Audit Smart Contract

For Token - Token Smart Money Project
By BECHAIN Strategy & Consulting

Table of Contents

Audit context

We analyzed in this report the effectiveness and safety of the code in order to bring you in a shortest time our technical recommendations and implementations suggests.

Smart contract auditing cannot unveil all existing flaws and/or vulnerabilities in the same way as an audit in which no flaws and/or vulnerabilities are found is not a guarantee for a secure smart contract. The objective of the audit is to find out flaws and/or vulnerabilities that were unobserved during development.

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Overview of the audit

We have found some issues related to the declaration of some "modifiers". So, we suggest to take our recommendations into consideration and modify the content.

Attack made to the contracts

There aren't critical issues in the smart contract audited.

Critical vulnerabilities found in the contracts

There aren't critical issues in the smart contract audited.

Medium vulnerabilities found in the contracts

Avoid expensive gas executions.

Low severity vulnerabilities found

- Is more recommended to use the latest version of solidity to enforce the compiler and prevent from any previous bugs/flaws.

  The current version is 0.4.25

- Verify the use of the modifier "**view**"

Smart contracts

Ownable Implementation (Ownable.sol)

```
pragma solidity ^0.4.24;


/**

COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


@title Ownable
@dev The Ownable contract has an owner address, and provides basic
authorization control
functions, this simplifies the implementation of "user permissions".

*/


contract Ownable {

  mapping(address => bool) public owner;

  event LogOwnershipTransferred(address indexed previousOwner,
address indexed newOwner);
  event LogAllowOwnership(address indexed allowedAddress);

  /**
   * @dev The Ownable constructor sets the original `owner` of the
contract to the sender
   * account.
   */
  constructor() public {
    owner[msg.sender] = true;
  }
  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(owner[msg.sender]);
    _;
  }

  /**
   * @dev Allows the current owner to transfer control of the
contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   * @return {"success" : "Returns true when successfully
transferred ownership"}
   */
  function transferOwnership(address newOwner) public onlyOwner
returns (bool success) {
    require(newOwner != address(0));
    emit LogOwnershipTransferred(msg.sender, newOwner);
    owner[newOwner] = true;
    owner[msg.sender] = false;
  }

  /**
   * @dev Allows interface contracts to access contract methods (e.g.
Storage contract)
   * @param allowedAddress The address of new owner
   * @return {"success" : "Returns true when successfully allowed
ownership"}
   */
  function allowOwnership(address allowedAddress) public onlyOwner
returns (bool success) {
    owner[allowedAddress] = true;
    emit LogAllowOwnership(allowedAddress);
    return true;
  }

}
```

1. Line 52 transferOwnership function

Warning:
You need to add a line at the end **return true**, to respect the return of the function.

```
function transferOwnership(address newOwner) public onlyOwner
returns (bool success) {
    require(newOwner != address(0));
    emit LogOwnershipTransferred(msg.sender, newOwner);
    owner[newOwner] = true;
    owner[msg.sender] = false;
    return true;
  }
```

Conclusion Ownable contract
No relevant issue on this contract.

SafeMath.sol

```
pragma solidity ^0.4.24;


/**
 * @title SafeMath
 * @notice Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @notice Multiplies two numbers, throws on overflow.
  * @param a Multiplier
  * @param b Multiplicand
  * @return {"result" : "Returns product"}
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256
result) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @notice Integer division of two numbers, truncating the quotient.
  * @param a Dividend
  * @param b Divisor
  * @return {"result" : "Returns quotient"}
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256
result) {
    // @dev assert(b > 0); // Solidity automatically throws when
dividing by 0
    uint256 c = a / b;
    // @dev assert(a == b * c + a % b); // There is no case in which
this doesn't hold
    return c;
  }

  /**
  * @notice Subtracts two numbers, throws on underflow.
  * @param a Subtrahend
  * @param b Minuend
  * @return {"result" : "Returns difference"}
 */
  function sub(uint256 a, uint256 b) internal pure returns (uint256
result) {
    // @dev throws on overflow (i.e. if subtrahend is greater than
minuend)
    assert(b <= a);
    return a - b;
  }

  /**
  * @notice Adds two numbers, throws on overflow.
  * @param a First addend
  * @param b Second addend
  * @return {"result" : "Returns summation"}
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256
result) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

Conclusion SafeMath contract

No relevant issue on this contract.
The openZeppelin's SafeMath.sol standard was chosen as the basis for development. It will be necessary to ensure the updating of future recommendations if this contract is brought to be reused.

TokenIOStorage.sol

```
pragma solidity ^0.4.24;

/*
COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

```
/**
@title TokenIO - ERC20 Compliant Smart Contract for Token, Inc.

@author  Ryan  Tate  <ryan.michael.tate@gmail.com>,  Sean  Pollock
<seanpollock3344@gmail.com>

@notice    Contract    uses    generalized    storage    contract,
`TokenIOStorage`, for
upgradeability of interface contract.
*/

import "./Ownable.sol";
import "./TokenIOStorage.sol";
import "./TokenIOLib.sol";

contract TokenIOAuthority is Ownable {

    /// @dev Set reference to TokenIOLib interface which proxies to
TokenIOStorage
    using TokenIOLib for TokenIOLib.Data;
    TokenIOLib.Data lib;

    /**
     * @notice Constructor method for Authority contract
     * @param _storageContract Ethereum Address of TokenIOStorage
contract
     */
    constructor(address _storageContract) public {
        /*
         * @notice Set the storage contract for the interface
         * @dev This contract will be unable to use the storage
constract until
         * @dev contract address is authorized with the storage
contract
         * @dev Once authorized, you can setRegisteredFirm and
setRegisteredAuthority
        */
        lib.Storage = TokenIOStorage(_storageContract);

        /// @dev set owner to contract initiator
        owner[msg.sender] = true;
    }

    /**
     * @notice Registers a firm as authorized true/false
     * @param firmName Name of firm
     * @param _authorized Authorization status
     * @return {"success" : "Returns true if lib.setRegisteredFirm
succeeds"}
     */
    function  setRegisteredFirm(string  firmName,  bool  _authorized)
public onlyAuthority(firmName, msg.sender) returns (bool success) {
        /// @notice set firm registration status
        require(lib.setRegisteredFirm(firmName, _authorized));
        return true;
    }

    /**
     * @notice Registers an authority asoociated with the given firm
as true/false
     * @param firmName Name of firm
     * @param authority Address of authority account
     * @param _authorized Authorization status
     *     @return     {"success"     :     "Returns     true     if
lib.setRegisteredAuthority succeeds"}
     */
    function   setRegisteredAuthority(string   firmName,   address
authority,   bool   _authorized)   public   onlyAuthority(firmName,
msg.sender) returns (bool success) {
        /// @notice set authority of firm to given status
        require(lib.setRegisteredAuthority(firmName,     authority,
_authorized));
        return true;
    }

    /**
     * @notice Gets firm asoociated with an authority address
     * @param authority Address of authority account
     * @return {"firm" : "name of firm"}
     */
    function  getFirmFromAuthority(address  authority)  public  view
returns (string firm) {
        return lib.getFirmFromAuthority(authority);
    }

    /**
     * @notice Gets status of firm registration
     * @param firmName Name of firm
     * @return {"status" : "Returns status of firm registration"}
     */
    function isRegisteredFirm(string firmName) public view returns
(bool status) {
        /// @notice check firm's registration status
        return lib.isRegisteredFirm(firmName);
    }

    /**
     *  @notice  Checks  if  an  authority  account  is  registered  to  a
given firm
     * @param firmName Name of firm
     * @param authority Address of authority account
     *   @return   {"registered"   :   "Returns   status   of   account
registration to firm"}
```

```
     */
    function isRegisteredToFirm(string firmName, address authority)
public view returns (bool registered) {
        /// @notice check if registered to firm
        return lib.isRegisteredToFirm(firmName, authority);
    }

    /**
     * @notice Gets status of authority registration
     * @param authority Address of authority account
     * @return { "registered" : "Returns true if account is a
registered authority" }
     */
    function isRegisteredAuthority(address authority) public view
returns (bool registered) {
        /// @notice check if registered authority
        return lib.isRegisteredAuthority(authority);
    }

    /**
     * @notice Sets contract which specifies fee parameters
     * @param feeContract Address of the fee contract
     * @return { "success" : "Returns true if
lib.setMasterFeeContract succeeds" }
     */
    function setMasterFeeContract(address feeContract) public
onlyOwner returns (bool success) {
        /// @notice set master fee contract
        require(lib.setMasterFeeContract(feeContract));
        return true;
      }


    modifier onlyAuthority(string firmName, address authority) {
        /// @notice throws if not an owner auuthority or not
registered to the given firm
        require(owner[authority] ||
lib.isRegisteredToFirm(firmName, authority));
        _;
    }

}
```

Conclusion TokenIOStorqge contract

No relevant issue on this contract.

