
150         token.mint(companyWallet, COMPANY_SHARE);
151 
152         // There shouldn't be any remaining presale tokens
153         // Remaining mainsale tokens will be lost (i.e. not minted)
154         // Remaining bonus tokens will be minted for the benefit of company
155         if (bonusRemaining > 0) {
156             token.mint(companyWallet, bonusRemaining);
157             bonusRemaining = 0;
158         }
159 
160         // Enable token trade
161         token.finishMinting();
162         token.unpause();
163 
164         isFinalized = true;
165 
166         emit Finalized();
167     }
168 
169     // false if the ico is not started, true if the ico is started and running, true if the ico is␣
↪completed
```

```
170     /// @dev Started (as required by Eidoo's ICOEngineInterface)
171     /// @return True iff mainsale start has passed
172     function started() public view returns (bool) {
173         return now >= openingTime;
174     }
175 
176     // false if the ico is not started, false if the ico is started and running, true if the ico is␣
↪completed
177     /// @dev Ended (as required by Eidoo's ICOEngineInterface)
178     /// @return True iff mainsale is finished
179     function ended() public view returns (bool) {
180         // Note: Even though we allow token holders to burn their tokens immediately after purchase,
↪ this won't
181         //       affect the early end via "sold out" as mainsaleRemaining is independent of token.
↪totalSupply.
182         return now > closingTime || mainsaleRemaining == 0;
183     }
184 
185     // time stamp of the starting time of the ico, must return 0 if it depends on the block number
186     /// @dev Start time (as required by Eidoo's ICOEngineInterface)
187     /// @return Block (Unix) timestamp of mainsale start time
188     function startTime() public view returns (uint) {
189         return openingTime;
190     }
191 
192     // time stamp of the ending time of the ico, must retrun 0 if it depends on the block number
193     /// @dev End time (as required by Eidoo's ICOEngineInterface)
194     /// @return Block (Unix) timestamp of mainsale latest end time
195     function endTime() public view returns (uint) {
196         return closingTime;
197     }
198 
199     // returns the total number of the tokens available for the sale, must not change when the ico␣
↪is started
200     /// @dev Total amount of tokens initially available for purchase during mainsale (excluding␣
↪bonus tokens)
201     /// @return Integral token units
202     function totalTokens() public view returns (uint) {
203         return MAINSALE_CAP;
204     }
205 
206     // returns the number of the tokens available for the ico. At the moment that the ico starts it␣
↪must be
207     // equal to totalTokens(), then it will decrease. It is used to calculate the percentage of␣
↪sold tokens as
208     // remainingTokens() / totalTokens()
209     /// @dev Remaining amount of tokens available for purchase during mainsale (excluding bonus␣
↪tokens)
210     /// @return Integral token units
211     function remainingTokens() public view returns (uint) {
212         return mainsaleRemaining;
213     }
214 
215     // return the price as number of tokens released for each ether
216     /// @dev Price (as required by Eidoo's ICOEngineInterface); actually the inverse of a "price"
217     /// @return Rate in integral token units per wei
218     function price() public view returns (uint) {
219         return rate;
220     }
221 
222     /// @dev Release purchased tokens to buyers during mainsale (as required by Eidoo's␣
↪ICOEngineInterface)
223     /// @param buyer Ethereum address of purchaser
```

```
224     /// @param signer Ethereum address of signer
225     /// @return Always true, failures will be indicated by transaction reversal
226     function releaseTokensTo(address buyer, address signer) internal returns (bool) {
227         require(started() && !ended());
228 
229         uint value = msg.value;
230         uint refund = 0;
231 
232         uint tokens = value.mul(rate);
233         uint bonus = 0;
234 
235         // (Last) buyer whose purchase would exceed available mainsale tokens will be partially␣
↪refunded
236         if (tokens > mainsaleRemaining) {
237             uint valueOfRemaining = mainsaleRemaining.div(rate);
238 
239             refund = value.sub(valueOfRemaining);
240             value = valueOfRemaining;
241             tokens = mainsaleRemaining;
242             // Note:
243             // To be 100% accurate the buyer should receive only a token amount that corresponds to␣
↪valueOfRemaining,
244             // i.e. tokens = valueOfRemaining.mul(rate), because of mainsaleRemaining may not be a␣
↪multiple of rate
245             // (due to regular adaption to the ether/fiat exchange rate).
246             // Nevertheless, we deliver all mainsaleRemaining tokens as the worth of these␣
↪additional tokens at time
247             // of purchase is less than a wei and the gas costs of a correct solution, i.e.␣
↪calculate value * rate
248             // again, would exceed this by several orders of magnitude.
249         }
250 
251         // Purchases signed via Eidoo's platform will receive additional 5% bonus tokens
252         if (signer == eidooSigner) {
253             bonus = tokens.div(20);
254         }
255 
256         mainsaleRemaining = mainsaleRemaining.sub(tokens);
257         bonusRemaining = bonusRemaining.sub(bonus);
258 
259         token.mint(buyer, tokens.add(bonus));
260         wallet.transfer(value);
261         if (refund > 0) {
262             buyer.transfer(refund);
263 
264             emit BuyerRefunded(buyer, refund);
265         }
266 
267         emit TokenPurchased(buyer, value, tokens.add(bonus));
268 
269         return true;
270     }
271 
272 }
```