TokenPlatform Documentation

Release

SICOS

Aug 28, 2018

CONTENTS

CHAPTER

ONE

WHITELIST

```
pragma solidity 0.4.24;

import "../node_modules/zeppelin-solidity/contracts/ownership/Ownable.sol";


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

    mapping(address => bool) public admins;
    mapping(address => bool) public isWhitelisted;

    /// @dev Log entry on admin added
    /// @param admin An Ethereum address
    event AdminAdded(address admin);

    /// @dev Log entry on admin removed
    /// @param admin An Ethereum address
    event AdminRemoved(address admin);

    /// @dev Log entry on investor added
    /// @param admin An Ethereum address
    /// @param investor An Ethereum address
    event InvestorAdded(address admin, address investor);

    /// @dev Log entry on investor removed
    /// @param admin An Ethereum address
    /// @param investor An Ethereum address
    event InvestorRemoved(address admin, address investor);

    /// @dev Only admin
    modifier onlyAdmin() {
        require(admins[msg.sender], "Operation is restricted to whitelist admin.");
        _;
    }

    /// @dev Add admin
    /// @param _admin An Ethereum address
    function addAdmin(address _admin) public onlyOwner {
        require(_admin != address(0x0), "Whitelist admin address must not be zero.");

        if (!admins[_admin]) {
            admins[_admin] = true;

            emit AdminAdded(_admin);
        }
    }

    /// @dev Remove admin
```

```
    /// @param _admin An Ethereum address
    function removeAdmin(address _admin) public onlyOwner {
        require(_admin != address(0x0), "Whitelist admin address must not be zero.");

        if (admins[_admin]) {
            admins[_admin] = false;

            emit AdminRemoved(_admin);
        }
    }

    /// @dev Add to whitelist
    /// @param _investors A list where each entry is an Ethereum address
    function addToWhitelist(address[] _investors) public onlyAdmin {
        for (uint256 i = 0; i < _investors.length; i++) {
            if (!isWhitelisted[_investors[i]]) {
                isWhitelisted[_investors[i]] = true;

                emit InvestorAdded(msg.sender, _investors[i]);
            }
        }
    }

    /// @dev Remove from whitelist
    /// @param _investors A list where each entry is an Ethereum address
    function removeFromWhitelist(address[] _investors) public onlyAdmin {
        for (uint256 i = 0; i < _investors.length; i++) {
            if (isWhitelisted[_investors[i]]) {
                isWhitelisted[_investors[i]] = false;

                emit InvestorRemoved(msg.sender, _investors[i]);
            }
        }
    }

}
```

CHAPTER

TWO

KEYRECOVERER

```
pragma solidity 0.4.24;

import "../node_modules/zeppelin-solidity/contracts/ownership/Ownable.sol";
import "./KeyRecoverable.sol";


/// @title SicosToken
/// @author C+B
contract KeyRecoverer is Ownable {

     // Indices of tokens within array. Note: There's no valid token at index 0.
    mapping(address => uint) public indices;
     // Array of tokens. Note: At index 0 is a placeholder that shouldn't be removed ever.
    address[] public tokens;

    /// @dev Constructor
    constructor() public {
        tokens.push(address(0x0));  // Placeholder at index 0.
    }

    /// @dev Check if a token is registered here
    /// @param _token Ethereum address of token contract instance
    /// @return True or false
    function containsToken(address _token) public view returns (bool) {
        return indices[_token] > 0;
    }

    /// @dev Register a key recoverable token
    /// @param _token Ethereum address of token contract instance
    function addToken(address _token) public onlyOwner {
        require(_token != address(0x0), "Token address must not be zero.");
        require(!containsToken(_token), "Token may only be added once.");

        indices[_token] = tokens.length;
        tokens.push(_token);
    }

    /// @dev Unregister a key recoverable token
    /// @param _token Ethereum address of token contract instance
    function removeToken(address _token) public onlyOwner {
        require(_token != address(0x0), "Token address must not be zero.");
        require(containsToken(_token), "Token has not been added before.");

        // Array index of token to delete.
        uint index = indices[_token];

        // Remove token from array.
        tokens[index] = tokens[tokens.length - 1];
        tokens.length = tokens.length - 1;
```

```
        // Update token indices.
        indices[tokens[index]] = index;
        delete indices[_token];
    }

    /// @dev Recover key for an investor in all tokens that are registered here
    /// @param _oldAddress Old Ethereum address of the investor
    /// @param _newAddress New Ethereum address of the investor
    function recoverKey(address _oldAddress, address _newAddress) public onlyOwner {
        for (uint i = 1; i < tokens.length; i++) {
            if (KeyRecoverable(tokens[i]).keyRecoverer() == address(this)) {
                KeyRecoverable(tokens[i]).recoverKey(_oldAddress, _newAddress);
            }
        }
    }

    /// @dev Check if this instance is the keyRecoverer of all registered tokens.
    /// @return True or false
    function checkTokens() public view onlyOwner returns (bool) {
        for (uint i = 1; i < tokens.length; i++) {
            if (KeyRecoverable(tokens[i]).keyRecoverer() == address(this)) {
                return false;
            }
        }
        return true;
    }

}
```

