Vreo ICO Solidity Smart Contracts

Release 1

Sicos et al.

Jun 06, 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 = "Vreo MTC";
    string public symbol = "MTC";
    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

VREOTOKENSALE

```
pragma solidity 0.4.24;

import "../zeppelin/crowdsale/distribution/FinalizableCrowdsale.sol";
import "../zeppelin/crowdsale/emission/MintedCrowdsale.sol";
import "./VreoTokenBounty.sol";
import "./IconiqInterface.sol";


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

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

    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

    uint public constant TOTAL_TOKEN_CAP_OF_SALE = 450000000e18;  // = 450.000.000 e18

    // Extra token percentages
    uint public constant EXTRA_TOKEN_PCT_IN_ICONIQ_SALE = 20;
    uint public constant EXTRA_TOKEN_PCT_IN_VREO_PRESALE = 15;

    // Minimum duration of KYC verification before finalization
    uint public constant MINIMUM_KYC_VERIFICATION_PERIOD = 14 days;

    // Amount of available tokens
    uint public remainingTokensForSale;

    // Opening and closing times of different sale periods
    uint public openingTimeOfIconiqSale;
    uint public closingTimeOfIconiqSale;
    uint public openingTimeOfVreoPresale;
    uint public closingTimeOfVreoPresale;
    uint public openingTimeOfPublicSale;
    uint public closingTimeOfPublicSale;

    IconiqInterface public iconiq;
    address public teamAccount;
    address public advisorsAccount;
    address public legalsAccount;
    VreoTokenBounty public bounty;

