Audit

By OCamlPro

February 4, 2022

Table of all Found Issues

Contents

Chapter 1

Only for Auditors

1.1 To edit this documents

In the report.tex file, choose:

- **\soldraftfalse** to remove draft mode (watermarks, advises)
- **\solmodulestrue** to display modules by chapter instead of contracts
- **\soltablestrue** to display tables for parameters and returns
- **\solissuesfalse** to remove the table of issues

Issues can be entered with:

- **\issueCritical{title}{text}**
- **\issueMajor{title}{text}**
- **\issueMinor{title}{text}**

1.2 General Auditing Rules

- Check that types have the correct integer types (Pubkey : uint256, Amount: uint128, Time: uint64).
- Naming conventions: constants should for example be all uppercase, static variables should start with a prefix like `s_`, globals should start with a prefix like `g_` or `m_`, internal functions should start with a prefix `_`.
- Numbers should not appear in source, but be defined as constants.
- In constant definitions, verify that 2 consecutive errors have not the same error (common copy-paste error)

- Constants for amounts should be expressed in `ton` to prevent too many zeroes.
- Modifiers with `tvm.accept` must always check the source of the message
- Constructors with arguments must always check the source of the message to prevent anybody from calling the constructor and set variables instead of the real owner
- Failures should never happen after `tvm.accept` (such as `require`, division by zero, overflows, etc.)
- Most arguments should be protected by a `require`
- Before sending a message, the function should check that it has enough gas (to prevent a partial failure during the message sending phase)
- `tvm.accept` should only be called after verifying that the sender of the message if the contracts' owner

Chapter 2

Introduction

This is a security audit of the "..." smart contract, whose source code is available at `https://github.com/...`, commit `....` This security audit is provided as a submission to Formal Method Sub-Governance Contest `https://formet.gov.freeton.org/....`

During this audit, we used the following classification of our findings, into three kinds of issues:

- **Critical Issues:** such issues can lead to taking ownership of resources (tokens, contracts), or total disabling of the service;
- **Major Issues:** such issues can lead to a decrease in the quality of the service, or temporary loss of availability;
- **Minor Issues:** Such issues do not impact the service itself. For example, code improvements to improve readability, to improve sharing, etc.

We found XX critical issues, YY major issues and ZZ minor issues during this audit of the contracts.

- **XX Critical Issues:**
- **YY Major Issues:**
- **ZZ Minor Issues:**

For easier access to issues, we provided a table of issues at the beginning of the document.

Chapter 3

Overview

Chapter 4

Library CostConstants

Contents

4.1 Overview

In file `CostConstants.sol`

4.2 Constant Definitions

```
4     uint128 constant TOKEN_INITIAL_UPDATE_PRICE = 0.2 ton;
5     uint128 constant FETCH_TIP3_ROOT_INFORMATION = 0.2 ton;
6     uint128 constant SEND_TO_TIP3_DEPLOYER = 1.5 ton;
7     uint128 constant USE_TO_DEPLOY_TIP3_ROOT = 1 ton;
8     uint128 constant NOTIFY_CONTRACT_CONTROLLER = 0.2 ton;
```

Chapter 5

Library FPO

Contents

5.1 Overview

In file `FloatingPointOperations.sol`

5.2 Constant Definitions

```
9     uint256 constant bits224 = 2**224;

10    uint256 constant bits192 = 2**192;
```

```
11      uint256 constant bits160 = 2**160;
12      uint256 constant bits128 = 2**128;
13      uint256 constant bits96 = 2**96;
14      uint256 constant bits64 = 2**64;
15      uint256 constant bits32 = 2**32;
```

5.3 Internal Method Definitions

5.3.1 Function eq

- TODO

```
53      function eq(fraction a, fraction b) internal pure returns(bool)
            {
54          return ((a.nom == b.nom) && (a.denom == b.denom));
55      }
```

5.3.2 Function fAdd

- TODO

```
33      function fAdd(fraction a, fraction b) internal pure returns (
            fraction) {
34          return fraction (a.nom * b.denom + b.nom * a.denom, a.denom
                * b.denom);
35      }
```

5.3.3 Function fDiv

- TODO

```
29      function fDiv(fraction a, fraction b) internal pure returns(
            fraction) {
30          return fraction(a.nom * b.denom, a.denom * b.nom);
31      }
```

5.3.4 Function fMul

- TODO

```
17      function fMul(fraction a, fraction b) internal pure returns (
            fraction) {
18          return fraction(a.nom*b.nom, a.denom*b.denom);
19      }
```