TokenIOCurrencyAuthority.sol

```
pragma solidity ^0.4.24;


/*

COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

*/

import "./Ownable.sol";
import "./TokenIOStorage.sol";
import "./TokenIOLib.sol";

/*
@title TokenIO - ERC20 Compliant Smart Contract for Token, Inc.

@author Ryan Tate <ryan.michael.tate@gmail.com>, Sean Pollock
<seanpollock3344@gmail.com>

@notice Contract uses generalized storage contract,
`TokenIOStorage`, for
upgradeability of interface contract.
*/

contract TokenIOCurrencyAuthority is Ownable {

    /// @dev Set reference to TokenIOLib interface which proxies to
TokenIOStorage */
    using TokenIOLib for TokenIOLib.Data;
    TokenIOLib.Data lib;

    /**
     * @notice Constructor method for CurrencyAuthority contract
     * @param _storageContract Address of TokenIOStorage contract
     */
    constructor(address _storageContract) public {
        /**
         * @notice Set the storage contract for the interface
         * @dev This contract will be unable to use the storage
constract until
         * @dev Contract address is authorized with the storage
contract
         */
        lib.Storage = TokenIOStorage(_storageContract);

        // @dev set owner to contract initiator
        owner[msg.sender] = true;
    }

    /**
     * @notice Gets balance of sepcified account for a given currency
     * @param currency Currency symbol 'USDx'
     * @param account Sepcified account address
     * @return { "balance": "Returns account balance"}
     */
    function getTokenBalance(string currency, address account)
public view returns (uint balance) {
      return lib.getTokenBalance(currency, account);
    }

    /**
     * @notice Gets total supply of specified currency
     * @param currency Currency symbol 'USDx'
     * @return { "supply": "Returns total supply of currency"}
     */
    function getTokenSupply(string currency) public view returns
(uint supply) {
      return lib.getTokenSupply(currency);
    }

    /**
     * @notice Updates account status. false: frozen, true: un-
frozen
     * @param account Sepcified account address
     * @param isAllowed Frozen status
     * @param issuerFirm Name of the issuer firm with authority on
account holder;
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function freezeAccount(address account, bool isAllowed, string
issuerFirm) public onlyAuthority(issuerFirm, msg.sender) returns
(bool success) {
        // @notice updates account status
        // @dev !!! mutates storage state
        require(lib.setAccountStatus(account,           isAllowed,
issuerFirm));
        return true;
    }
```

```
* @notice Sets approval status of specified account
     * @param account Sepcified account address
     * @param isApproved Frozen status
     * @param issuerFirm Name of the issuer firm with authority on
account holder;
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function approveKYC(address account, bool isApproved, uint
limit, string issuerFirm) public onlyAuthority(issuerFirm,
msg.sender) returns (bool success) {
        // @notice updates kyc approval status
        // @dev !!! mutates storage state
        require(lib.setKYCApproval(account, isApproved,
issuerFirm));
        // @notice updates account statuss
        // @dev !!! mutates storage state
        require(lib.setAccountStatus(account, isApproved,
issuerFirm));
        require(lib.setAccountSpendingLimit(account, limit));
        require(lib.setAccountSpendingPeriod(account, (now +
86400)));
        return true;
    }

    /**
     * @notice Approves account and deposits specified amount of
given currency
     * @param currency Currency symbol of amount to be deposited;
     * @param account Ethereum address of account holder;
     * @param amount Deposit amount for account holder;
     * @param issuerFirm Name of the issuer firm with authority on
account holder;
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function approveKYCAndDeposit(string currency, address account,
uint amount, uint limit, string issuerFirm) public
onlyAuthority(issuerFirm, msg.sender) returns (bool success) {
        /// @notice updates kyc approval status
        /// @dev !!! mutates storage state
        require(lib.setKYCApproval(account, true, issuerFirm));
        /// @notice updates kyc approval status
        /// @dev !!! mutates storage state
        require(lib.setAccountStatus(account, true, issuerFirm));
        require(lib.deposit(currency, account, amount,
issuerFirm));
        require(lib.setAccountSpendingLimit(account, limit));
        require(lib.setAccountSpendingPeriod(account, (now +
86400)));
        return true;
    }
    /**
     * @notice Sets the spending limit for a given account
     * @param account Ethereum address of account holder;
     * @param limit Spending limit amount for account;
     * @param issuerFirm Name of the issuer firm with authority on
account holder;
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function setAccountSpendingLimit(address account, uint limit,
string issuerFirm) public onlyAuthority(issuerFirm, msg.sender)
returns (bool success) {
      require(lib.setAccountSpendingLimit(account, limit));
      return true;
    }

    /**
     * @notice Returns the periodic remaining spending amount for
an account
     * @param  account Ethereum address of account holder;
     * @return {"spendingRemaining" : "Returns the remaining
spending amount for the account"}
     */
    function getAccountSpendingRemaining(address account) public
view returns (uint spendingRemaining) {
      return lib.getAccountSpendingRemaining(account);
    }

    /**
     * @notice Return the spending limit for an account
     * @param  account Ethereum address of account holder
     * @return {"spendingLimit" : "Returns the remaining daily
spending limit of the account"}
     */
    function getAccountSpendingLimit(address account) public view
returns (uint spendingLimit) {
      return lib.getAccountSpendingLimit(account);
    }

    /**
     * @notice Set the foreign currency exchange rate to USD in basis
points
     * @dev NOTE: This value should always be relative to USD pair;
e.g. JPY/USD, GBP/USD, etc.
     * @param currency The TokenIO currency symbol (e.g. USDx, JPYx,
GBPx)
     * @param bpsRate Basis point rate of foreign currency exchange
rate to USD
     * @param issuerFirm Firm setting the foreign currency exchange
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function  setFxBpsRate(string  currency,  uint  bpsRate,  string
issuerFirm)  public  onlyAuthority(issuerFirm,  msg.sender)  returns
(bool success) {
      require(lib.setFxUSDBPSRate(currency, bpsRate));
      return true;
    }

    /**
     * @notice Return the foreign currency USD exchanged amount
     * @param currency The TokenIO currency symbol (e.g. USDx, JPYx,
GBPx)
     * @param fxAmount Amount of foreign currency to exchange into
USD
     * @return {"usdAmount" : "Returns the foreign currency amount
in USD"}
     */
    function getFxUSDAmount(string currency, uint fxAmount) public
view returns (uint usdAmount) {
      return lib.getFxUSDAmount(currency, fxAmount);
    }

    /**
     * @notice Updates to new forwarded account
     * @param originalAccount [address]
     * @param updatedAccount [address]
     * @param issuerFirm Name of the issuer firm with authority on
account holder;
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function    approveForwardedAccount(address    originalAccount,
address     updatedAccount,     string     issuerFirm)     public
onlyAuthority(issuerFirm, msg.sender) returns (bool success) {
        // @notice updatesa forwarded account
        // @dev !!! mutates storage state
        require(lib.setForwardedAccount(originalAccount,
updatedAccount));
        return true;
    }

    /**
     * @notice Issues a specified account to recipient account of a
given currency
     * @param currency [string] currency symbol
     * @param amount [uint] issuance amount
     * @param issuerFirm Name of the issuer firm with authority on
account holder;
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function deposit(string currency, address account, uint amount,
string  issuerFirm)  public  onlyAuthority(issuerFirm,  msg.sender)
returns (bool success) {
        require(lib.verifyAccount(account));
        // @notice depositing tokens to account
        // @dev !!! mutates storage state
        require(lib.deposit(currency,       account,       amount,
issuerFirm));
        return true;
    }

    /**
     * @notice Withdraws a specified amount of tokens of a given
currency
     * @param currency Currency symbol
     * @param account Ethereum address of account holder
     * @param amount Issuance amount
     * @param issuerFirm Name of the issuer firm with authority on
account holder
     * @return { "success": "Returns true if successfully called
from another contract"}
     */
    function withdraw(string currency, address account, uint amount,
string  issuerFirm)  public  onlyAuthority(issuerFirm,  msg.sender)
returns (bool success) {
        require(lib.verifyAccount(account));
        // @notice withdrawing from account
        // @dev !!! mutates storage state
        require(lib.withdraw(currency,       account,       amount,
issuerFirm));
        return true;
    }

    /**
     * @notice Ensure only authorized currency firms and authorities
can modify protected methods
     * @dev authority must be registered to an authorized firm to
use protected methods
     */
    modifier onlyAuthority(string _firmName, address _authority) {
        // @notice throws if authority account is not registred to
the given firm
        require(lib.isRegisteredToFirm(_firmName, _authority));
        _;
    }

}
```

Conclusion TokenIOCurrencyAuthority contract

No relevant issue on this contract.

TOkenIOERC20.sol

```
pragma solidity ^0.4.24;

import "./Ownable.sol";
import "./TokenIOStorage.sol";
import "./TokenIOLib.sol";



/*
    COPYRIGHT 2018 Token, Inc.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
    INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
    PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN CONNECTION
    WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

    @title ERC20 Compliant Smart Contract for Token, Inc.

    @author Ryan Tate <ryan.michael.tate@gmail.com>, Sean Pollock
<seanpollock3344@gmail.com>

    @notice Contract uses generalized storage contract,
`TokenIOStorage`, for
    upgradeability of interface contract.

    @dev In the event that the main contract becomes deprecated, the
upgraded contract
    will be set as the owner of this contract, and use this
contract's storage to
    maintain data consistency between contract.
*/



contract TokenIOERC20 is Ownable {
    //// @dev Set reference to TokenIOLib interface which proxies
to TokenIOStorage
    using TokenIOLib for TokenIOLib.Data;
    TokenIOLib.Data lib;

    /**
    * @notice Constructor method for ERC20 contract
    * @param _storageContract    address of TokenIOStorage contract
    */
    constructor(address _storageContract) public {
        //// @dev Set the storage contract for the interface
        //// @dev This contract will be unable to use the storage
constract until
        //// @dev contract address is authorized with the storage
contract
        //// @dev Once authorized, Use the `setParams` method to set
storage values
        lib.Storage = TokenIOStorage(_storageContract);

        //// @dev set owner to contract initiator
        owner[msg.sender] = true;
    }


    /**
     @notice Sets erc20 globals and fee paramters
     @param _name Full token name  'USD by token.io'
     @param _symbol Symbol name 'USDx'
     @param _tla Three letter abbreviation 'USD'
     @param _version Release version 'v0.0.1'
     @param _decimals Decimal precision
     @param _feeContract Address of fee contract
     @return { "success" : "Returns true if successfully called from
another contract"}
     */
     function setParams(
        string _name,
        string _symbol,
        string _tla,
        string _version,
        uint _decimals,
        address _feeContract,
        uint _fxUSDBPSRate
    ) onlyOwner public returns (bool success) {
        require(lib.setTokenName(_name));
        require(lib.setTokenSymbol(_symbol));
        require(lib.setTokenTLA(_tla));
        require(lib.setTokenVersion(_version));
        require(lib.setTokenDecimals(_symbol, _decimals));
        require(lib.setFeeContract(_feeContract));
        require(lib.setFxUSDBPSRate(_symbol, _fxUSDBPSRate));
        return true;
    }

    /**
     * @notice Gets name of token
     * @return {"_name" : "Returns name of token"}
     */
    function name() public view returns (string _name) {
        return lib.getTokenName(address(this));
    }
    /**
     * @notice Gets symbol of token
     * @return {"_symbol" : "Returns symbol of token"}
     */
    function symbol() public view returns (string _symbol) {
        return lib.getTokenSymbol(address(this));
    }

    /**
     * @notice Gets three-letter-abbreviation of token
     * @return {"_tla" : "Returns three-letter-abbreviation of
token"}
     */
    function tla() public view returns (string _tla) {
        return lib.getTokenTLA(address(this));
    }

    /**
     * @notice Gets version of token
     * @return {"_version" : "Returns version of token"}
     */
    function version() public view returns (string _version) {
        return lib.getTokenVersion(address(this));
    }

    /**
     * @notice Gets decimals of token
     * @return {"_decimals" : "Returns number of decimals"}
     */
    function decimals() public view returns (uint _decimals) {
        return
lib.getTokenDecimals(lib.getTokenSymbol(address(this)));
    }

    /**
     * @notice Gets total supply of token
     * @return {"supply" : "Returns current total supply of token"}
     */
    function totalSupply() public view returns (uint supply) {
      return lib.getTokenSupply(lib.getTokenSymbol(address(this)));
    }

    /**
     * @notice Gets allowance that spender has with approver
     * @param account Address of approver
     * @param spender Address of spender
     * @return {"amount" : "Returns allowance of given account and
spender"}
     */
    function allowance(address account, address spender) public view
returns (uint amount) {
      return
lib.getTokenAllowance(lib.getTokenSymbol(address(this)),   account,
spender);
    }

    /**
     * @notice Gets balance of account
     * @param account Address for balance lookup
     * @return {"balance" : "Returns balance amount"}
     */
    function balanceOf(address account) public view returns (uint
balance) {
      return lib.getTokenBalance(lib.getTokenSymbol(address(this)),
account);
    }

    /**
     * @notice Gets fee parameters
     * @return {
     "bps":"Fee amount as a mesuare of basis points",
     "min":"Minimum fee amount",
     "max":"Maximum fee amount",
     "flat":"Flat fee amount",
     "contract":"Address of fee contract"
     }
     */
    function getFeeParams() public view returns (uint bps, uint min,
uint max, uint flat, address feeAccount) {
        return (
            lib.getFeeBPS(lib.getFeeContract(address(this))),
            lib.getFeeMin(lib.getFeeContract(address(this))),
            lib.getFeeMax(lib.getFeeContract(address(this))),
            lib.getFeeFlat(lib.getFeeContract(address(this))),
            lib.getFeeContract(address(this))
        );
    }

    /**
     * @notice Calculates fee of a given transfer amount
     * @param amount Amount to calculcate fee value
     * @return {"fees": "Returns the calculated transaction fees
based on the fee contract parameters"}
     */
    function calculateFees(uint amount) public view returns (uint
fees) {
      return   lib.calculateFees(lib.getFeeContract(address(this)),
amount);
    }

    /**
     * @notice transfers 'amount' from msg.sender to a receiving
account 'to'
     * @param to Receiving address
     * @param amount Transfer amount
     * @return {"success" : "Returns true if transfer succeeds"}
     */
    function transfer(address to, uint amount) public notDeprecated
returns (bool success) {
      /// @notice check that receiving account is verified
      require(lib.verifyAccounts(msg.sender, to));
      /// @notice send transfer through library
      /// @dev !!! mutates storage state
      require(lib.transfer(lib.getTokenSymbol(address(this)),   to,
amount, "0x0"));
      return true;
    }

    /**
     *  @notice  spender  transfers  from  approvers  account  to  the
reciving account
     * @param from Approver's address
     * @param to Receiving address
     * @param amount Transfer amount
     * @return {"success" : "Returns true if transferFrom succeeds"}
     */
    function  transferFrom(address  from,  address  to,  uint  amount)
public notDeprecated returns (bool success) {
        // @ notice checks from and to accounts are verified
        require(lib.verifyAccounts(from, to));
        /// @notice sends transferFrom through library
        /// @dev !!! mutates storage state

require(lib.transferFrom(lib.getTokenSymbol(address(this)),   from,
to, amount, "0x0"));
        return true;
    }

    /**
     * @notice approves spender a given amount
     * @param spender Spender's address
     * @param amount Allowance amount
     * @return {"success" : "Returns true if approve succeeds"}
     */
    function   approve(address   spender,   uint   amount)   public
notDeprecated returns (bool success) {
        ///  @notice  checks  approver  and  spenders  accounts  are
verified
        require(lib.verifyAccounts(msg.sender, spender));
        /// @notice sends approve through library
        /// @dev !!! mtuates storage states
        require(lib.approveAllowance(spender, amount));
        return true;
    }

    /**
     * @notice gets currency status of contract
     * @return {"deprecated" : "Returns true if deprecated, false
otherwise"}
     */
    function deprecateInterface() public onlyOwner returns (bool
deprecated) {
      require(lib.setDeprecatedContract(address(this)));
      return true;
    }

    modifier notDeprecated() {
      /// @notice throws if contract is deprecated
      require(!lib.isContractDeprecated(address(this)));
      _;
    }

}
```