CHAPTER

THREE

STOKRCROWDSALE

```
pragma solidity 0.4.24;

import "../node_modules/zeppelin-solidity/contracts/crowdsale/distribution/RefundableCrowdsale.sol";
import "./StokrToken.sol";


/// @title StokrCrowdsale
/// @author Autogenerated from a Dia UML diagram
contract StokrCrowdsale is RefundableCrowdsale {

    uint public tokenCap;
    uint public tokenRemaining;

    address public teamAccount;
    uint public teamShare;

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

    /// @dev Constructor
    /// @param _token Address of token contract
    /// @param _tokenCap Maximum amount of token (sale + share)
    /// @param _tokenGoal Minimum amount of sold tokens for a successful sale
    /// @param _openingTime Unix timestamp of crowdsale opening
    /// @param _closingTime Unix timestamp of crowdsale closing
    /// @param _rate Tokens per ether rate
    /// @param _wallet Multisig wallet for receiving invested ether
    constructor(StokrToken _token,
                uint _tokenCap,
                uint _tokenGoal,
                uint _openingTime,
                uint _closingTime,
                uint _rate,
                uint _teamShare,
                address _wallet)
        public
        RefundableCrowdsale(_tokenGoal)
        TimedCrowdsale(_openingTime, _closingTime)
        Crowdsale(_rate, _wallet, _token)
    {
        require(_teamShare <= _tokenCap, "Team share must not exceed token cap.");
        require(_tokenGoal <= _tokenCap - _teamShare, "Goal must be attainable.");

        tokenCap = _tokenCap;
        teamShare = _teamShare;
        tokenRemaining = _tokenCap - _teamShare;
    }
```

```
    /// @dev Distribute tokens.
    /// @param _accounts List of Ethereum addresses who will receive tokens
    /// @param _amounts List of token amounts per account
    function distributeTokens(address[] _accounts, uint[] _amounts) public onlyOwner {
        require(_accounts.length == _amounts.length, "Number of accounts and amounts must be equal.
↪");

        for (uint i = 0; i < _accounts.length; ++i) {
            _deliverTokens(_accounts[i], _amounts[i]);
        }
    }

    /// @dev Set rate
    /// @param _newRate A positive number
    function setRate(uint _newRate) public onlyOwner {
        // A rate change by an order of magnitude (or more) is likely a typo instead of intention
        // Note, this implicitly ensures the new rate cannot be set to zero
        require(rate / 10 < _newRate && _newRate < 10 * rate, "Rate change must be less than an␣
↪order of magnitude.");

        if (_newRate != rate) {
            emit RateChanged(rate, _newRate);
        }
        rate = _newRate;
    }

    /// @dev Set team account
    /// @param _teamAccount An Ethereum address.
    function setTeamAccount(address _teamAccount) public onlyOwner {
        require(_teamAccount != address(0x0), "Team account address must not be zero.");

        teamAccount = _teamAccount;
    }

    /// @dev Time remaining of open crowdsale.
    /// @return Duration in seconds, or 0 if crowdsale has ended.
    function timeRemaining() public view returns (uint) {
        if (now >= closingTime) {
            return 0;
        }

        return closingTime - now;
    }

    /// @dev Overridden RefundableCrowdsale.goalReached().
    /// @return Whether the desired token amount was sold or not
    function goalReached() public view returns (bool) {
        return tokenCap - teamShare - tokenRemaining >= goal;
    }

    /// @dev Extend parent behavior requiring beneficiary to be identical to msg.sender
    /// @param _beneficiary Token purchaser
    /// @param _weiAmount Amount of wei contributed
    function _preValidatePurchase(address _beneficiary, uint256 _weiAmount) internal {
        require(_beneficiary == msg.sender, "Message sender and beneficiary address must be the␣
↪same.");

        super._preValidatePurchase(_beneficiary, _weiAmount);
    }

    /// @dev Extend parent behavior by minting a tokens for the benefit of beneficiary.
    /// @param _beneficiary Token recipient
```

```
    /// @param _tokenAmount Token amount
    function _deliverTokens(address _beneficiary, uint256 _tokenAmount) internal {
        require(tokenRemaining >= _tokenAmount, "Amount of tokens to deliver exceeds remaining␣
↪amount.");

        tokenRemaining -= _tokenAmount;

        StokrToken(token).mint(_beneficiary, _tokenAmount);
    }

    /// @dev Extend parent behavior to finish the token minting.
    function finalization() internal {
        super.finalization();

        if (goalReached()) {
            require(teamAccount != address(0x0), "Team account has to be set prior to finalization.
↪");

            StokrToken(token).mint(teamAccount, teamShare);
            StokrToken(token).finishMinting();
        }
        else {
            StokrToken(token).destruct();
        }
    }

}
```