5.3.5 Function fNumAdd

- TODO

```
37      function fNumAdd(fraction a, uint256 b) internal pure returns (
            fraction) {
38          return fraction (a.nom + b*a.denom, a.denom);
39      }
```

5.3.6 Function fNumDiv

- TODO

```
25      function fNumDiv(fraction a, uint256 b) internal pure returns (
            fraction) {
26          return fraction(a.nom, a.denom * b);
27      }
```

5.3.7 Function fNumMul

- TODO

```
21      function fNumMul(fraction a, uint256 b) internal pure returns (
            fraction) {
22          return fraction(a.nom * b, a.denom);
23      }
```

5.3.8 Function fSub

- TODO

```
41      function fSub(fraction a, fraction b) internal pure returns (
            fraction) {
42          return fraction(a.nom * b.denom - b.nom * a.denom, a.denom
                * b.denom);
43      }
```

5.3.9 Function getMin

- TODO

```
57      function getMin(fraction a, fraction b) internal pure returns(
            fraction) {
58          if (a.nom * b.denom < b.nom * a.denom) {
59              return a;
60          } else {
61              return b;
62          }
63      }
```

5.3.10 Function isLarger

- TODO

```
45      function isLarger(fraction a, fraction b) internal pure returns
            (bool) {
46          return a.nom * b.denom > b.nom * a.denom;
47      }
```

5.3.11 Function lessThan

- TODO

```
65      function lessThan(fraction a, fraction b) internal pure returns
            (bool) {
66          return a.nom * b.denom < b.nom * a.denom;
67      }
```

5.3.12 Function simplify

- TODO

```
69      function simplify(fraction a) internal pure returns(fraction) {
70          // loosing ??? of presicion at most
71          if (a.nom / a.denom > 100e9) {
72              return fraction(a.nom / a.denom, 1);
73          } else {
74              // using bitshift for simultaneos division
75              // leaving up to 64 bits of information if nom & denom
                    > 2^64
76              if ( (a.nom >= bits224) && (a.denom >= bits224) ) {
77                  return fraction(a.nom / bits160, a.denom / bits160)
                        ;
78              }
79
80              if ( (a.nom >= bits192) && (a.denom >= bits192) ) {
81                  return fraction(a.nom / bits128, a.denom / bits128)
                        ;
82              }
83
84              if ( (a.nom >= bits160) && (a.denom >= bits160) ) {
85                  return fraction(a.nom / bits96, a.denom / bits96);
86              }
87
88              if ( (a.nom >= bits128) && (a.denom >= bits128) ) {
89                  return fraction(a.nom / bits64, a.denom / bits64);
90              }
91
92              if ( (a.nom >= bits96) && (a.denom >= bits96) ) {
93                  return fraction(a.nom / bits32, a.denom / bits32);
94              }
```

```
95
96              return a;
97          }
98      }
```

5.3.13 Function toNum

- TODO

```
49      function toNum(fraction a) internal pure returns(uint256) {
50          return a.nom / a.denom;
51      }
```

Chapter 6

Library MarketErrorCodes

Contents

6.1 Overview

In file `MarketErrorCodes.sol`

6.2 Constant Definitions

```
4     uint8 constant ERROR_MSG_SENDER_IS_NOT_SELF = 100;
5     uint8 constant ERROR_MSG_SENDER_IS_NOT_ROOT = 101;
6     uint8 constant ERROR_MSG_SENDER_IS_NOT_REAL_TOKEN = 102;
7     uint8 constant ERROR_MSG_SENDER_IS_NOT_VIRTUAL_TOKEN = 103;
8     uint8 constant ERROR_MSG_SENDER_IS_NOT_OWNER = 104;
9     uint8 constant ERROR_MSG_SENDER_IS_NOT_ORACLE = 110;
10    uint8 constant ERROR_MSG_SENDER_IS_NOT_TIP3_DEPLOYER = 111;
11    uint8 constant ERROR_MSG_SENDER_IS_NOT_USER_ACCOUNT_MANAGER =
          112;
12    uint8 constant ERROR_MSG_SENDER_IS_NOT_TIP3_WALLET_CONTROLLER =
           113;
13    uint8 constant ERROR_INVALID_CONTRACT_TYPE = 200;
```

Chapter 7

Library MarketMath

Contents

7.1 Overview

In file `MarketMath.sol`

7.2 Internal Method Definitions

7.2.1 Function calculateBorrowingRate

- TODO

```
8     function calculateBorrowingRate(uint256 currentPool, uint256
          totalBorrowed, uint256 totalReserves, uint256 totalSupply)
9         internal pure returns (uint256)
10    {
11
12    }
```