Conclusion TOkenIOERC20 contract

```
No relevant issue on this contract.
```

TokenIOFeeContract.sol

```
pragma solidity ^0.4.24;

import "./Ownable.sol";
import "./TokenIOStorage.sol";
import "./TokenIOLib.sol";


/*
COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


@title Standard Fee Contract for Token, Inc. Smart Money System

@author Ryan Tate <ryan.michael.tate@gmail.com>, Sean Pollock
<seanpollock3344@gmail.com>

@notice Contract uses generalized storage contract,
`TokenIOStorage`, for
```

```
upgradeability of interface contract.

@dev In the event that the main contract becomes deprecated, the
upgraded contract
will be set as the owner of this contract, and use this contract's
storage to
maintain data consistency between contract.

*/



contract TokenIOFeeContract is Ownable {

     /// @dev Set reference to TokenIOLib interface which proxies to
TokenIOStorage
     using TokenIOLib for TokenIOLib.Data;
     TokenIOLib.Data lib;



     /**
     * @notice Constructor method for ERC20 contract
     * @param _storageContract    address of TokenIOStorage contract
     */
     constructor(address _storageContract) public {
               /// @dev Set the storage contract for the interface
               /// @dev NOTE: This contract will be unable to use
the storage constract until
               /// @dev contract address is authorized with the
storage contract
               /// @dev Once authorized, Use the `setParams` method
to set storage values
               lib.Storage = TokenIOStorage(_storageContract);

               /// @dev set owner to contract initiator
               owner[msg.sender] = true;
     }

     /**
      * @notice Set Fee Parameters for Fee Contract
      * @dev The min, max, flat transaction fees should be relative
to decimal precision
      * @param feeBps Basis points transaction fee
      * @param feeMin Minimum transaction fees
      * @param feeMax Maximum transaction fee
      * @param feeFlat Flat transaction fee
      * returns {"success" : "Returns true if successfully called
from another contract"}
      */
     function setFeeParams(uint feeBps, uint feeMin, uint feeMax,
uint feeFlat) public onlyOwner returns (bool success) {
          require(lib.setFeeBPS(feeBps));
          require(lib.setFeeMin(feeMin));
          require(lib.setFeeMax(feeMax));
          require(lib.setFeeFlat(feeFlat));
          return true;
     }

     /**
          * @notice Gets fee parameters
          * @return {
          "bps":"Returns    the    basis    points    fee    of    the
TokenIOFeeContract",
          "min":"Returns the min fee of the TokenIOFeeContract",
          "max":"Returns the max fee of the TokenIOFeeContract",
          "flat":"Returns the flat fee of the TokenIOFeeContract",
          "feeContract": "Address of this contract"
     }
     */
     function getFeeParams() public view returns (uint bps, uint
min, uint max, uint flat, address feeContract) {
               return (
                         lib.getFeeBPS(address(this)),
                         lib.getFeeMin(address(this)),
                         lib.getFeeMax(address(this)),
                         lib.getFeeFlat(address(this)),
                         address(this)
               );
     }

     /**
      * @notice Returns balance of this contract associated with
currency symbol.
      * @param  currency Currency symbol of the token (e.g. USDx,
JYPx, GBPx)
      *  @return  {"balance":  "Balance  of  TokenIO  TSM  currency
account"}
      */
     function   getTokenBalance(string   currency)   public   view
returns(uint balance) {
          return lib.getTokenBalance(currency, address(this));
     }

     /** @notice Calculates fee of a given transfer amount
      * @param amount transfer amount
      * @return { "fees": "Returns the fees associated with this
contract"}
      */
     function calculateFees(uint amount) public view returns (uint
fees) {
          return lib.calculateFees(address(this), amount);
     }



     /**
      *  @notice  Transfer  collected  fees  to  another  account;
onlyOwner
```

```
      * @param  currency Currency symbol of the token (e.g. USDx,
JYPx, GBPx)
      * @param  to          Ethereum address of account to send
token amount to
      * @param  amount     Amount of tokens to transfer
      * @param  data            Arbitrary bytes data message to
include in transfer
      * @return {"success": "Returns ture if successfully called
from another contract"}
      */
     function transferCollectedFees(string currency, address to,
uint amount, bytes data) public onlyOwner returns (bool success) {
          require(lib.verifyAccount(to));
          require(lib.forceTransfer(currency, address(this), to,
amount, data));
          return true;
     }


}
```

Conclusion TokenIOFeeContract contract

No relevant issue on this contract.

TokenIOFX.sol

```
pragma solidity ^0.4.24;
/* pragma experimental ABIEncoderV2; */


import "./Ownable.sol";
import "./TokenIOLib.sol";
import "./TokenIOStorage.sol";


/**
COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


@title Currency FX Contract for Token, Inc. Smart Money System
```

```
@author  Ryan  Tate  <ryan.michael.tate@gmail.com>,  Sean  Pollock
<seanpollock3344@gmail.com>

@notice   Contract   uses   generalized   storage   contract,
`TokenIOStorage`, for
upgradeability of interface contract.

@dev In the event that the main contract becomes deprecated, the
upgraded contract
will be set as the owner of this contract, and use this contract's
storage to
maintain data consistency between contract.

*/

contract TokenIOFX is Ownable {

  /// @dev Set reference to TokenIOLib interface which proxies to
TokenIOStorage
  using TokenIOLib for TokenIOLib.Data;
  TokenIOLib.Data lib;


  /**
     * @notice Constructor method for ERC20 contract
     * @param _storageContract Address of TokenIOStorage contract
     */
     constructor(address _storageContract) public {
               /// @dev Set the storage contract for the interface
               /// @dev NOTE: This contract will be unable to use
the storage constract until
               ///  @dev  contract  address  is  authorized  with  the
storage contract
               /// @dev Once authorized, Use the `setParams` method
to set storage values
               lib.Storage = TokenIOStorage(_storageContract);

               /// @dev set owner to contract initiator
               owner[msg.sender] = true;
     }


  /**
    * @notice Accepts a signed fx request to swap currency pairs at
a given amount;
    * @dev This method can be called directly between peers.
    *  @param  requester address Requester is the orginator of the
offer and must
    * match the signature of the payload submitted by the fulfiller
    * @param  symbolA    Symbol of the currency desired
    * @param  symbolB    Symbol of the currency offered
    * @param  valueA     Amount of the currency desired
```

```
   * @param  valueB     Amount of the currency offered
   * @param  sigV        Ethereum secp256k1 signature V value; used
by ecrecover()
   * @param  sigR        Ethereum secp256k1 signature R value; used
by ecrecover()
   * @param  sigS        Ethereum secp256k1 signature S value; used
by ecrecover()
   * @param  expiration Expiration of the offer; Offer is good until
expired
   * @return {"success" : "Returns true if successfully called from
another contract"}
   */
  function swap(
    address requester,
    string symbolA,
    string symbolB,
    uint valueA,
    uint valueB,
    uint8 sigV,
    bytes32 sigR,
    bytes32 sigS,
    uint expiration
  ) public returns (bool success) {
    require(lib.execSwap(requester,  symbolA,  symbolB,  valueA,
valueB, sigV, sigR, sigS, expiration));
    return true;
  }

}
```

Conclusion TokenIOFX contract

No relevant issue on this contract.

TokenIONameSpace.sol

```
pragma solidity ^0.4.24;


/**
COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
@title Token Name Space Interface for Token, Inc. Smart Money System

@author  Ryan  Tate  <ryan.michael.tate@gmail.com>,  Sean  Pollock
<seanpollock3344@gmail.com>

@notice    Contract    uses    generalized    storage    contract,
`TokenIOStorage`, for
upgradeability of interface contract.

@dev In the event that the main contract becomes deprecated, the
upgraded contract
will be set as the owner of this contract, and use this contract's
storage to
maintain data consistency between contract.

*/

import "./Ownable.sol";
import "./TokenIOStorage.sol";
import "./TokenIOLib.sol";

contract TokenIONameSpace is Ownable {

    /// @dev Set reference to TokenIOLib interface which proxies to
TokenIOStorage
    using TokenIOLib for TokenIOLib.Data;
    TokenIOLib.Data lib;

    /**
     * @notice Constructor method for ERC20 contract
     * @param _storageContract    address of TokenIOStorage contract
     */
     constructor(address _storageContract) public {
               /// @dev Set the storage contract for the interface
               /// @dev NOTE: This contract will be unable to use
the storage constract until
               ///  @dev  contract  address  is  authorized  with  the
storage contract
               /// @dev Once authorized, Use the `setParams` method
to set storage values
               lib.Storage = TokenIOStorage(_storageContract);

               /// @dev set owner to contract initiator
               owner[msg.sender] = true;
     }

    /**
     * @notice Returns the address of the contract associated with
the currency symbol
     * @notice This method may be deprecated or refactored to allow
for multiple interfaces
     * @param  currency string Currency symbol of the token (e.g.
USDx, JYPx, GBPx)
```

```
     *  @return  {"contractAddress":  "Returns  the  token  contract
address associated with the currency"}
     */
    function getTokenNameSpace(string currency) public view returns
(address contractAddress) {
        return lib.getTokenNameSpace(currency);
    }

}
```

Conclusion TokenIONameSpace contract

No relevant issue on this contract.