```

```
    mapping(address => Investment) public investments;

    /// @dev Log entry on rate changed
    /// @param newRate A positive number
    event RateChanged(uint newRate);

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

    /// @dev Log entry on investor falsified
    /// @param investor An Ethereum address
    event InvestorFalsified(address investor);

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

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

    /// @dev Constructor
    /// @param _token A VreoToken
    /// @param _openingTimeOfIconiqSale A positive number
    /// @param _closingTimeOfIconiqSale A positive number
    /// @param _openingTimeOfVreoPresale A positive number
    /// @param _closingTimeOfVreoPresale A positive number
    /// @param _openingTimeOfPublicSale A positive number
    /// @param _closingTimeOfPublicSale A positive number
    /// @param _rate A positive number
    /// @param _iconiq An IconiqInterface
    /// @param _teamAccount An Ethereum address
    /// @param _advisorsAccount An Ethereum address
    /// @param _legalsAccount An Ethereum address
    /// @param _bounty A VreoTokenBounty
    /// @param _wallet An Ethereum address
    constructor(
        VreoToken _token,
        uint _openingTimeOfIconiqSale,
        uint _closingTimeOfIconiqSale,
        uint _openingTimeOfVreoPresale,
        uint _closingTimeOfVreoPresale,
        uint _openingTimeOfPublicSale,
        uint _closingTimeOfPublicSale,
        uint _rate,
        IconiqInterface _iconiq,
        address _teamAccount,
        address _advisorsAccount,
        address _legalsAccount,
        VreoTokenBounty _bounty,
        address _wallet
    )
        public
        Crowdsale(_rate, _wallet, _token)
        TimedCrowdsale(_openingTimeOfIconiqSale, _closingTimeOfPublicSale)
    {
        // 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);

        // Ensure strict timing order
        require(now < _openingTimeOfIconiqSale
                && _openingTimeOfIconiqSale < _closingTimeOfIconiqSale
                && _closingTimeOfIconiqSale < _openingTimeOfVreoPresale
                && _openingTimeOfVreoPresale < _closingTimeOfVreoPresale
                && _closingTimeOfVreoPresale < _openingTimeOfPublicSale
                && _openingTimeOfPublicSale < _closingTimeOfPublicSale);

        // Sanity check of addresses
        require(address(_iconiq) != address(0)
                && _teamAccount != address(0)
                && _advisorsAccount != address(0)
                && _legalsAccount != address(0)
                && address(_bounty) != address(0));

        remainingTokensForSale = TOTAL_TOKEN_CAP_OF_SALE;

        openingTimeOfIconiqSale = _openingTimeOfIconiqSale;
        openingTimeOfIconiqSale = _closingTimeOfIconiqSale;
        openingTimeOfVreoPresale = _openingTimeOfVreoPresale;
        closingTimeOfVreoPresale = _closingTimeOfVreoPresale;
        openingTimeOfPublicSale = _openingTimeOfPublicSale;
        closingTimeOfPublicSale = _closingTimeOfPublicSale;

        iconiq = _iconiq;
        teamAccount = _teamAccount;
        advisorsAccount = _advisorsAccount;
        legalsAccount = _legalsAccount;
        bounty = _bounty;
    }

    /// @dev Destroy
    function destroy() public onlyOwner {
        // TBD
    }

    /// @dev Set rate
    /// @param _newRate A positive number
    function setRate(uint _newRate) public onlyOwner {
        require(_newRate > 0);

        rate = _newRate;

        emit RateChanged(_newRate);
    }

    /// @dev Verify investors
    /// @param _investors A list where each entry is 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;

                if (investment.amount > 0) {
                    fulfillInvestment(investor, investment);
                }
```

```
                emit InvestorVerified(investor);
            }
        }
    }

    /// @dev Falsify investors
    /// @param _investors A list where each entry is an Ethereum address
    function falsifyInvestors(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 = false;

                emit InvestorFalsified(investor);
            }
        }
    }

    /// @dev Withdraw
    function withdraw() public {
        require(hasClosed());

        Investment storage investment = investments[msg.sender];
        investment.amount = 0;

        uint value = investment.value;

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

            msg.sender.transfer(value);

            emit Withdrawn(msg.sender, value);
        }
    }

    function fulfillInvestment(address _investor, Investment _investment) internal {
        uint value = _investment.value;
        uint amount = _investment.amount;

        if (amount > remainingTokensForSale) {
            value = value.mul(remainingTokensForSale).div(amount);
            amount = remainingTokensForSale;
        }

        // Dev note: no overflow possible

        _investment.value -= value;
        _investment.amount -= amount;

        remainingTokensForSale -= amount;

        wallet.transfer(value);
        _deliverTokens(_investor, amount);
    }

    /// @dev Pre validate purchase
    /// @param _beneficiary An Ethereum address
    /// @param _weiAmount A positive number
    function _preValidatePurchase(address _beneficiary, uint _weiAmount) internal {
        require(openingTimeOfIconiqSale <= now && now <= closingTimeOfIconiqSale && iconiq.
↪isAllowed(_beneficiary)
```

```
            || openingTimeOfVreoPresale <= now && now <= closingTimeOfVreoPresale
            || openingTimeOfPublicSale <= now && now <= closingTimeOfPublicSale);

        super._preValidatePurchase(_beneficiary, _weiAmount);
    }

    /// @dev Post validate purchase
    /// @param _beneficiary An Ethereum address
    /// @param _weiAmount A positive number
    function _postValidatePurchase(address _beneficiary, uint _weiAmount) internal {
        // Nothing to do here...
    }

    /// @dev Deliver tokens
    /// @param _beneficiary An Ethereum address
    /// @param _tokenAmount A positive number
    function _deliverTokens(address _beneficiary, uint _tokenAmount) internal {
        super._deliverTokens(_beneficiary, _tokenAmount);

        emit TokensDelivered(_beneficiary, _tokenAmount);
    }

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

        investment.value = investment.value.add(msg.value);
        investment.amount = investment.amount.add(_tokenAmount);

        if (investment.isVerified) {
            fulfillInvestment(_beneficiary, investment);
        }
    }

    /// @dev Update purchasing state
    /// @param _beneficiary An Ethereum address
    /// @param _weiAmount A positive number
    function _updatePurchasingState(address _beneficiary, uint _weiAmount) internal {
        // Nothing to do here...
    }

    /// @dev Get token amount
    /// @param _weiAmount A positive number
    /// @return A positive number
    function _getTokenAmount(uint _weiAmount) internal view returns (uint) {
        uint amount = super._getTokenAmount(_weiAmount);

        if (now <= closingTimeOfIconiqSale) {
            return amount.mul(100 + EXTRA_TOKEN_PCT_IN_ICONIQ_SALE).div(100);
        }

        if (now <= closingTimeOfVreoPresale) {
            return amount.mul(100 + EXTRA_TOKEN_PCT_IN_VREO_PRESALE).div(100);
        }

        return amount;
    }

    /// @dev Forward funds
    function _forwardFunds() internal {
        // 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 }
```

CHAPTER

THREE

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 Log entry on token distributed
    /// @param recipient An Ethereum address
    /// @param amount A positive number
    event TokenDistributed(address recipient, uint amount);

    /// @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) {
            require(_amounts[i] <= token.balanceOf(this));  // TODO: superfluous

            token.transfer(_recipients[i], _amounts[i]);

            emit TokenDistributed(_recipients[i], _amounts[i]);
        }
    }

}
```

CHAPTER

FOUR

ICONIQINTERFACE

```
pragma solidity 0.4.24;


/// @title IconiqInterface
/// @author Autogenerated from a Dia UML diagram
interface IconiqInterface {

    /// @dev Is allowed
    /// @param _account An Ethereum address
    /// @return True or false
    function isAllowed(address _account) external view returns (bool);

}
```