CHAPTER

FOUR

WHITELISTED

```
pragma solidity 0.4.24;

import "../node_modules/zeppelin-solidity/contracts/ownership/Ownable.sol";
import "./Whitelist.sol";


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

    Whitelist public whitelist;

    /// @dev Log entry on whitelist changed
    /// @param newWhitelist An Ethereum address
    event WhitelistChanged(address newWhitelist);

    /// @dev Ensure only whitelisted
    modifier onlyWhitelisted(address _address) {
        require(whitelist.isWhitelisted(_address), "Address must be whitelisted.");
        _;
    }

    /// @dev Constructor
    /// @param _whitelist An Ethereum address
    constructor(Whitelist _whitelist) public {
        setWhitelist(_whitelist);
    }

    /// @dev Set whitelist
    /// @param _newWhitelist An Ethereum address
    function setWhitelist(Whitelist _newWhitelist) public onlyOwner {
        require(_newWhitelist != address(0x0), "Whitelist address must not be zero.");

        if (address(whitelist) != address(0x0) && address(_newWhitelist) != address(whitelist)) {
            emit WhitelistChanged(_newWhitelist);
        }
        whitelist = Whitelist(_newWhitelist);
    }

}
```

CHAPTER

FIVE

KEYRECOVERABLE

```
pragma solidity 0.4.24;

import "../node_modules/zeppelin-solidity/contracts/ownership/Ownable.sol";


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

    address public keyRecoverer;

    /// @dev Log entry on key recoverer changed
    /// @param newKeyRecoverer An Ethereum address
    event KeyRecovererChanged(address newKeyRecoverer);

    /// @dev Log entry on key recovered
    /// @param oldAddress An Ethereum address
    /// @param newAddress An Ethereum address
    event KeyRecovered(address oldAddress, address newAddress);

    /// @dev Ensure only key recoverer
    modifier onlyKeyRecoverer() {
        require(msg.sender == keyRecoverer, "Operation is restricted to key recoverer.");
        _;
    }

    /// @dev Constructor
    /// @param _keyRecoverer An Ethereum address
    constructor(address _keyRecoverer) public {
        setKeyRecoverer(_keyRecoverer);
    }

    /// @dev Set key recoverer
    /// @param _newKeyRecoverer An Ethereum address
    function setKeyRecoverer(address _newKeyRecoverer) public onlyOwner {
        require(_newKeyRecoverer != address(0x0), "Key recoverer address must not be zero.");

        if (keyRecoverer != address(0x0) && _newKeyRecoverer != keyRecoverer) {
            emit KeyRecovererChanged(_newKeyRecoverer);
        }
        keyRecoverer = _newKeyRecoverer;
    }

    /// @dev Recover key
    /// @param _oldAddress An Ethereum address
    /// @param _newAddress An Ethereum address
    function recoverKey(address _oldAddress, address _newAddress) public;

}
```

CHAPTER

SIX

PROFITSHARING

```
pragma solidity 0.4.24;

import "../node_modules/zeppelin-solidity/contracts/ownership/Ownable.sol";
import "../node_modules/zeppelin-solidity/contracts/token/ERC20/ERC20.sol";
import "../node_modules/zeppelin-solidity/contracts/math/SafeMath.sol";


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

    using SafeMath for uint;

    struct InvestorAccount {
        uint balance;
        uint lastTotalProfits;
        uint profitShare;
    }

    mapping(address => InvestorAccount) public accounts;

    address public profitDepositor;