7.2.2 Function calculateExchangeRate

- TODO

```
14      function calculateExchangeRate(uint256 currentPool, uint256
            totalBorrowed, uint256 totalReserves, uint256 totalSupply)
15          internal pure returns (uint256)
16      {
17          return math.div(currentPool - totalReserves + totalBorrowed
                , totalSupply);
18      }
```

7.2.3 Function calculateUtilizationRate

- TODO

```
4       function calculateUtilizationRate(uint256 currentPool, uint256
            totalBorrowed) internal pure returns (uint256) {
5
6       }
```

7.2.4 Function recalculateState

- TODO

```
20      function recalculateState(uint256 currentPool, uint256
            totalBorrowed, uint256 totalReserves, uint256 totalSupply)
21          internal pure
22      {
23          // uint256 exchangeRate = calculateExchangeRate(currentPool
                , totalBorrowed, totalReserves, totalSupply);
24      }
```

Chapter 8

Library MarketOperationCodes

Contents

8.1 Overview

In file `MarketOperationCodes.sol`

8.2 Constant Definitions

```
4     uint8 constant SUPPLY_TOKENS = 0;
5     uint8 constant WITHDRAW_TOKENS = 1;
6     uint8 constant BORROW_TOKENS = 2;
7     uint8 constant REPAY_LOAN = 3;
8     uint8 constant LIQUIDATE_LOAN = 4;
10    uint8 constant RESUME_SUPPLY_TOKENS = 10;
11    uint8 constant RESUME_WITHDRAW_TOKENS = 11;
12    uint8 constant RESUME_BORROW_TOKENS = 12;
13    uint8 constant RESUME_REPAY_LOAN = 13;
```

```
14      uint8 constant RESUME_LIQUIDATE_LOAN = 14;

16      uint8 constant WRITE_SUPPLY_TOKENS = 20;
17      uint8 constant WRITE_WITHDRAW_TOKENS = 21;
18      uint8 constant WRITE_BORROW_TOKENS = 22;
19      uint8 constant WRITE_REPAY_LOAN = 23;
20      uint8 constant WRITE_LIQUIDATE_LOAN = 24;

22      uint8 constant RESPONSE_SUPPLY_TOKENS = 30;
23      uint8 constant RESPONSE_WITHDRAW_TOKENS = 31;
24      uint8 constant RESPONSE_BORROW_TOKENS = 32;
25      uint8 constant RESPONSE_REPAY_LOAN = 33;
26      uint8 constant RESPONSE_LIQUIDATE_LOAN = 34;

28      uint8 constant BORROW_FINALIZE = 40;

30      uint8 constant REQUEST_INDEX_UPDATE = 50;

32      uint8 constant INDEX_UPDATE_RESPONSE = 60;
```

Chapter 9

Library MarketOperations

Contents

9.1 Overview

In file `MarketOperations.sol`

9.2 Internal Method Definitions

9.2.1 Function calculateBorrowInterestRate

- TODO

```
13      function calculateBorrowInterestRate(fraction baseRate, uint256
            realTokenBalance, uint256 totalBorrowed, fraction
        utilizationMultiplier) internal returns (fraction) {
14          fraction bir;
15
16          fraction utilizationRate = fraction(totalBorrowed,
                totalBorrowed + realTokenBalance);
```

```
17
18          bir = utilizationRate.fMul(utilizationMultiplier);
19          bir = bir.fAdd(baseRate);
20
21          return bir;
22      }
```

9.2.2 Function calculateExchangeRate

- TODO

```
24      function calculateExchangeRate(uint256 currentPoolBalance,
            uint256 totalBorrowed, uint256 totalReserve, uint256
            vTokenSupply) internal pure returns(fraction) {
25          return fraction(currentPoolBalance + totalBorrowed -
                totalReserve, vTokenSupply);
26      }
```

9.2.3 Function calculateNewIndex

- TODO

```
37      function calculateNewIndex(fraction index, fraction bir,
            uint256 dt) internal returns (fraction) {
38          fraction index_;
39          index_ = bir.fNumMul(dt);
40          index_ = index_.fNumAdd(1);
41          index_ = index_.fAdd(index);
42          return index_;
43      }
```

9.2.4 Function calculateReserves

- TODO

```
52      function calculateReserves(uint256 reserveOld, uint256
            totalBorrowedOld, fraction bir, fraction reserveFactor,
            uint256 dt) internal returns (uint256) {
53          fraction res = bir;
54          res = res.fNumMul(dt);
55          res = res.fMul(reserveFactor);
56          res = res.fNumMul(totalBorrowedOld);
57          res = res.fNumAdd(reserveOld);
58          return res.toNum();
59      }
```