TokenIOLib.sol

```
pragma solidity ^0.4.24;
pragma experimental ABIEncoderV2;

/**
COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

@title TokenIOLib

@author  Ryan  Tate  <ryan.michael.tate@gmail.com>,  Sean  Pollock
<seanpollock3344@gmail.com>

@notice This library proxies the TokenIOStorage contract for the
interface contract,
allowing the library and the interfaces to remain stateless, and
share a universally
available storage contract between interfaces.
*/
import "./SafeMath.sol";
import "./TokenIOStorage.sol";
library TokenIOLib {
  /// @dev all math operating are using SafeMath methods to check
for overflow/underflows
 using SafeMath for uint;
// @dev the Data struct uses the Storage contract for stateful
setters
```

```
struct Data {
    TokenIOStorage Storage;
  }

  event LogApproval(address indexed owner, address indexed spender,
uint amount);
  event LogDeposit(string currency, address indexed account, uint
amount, string issuerFirm);
  event LogWithdraw(string currency, address indexed account, uint
amount, string issuerFirm);
  event LogTransfer(string currency, address indexed from, address
indexed to, uint amount, bytes data);
  event LogKYCApproval(address indexed account, bool status, string
issuerFirm);
  event LogAccountStatus(address indexed account, bool status,
string issuerFirm);
  event LogFxSwap(string tokenASymbol,string tokenBSymbol,uint
tokenAValue,uint tokenBValue, uint expiration, bytes32
transactionHash);
  event LogAccountForward(address indexed originalAccount, address
indexed forwardedAccount);
  event LogNewAuthority(address indexed authority, string
issuerFirm);

  /**
   * @notice Set the token name for Token interfaces
   * @dev This method must be set by the token interface's
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param tokenName Name of the token contract
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setTokenName(Data storage self, string tokenName)
internal returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('token.name',
address(this)));
    self.Storage.setString(id, tokenName);
    return true;
  }


  /**
* @notice Set the token symbol for Token interfaces
   * @dev This method must be set by the token interface's
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
* @param tokenSymbol Symbol of the token contract
   * @return {"success" : "Returns true when successfully called
from another contract"}
```

```
 */
  function  setTokenSymbol(Data  storage  self,  string  tokenSymbol) 
internal returns (bool success) {
    bytes32    id    =    keccak256(abi.encodePacked('token.symbol', 
address(this)));
    self.Storage.setString(id, tokenSymbol);
    return true;
  }

  /**
   * @notice Set the token three letter abreviation (TLA) for Token 
interfaces
   *  @dev  This  method  must  be  set  by  the  token  interface's 
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param tokenTLA TLA of the token contract
   *  @return  {"success"  :  "Returns  true  when  successfully  called 
from another contract"}
   */
  function setTokenTLA(Data storage self, string tokenTLA) internal 
returns  (bool success)  {
    bytes32     id     =     keccak256(abi.encodePacked('token.tla', 
address(this)));
    self.Storage.setString(id, tokenTLA);
    return true;
  }

  /**
   * @notice Set the token version for Token interfaces
   *  @dev  This  method  must  be  set  by  the  token  interface's 
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param tokenVersion Semantic (vMAJOR.MINOR.PATCH | e.g. v0.1.0) 
version of the token contract
   *  @return  {"success"  :  "Returns  true  when  successfully  called 
from another contract"}
   */
  function  setTokenVersion(Data  storage  self,  string  tokenVersion) 
internal returns  (bool success) {
    bytes32    id    =    keccak256(abi.encodePacked('token.version', 
address(this)));
    self.Storage.setString(id, tokenVersion);
    return true;
  }

  /**
   * @notice Set the token decimals for Token interfaces

   *  @dev  This  method  must  be  set  by  the  token  interface's 
setParams() method
```

```
 * @dev | This method has an `internal` view
   * @dev This method is not set to the address of the contract,
rather is maped to currency
   * @dev To derive decimal value, divide amount by 10^decimal
representation (e.g. 10132 / 10**2 == 101.32)
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param tokenDecimals Decimal representation of the token
contract unit amount
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setTokenDecimals(Data storage self, string currency, uint
tokenDecimals) internal returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('token.decimals',
currency));
    self.Storage.setUint(id, tokenDecimals);
    return true;
  }

  /**
   * @notice Set basis point fee for contract interface
   * @dev Transaction fees can be set by the TokenIOFeeContract
   * @dev Fees vary by contract interface specified `feeContract`
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param feeBPS Basis points fee for interface contract
transactions
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setFeeBPS(Data storage self, uint feeBPS) internal
returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('fee.bps',
address(this)));
    self.Storage.setUint(id, feeBPS);
    return true;
  }

  /**
   * @notice Set minimum fee for contract interface
   * @dev Transaction fees can be set by the TokenIOFeeContract
   * @dev Fees vary by contract interface specified `feeContract`
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param feeMin Minimum fee for interface contract transactions
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setFeeMin(Data storage self, uint feeMin) internal
returns (bool success) {
```

```
bytes32 id = keccak256(abi.encodePacked('fee.min', address(this)));
    self.Storage.setUint(id, feeMin);
    return true;
  }

  /**
   * @notice Set maximum fee for contract interface
   * @dev Transaction fees can be set by the TokenIOFeeContract
   * @dev Fees vary by contract interface specified `feeContract`
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param feeMax Maximum fee for interface contract transactions
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setFeeMax(Data storage self, uint feeMax) internal
returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('fee.max',
address(this)));
    self.Storage.setUint(id, feeMax);
    return true;
  }

  /**
   * @notice Set flat fee for contract interface
   * @dev Transaction fees can be set by the TokenIOFeeContract
   * @dev Fees vary by contract interface specified `feeContract`
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param feeFlat Flat fee for interface contract transactions
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setFeeFlat(Data storage self, uint feeFlat) internal
returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('fee.flat',
address(this)));
    self.Storage.setUint(id, feeFlat);
    return true;
  }

  /**
   * @notice Set fee contract for a contract interface
   * @dev feeContract must be a TokenIOFeeContract storage approved
contract
   * @dev Fees vary by contract interface specified `feeContract`
   * @dev | This method has an `internal` view
   * @dev | This must be called directly from the interface contract
   * @param self Internal storage proxying TokenIOStorage contract
   * @param feeContract Set the fee contract for `this` contract
address interface
```

```
 * @return {"success" : "Returns true when successfully called from
another contract"}
   */
  function setFeeContract(Data storage self, address feeContract)
internal returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('fee.account',
address(this)));
    self.Storage.setAddress(id, feeContract);
    return true;
  }

  /**
   * @notice Set contract interface associated with a given TokenIO
currency symbol (e.g. USDx)
   * @dev | This should only be called once from a token interface
contract;
   * @dev | This method has an `internal` view
   * @dev | This method is experimental and may be
deprecated/refactored
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setTokenNameSpace(Data storage self, string currency)
internal returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('token.namespace',
currency));
    self.Storage.setAddress(id, address(this));
    return true;
  }

  /**
   * @notice Set the KYC approval status (true/false) for a given
account
   * @dev | This method has an `internal` view
   * @dev | Every account must be KYC'd to be able to use transfer()
& transferFrom() methods
   * @dev | To gain approval for an account, register at
https://tsm.token.io/sign-up
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of account holder
   * @param isApproved Boolean (true/false) KYC approval status for
a given account
   * @param issuerFirm Firm name for issuing KYC approval
   * @return {"success" : "Returns true when successfully called
from another contract"}
   */
  function setKYCApproval(Data storage self, address account, bool
isApproved, string issuerFirm) internal returns (bool success) {
 bytes32     id     =     keccak256(abi.encodePacked('account.kyc',
getForwardedAccount(self, account)));
      self.Storage.setBool(id, isApproved);

      /// @dev NOTE: Issuer is logged for setting account KYC status
      emit LogKYCApproval(account, isApproved, issuerFirm);
      return true;
  }

  /**
   * @notice Set the global approval status (true/false) for a given
account
   * @dev | This method has an `internal` view
   *  @dev  |  Every  account  must  be  permitted  to  be  able  to  use
transfer() & transferFrom() methods
   *   @dev   |   To   gain   approval   for   an   account,   register   at
https://tsm.token.io/sign-up
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of account holder
   * @param isAllowed Boolean (true/false) global status for a given
account
   * @param issuerFirm Firm name for issuing approval
   *  @return  {"success"  :  "Returns  true  when  successfully  called
from another contract"}
   */
  function setAccountStatus(Data storage self, address account, bool
isAllowed, string issuerFirm) internal returns (bool success) {
    bytes32   id   =   keccak256(abi.encodePacked('account.allowed',
getForwardedAccount(self, account)));
    self.Storage.setBool(id, isAllowed);

    /// @dev NOTE: Issuer is logged for setting account status
    emit LogAccountStatus(account, isAllowed, issuerFirm);
    return true;
  }


  /**
   * @notice Set a forwarded address for an account.
   * @dev | This method has an `internal` view
   * @dev | Forwarded accounts must be set by an authority in case
of account recovery;
   *  @dev  |  Additionally,  the  original  owner  can  set  a  forwarded
account (e.g. add a new device, spouse, dependent, etc)
   *  @dev  |  All  transactions  will  be  logged  under  the  same  KYC
information as the original account holder;
   * @param self Internal storage proxying TokenIOStorage contract
   * @param originalAccount Original registered Ethereum address of
the account holder
   *  @param  forwardedAccount  Forwarded  Ethereum  address  of  the
account holder
```

```
 * @return {"success" : "Returns true when successfully called from
another contract"}
   */
  function   setForwardedAccount(Data   storage   self,   address
originalAccount, address forwardedAccount) internal returns (bool
success) {
    bytes32   id   =   keccak256(abi.encodePacked('master.account',
forwardedAccount));
    self.Storage.setAddress(id, originalAccount);
    return true;
  }

  /**
   * @notice Get the original address for a forwarded account
   * @dev | This method has an `internal` view
   *  @dev  |  Will  return  the  registered  account  for  the  given
forwarded account
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of account holder
   *  @return  {  "registeredAccount"  :  "Will  return  the  original
account of a forwarded account or the account itself if no account
found"}
   */
  function getForwardedAccount(Data storage self, address account)
internal view returns (address registeredAccount) {
    bytes32   id   =   keccak256(abi.encodePacked('master.account',
account));
    address originalAccount = self.Storage.getAddress(id);
    if (originalAccount != 0x0) {
      return originalAccount;
    } else {
      return account;
    }
  }

  /**
   * @notice Get KYC approval status for the account holder
   * @dev | This method has an `internal` view
   * @dev | All forwarded accounts will use the original account's
status
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of account holder
   *  @return  {  "status"  :  "Returns  the  KYC  approval  status  for  an
account holder" }
   */
  function   getKYCApproval(Data   storage   self,   address   account)
internal view returns (bool status) {
      bytes32   id   =   keccak256(abi.encodePacked('account.kyc',
getForwardedAccount(self, account)));
      return self.Storage.getBool(id);
  }
```

```
/**
   * @notice Get global approval status for the account holder