    uint public totalProfits;

    // As long as the total supply isn't fixed, i.e. new tokens can appear out of thin air,
    // the investors' profit shares aren't determined.
    bool public totalSupplyIsFixed;
    uint internal totalSupply_;

    event ProfitDepositorChanged(address newProfitDepositor);

    /// @dev Log entry on profit deposited
    /// @param depositor An Ethereum address
    /// @param amount A positive number
    event ProfitDeposited(address depositor, uint amount);

    /// @dev Log entry on profit share updated
    /// @param investor An Ethereum address
    /// @param amount A positive number
    event ProfitShareUpdated(address investor, uint amount);

    /// @dev Log entry on profit withdrawal
    /// @param investor An Ethereum address
    /// @param amount A positive number
    event ProfitWithdrawal(address investor, uint amount);

    /// @dev Ensure only depositor
    modifier onlyProfitDepositor() {
        require(msg.sender == profitDepositor, "Operation is restricted to profit depositor.");
```

```
        _;
    }

    /// @dev Constructor
    /// @param _profitDepositor An Ethereum address
    constructor(address _profitDepositor) public {
        setProfitDepositor(_profitDepositor);
    }

    /// @dev Change profit depositor
    /// @param _newProfitDepositor An Ethereum address
    function setProfitDepositor(address _newProfitDepositor) public onlyOwner {
        require(_newProfitDepositor != address(0x0), "Profit depositor address must not be zero.");

        if (profitDepositor != address(0x0) && _newProfitDepositor != profitDepositor) {
            emit ProfitDepositorChanged(_newProfitDepositor);
        }
        profitDepositor = _newProfitDepositor;
    }

    /// @dev Deposit profit
    function depositProfit() public payable onlyProfitDepositor {
        totalProfits = totalProfits.add(msg.value);

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

    /// @dev Profit share owing
    /// @param _investor An Ethereum address
    /// @return A positive number
    function profitShareOwing(address _investor) public view returns (uint) {
        if (!totalSupplyIsFixed || totalSupply_ == 0) {
            return 0;
        }
        return totalProfits.sub(accounts[_investor].lastTotalProfits)
                           .mul(accounts[_investor].balance)
                           .div(totalSupply_);  // <- The linter doesn't like this.
    }

    /// @dev Update profit share
    /// @param _investor An Ethereum address
    function updateProfitShare(address _investor) public {
        require(totalSupplyIsFixed, "Total supply must be fixed prior to update profit share.");

        uint additionalProfitShare = profitShareOwing(_investor);

        accounts[_investor].lastTotalProfits = totalProfits;
        accounts[_investor].profitShare = accounts[_investor].profitShare.
↪add(additionalProfitShare);

        emit ProfitShareUpdated(_investor, additionalProfitShare);
    }

    /// @dev Withdraw profit share
    function withdrawProfitShare() public {
        updateProfitShare(msg.sender);

        uint withdrawnProfitShare = accounts[msg.sender].profitShare;

        accounts[msg.sender].profitShare = 0;
        msg.sender.transfer(withdrawnProfitShare);

        emit ProfitWithdrawal(msg.sender, withdrawnProfitShare);
```

```
    }

}
```

