# ORS TokenSale Solidity Smart Contracts

***Release 2***

**Sicos et al.**

**May 07, 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
    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 New rate in integral token units per wei
55      event RateChanged(uint newRate);
56
57      /// @dev Log entry on token purchased
58      /// @param buyer Ethereum address of token purchaser
59      /// @param value Worth in wei of purchased token amount
60      /// @param tokens Number of integral token units
61      event TokenPurchased(address indexed buyer, uint value, uint tokens);
62
63      /// @dev Log entry on buyer refunded upon token purchase
64      /// @param buyer Ethereum address of token purchaser
65      /// @param value Worth of refund of wei
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 Rate in integral token units per wei
74      /// @param _openingTime Block (Unix) timestamp of mainsale start time
75      /// @param _closingTime Block (Unix) timestamp of mainsale latest end time
76      /// @param _wallet Ethereum account who will receive sent ether upon token purchase during␣
    ↪mainsale
77      /// @param _teamWallet Ethereum account of team who will receive team share upon finalization
78      /// @param _advisorsWallet Ethereum account of advisors who will receive advisors share upon␣
    ↪finalization
79      /// @param _companyWallet Ethereum account of company who will receive company share upon␣
    ↪finalization
80      /// @param _kycSigners List of KYC signers' Ethereum addresses
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, i.e. adjust to changes of fiat/ether exchange rates
116     /// @param newRate Rate in integral token units per wei
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 presold tokens and bonus tokens to investors
128     /// @param investors List of investors' Ethereum addresses
129     /// @param tokens List of integral token amounts each investors will receive
130     /// @param bonuses List of integral bonus token amounts each investor will receive
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, i.e. end token minting phase and enable token trading
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 (as required by Eidoo's ICOEngineInterface)
172     /// @return True iff mainsale start has passed
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 (as required by Eidoo's ICOEngineInterface)
179     /// @return True iff mainsale is finished
180     function ended() public view returns (bool) {
181         // Note: Even though we allow token holders to burn their tokens immediately after purchase,
↪ this won't
182         //       affect the early end via "sold out" as mainsaleRemaining is independent of token.
↪totalSupply.
183         return now > closingTime || mainsaleRemaining == 0;
184     }
185 
186     // time stamp of the starting time of the ico, must return 0 if it depends on the block number
187     /// @dev Start time (as required by Eidoo's ICOEngineInterface)
188     /// @return Block (Unix) timestamp of mainsale start time
189     function startTime() public view returns (uint) {
190         return openingTime;
191     }
192 
193     // time stamp of the ending time of the ico, must retrun 0 if it depends on the block number
194     /// @dev End time (as required by Eidoo's ICOEngineInterface)
195     /// @return Block (Unix) timestamp of mainsale latest end time
196     function endTime() public view returns (uint) {
197         return closingTime;
198     }
199 
200     // returns the total number of the tokens available for the sale, must not change when the ico␣
↪is started
201     /// @dev Total amount of tokens initially available for purchase during mainsale (excluding␣
↪bonus tokens)
202     /// @return Integral token units
203     function totalTokens() public view returns (uint) {
204         return MAINSALE_CAP;
205     }
206 
207     // returns the number of the tokens available for the ico. At the moment that the ico starts it␣
↪must be
208     // equal to totalTokens(), then it will decrease. It is used to calculate the percentage of␣
↪sold tokens as
209     // remainingTokens() / totalTokens()
210     /// @dev Remaining amount of tokens available for purchase during mainsale (excluding bonus␣
↪tokens)
211     /// @return Integral token units
212     function remainingTokens() public view returns (uint) {
213         return mainsaleRemaining;
214     }
215 
216     // return the price as number of tokens released for each ether
217     /// @dev Price (as required by Eidoo's ICOEngineInterface); actually the inverse of a "price"
218     /// @return Rate in integral token units per wei
219     function price() public view returns (uint) {
220         return rate;
221     }
222 
223     /// @dev Release purchased tokens to buyers during mainsale (as required by Eidoo's␣
↪ICOEngineInterface)
```

```
224     /// @param buyer Ethereum address of purchaser
225     /// @param signer Ethereum address of signer
226     /// @return Always true, failures will be indicated by transaction reversal
227     function releaseTokensTo(address buyer, address signer) internal returns (bool) {
228         require(started() && !ended());
229 
230         uint value = msg.value;
231         uint refund = 0;
232 
233         uint tokens = value.mul(rate);
234         uint bonus = 0;
235 
236         // (Last) buyer whose purchase would exceed available mainsale tokens will be partially␣
↪refunded
237         if (tokens > mainsaleRemaining) {
238             uint valueOfRemaining = mainsaleRemaining.div(rate);
239 
240             refund = value.sub(valueOfRemaining);
241             value = valueOfRemaining;
242             tokens = mainsaleRemaining;
243             // Note:
244             // To be 100% accurate the buyer should receive only a token amount that corresponds to␣
↪valueOfRemaining,
245             // i.e. tokens = valueOfRemaining.mul(rate), because of mainsaleRemaining may not be a␣
↪multiple of rate
246             // (due to regular adaption to the ether/fiat exchange rate).
247             // Nevertheless, we deliver all mainsaleRemaining tokens as the worth of these␣
↪additional tokens at time
248             // of purchase is less than a wei and the gas costs of a correct solution, i.e.␣
↪calculate value * rate
249             // again, would exceed this by several orders of magnitude.
250         }
251 
252         // Purchases signed via Eidoo's platform will receive additional 5% bonus tokens
253         if (signer == eidooSigner) {
254             bonus = tokens.div(20);
255         }
256 
257         mainsaleRemaining = mainsaleRemaining.sub(tokens);
258         bonusRemaining = bonusRemaining.sub(bonus);
259 
260         token.mint(buyer, tokens.add(bonus));
261         wallet.transfer(value);
262         if (refund > 0) {
263             buyer.transfer(refund);
264 
265             emit BuyerRefunded(buyer, refund);
266         }
267 
268         emit TokenPurchased(buyer, value, tokens.add(bonus));
269 
270         return true;
271     }
272 
273 }
```