   * @dev | This method has an `internal` view
   * @dev | All forwarded accounts will use the original account's
status
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of account holder
   * @return { "status" : "Returns the global approval status for
an account holder" }
   */
  function getAccountStatus(Data storage self, address account)
internal view returns (bool status) {
    bytes32 id = keccak256(abi.encodePacked('account.allowed',
getForwardedAccount(self, account)));
    return self.Storage.getBool(id);
  }

  /**
   * @notice Get the contract interface address associated with token
symbol
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @return { "contractAddress" : "Returns the contract interface
address for a symbol" }
   */
  function getTokenNameSpace(Data storage self, string currency)
internal view returns (address contractAddress) {
    bytes32 id = keccak256(abi.encodePacked('token.namespace',
currency));
    return self.Storage.getAddress(id);
  }

  /**
   * @notice Get the token name for Token interfaces
   * @dev This method must be set by the token interface's
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the queryable
interface
   * @return {"tokenName" : "Name of the token contract"}
   */
  function getTokenName(Data storage self, address contractAddress)
internal view returns (string tokenName) {
    bytes32 id = keccak256(abi.encodePacked('token.name',
contractAddress));
    return self.Storage.getString(id);
  }

  /**
   * @notice Get the token symbol for Token interfaces
* @dev This method must be set by the token interface's setParams()
method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   *  @param  contractAddress  Contract  address  of  the  queryable
interface
   * @return {"tokenSymbol" : "Symbol of the token contract"}
   */
  function     getTokenSymbol(Data     storage     self,     address
contractAddress) internal view returns (string tokenSymbol) {
    bytes32    id    =    keccak256(abi.encodePacked('token.symbol',
contractAddress));
    return self.Storage.getString(id);
  }

  /**
   * @notice Get the token Three letter abbreviation (TLA) for Token
interfaces
   *  @dev  This  method  must  be  set  by  the  token  interface's
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   *  @param  contractAddress  Contract  address  of  the  queryable
interface
   * @return {"tokenTLA" : "TLA of the token contract"}
   */
  function getTokenTLA(Data storage self, address contractAddress)
internal view returns (string tokenTLA) {
    bytes32     id     =     keccak256(abi.encodePacked('token.tla',
contractAddress));
    return self.Storage.getString(id);
  }

  /**
   * @notice Get the token version for Token interfaces
   *  @dev  This  method  must  be  set  by  the  token  interface's
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   *  @param  contractAddress  Contract  address  of  the  queryable
interface
   *  @return  {"tokenVersion"  :  "Semantic  version  of  the  token
contract"}
   */
  function     getTokenVersion(Data     storage     self,     address
contractAddress) internal view returns (string) {
    bytes32    id    =    keccak256(abi.encodePacked('token.version',
contractAddress));
    return self.Storage.getString(id);
  }

  /**
```

```
* @notice Get the token decimals for Token interfaces
   * @dev This method must be set by the token interface's
setParams() method
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @return {"tokenDecimals" : "Decimals of the token contract"}
   */
  function getTokenDecimals(Data storage self, string currency)
internal view returns (uint tokenDecimals) {
    bytes32 id = keccak256(abi.encodePacked('token.decimals',
currency));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Get the basis points fee of the contract address;
typically TokenIOFeeContract
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the queryable
interface
   * @return { "feeBps" : "Returns the basis points fees associated
with the contract address"}
   */
  function getFeeBPS(Data storage self, address contractAddress)
internal view returns (uint feeBps) {
    bytes32 id = keccak256(abi.encodePacked('fee.bps',
contractAddress));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Get the minimum fee of the contract address; typically
TokenIOFeeContract
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the queryable
interface
   * @return { "feeMin" : "Returns the minimum fees associated with
the contract address"}
   */
  function getFeeMin(Data storage self, address contractAddress)
internal view returns (uint feeMin) {
    bytes32 id = keccak256(abi.encodePacked('fee.min',
contractAddress));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Get the maximum fee of the contract address; typically
TokenIOFeeContract
```

```
* @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the queryable
interface
   * @return { "feeMax" : "Returns the maximum fees associated with
the contract address"}
   */
  function getFeeMax(Data storage self, address contractAddress)
internal view returns (uint feeMax) {
    bytes32 id = keccak256(abi.encodePacked('fee.max',
contractAddress));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Get the flat fee of the contract address; typically
TokenIOFeeContract
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the queryable
interface
   * @return { "feeFlat" : "Returns the flat fees associated with
the contract address"}
   */
  function getFeeFlat(Data storage self, address contractAddress)
internal view returns (uint feeFlat) {
    bytes32 id = keccak256(abi.encodePacked('fee.flat',
contractAddress));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Set the master fee contract used as the default fee
contract when none is provided
   * @dev | This method has an `internal` view
   * @dev | This value is set in the TokenIOAuthority contract
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the queryable
interface
   * @return { "success" : "Returns true when successfully called
from another contract"}
   */
  function setMasterFeeContract(Data storage self, address
contractAddress) internal returns (bool success) {
    bytes32 id =
keccak256(abi.encodePacked('fee.contract.master'));
    require(self.Storage.setAddress(id, contractAddress));
    return true;
  }

  /**
```

```
* @notice Get the master fee contract set via the TokenIOAuthority
contract
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @return { "masterFeeContract" : "Returns the master fee contract
set for TSM."}
   */
  function getMasterFeeContract(Data storage self) internal view
returns (address masterFeeContract) {
    bytes32                           id                           =
keccak256(abi.encodePacked('fee.contract.master'));
    return self.Storage.getAddress(id);
  }

  /**
   * @notice Get the fee contract set for a contract interface
   * @dev | This method has an `internal` view
   * @dev | Custom fee pricing can be set by assigning a fee contract
to transactional contract interfaces
   * @dev | If a fee contract has not been set by an interface
contract, then the master fee contract will be returned
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the queryable
interface
   * @return { "feeContract" : "Returns the fee contract associated
with a contract interface"}
   */
  function   getFeeContract(Data   storage   self,   address
contractAddress) internal view returns (address feeContract) {
    bytes32   id   =   keccak256(abi.encodePacked('fee.account',
contractAddress));

    address feeAccount = self.Storage.getAddress(id);
    if (feeAccount == 0x0) {
      return getMasterFeeContract(self);
    } else {
      return feeAccount;
    }
  }

  /**
   * @notice Get the token supply for a given TokenIO TSM currency
symbol (e.g. USDx)
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @return { "supply" : "Returns the token supply of the given
currency"}
   */
  function  getTokenSupply(Data  storage  self,  string  currency)
internal view returns (uint supply) {
```

```
bytes32 id = keccak256(abi.encodePacked('token.supply', currency));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Get the token spender allowance for a given account
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of account holder
   * @param spender Ethereum address of spender
   * @return { "allowance" : "Returns the allowance of a given
spender for a given account"}
   */
  function getTokenAllowance(Data storage self, string currency,
address account, address spender) internal view returns (uint
allowance) {
    bytes32 id = keccak256(abi.encodePacked('token.allowance',
currency, getForwardedAccount(self, account),
getForwardedAccount(self, spender)));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Get the token balance for a given account
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param account Ethereum address of account holder
   * @return { "balance" : "Return the balance of a given account
for a specified currency"}
   */
  function getTokenBalance(Data storage self, string currency,
address account) internal view returns (uint balance) {
    bytes32 id = keccak256(abi.encodePacked('token.balance',
currency, getForwardedAccount(self, account)));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Get the frozen token balance for a given account
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param account Ethereum address of account holder
   * @return { "frozenBalance" : "Return the frozen balance of a
given account for a specified currency"}
   */
  function getTokenFrozenBalance(Data storage self, string currency,
address account) internal view returns (uint frozenBalance) {
    bytes32 id = keccak256(abi.encodePacked('token.frozen',
currency, getForwardedAccount(self, account)));
    return self.Storage.getUint(id);
```

```
}

  /**
   * @notice Set the frozen token balance for a given account
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param account Ethereum address of account holder
   * @param amount Amount of tokens to freeze for account
   * @return { "success" : "Return true if successfully called from
another contract"}
   */
  function setTokenFrozenBalance(Data storage self, string currency,
address account, uint amount) internal view returns (bool success)
{
    bytes32   id   =   keccak256(abi.encodePacked('token.frozen',
currency, getForwardedAccount(self, account)));
    require(self.Storage.setUint(id, amount));
    return true;
  }

  /**
   * @notice Set the frozen token balance for a given account
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Contract address of the fee contract
   * @param amount Transaction value
   * @return { "calculatedFees" : "Return the calculated transaction
fees for a given amount and fee contract" }
   */
  function calculateFees(Data storage self, address contractAddress,
uint amount) internal view returns (uint calculatedFees) {

    uint                          maxFee                          =
self.Storage.getUint(keccak256(abi.encodePacked('fee.max',
contractAddress)));
    uint                          minFee                          =
self.Storage.getUint(keccak256(abi.encodePacked('fee.min',
contractAddress)));
    uint                          bpsFee                          =
self.Storage.getUint(keccak256(abi.encodePacked('fee.bps',
contractAddress)));
    uint                          flatFee                         =
self.Storage.getUint(keccak256(abi.encodePacked('fee.flat',
contractAddress)));
    uint fees = ((amount.mul(bpsFee)).div(10000)).add(flatFee);

    if (fees > maxFee) {
      return maxFee;
    } else if (fees < minFee) {
      return minFee;
    } else {
```

```
return fees;
    }
  }

  /**
   * @notice Verified KYC and global status for two accounts and
return true or throw if either account is not verified
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param accountA Ethereum address of first account holder to
verify
   * @param accountB Ethereum address of second account holder to
verify
   * @return { "verified" : "Returns true if both accounts are
successfully verified" }
   */
  function verifyAccounts(Data storage self, address accountA,
address accountB) internal returns (bool verified) {
    require(verifyAccount(self, accountA));
    require(verifyAccount(self, accountB));
    return true;
  }

  /**
   * @notice Verified KYC and global status for a single account and
return true or throw if account is not verified
   * @dev | This method has an `internal` view
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of account holder to verify
   * @return { "verified" : "Returns true if account is successfully
verified" }
   */
  function verifyAccount(Data storage self, address account)
internal returns (bool verified) {
    require(getKYCApproval(self, account));
    require(getAccountStatus(self, account));
    return true;
  }


  /**
   * @notice Transfer an amount of currency token from msg.sender
account to another specified account
   * @dev This function is called by an interface that is accessible