CHAPTER

SEVEN

MINTABLETOKEN

```
pragma solidity 0.4.24;

import "./ProfitSharing.sol";
import "./Whitelisted.sol";


/// @title MintableToken
/// @author Autogenerated from a Dia UML diagram
/// @dev A mintable token is a token that can be minted
contract MintableToken is ERC20, ProfitSharing, Whitelisted {

    address public minter;
    uint public numberOfInvestors = 0;

    /// @dev Log entry on mint
    /// @param to An Ethereum address
    /// @param amount A positive number
    event Minted(address to, uint amount);

    /// @dev Log entry on mint finished
    event MintFinished();

    /// @dev Ensure only minter
    modifier onlyMinter() {
        require(msg.sender == minter, "Operation is restricted to minter.");
        _;
    }

    /// @dev Ensure can mint
    modifier canMint() {
        require(!totalSupplyIsFixed, "Total token supply must not be fixed.");
        _;
    }

    /// @dev Ensure not minting
    modifier notMinting() {
        require(totalSupplyIsFixed, "Total token supply must not change.");
        _;
    }

    /// @dev Set minter
    /// @param _minter An Ethereum address
    function setMinter(address _minter) public onlyOwner {
        require(minter == address(0x0), "Minter may only be set once.");
        require(_minter != address(0x0), "Minter address must not be zero.");

        minter = _minter;
    }
```

```
    /// @dev Mint
    /// @param _to An Ethereum address
    /// @param _amount A positive number
    function mint(address _to, uint _amount) public onlyMinter canMint onlyWhitelisted(_to) {
        if (accounts[_to].balance == 0) {
            numberOfInvestors++;
        }

        totalSupply_ = totalSupply_.add(_amount);
        accounts[_to].balance = accounts[_to].balance.add(_amount);

        emit Minted(_to, _amount);
        emit Transfer(address(0x0), _to, _amount);
    }

    /// @dev Finish minting
    function finishMinting() public onlyMinter canMint {
        totalSupplyIsFixed = true;

        emit MintFinished();
    }

    /// @dev Minting finished
    /// @return True or false
    function mintingFinished() public view returns (bool) {
        return totalSupplyIsFixed;
    }

}
```

