Vreo ICO Solidity Smart Contracts

Release 1

Sicos et al.

Jun 14, 2018

CONTENTS

CHAPTER

ONE

VREOTOKEN

```
pragma solidity 0.4.24;

import "../zeppelin/token/ERC20/CappedToken.sol";
import "../zeppelin/token/ERC20/PausableToken.sol";
import "../zeppelin/token/ERC20/BurnableToken.sol";


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

    uint public constant TOTAL_TOKEN_CAP = 700000000e18;  // = 700.000.000 e18

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

    /// @dev Constructor
    constructor() public CappedToken(TOTAL_TOKEN_CAP) {
    }

    /// @dev Burn
    /// @param _value A positive number
    function burn(uint _value) public whenNotPaused {
        super.burn(_value);
    }

}
```

CHAPTER

TWO

POSTKYCCROWDSALE

```
pragma solidity 0.4.24;

import "../zeppelin/ownership/Ownable.sol";
import "../zeppelin/crowdsale/Crowdsale.sol";


/// @title PostKYCCrowdsale
/// @author Autogenerated from a Dia UML diagram
contract PostKYCCrowdsale is Crowdsale, Ownable {

    struct Investment {
        bool isVerified;         // wether or not the investor passed the KYC process
        uint totalWeiInvested;   // invested wei
        uint pendingTokenAmount; // amount of token quantums the investor wants to purchase
    }

    mapping(address => Investment) public investments;

    /// @dev Log entry on investor verified
    /// @param investor An Ethereum address
    event InvestorVerified(address investor);

    /// @dev Log entry on tokens delivered
    /// @param investor An Ethereum address
    /// @param amount A positive number
    event TokensDelivered(address investor, uint amount);

    /// @dev Log entry on investment withdrawn
    /// @param investor An Ethereum address
    /// @param value A positive number
    event InvestmentWithdrawn(address investor, uint value);

    /// @dev Verify investors
    /// @param _investors An Ethereum address
    function verifyInvestors(address[] _investors) public onlyOwner {
        for (uint i = 0; i < _investors.length; ++i) {
            address investor = _investors[i];
            Investment storage investment = investments[investor];

            if (!investment.isVerified) {
                investment.isVerified = true;

                emit InvestorVerified(investor);

                uint pendingTokenAmount = investment.pendingTokenAmount;

                if (pendingTokenAmount > 0) {
                    investment.pendingTokenAmount = 0;

```

```
                    _forwardFunds(investment.totalWeiInvested);
                    _deliverTokens(investor, pendingTokenAmount);

                    emit TokensDelivered(investor, pendingTokenAmount);
                }
            }
        }
    }

    /// @dev Withdraw investment
    function withdrawInvestment() public {
        Investment storage investment = investments[msg.sender];

        require(!investment.isVerified);

        uint totalWeiInvested = investment.totalWeiInvested;

        require(totalWeiInvested > 0);

        investment.totalWeiInvested = 0;
        investment.pendingTokenAmount = 0;

        msg.sender.transfer(totalWeiInvested);

        emit InvestmentWithdrawn(msg.sender, totalWeiInvested);
    }

    /// @dev Pre validate purchase
    /// @param _beneficiary An Ethereum address
    /// @param _weiAmount A positive number
    function _preValidatePurchase(address _beneficiary, uint _weiAmount) internal {
        require(_beneficiary == msg.sender);

        super._preValidatePurchase(_beneficiary, _weiAmount);
    }

    /// @dev Process purchase
    /// @param _beneficiary An Ethereum address
    /// @param _tokenAmount A positive number
    function _processPurchase(address _beneficiary, uint _tokenAmount) internal {
        Investment storage investment = investments[msg.sender];

        if (investment.isVerified) {
            _deliverTokens(_beneficiary, _tokenAmount);

            emit TokensDelivered(_beneficiary, _tokenAmount);
        } else {
            investment.totalWeiInvested = investment.totalWeiInvested.add(msg.value);
            investment.pendingTokenAmount = investment.pendingTokenAmount.add(_tokenAmount);
        }
    }

    /// @dev Forward funds
    function _forwardFunds() internal {
        // Ensure the investor was verified, i.e. his purchased tokens were delivered,
        // before forwarding funds.
        if (investments[msg.sender].isVerified) {
            super._forwardFunds();
        }
    }

    /// @dev Forward funds
    /// @param _weiAmount A positive number
```