directly to the account holder
   * @dev | This method has an `internal` view
   * @dev | This method uses `forceTransfer()` low-level api
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param to Ethereum address of account to send currency amount
to
   * @param amount Value of currency to transfer
```

```
 * @param data Arbitrary bytes data to include with the transaction
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function transfer(Data storage self, string currency, address to,
uint amount, bytes data) internal returns (bool success) {
    require(address(to) != 0x0);

    address feeContract = getFeeContract(self, address(this));
    uint fees = calculateFees(self, feeContract, amount);

    require(setAccountSpendingAmount(self,              msg.sender,
getFxUSDAmount(self, currency, amount)));
    require(forceTransfer(self, currency, msg.sender, to, amount,
data));
    /// @dev "0x547846656573" == "TxFees"
    require(forceTransfer(self, currency, msg.sender, feeContract,
fees, "0x547846656573"));

    return true;
  }

  /**
   * @notice Transfer an amount of currency token from account to
another specified account via an approved spender account
   * @dev This function is called by an interface that is accessible
directly to the account spender
   * @dev | This method has an `internal` view
   * @dev | Transactions will fail if the spending amount exceeds
the daily limit
   * @dev | This method uses `forceTransfer()` low-level api
   * @dev | This method implements ERC20 transferFrom() method with
approved spender behavior
   * @dev | msg.sender == spender; `updateAllowance()` reduces
approved limit for account spender
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param from Ethereum address of account to send currency amount
from
   * @param to Ethereum address of account to send currency amount
to
   * @param amount Value of currency to transfer
   * @param data Arbitrary bytes data to include with the transaction
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function transferFrom(Data storage self, string currency, address
from, address to, uint amount, bytes data) internal returns (bool
success) {
    require(address(to) != 0x0);

    address feeContract = getFeeContract(self, address(this));
 uint fees = calculateFees(self, feeContract, amount);
    /// @dev NOTE: This transaction will fail if the spending amount
exceeds the daily limit
    require(setAccountSpendingAmount(self,                    from,
getFxUSDAmount(self, currency, amount)));

    /// @dev Attempt to transfer the amount
    require(forceTransfer(self, currency, from, to, amount, data));

    /// @dev "0x547846656573" == "TxFees"
    require(forceTransfer(self, currency, from, feeContract, fees,
"0x547846656573"));

    /// @dev Attempt to update the spender allowance
    require(updateAllowance(self, currency, from, amount));

    return true;
  }

  /**
   * @notice Low-level transfer method
   * @dev | This method has an `internal` view
   * @dev | This method does not include fees or approved allowances.
   * @dev | This method is only for authorized interfaces to use
(e.g. TokenIOFX)
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param from Ethereum address of account to send currency amount
from
   * @param to Ethereum address of account to send currency amount
to
   * @param amount Value of currency to transfer
   * @param data Arbitrary bytes data to include with the transaction
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function forceTransfer(Data storage self, string currency, address
from, address to, uint amount, bytes data) internal returns (bool
success) {
    require(address(to) != 0x0);

    bytes32  id_a  =  keccak256(abi.encodePacked('token.balance',
currency, getForwardedAccount(self, from)));
    bytes32  id_b  =  keccak256(abi.encodePacked('token.balance',
currency, getForwardedAccount(self, to)));

    require(self.Storage.setUint(id_a,
self.Storage.getUint(id_a).sub(amount)));
    require(self.Storage.setUint(id_b,
self.Storage.getUint(id_b).add(amount)));
```

```
emit LogTransfer(currency, from, to, amount, data);

    return true;
  }

  /**
   * @notice Low-level method to update spender allowance for account
   * @dev | This method is called inside the `transferFrom()` method
   * @dev | msg.sender == spender address
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param account Ethereum address of account holder
   * @param amount Value to reduce allowance by (i.e. the amount
spent)
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function updateAllowance(Data storage self, string currency,
address account, uint amount) internal returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('token.allowance',
currency,          getForwardedAccount(self,          account),
getForwardedAccount(self, msg.sender)));
    require(self.Storage.setUint(id,
self.Storage.getUint(id).sub(amount)));
    return true;
  }

  /**
   * @notice Low-level method to set the allowance for a spender
   * @dev | This method is called inside the `approve()` ERC20 method
   * @dev | msg.sender == account holder
   * @param self Internal storage proxying TokenIOStorage contract
   * @param spender Ethereum address of account spender
   * @param amount Value to set for spender allowance
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function approveAllowance(Data storage self, address spender, uint
amount) internal returns (bool success) {
    string memory currency = getTokenSymbol(self, address(this));

    bytes32 id_a = keccak256(abi.encodePacked('token.allowance',
currency,         getForwardedAccount(self,         msg.sender),
getForwardedAccount(self, spender)));
    bytes32 id_b = keccak256(abi.encodePacked('token.balance',
currency, getForwardedAccount(self, msg.sender)));

    require(self.Storage.getUint(id_a) == 0 || amount == 0);
    require(self.Storage.getUint(id_b) >= amount);
    require(self.Storage.setUint(id_a, amount));

    emit LogApproval(msg.sender, spender, amount);
```

```
return true;
  }

  /**
   * @notice Deposit an amount of currency into the Ethereum account
holder
   * @dev | The total supply of the token increases only when new
funds are deposited 1:1
   * @dev | This method should only be called by authorized issuer
firms
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param account Ethereum address of account holder to deposit
funds for
   * @param amount Value of currency to deposit for account
   * @param issuerFirm Name of the issuing firm authorizing the
deposit
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function deposit(Data storage self, string currency, address
account, uint amount, string issuerFirm) internal returns (bool
success) {
    bytes32 id_a = keccak256(abi.encodePacked('token.balance',
currency, getForwardedAccount(self, account)));
    bytes32 id_b = keccak256(abi.encodePacked('token.issued',
currency, issuerFirm));
    bytes32 id_c = keccak256(abi.encodePacked('token.supply',
currency));


    require(self.Storage.setUint(id_a,
self.Storage.getUint(id_a).add(amount)));
    require(self.Storage.setUint(id_b,
self.Storage.getUint(id_b).add(amount)));
    require(self.Storage.setUint(id_c,
self.Storage.getUint(id_c).add(amount)));

    emit LogDeposit(currency, account, amount, issuerFirm);

    return true;

  }

  /**
   * @notice Withdraw an amount of currency from the Ethereum account
holder
   * @dev | The total supply of the token decreases only when new
funds are withdrawn 1:1
   * @dev | This method should only be called by authorized issuer
firms
```

```
* @param self Internal storage proxying TokenIOStorage contract
   * @param currency TokenIO TSM currency symbol (e.g. USDx)
   * @param account Ethereum address of account holder to deposit
funds for
   * @param amount Value of currency to withdraw for account
   * @param issuerFirm Name of the issuing firm authorizing the
withdraw
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function withdraw(Data storage self, string currency, address
account, uint amount, string issuerFirm) internal returns (bool
success) {
    bytes32 id_a = keccak256(abi.encodePacked('token.balance',
currency, getForwardedAccount(self, account)));
    bytes32 id_b = keccak256(abi.encodePacked('token.issued',
currency, issuerFirm)); // possible for issuer to go negative
    bytes32 id_c = keccak256(abi.encodePacked('token.supply',
currency));

    require(self.Storage.setUint(id_a,
self.Storage.getUint(id_a).sub(amount)));
    require(self.Storage.setUint(id_b,
self.Storage.getUint(id_b).sub(amount)));
    require(self.Storage.setUint(id_c,
self.Storage.getUint(id_c).sub(amount)));

    emit LogWithdraw(currency, account, amount, issuerFirm);

    return true;

  }

  /**
   * @notice Method for setting a registered issuer firm
   * @dev | Only Token, Inc. and other authorized institutions may
set a registered firm
   * @dev | The TokenIOAuthority.sol interface wraps this method
   * @dev | If the registered firm is unapproved; all authorized
addresses of that firm will also be unapproved
   * @param self Internal storage proxying TokenIOStorage contract
   * @param issuerFirm Name of the firm to be registered
   * @param approved Approval status to set for the firm (true/false)
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function setRegisteredFirm(Data storage self, string issuerFirm,
bool approved) internal returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('registered.firm',
issuerFirm));
    require(self.Storage.setBool(id, approved));
    return true;
}
  /**
   * @notice Method for setting a registered issuer firm authority
   * @dev | Only Token, Inc. and other approved institutions may set
a registered firm
   * @dev | The TokenIOAuthority.sol interface wraps this method
   * @dev | Authority can only be set for a registered issuer firm
   * @param self Internal storage proxying TokenIOStorage contract
   * @param issuerFirm Name of the firm to be registered to authority
   * @param authorityAddress Ethereum address of the firm authority
to be approved
   * @param approved Approval status to set for the firm authority
(true/false)
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function   setRegisteredAuthority(Data   storage   self,   string
issuerFirm,   address   authorityAddress,   bool   approved)   internal
returns (bool success) {
    require(isRegisteredFirm(self, issuerFirm));
    bytes32                          id_a                          =
keccak256(abi.encodePacked('registered.authority',      issuerFirm,
authorityAddress));
    bytes32                          id_b                          =
keccak256(abi.encodePacked('registered.authority.firm',
authorityAddress));

    require(self.Storage.setBool(id_a, approved));
    require(self.Storage.setString(id_b, issuerFirm));

    return true;
  }

  /**
   * @notice Get the issuer firm registered to the authority Ethereum
address
   * @dev | Only one firm can be registered per authority
   * @param self Internal storage proxying TokenIOStorage contract
   * @param authorityAddress Ethereum address of the firm authority
to query
   *  @return  {  "issuerFirm"  :  "Name  of  the  firm  registered  to
authority" }
   */
  function   getFirmFromAuthority(Data   storage   self,   address
authorityAddress) internal view returns (string issuerFirm) {
    bytes32                           id                           =
keccak256(abi.encodePacked('registered.authority.firm',
getForwardedAccount(self, authorityAddress)));
    return self.Storage.getString(id);
  }

  /**
```

```
* @notice Return the boolean (true/false) registration status for
an issuer firm
   * @param self Internal storage proxying TokenIOStorage contract
   * @param issuerFirm Name of the issuer firm
   * @return { "registered" : "Return if the issuer firm has been
registered" }
   */
  function isRegisteredFirm(Data storage self, string issuerFirm)
internal view returns (bool registered) {
    bytes32 id = keccak256(abi.encodePacked('registered.firm',
issuerFirm));
    return self.Storage.getBool(id);
  }

  /**
   * @notice Return the boolean (true/false) status if an authority
is registered to an issuer firm
   * @param self Internal storage proxying TokenIOStorage contract
   * @param issuerFirm Name of the issuer firm
   * @param authorityAddress Ethereum address of the firm authority
to query
   * @return { "registered" : "Return if the authority is registered
with the issuer firm" }
   */
  function isRegisteredToFirm(Data storage self, string issuerFirm,
address authorityAddress) internal view returns (bool registered) {
    bytes32 id = keccak256(abi.encodePacked('registered.authority',
issuerFirm, getForwardedAccount(self, authorityAddress)));
    return self.Storage.getBool(id);
  }

  /**
   * @notice Return if an authority address is registered
   * @dev | This also checks the status of the registered issuer
firm
   * @param self Internal storage proxying TokenIOStorage contract
   * @param authorityAddress Ethereum address of the firm authority
