TokenPlatform Documentation

Release

Sicos

Apr 18, 2018

CONTENTS

CHAPTER

ONE

WHITELIST

```
pragma solidity 0.4.19;

import "../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]);
        _;
    }

    /// @dev Add admin
    /// @param _admin An Ethereum address
    function addAdmin(address _admin) public onlyOwner {
        if (!admins[_admin]) {
            admins[_admin] = true;
            AdminAdded(_admin);
        }
    }

    /// @dev Remove admin
    /// @param _admin An Ethereum address
    function removeAdmin(address _admin) public onlyOwner {
        if (admins[_admin]) {
            admins[_admin] = false;
            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;
                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;
                InvestorRemoved(msg.sender, _investors[i]);
            }
        }
    }

}
```

CHAPTER

TWO

KEYRECOVERER

```
pragma solidity 0.4.19;

import "../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
    function KeyRecoverer() 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) && !containsToken(_token));

        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) && containsToken(_token));

        // 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

SICOSCROWDSALE

```
pragma solidity 0.4.19;

import "../zeppelin-solidity/contracts/crowdsale/distribution/FinalizableCrowdsale.sol";
import "../zeppelin-solidity/contracts/crowdsale/validation/CappedCrowdsale.sol";
import "./MintableToken.sol";


/// @title SicosCrowdsale
/// @author Autogenerated from a Dia UML diagram
contract SicosCrowdsale is FinalizableCrowdsale, CappedCrowdsale {

    /// @dev Crowdsale
    /// @param _token An Ethereum address
    /// @param _startTime A positive number
    /// @param _endTime A positive number
    /// @param _rate A positive number
    /// @param _wallet An Ethereum address
    function SicosCrowdsale(MintableToken _token,
                            uint _startTime,
                            uint _endTime,
                            uint _rate,
                            uint _cap,
                            address _wallet)
        public
        CappedCrowdsale(_cap)
        TimedCrowdsale(_startTime, _endTime)
        Crowdsale(_rate, _wallet, _token)
    {}

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

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

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

    /// @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 {
        super._preValidatePurchase(_beneficiary, _weiAmount);

        require(_beneficiary == msg.sender);
    }

    /// @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 {
        MintableToken(token).mint(_beneficiary, _tokenAmount);
    }

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

        MintableToken(token).finishMinting();
    }

}
```

CHAPTER

FOUR

WHITELISTED

```
pragma solidity 0.4.19;

import "../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));
        _;
    }

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

    /// @dev Set whitelist
    /// @param _newWhitelist An Ethereum address
    function setWhitelist(address _newWhitelist) public onlyOwner {
        require(_newWhitelist != address(0x0));

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

}
```

CHAPTER

FIVE

KEYRECOVERABLE

```
pragma solidity 0.4.19;

import "../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);
        _;
    }

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

    /// @dev Set key recoverer
    /// @param _newKeyRecoverer An Ethereum address
    function setKeyRecoverer(address _newKeyRecoverer) public onlyOwner {
        require(_newKeyRecoverer != address(0x0));

        if (keyRecoverer != address(0x0) && _newKeyRecoverer != keyRecoverer) {
            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 0.4.19;

import "../zeppelin-solidity/contracts/ownership/Ownable.sol";
import "../zeppelin-solidity/contracts/token/ERC20/ERC20.sol";
import "../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);
        _;
    }

    /// @dev Constructor
    /// @param _profitDepositor An Ethereum address
    function ProfitSharing(address _profitDepositor) {
        setProfitDepositor(_profitDepositor);
    }

    /// @dev Change profit depositor
    /// @param _newProfitDepositor An Ethereum address
    function setProfitDepositor(address _newProfitDepositor) public onlyOwner {
        require(_newProfitDepositor != address(0x0));

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

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

        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) {
        return totalSupplyIsFixed && totalSupply_ > 0
               ? totalProfits.sub(accounts[_investor].lastTotalProfits)
                             .mul(accounts[_investor].balance)
                             .div(totalSupply_)  // <- The linter doesn't like this.
               : 0;
    }

    /// @dev Update profit share
    /// @param _investor An Ethereum address
    function updateProfitShare(address _investor) public {
        require(totalSupplyIsFixed);

        uint additionalProfitShare =  profitShareOwing(_investor);

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

        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);

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

```

}
```

CHAPTER

SEVEN

MINTABLETOKEN

```
pragma solidity 0.4.19;

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;

    /// @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);
        _;
    }

    /// @dev Ensure can mint
    modifier canMint() {
        require(!totalSupplyIsFixed);
        _;
    }

    /// @dev Ensure not minting
    modifier notMinting() {
        require(totalSupplyIsFixed);
        _;
    }

    /// @dev Set minter
    /// @param _minter An Ethereum address
    function setMinter(address _minter) public onlyOwner {
        require(_minter != address(0x0) && minter == address(0x0));

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

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

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

        MintFinished();
    }

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

}
```

CHAPTER

EIGHT

SICOSTOKEN

```
pragma solidity 0.4.19;

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


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

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

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

    /// @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);

        updateProfitShare(_oldAddress);

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

        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;

        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) 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) returns (bool) {
        require(_value <= allowance_[_from][msg.sender]);

        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(0));
        require(_value <= accounts[_from].balance);

        updateProfitShare(_from);
        updateProfitShare(_to);

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

        Transfer(_from, _to, _value);

        return true;
    }

}
```

CHAPTER

NINE

SAMPLETOKEN

```
pragma solidity 0.4.19;

import "./SicosToken.sol";


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

    string public name = "Sample Sicos Token";
    string public symbol = "SAM";
    uint public decimal = 18;

}
```