Vreo ICO Solidity Smart Contracts

Release 1

Sicos et al.

Jun 08, 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

TOKENCAPPEDCROWDSALE

```
pragma solidity 0.4.24;

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


/// @title TokenCappedCrowdsale
/// @author Autogenerated from a Dia UML diagram
contract TokenCappedCrowdsale is Crowdsale {

    uint public remainingTokens;

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

    /// @dev Deliver tokens
    /// @param _beneficiary An Ethereum address
    /// @param _tokenAmount A positive number
    function _deliverTokens(address _beneficiary, uint _tokenAmount) internal {
        // SafeMath will reject on underflow.
        remainingTokens = remainingTokens.sub(_tokenAmount);

        super._deliverTokens(_beneficiary, _tokenAmount);
    }

}
```

CHAPTER

THREE

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 weiAmount;   // invested wei
        uint tokenAmount; // 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 withdrawn
    /// @param investor An Ethereum address
    /// @param value A positive number
    event Withdrawn(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 weiAmount = investment.weiAmount;
                uint tokenAmount = investment.tokenAmount;

                if (weiAmount > 0) {
                    investment.weiAmount = 0;
                    investment.tokenAmount = 0;

                    _forwardFunds(weiAmount);
                    _deliverTokens(investor, tokenAmount);

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

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

        require(weiAmount > 0);

        investment.weiAmount = 0;
        investment.tokenAmount = 0;

        msg.sender.transfer(weiAmount);

        emit Withdrawn(msg.sender, weiAmount);
    }

    /// @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, uint _tokenAmount) internal {
        if (investments[msg.sender].isVerified) {
            _deliverTokens(msg.sender, _tokenAmount);

            emit TokensDelivered(msg.sender, _tokenAmount);
        } else {
            investments[msg.sender].weiAmount = msg.value;
            investments[msg.sender].tokenAmount = _tokenAmount;
        }
    }

    /// @dev Forward funds
    function _forwardFunds() internal {
        if (investments[msg.sender].isVerified) {
            super._forwardFunds();
        }
    }

    /// @dev Forward funds
    /// @param _weiAmount A positive number
    function _forwardFunds(uint _weiAmount) internal {
        wallet.transfer(_weiAmount);
    }

}
```

CHAPTER

FOUR

VREOTOKENSALE

```
pragma solidity 0.4.24;

import "../zeppelin/crowdsale/distribution/FinalizableCrowdsale.sol";
import "../zeppelin/crowdsale/emission/MintedCrowdsale.sol";
import "./TokenCappedCrowdsale.sol";
import "./PostKYCCrowdsale.sol";
import "./IconiqInterface.sol";
import "./VreoToken.sol";


/// @title VreoTokenSale
/// @author Autogenerated from a Dia UML diagram
contract VreoTokenSale is PostKYCCrowdsale, TokenCappedCrowdsale, 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       = 20;  // 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

    uint public constant MINIMUM_LIFETIME_AFTER_END = 365 days;

    IconiqInterface public iconiq;
    address public teamAddress;
    address public advisorsAddress;
    address public legalsAddress;
    address public bountyAddress;

    /// @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 _iconiq 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,
        IconiqInterface _iconiq,
        address _teamAddress,
        address _advisorsAddress,
        address _legalsAddress,
        address _bountyAddress,
        address _wallet
    )
        public
        Crowdsale(_rate, _wallet, _token)
        TokenCappedCrowdsale(TOTAL_TOKEN_CAP_OF_SALE)
        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(_iconiq) != address(0)
                && _teamAddress != address(0)
                && _advisorsAddress != address(0)
                && _legalsAddress != address(0)
                && _bountyAddress != address(0));

        iconiq = _iconiq;
        teamAddress = _teamAddress;
        advisorsAddress = _advisorsAddress;
        legalsAddress = _legalsAddress;
        bountyAddress = _bountyAddress;
    }

    /// @dev Destroy
    function liquidate() public onlyOwner {
        require(now >= KYC_VERIFICATION_END_TIME + MINIMUM_LIFETIME_AFTER_END);

        owner.transfer(address(this).balance);
        //selfdestruct(owner);
    }

    /// @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]);
        }

        remainingTokens = remainingTokens.sub(totalAmount);
    }

    /// @dev Set rate
    /// @param _newRate A positive number
    function setRate(uint _newRate) public onlyOwner {
        // A rate change by a magnitude order of ten and above is rather a typo than intention.
        // If it was indeed desired, several setRate transactions have to be sent.
        require(rate / 10 < _newRate && _newRate < 10 * rate);

        rate = _newRate;

        emit RateChanged(_newRate);
    }

    /// @dev Pre validate purchase
    /// @param _beneficiary An Ethereum address
    /// @param _weiAmount A positive number
    function _preValidatePurchase(address _beneficiary, uint _weiAmount) internal {
        require(ICONIQ_SALE_OPENING_TIME <= now && now <= ICONIQ_SALE_CLOSING_TIME && iconiq.
↪isAllowed(msg.sender)
                || VREO_SALE_OPENING_TIME <= now && now <= VREO_SALE_CLOSING_TIME);

        super._preValidatePurchase(_beneficiary, _weiAmount);
    }

    /// @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 <= ICONIQ_SALE_CLOSING_TIME) {
            return amount.mul(100 + BONUS_PCT_IN_ICONIQ_SALE).div(100);
        }

        if (now <= VREO_SALE_PHASE_1_END_TIME) {
            return amount.mul(100 + BONUS_PCT_IN_VREO_SALE_PHASE_1).div(100);
        }

        if (now <= VREO_SALE_PHASE_2_END_TIME) {
            return amount.mul(100 + BONUS_PCT_IN_VREO_SALE_PHASE_2).div(100);
        }

        return amount;  // No bonus
    }

    /// @dev Finalization
    function finalization() internal {
        require(now >= KYC_VERIFICATION_END_TIME);

        MintableToken(token).mint(teamAddress, TOKEN_SHARE_OF_TEAM);
        MintableToken(token).mint(advisorsAddress, TOKEN_SHARE_OF_ADVISORS);
        MintableToken(token).mint(legalsAddress, TOKEN_SHARE_OF_LEGALS);
        MintableToken(token).mint(bountyAddress, TOKEN_SHARE_OF_BOUNTY);

        VreoToken(token).finishMinting();
        VreoToken(token).unpause();

        super.finalization();
    }

}
```

CHAPTER

FIVE

VREOTOKENBOUNTY

```solidity
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]);
        }
    }

}
```

CHAPTER

SIX

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);

}
```