to query
   * @return { "registered" : "Return if the authority is
registered" }
   */
  function isRegisteredAuthority(Data storage self, address
authorityAddress) internal view returns (bool registered) {
    bytes32 id = keccak256(abi.encodePacked('registered.authority',
getFirmFromAuthority(self, getForwardedAccount(self,
authorityAddress)), getForwardedAccount(self, authorityAddress)));
    return self.Storage.getBool(id);
  }

  /**
   * @notice Return boolean transaction status if the transaction
has been used
```

```
* @param self Internal storage proxying TokenIOStorage contract
   * @param txHash keccak256 ABI tightly packed encoded hash digest
of tx params
   * @return {"txStatus": "Returns true if the tx hash has already
been set using `setTxStatus()` method"}
   */
  function getTxStatus(Data storage self, bytes32 txHash) internal
view returns (bool txStatus) {
    bytes32 id = keccak256(abi.encodePacked('tx.status', txHash));
    return self.Storage.getBool(id);
  }

  /**
   * @notice Set transaction status if the transaction has been used
   * @param self Internal storage proxying TokenIOStorage contract
   * @param txHash keccak256 ABI tightly packed encoded hash digest
of tx params
   * @return { "success" : "Return true if successfully called from
another contract" }
   */
  function setTxStatus(Data storage self, bytes32 txHash) internal
returns (bool success) {
    bytes32 id = keccak256(abi.encodePacked('tx.status', txHash));
    require(self.Storage.setBool(id, true));
    return true;
  }

  /**
   * @notice Accepts a signed fx request to swap currency pairs at
a given amount;
   * @dev | This method can be called directly between peers
   * @dev | This method does not take transaction fees from the swap
   * @param self Internal storage proxying TokenIOStorage contract
   * @param  requester address Requester is the orginator of the
offer and must
   * match the signature of the payload submitted by the fulfiller
   * @param  symbolA    Symbol of the currency desired
   * @param  symbolB    Symbol of the currency offered
   * @param  valueA     Amount of the currency desired
   * @param  valueB     Amount of the currency offered
   * @param  sigV        Ethereum secp256k1 signature V value; used
by ecrecover()
   * @param  sigR        Ethereum secp256k1 signature R value; used
by ecrecover()
   * @param  sigS        Ethereum secp256k1 signature S value; used
by ecrecover()
   * @param  expiration Expiration of the offer; Offer is good until
expired
   * @return {"success" : "Returns true if successfully called from
another contract"}
   */
  function execSwap(
    Data storage self,
    address requester,
    string symbolA,
    string symbolB,
    uint valueA,
    uint valueB,
    uint8 sigV,
    bytes32 sigR,
    bytes32 sigS,
    uint expiration
  ) internal returns (bool success) {

    bytes32   fxTxHash   =   keccak256(abi.encodePacked(requester,
symbolA, symbolB, valueA, valueB, expiration));
    require(verifyAccounts(self, msg.sender, requester));

    /// @dev Ensure transaction has not yet been used;
    require(!getTxStatus(self, fxTxHash));

    /// @dev Immediately set this transaction to be confirmed before
updating any params;
    require(setTxStatus(self, fxTxHash));

    /// @dev Ensure contract has not yet expired;
    require(expiration >= now);

    /// @dev Recover the address of the signature from the hashed
digest;
    /// @dev Ensure it equals the requester's address
    require(ecrecover(fxTxHash, sigV, sigR, sigS) == requester);

    /// @dev Transfer funds from each account to another.
    require(forceTransfer(self,  symbolA,  msg.sender,  requester,
valueA, "0x0"));
    require(forceTransfer(self,  symbolB,  requester,  msg.sender,
valueB, "0x0"));

    emit LogFxSwap(symbolA, symbolB, valueA, valueB, expiration,
fxTxHash);

    return true;
  }

  /**
   * @notice Deprecate a contract interface
   * @dev | This is a low-level method to deprecate a contract
interface.
   * @dev | This is useful if the interface needs to be updated or
becomes out of date
   * @param self Internal storage proxying TokenIOStorage contract
   * @param contractAddress Ethereum address of the contract
interface
```

```
* @return {"success" : "Returns true if successfully called from
another contract"}
   */
  function  setDeprecatedContract(Data  storage  self,  address
contractAddress) internal returns (bool success) {
    bytes32   id   =   keccak256(abi.encodePacked('depcrecated',
contractAddress));
    require(self.Storage.setBool(id, true));
    return true;
  }

  /**
   * @notice Return the deprecation status of a contract
   * @param self Internal storage proxying TokenIOStorage contract
   *  @param  contractAddress  Ethereum  address  of  the  contract
interface
   * @return {"status" : "Return deprecation status (true/false) of
the contract interface"}
   */
  function  isContractDeprecated(Data  storage  self,  address
contractAddress) internal view returns (bool status) {
    bytes32   id   =   keccak256(abi.encodePacked('depcrecated',
contractAddress));
    return self.Storage.getBool(id);
  }

  /**
   * @notice Set the Account Spending Period Limit as UNIX timestamp
   * @dev | Each account has it's own daily spending limit
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
   * @param period Unix timestamp of the spending period
   * @return {"success" : "Returns true is successfully called from
a contract"}
   */
  function  setAccountSpendingPeriod(Data  storage  self,  address
account, uint period) internal returns (bool success) {
    bytes32                          id                          =
keccak256(abi.encodePacked('limit.spending.period', account));
    require(self.Storage.setUint(id, period));
    return true;
  }

  /**
   * @notice Get the Account Spending Period Limit as UNIX timestamp
   * @dev | Each account has it's own daily spending limit
   * @dev | If the current spending period has expired, it will be
set upon next `transfer()`
   * or `transferFrom()` request
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
* @return {"period" : "Returns Unix timestamp of the current spending
period"}
   */
  function  getAccountSpendingPeriod(Data  storage  self,  address
account) internal view returns (uint period) {
    bytes32                          id                          =
keccak256(abi.encodePacked('limit.spending.period', account));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Set the account spending limit amount
   * @dev | Each account has it's own daily spending limit
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
   * @param limit Spending limit amount
   * @return {"success" : "Returns true is successfully called from
a contract"}
   */
  function  setAccountSpendingLimit(Data  storage  self,  address
account, uint limit) internal returns (bool success) {
    bytes32                          id                          =
keccak256(abi.encodePacked('account.spending.limit', account));
    require(self.Storage.setUint(id, limit));
    return true;
  }

  /**
   * @notice Get the account spending limit amount
   * @dev | Each account has it's own daily spending limit
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
   * @return {"limit" : "Returns the account spending limit amount"}
   */
  function  getAccountSpendingLimit(Data  storage  self,  address
account) internal view returns (uint limit) {
    bytes32                          id                          =
keccak256(abi.encodePacked('account.spending.limit', account));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Set the account spending amount for the daily period
   * @dev | Each account has it's own daily spending limit
   * @dev | This transaction will throw if the new spending amount
is greater than the limit
   *  @dev  |  This  method  is  called  in  the  `transfer()`  and
`transferFrom()` methods
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
   * @param amount Set the amount spent for the daily period
* @return {"success" : "Returns true is successfully called from a
contract"}
   */
  function  setAccountSpendingAmount(Data  storage  self,  address
account, uint amount) internal returns (bool success) {

    /// @dev NOTE: Always ensure the period is current when checking
the daily spend limit
    require(updateAccountSpendingPeriod(self, account));
    uint     updatedAmount     =     getAccountSpendingAmount(self,
account).add(amount);

    /// @dev Ensure the spend limit is greater than the amount spend
for the period
    require(getAccountSpendingLimit(self,        account)        >=
updatedAmount);

    /// @dev Update the spending period amount if within limit
    bytes32                          id                            =
keccak256(abi.encodePacked('account.spending.amount',      account,
getAccountSpendingPeriod(self, account)));
    require(self.Storage.setUint(id, updatedAmount));
    return true;
  }

  /**
   * @notice Low-level API to ensure the account spending period is
always current
   *    @dev    |    This    method    is    internally    called    by
`setAccountSpendingAmount()` to ensure
   * spending period is always the most current daily period.
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
   * @return {"success" : "Returns true is successfully called from
a contract"}
   */
  function  updateAccountSpendingPeriod(Data  storage  self,  address
account) internal returns (bool success) {
    uint begDate = getAccountSpendingPeriod(self, account);
    if (begDate > now) {
      return true;
    } else {
      uint duration = 86400; // 86400 Seconds in a Day
      require(setAccountSpendingPeriod(self,              account,
begDate.add(((now.sub(begDate)).div(duration).add(1)).mul(duration
))));
      return true;
    }
  }

  /**
   * @notice Return the amount spent during the current period
* @dev | Each account has it's own daily spending limit
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
   * @return {"amount" : "Returns the amount spent by the account
during the current period"}
   */
  function  getAccountSpendingAmount(Data  storage  self,  address
account) internal view returns (uint amount) {
    bytes32                          id                          =
keccak256(abi.encodePacked('account.spending.amount',     account,
getAccountSpendingPeriod(self, account)));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Return the amount remaining during the current period
   * @dev | Each account has it's own daily spending limit
   * @param self Internal storage proxying TokenIOStorage contract
   * @param account Ethereum address of the account holder
   * @return {"amount" : "Returns the amount remaining by the account
during the current period"}
   */
  function getAccountSpendingRemaining(Data storage self, address
account) internal view returns (uint remainingLimit) {
    return                          getAccountSpendingLimit(self,
account).sub(getAccountSpendingAmount(self, account));
  }

  /**
   * @notice Set the foreign currency exchange rate to USD in basis
points
   * @dev | This value should always be relative to USD pair; e.g.
JPY/USD, GBP/USD, etc.
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency The TokenIO currency symbol (e.g. USDx, JPYx,
GBPx)
   * @param bpsRate Basis point rate of foreign currency exchange
rate to USD
   * @return { "success": "Returns true if successfully called from
another contract"}
   */
  function setFxUSDBPSRate(Data storage self, string currency, uint
bpsRate) internal returns (bool success) {
    bytes32    id    =    keccak256(abi.encodePacked('fx.usd.rate',
currency));
    require(self.Storage.setUint(id, bpsRate));
    return true;
  }

  /**
   * @notice Return the foreign currency USD exchanged amount in
basis points
```

```
* @param self Internal storage proxying TokenIOStorage contract
   * @param currency The TokenIO currency symbol (e.g. USDx, JPYx,
GBPx)
   * @return {"usdAmount" : "Returns the foreign currency amount in
USD"}
   */
  function  getFxUSDBPSRate(Data  storage  self,  string  currency)
internal view returns (uint bpsRate) {
    bytes32    id    =    keccak256(abi.encodePacked('fx.usd.rate',
currency));
    return self.Storage.getUint(id);
  }

  /**
   * @notice Return the foreign currency USD exchanged amount
   * @param self Internal storage proxying TokenIOStorage contract
   * @param currency The TokenIO currency symbol (e.g. USDx, JPYx,
GBPx)
   * @param fxAmount Amount of foreign currency to exchange into USD
   * @return {"amount" : "Returns the foreign currency amount in
USD"}
   */
  function getFxUSDAmount(Data storage self, string currency, uint
fxAmount) internal view returns (uint amount) {
    uint usdDecimals = getTokenDecimals(self, 'USDx');
    uint fxDecimals = getTokenDecimals(self, currency);
    /// @dev ensure decimal precision is normalized to USD decimals
    uint    usdAmount    =    ((fxAmount.mul(getFxUSDBPSRate(self,
currency)).div(10000)).mul(10**usdDecimals)).div(10**fxDecimals);
    return usdAmount;
  }
}
```

