pragma solidity 0.4.24;

library SafeMath {

function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {

if (a == 0) {

return 0;

}

c = a \* b;

assert(c / a == b);

return c;

}

function div(uint256 a, uint256 b) internal pure returns (uint256) {

return a / b;

}

function sub(uint256 a, uint256 b) internal pure returns (uint256) {

assert(b <= a);

return a - b;

}

function add(uint256 a, uint256 b) internal pure returns (uint256 c) {

c = a + b;

assert(c >= a);

return c;

}

}

contract customIcoToken{

using SafeMath for uint256;

/\* Events \*/

event Transfer(address indexed from, address indexed to, uint256 value);

event Approval(address indexed owner, address indexed spender, uint256 value);

event LogRefund(address indexed \_to, uint256 \_value);

event CreateToken(address indexed \_to, uint256 \_value);

// metadata

string public name;

string public symbol;

uint256 public decimals;

// contracts

address public ethFundDeposit; // deposit address for ETH

address public tokenFundDeposit;

// crowdsale parameters

bool public isFinalized; // switched to true in operational state

uint256 public fundingStartBlock;

uint256 public fundingEndBlock;

uint256 public tokenFund;

uint256 public tokenExchangeRate;

uint256 public tokenCreationCap;

uint256 public tokenCreationMin;

/\* Storage \*/

mapping(address => uint256) balances;

mapping (address => mapping (address => uint256)) internal allowed;

uint256 public totalSupply;

/\* Getters \*/

function totalSupply() public view returns (uint256) {

return totalSupply;

}

function balanceOf(address \_owner) public view returns (uint256 balance) {

return balances[\_owner];

}

function allowance(address \_owner, address \_spender) public view returns (uint256) {

return allowed[\_owner][\_spender];

}

/\* Methods \*/

function transferFrom(address \_from, address \_to, uint256 \_value) public returns (bool) {

require(\_to != address(0));

require(\_value <= balances[\_from]);

require(\_value <= allowed[\_from][msg.sender]);

balances[\_from] = balances[\_from].sub(\_value);

balances[\_to] = balances[\_to].add(\_value);

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

emit Transfer(\_from, \_to, \_value);

return true;

}

function approve(address \_spender, uint256 \_value) public returns (bool) {

allowed[msg.sender][\_spender] = \_value;

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

return true;

}

function increaseApproval(address \_spender, uint \_addedValue) public returns (bool) {

allowed[msg.sender][\_spender] = allowed[msg.sender][\_spender].add(\_addedValue);

emit Approval(msg.sender, \_spender, allowed[msg.sender][\_spender]);

return true;

}

function decreaseApproval(address \_spender, uint \_subtractedValue) public returns (bool) {

uint oldValue = allowed[msg.sender][\_spender];

if (\_subtractedValue > oldValue) {

allowed[msg.sender][\_spender] = 0;

} else {

allowed[msg.sender][\_spender] = oldValue.sub(\_subtractedValue);

}

emit Approval(msg.sender, \_spender, allowed[msg.sender][\_spender]);

return true;

}

function transfer(address \_to, uint256 \_value) public returns (bool) {

require(\_to != address(0));

require(\_value <= balances[msg.sender]);

// SafeMath.sub will throw if there is not enough balance.

balances[msg.sender] = balances[msg.sender].sub(\_value);

balances[\_to] = balances[\_to].add(\_value);

emit Transfer(msg.sender, \_to, \_value);

return true;

}

/\* Crowdsale methods \*/

/// @dev Accepts ether and creates new tokens.

function createTokens() payable external {

require (isFinalized == false);

require(block.number > fundingStartBlock);

require(block.number < fundingEndBlock);

require(msg.value > 0);

uint256 tokens = msg.value.mul(tokenExchangeRate);

uint256 checkedSupply = totalSupply.add(tokens);

// return money if something goes wrong

require(tokenCreationCap >= checkedSupply); // odd fractions won't be found

totalSupply = checkedSupply;

balances[msg.sender] += tokens; // safeAdd not needed; bad semantics to use here

emit CreateToken(msg.sender, tokens); // logs token creation

}

/// @dev Ends the funding period and sends the ETH home

function finalize() external {

require(isFinalized == false);

require(msg.sender == ethFundDeposit);

require(totalSupply > tokenCreationMin); // have to sell minimum to move to operational

require(block.number > fundingEndBlock || totalSupply == tokenCreationCap);

// move to operational

isFinalized = true;

assert(ethFundDeposit.send(address(this).balance)); // send the eth

}

/// @dev Allows contributors to recover their ether in the case of a failed funding campaign.

function refund() external {

require(isFinalized == false); // prevents refund if operational

require(block.number > fundingEndBlock); // prevents refund until sale period is over

require(totalSupply < tokenCreationMin); // no refunds if we sold enough

require(msg.sender != tokenFundDeposit); // team not entitled to a refund

uint256 tokenVal = balances[msg.sender];

require(tokenVal > 0);

balances[msg.sender] = 0;

totalSupply = totalSupply.sub(tokenVal); // extra safe

uint256 ethVal = tokenVal / tokenExchangeRate; // should be safe; previous throws covers edges

emit LogRefund(msg.sender, ethVal); // log it

assert(msg.sender.send(ethVal)); // if you're using a contract; make sure it works with .send gas limits

}

constructor(

string \_name,

string \_symbol,

uint8 \_decimals,

address \_ethFundDeposit,

address \_tokenFundDeposit,

uint256 \_tokenFund,

uint256 \_tokenExchangeRate,

uint256 \_tokenCreationCap,

uint256 \_tokenCreationMin,

uint256 \_fundingStartBlock,

uint256 \_fundingEndBlock) public

{

name = \_name;

symbol = \_symbol;

decimals = \_decimals;

isFinalized = false; //controls pre through crowdsale state

ethFundDeposit = \_ethFundDeposit;

tokenFundDeposit = \_tokenFundDeposit;

tokenFund = \_tokenFund\*10\*\*decimals;

tokenExchangeRate = \_tokenExchangeRate;

tokenCreationCap = \_tokenCreationCap\*10\*\*decimals;

tokenCreationMin = \_tokenCreationMin\*10\*\*decimals;

fundingStartBlock = \_fundingStartBlock;

fundingEndBlock = \_fundingEndBlock;

totalSupply = tokenFund;

balances[tokenFundDeposit] = tokenFund;

emit CreateToken(tokenFundDeposit, tokenFund);

}

}