CHAPTER

EIGHT

STOKRTOKEN

```
pragma solidity 0.4.24;

import "./MintableToken.sol";
import "./KeyRecoverable.sol";
import "./Whitelisted.sol";


/// @title StokrToken
/// @author Autogenerated from a Dia UML diagram
contract StokrToken is MintableToken, KeyRecoverable {

    mapping(address => mapping(address => uint)) internal allowance_;

    /// @dev Constructor
    /// @param _whitelist An Ethereum address
    /// @param _keyRecoverer An Ethereum address
    constructor(Whitelist _whitelist, address _profitDepositor, address _keyRecoverer)
        public
        Whitelisted(_whitelist)
        ProfitSharing(_profitDepositor)
        KeyRecoverable(_keyRecoverer)
    {}

    /// @dev Self destruct
    function destruct() public onlyMinter {
        selfdestruct(owner);
    }

    /// @dev Recover key
    /// @param _oldAddress An Ethereum address
    /// @param _newAddress An Ethereum address
    function recoverKey(address _oldAddress, address _newAddress)
        public
        onlyKeyRecoverer
        onlyWhitelisted(_oldAddress)
        onlyWhitelisted(_newAddress)
    {
        // Ensure that new address is *not* an existing account.
        // Check for account.profitShare is not needed because of following implication:
        //   (account.lastTotalProfits == 0) ==> (account.profitShare == 0)
        require(accounts[_newAddress].balance == 0 && accounts[_newAddress].lastTotalProfits == 0,
                "New account address must not be an already existing account.");

        updateProfitShare(_oldAddress);

        accounts[_newAddress] = accounts[_oldAddress];
        delete accounts[_oldAddress];

        emit KeyRecovered(_oldAddress, _newAddress);
```

```
    }

    /// @dev Total supply
    /// @return A positive number
    function totalSupply() public view returns (uint) {
        return totalSupply_;
    }

    /// @dev Balance of
    /// @param _investor An Ethereum address
    /// @return A positive number
    function balanceOf(address _investor) public view returns (uint) {
        return accounts[_investor].balance;
    }

    /// @dev Allowance
    /// @param _investor An Ethereum address
    /// @param _spender An Ethereum address
    /// @return A positive number
    function allowance(address _investor, address _spender) public view returns (uint) {
        return allowance_[_investor][_spender];
    }

    /// @dev Approve
    /// @param _spender An Ethereum address
    /// @param _value A positive number
    /// @return True or false
    function approve(address _spender, uint _value)
        public
        onlyWhitelisted(msg.sender)
        notMinting
        returns (bool)
    {
        allowance_[msg.sender][_spender] = _value;

        emit Approval(msg.sender, _spender, _value);

        return true;
    }

    /// @dev Transfer
    /// @param _to An Ethereum address
    /// @param _value A positive number
    /// @return True or false
    function transfer(address _to, uint _value) public returns (bool) {
        return _transfer(msg.sender, _to, _value);
    }

    /// @dev Transfer from
    /// @param _from An Ethereum address
    /// @param _to An Ethereum address
    /// @param _value A positive number
    /// @return True or false
    function transferFrom(address _from, address _to, uint _value) public returns (bool) {
        require(_value <= allowance_[_from][msg.sender], "Amount to transfer must not exceed␣
↪allowance.");

        allowance_[_from][msg.sender] = allowance_[_from][msg.sender].sub(_value);

        return _transfer(_from, _to, _value);
    }

    /// @dev Transfer
```

```
    /// @param _from An Ethereum address
    /// @param _to An Ethereum address
    /// @param _value A positive number
    /// @return True or false
    function _transfer(address _from, address _to, uint _value)
        internal
        onlyWhitelisted(_from)
        onlyWhitelisted(_to)
        notMinting
        returns (bool)
    {
        require(_to != address(0x0), "Recipient address must not be zero.");
        require(_value <= accounts[_from].balance, "Amount to transfer must not exceed balance.");

        updateProfitShare(_from);
        updateProfitShare(_to);

        accounts[_from].balance = accounts[_from].balance.sub(_value);
        accounts[_to].balance = accounts[_to].balance.add(_value);

        emit Transfer(_from, _to, _value);

        return true;
    }

}
```

CHAPTER

NINE

SAMPLETOKEN

```
pragma solidity 0.4.24;

import "./StokrToken.sol";


/// @title StokrToken
/// @author Autogenerated from a Dia UML diagram
contract SampleToken is StokrToken {

    string public name = "Sample Stokr Token";
    string public symbol = "SAM";
    uint8 public decimals = 18;

    constructor(
        string _name,
        string _symbol,
        Whitelist _whitelist,
        address _profitDepositor,
        address _keyRecoverer
    )
        StokrToken(_whitelist, _profitDepositor, _keyRecoverer)
    {
        name = _name;
        symbol = _symbol;
    }

}
```