1. **Line 548 function setTokenFrozenBalance**

Function is declared as view, but this function called in line 550:

require(self.Storage.setUint(id, amount));

modifies the state of a variable. You need to remove the modifier **view.**

2. **Line 600 function verifyAccount**

Function state mutability can be restricted to view

function verifyAccount(Data storage self, address account) internal **view** returns (bool verified)

3. **Line 938 Now operator**

Now does not mean current time.
Now is an alias for block.timestamp and the Block.timestamp can be influenced by miners to a certain degree, be careful.

TokenIOStorage.sol

```
pragma solidity ^0.4.24;

import "./Ownable.sol";

/**

COPYRIGHT 2018 Token, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


@title TokenIOStorage - Serves as derived contract for TokenIO
contract and
is used to upgrade interfaces in the event of deprecating the main
contract.

@author Ryan Tate <ryan.michael.tate@gmail.com>

@notice Storage contract

@dev In the event that the main contract becomes deprecated, the
upgraded contract
will be set as the owner of this contract, and use this contract's
storage to
maintain data consistency between contract.

@notice NOTE: This contract is based on the RocketPool Storage
Contract,
found here: https://github.com/rocket-
pool/rocketpool/blob/master/contracts/RocketStorage.sol
And this medium article: https://medium.com/rocket-pool/upgradable-
solidity-contract-design-54789205276d

Changes:
 - setting primitive mapping view to internal;
 - setting method views to public;

 @dev NOTE: When deprecating the main TokenIO contract, the upgraded
contract
 must take ownership of the TokenIO contract, it will require using
the public methods
```

```
 to update changes to the underlying data. The updated contract must
use a
 standard call to original TokenIO contract such that the  request
is made from
 the upgraded contract and not the transaction origin (tx.origin)
of the signing
 account.


 @dev NOTE: The reasoning for using the storage contract is to
abstract the interface
 from the data of the contract on chain, limiting the need to migrate
data to
 new contracts.

*/
contract TokenIOStorage is Ownable {


    /// @dev mapping for Primitive Data Types;
          /// @notice primitive data mappings have `internal` view;
          /// @dev only the derived contract can use the internal
methods;
          /// @dev key == `keccak256(param1, param2...)`
          /// @dev Nested mapping can be achieved using multiple
params in keccak256 hash;
    mapping(bytes32 => uint256)    internal uIntStorage;
    mapping(bytes32 => string)     internal stringStorage;
    mapping(bytes32 => address)    internal addressStorage;
    mapping(bytes32 => bytes)      internal bytesStorage;
    mapping(bytes32 => bool)       internal boolStorage;
    mapping(bytes32 => int256)     internal intStorage;

    constructor() public {
                    ///  @notice  owner  is  set  to  msg.sender  by
default
                    /// @dev consider removing in favor of setting
ownership in inherited
                    /// contract
        owner[msg.sender] = true;
    }

    /// @dev Set Key Methods

    /**
     * @notice Set value for Address associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @param _value The Address value to be set
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
```

```
function setAddress(bytes32 _key, address _value) public onlyOwner
returns (bool success) {
        addressStorage[_key] = _value;
        return true;
    }

    /**
     * @notice Set value for Uint associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @param _value The Uint value to be set
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function setUint(bytes32 _key, uint _value) public onlyOwner
returns (bool success) {
        uIntStorage[_key] = _value;
        return true;
    }

    /**
     * @notice Set value for String associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @param _value The String value to be set
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function setString(bytes32 _key, string _value) public onlyOwner
returns (bool success) {
        stringStorage[_key] = _value;
        return true;
    }

    /**
     * @notice Set value for Bytes associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @param _value The Bytes value to be set
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function setBytes(bytes32 _key, bytes _value) public onlyOwner
returns (bool success) {
        bytesStorage[_key] = _value;
        return true;
    }

    /**
     * @notice Set value for Bool associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @param _value The Bool value to be set
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
```

```
function setBool(bytes32 _key, bool _value) public onlyOwner returns
(bool success) {
        boolStorage[_key] = _value;
        return true;
    }

    /**
     * @notice Set value for Int associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @param _value The Int value to be set
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function  setInt(bytes32  _key,  int  _value)  public  onlyOwner
returns (bool success) {
        intStorage[_key] = _value;
        return true;
    }

    /// @dev Delete Key Methods
          /// @dev delete methods may be unnecessary; Use set methods
to set values
          /// to default?

    /**
     * @notice Delete value for Address associated with bytes32 id
key
     * @param _key Pointer identifier for value in storage
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function  deleteAddress(bytes32  _key)  public  onlyOwner  returns
(bool success) {
        delete addressStorage[_key];
        return true;
    }

    /**
     * @notice Delete value for Uint associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function deleteUint(bytes32 _key) public onlyOwner returns (bool
success) {
        delete uIntStorage[_key];
        return true;
    }

    /**
     * @notice Delete value for String associated with bytes32 id
key
```

```
* @param _key Pointer identifier for value in storage
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function deleteString(bytes32 _key) public onlyOwner returns
(bool success) {
        delete stringStorage[_key];
        return true;
    }

    /**
     * @notice Delete value for Bytes associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function deleteBytes(bytes32 _key) public onlyOwner returns
(bool success) {
        delete bytesStorage[_key];
        return true;
    }

    /**
     * @notice Delete value for Bool associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function deleteBool(bytes32 _key) public onlyOwner returns (bool
success) {
        delete boolStorage[_key];
        return true;
    }

    /**
     * @notice Delete value for Int associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "success" : "Returns true when successfully called
from another contract" }
     */
    function deleteInt(bytes32 _key) public onlyOwner returns (bool
success) {
        delete intStorage[_key];
        return true;
    }

    /// @dev Get Key Methods

    /**
     * @notice Get value for Address associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
* @return { "_value" : "Returns the Address value associated with
the id key" }
     */
    function getAddress(bytes32 _key) public view returns (address
_value) {
        return addressStorage[_key];
    }

    /**
     * @notice Get value for Uint associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "_value" : "Returns the Uint value associated with
the id key" }
     */
    function getUint(bytes32 _key) public view returns (uint _value)
{
        return uIntStorage[_key];
    }

    /**
     * @notice Get value for String associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "_value" : "Returns the String value associated
with the id key" }
     */
    function getString(bytes32 _key) public view returns (string
_value) {
        return stringStorage[_key];
    }

    /**
     * @notice Get value for Bytes associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "_value" : "Returns the Bytes value associated with
the id key" }
     */
    function getBytes(bytes32 _key) public view returns (bytes
_value) {
        return bytesStorage[_key];
    }

    /**
     * @notice Get value for Bool associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "_value" : "Returns the Bool value associated with
the id key" }
     */
    function getBool(bytes32 _key) public view returns (bool _value)
{
        return boolStorage[_key];
    }
```

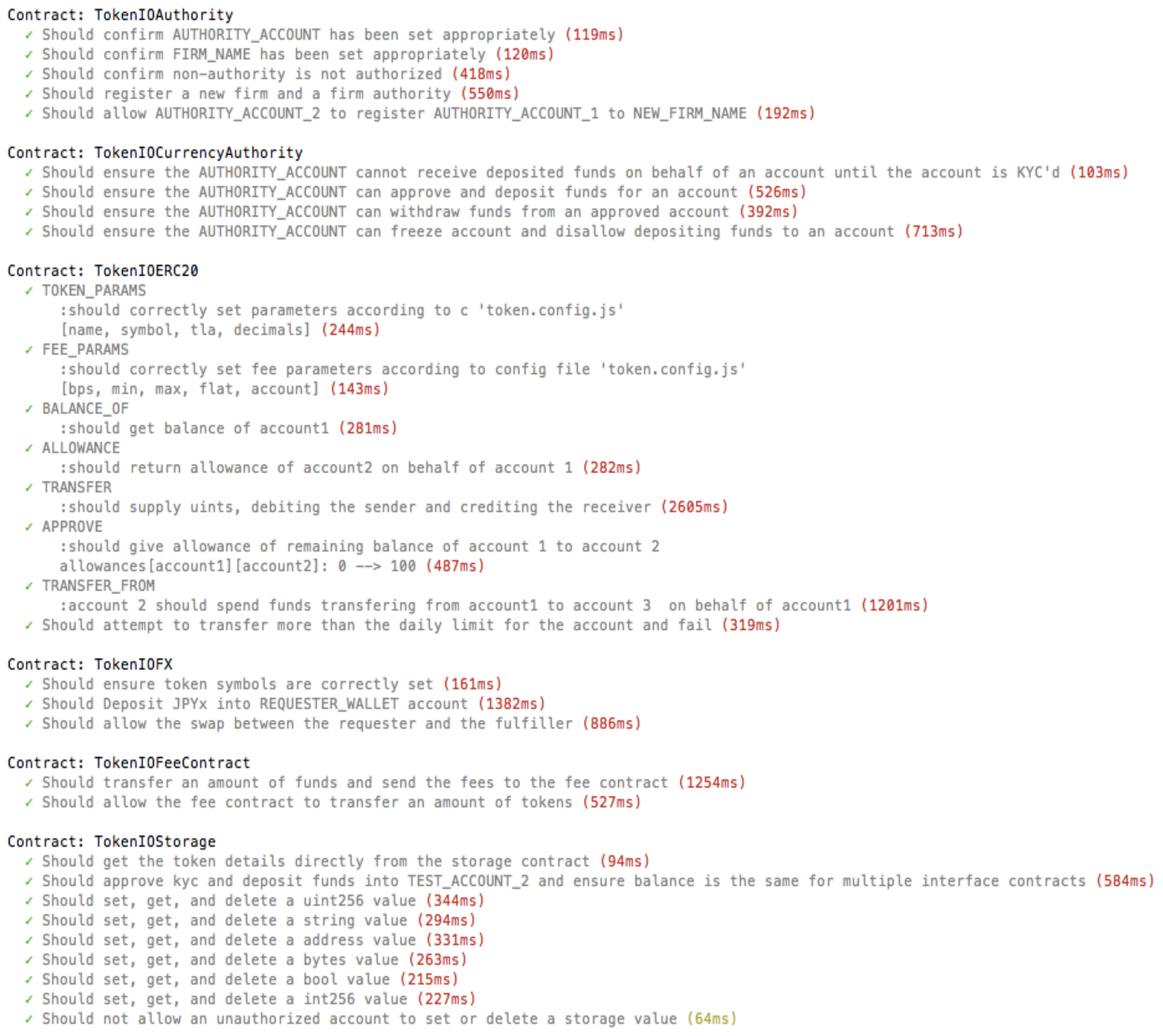
```
 /**
     * @notice Get value for Int associated with bytes32 id key
     * @param _key Pointer identifier for value in storage
     * @return { "_value" : "Returns the Int value associated with
the id key" }
     */
    function getInt(bytes32 _key) public view returns (int _value)
{
        return intStorage[_key];
    }

}
```

Conclusion TokenIOStorqge contract

```
No relevant issue on this contract.
```

Testing

- **Localhost Tests are passing without fail**



```
Contract: TokenIOAuthority
  ✓ Should confirm AUTHORITY_ACCOUNT has been set appropriately (119ms)
  ✓ Should confirm FIRM_NAME has been set appropriately (120ms)
  ✓ Should confirm non-authority is not authorized (418ms)
  ✓ Should register a new firm and a firm authority (550ms)
  ✓ Should allow AUTHORITY_ACCOUNT_2 to register AUTHORITY_ACCOUNT_1 to NEW_FIRM_NAME (192ms)

Contract: TokenIOCurrencyAuthority
  ✓ Should ensure the AUTHORITY_ACCOUNT cannot receive deposited funds on behalf of an account until the account is KYC'd (103ms)
  ✓ Should ensure the AUTHORITY_ACCOUNT can approve and deposit funds for an account (526ms)
  ✓ Should ensure the AUTHORITY_ACCOUNT can withdraw funds from an approved account (392ms)
  ✓ Should ensure the AUTHORITY_ACCOUNT can freeze account and disallow depositing funds to an account (713ms)

Contract: TokenIOERC20
  ✓ TOKEN_PARAMS
      :should correctly set parameters according to c 'token.config.js'
      [name, symbol, tla, decimals] (244ms)
  ✓ FEE_PARAMS
      :should correctly set fee parameters according to config file 'token.config.js'
      [bps, min, max, flat, account] (143ms)
  ✓ BALANCE_OF
      :should get balance of account1 (281ms)
  ✓ ALLOWANCE
      :should return allowance of account2 on behalf of account 1 (282ms)
  ✓ TRANSFER
      :should supply uints, debiting the sender and crediting the receiver (2605ms)
  ✓ APPROVE
      :should give allowance of remaining balance of account 1 to account 2
      allowances[account1][account2]: 0 --> 100 (487ms)
  ✓ TRANSFER_FROM
      :account 2 should spend funds transfering from account1 to account 3  on behalf of account1 (1201ms)
  ✓ Should attempt to transfer more than the daily limit for the account and fail (319ms)

Contract: TokenIOFX
  ✓ Should ensure token symbols are correctly set (161ms)
  ✓ Should Deposit JPYx into REQUESTER_WALLET account (1382ms)
  ✓ Should allow the swap between the requester and the fulfiller (886ms)

Contract: TokenIOFeeContract
  ✓ Should transfer an amount of funds and send the fees to the fee contract (1254ms)
  ✓ Should allow the fee contract to transfer an amount of tokens (527ms)

Contract: TokenIOStorage
  ✓ Should get the token details directly from the storage contract (94ms)
  ✓ Should approve kyc and deposit funds into TEST_ACCOUNT_2 and ensure balance is the same for multiple interface contracts (584ms)
  ✓ Should set, get, and delete a uint256 value (344ms)
  ✓ Should set, get, and delete a string value (294ms)
  ✓ Should set, get, and delete a address value (331ms)
  ✓ Should set, get, and delete a bytes value (263ms)
  ✓ Should set, get, and delete a bool value (215ms)
  ✓ Should set, get, and delete a int256 value (227ms)
  ✓ Should not allow an unauthorized account to set or delete a storage value (64ms)
```

- **Ropsten testnet Tests are not passing**