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

CHAPTER

THREE

VREOTOKENSALE

```
pragma solidity 0.4.24;

import "../zeppelin/token/ERC20/ERC20Basic.sol";
import "../zeppelin/crowdsale/distribution/FinalizableCrowdsale.sol";
import "../zeppelin/crowdsale/emission/MintedCrowdsale.sol";
import "./PostKYCCrowdsale.sol";
import "./VreoToken.sol";


/// @title VreoTokenSale
/// @author Autogenerated from a Dia UML diagram
contract VreoTokenSale is PostKYCCrowdsale, FinalizableCrowdsale, MintedCrowdsale {

    // Maxmimum number of tokens sold in Presale+Iconiq+Vreo sales
    uint public constant TOTAL_TOKEN_CAP_OF_SALE = 450000000e18;  // = 450.000.000 e18

    // Extra tokens minted upon finalization
    uint public constant TOKEN_SHARE_OF_TEAM     =  85000000e18;  // =  85.000.000 e18
    uint public constant TOKEN_SHARE_OF_ADVISORS =  58000000e18;  // =  58.000.000 e18
    uint public constant TOKEN_SHARE_OF_LEGALS   =  57000000e18;  // =  57.000.000 e18
    uint public constant TOKEN_SHARE_OF_BOUNTY   =  50000000e18;  // =  50.000.000 e18

    // Extra token percentages
    uint public constant BONUS_PCT_IN_ICONIQ_SALE       = 30;  // TBD
    uint public constant BONUS_PCT_IN_VREO_SALE_PHASE_1 = 20;
    uint public constant BONUS_PCT_IN_VREO_SALE_PHASE_2 = 10;

    // Date/time constants
    uint public constant ICONIQ_SALE_OPENING_TIME   = 1530432000;  // 2018-07-01 10:00:00 CEST
    uint public constant ICONIQ_SALE_CLOSING_TIME   = 1531598400;  // 2018-07-14 22:00:00 CEST
    uint public constant VREO_SALE_OPENING_TIME     = 1532160000;  // 2018-07-21 10:00:00 CEST
    uint public constant VREO_SALE_PHASE_1_END_TIME = 1532462400;  // 2018-07-24 22:00:00 CEST
    uint public constant VREO_SALE_PHASE_2_END_TIME = 1533153600;  // 2018-08-01 22:00:00 CEST
    uint public constant VREO_SALE_CLOSING_TIME     = 1534622400;  // 2018-08-18 22:00:00 CEST
    uint public constant KYC_VERIFICATION_END_TIME  = 1535832000;  // 2018-09-01 22:00:00 CEST

    // Max amount of wei ICONIQ investors can buy per ICONIQ TOKEN_SHARE_OF_TEAM
    uint public constant WEI_INVESTABLE_PER_ICONIQTOKEN = 1000;


    ERC20Basic public iconiqToken;
    address public teamAddress;
    address public advisorsAddress;
    address public legalsAddress;
    address public bountyAddress;

    uint public remainingTokensForSale;

    /// @dev Log entry on rate changed
```

```
    /// @param newRate A positive number
    event RateChanged(uint newRate);

    /// @dev Constructor
    /// @param _token A VreoToken
    /// @param _rate A positive number
    /// @param _iconiqToken An IconiqInterface
    /// @param _teamAddress An Ethereum address
    /// @param _advisorsAddress An Ethereum address
    /// @param _legalsAddress An Ethereum address
    /// @param _bountyAddress A VreoTokenBounty
    /// @param _wallet An Ethereum address
    constructor(
        VreoToken _token,
        uint _rate,
        ERC20Basic _iconiqToken,
        address _teamAddress,
        address _advisorsAddress,
        address _legalsAddress,
        address _bountyAddress,
        address _wallet
    )
        public
        Crowdsale(_rate, _wallet, _token)
        TimedCrowdsale(ICONIQ_SALE_OPENING_TIME, VREO_SALE_CLOSING_TIME)
    {
        // Token sanity check
        require(_token.cap() >= TOTAL_TOKEN_CAP_OF_SALE
                                + TOKEN_SHARE_OF_TEAM
                                + TOKEN_SHARE_OF_ADVISORS
                                + TOKEN_SHARE_OF_LEGALS
                                + TOKEN_SHARE_OF_BOUNTY);

        // Sanity check of addresses
        require(address(_iconiqToken) != address(0)
                && _teamAddress != address(0)
                && _advisorsAddress != address(0)
                && _legalsAddress != address(0)
                && _bountyAddress != address(0));

        iconiqToken = _iconiqToken;
        teamAddress = _teamAddress;
        advisorsAddress = _advisorsAddress;
        legalsAddress = _legalsAddress;
        bountyAddress = _bountyAddress;

        remainingTokensForSale = TOTAL_TOKEN_CAP_OF_SALE;
    }

    /// @dev Distribute presale
    /// @param _investors A list where each entry is an Ethereum address
    /// @param _amounts A list where each entry is a positive number
    function distributePresale(address[] _investors, uint[] _amounts) public onlyOwner {
        require(_investors.length == _amounts.length);

        uint totalAmount = 0;

        for (uint i = 0; i < _investors.length; ++i) {
            VreoToken(token).mint(_investors[i], _amounts[i]);
            totalAmount = totalAmount.add(_amounts[i]);
        }

        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.mul(WEI_INVESTABLE_PER_ICONIQTOKEN);
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 ||␣
↪vreoSaleOngoing());
160     }
161
162     /// @dev Get token amount
163     /// @param _weiAmount A positive number
164     /// @return A positive number
165     function _getTokenAmount(uint _weiAmount) internal view returns (uint) {
166         uint tokenAmount = super._getTokenAmount(_weiAmount);
167
168         if (now <= ICONIQ_SALE_CLOSING_TIME) {
169             return tokenAmount.mul((100 + BONUS_PCT_IN_ICONIQ_SALE) / 100);
170         }
171
172         if (now <= VREO_SALE_PHASE_1_END_TIME) {
173             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 }
```

CHAPTER

FOUR

VREOTOKENBOUNTY

```
pragma solidity 0.4.24;

import "../zeppelin/ownership/Ownable.sol";
import "./VreoToken.sol";


/// @title VreoTokenBounty
/// @author Autogenerated from a Dia UML diagram
contract VreoTokenBounty is Ownable {

    VreoToken public token;

    /// @dev Constructor
    /// @param _token A VreoToken
    constructor(VreoToken _token) public {
        require(address(_token) != address(0));

        token = _token;
    }

    /// @dev Distribute tokens
    /// @param _recipients A list where each entry is an Ethereum address
    /// @param _amounts A list where each entry is a positive number
    function distributeTokens(address[] _recipients, uint[] _amounts) public onlyOwner {
        require(_recipients.length == _amounts.length);

        for (uint i = 0; i < _recipients.length; ++i) {
            token.transfer(_recipients[i], _amounts[i]);
        }
    }

}
```