9.2.5 Function calculateTotalBorrowed

- TODO

```
45      function calculateTotalBorrowed(uint256 totalBorrowed, fraction
            oldIndex, fraction newIndex) internal returns (uint256) {
46          fraction tb_;
47          tb_ = totalBorrowed.numFDiv(oldIndex);
48          tb_ = tb_.fMul(newIndex);
49          return tb_.toNum();
50      }
```

9.2.6 Function calculateTotalReserves

- TODO

```
28      function calculateTotalReserves(uint256 totalReserve, uint256
            totalBorrowed, fraction r, fraction reserveFactor, uint256
            t) internal returns (fraction) {
29          fraction tr;
30          tr = r.fNumMul(t);
31          tr = tr.fMul(reserveFactor);
32          tr = tr.fNumMul(totalBorrowed);
33          tr = tr.fNumAdd(totalReserve);
34          return tr;
35      }
```

9.2.7 Function calculateU

- TODO

```
9       function calculateU(uint256 totalBorrowed, uint256 realTokens)
            internal pure returns (fraction) {
10          return fraction(totalBorrowed, totalBorrowed + realTokens);
11      }
```

9.2.7.0.1 Some functions inherited by using

Chapter 10

Library MarketToUserPayloads

Contents

10.1 Overview

In file `MarketPayloads.sol`

10.2 Internal Method Definitions

10.2.1 Function createBorrowPayload

- TODO

```
26      function createBorrowPayload(uint32 marketId, uint256
            tokensToBorrow, address userTargetWallet) internal pure
            returns (TvmCell) {
27          TvmBuilder tb;
28          tb.store(MarketOperationCodes.BORROW_TOKENS);
29          TvmBuilder op;
30          op.store(marketId);
31          op.store(tokensToBorrow);
32          op.store(userTargetWallet);
33          tb.storeRef(op.toCell());
34          return tb.toCell();
35      }
```

10.2.2 Function createIndexUpdateRequest

- TODO

```
88      function createIndexUpdateRequest(address tonWallet, uint32
            marketId, mapping (uint32=>bool) upd, address
            userTip3Wallet, uint256 amountToBorrow) internal pure
            returns(TvmCell) {
89          TvmBuilder tb;
90          tb.store(MarketOperationCodes.REQUEST_INDEX_UPDATE);
91          TvmBuilder op;
92          op.store(tonWallet);
93          op.store(marketId);
94          op.store(upd);
95          op.store(userTip3Wallet);
96          op.store(amountToBorrow);
97          tb.store(op.toCell());
98          return tb.toCell();
99      }
```

10.2.3 Function createIndexUpdateResponse

- TODO

```
106     function createIndexUpdateResponse(uint32 marketId, address
            userTip3Wallet, uint256 amountToBorrow, mapping (uint32=>
            bool) upd) internal pure returns(TvmCell) {
107         TvmBuilder tb;
108         tb.store(MarketOperationCodes.INDEX_UPDATE_RESPONSE);
109         TvmBuilder op;
110         op.store(marketId);
111         op.store(userTip3Wallet);
112         op.store(amountToBorrow);
113         op.store(upd);
114         tb.store(op);
115         return tb.toCell();
116     }
```

10.2.4 Function createRepayPayload

- TODO

```
37      function createRepayPayload(uint32 marketId, uint256
            tokensToRepay) internal pure returns (TvmCell) {
38
39      }
```

10.2.5 Function createSupplyPayload

- TODO

```
9       function createSupplyPayload(uint32 marketId, uint256
            providedTokens, uint128 realTokens, address userTIP3Wallet)
             internal pure returns (TvmCell) {
10          TvmBuilder tb;
11          tb.store(MarketOperationCodes.SUPPLY_TOKENS);
12          TvmBuilder op;
13          op.store(marketId);
14          op.store(providedTokens);
15          op.store(realTokens);
16          op.store(userTIP3Wallet);
17          tb.store(op.toCell());
18          return tb.toCell();
19      }
```

10.2.6 Function decodeBorrow

- TODO

```
66      function decodeBorrow(TvmCell args) internal pure returns (
            address, uint32, address, uint256, mapping(uint32 =>
            uint256), mapping(uint32 => uint256)) {
67          TvmSlice ts = args.toSlice();
68          return ts.decode(address, uint32, address, uint256, mapping
                (uint32 => uint256), mapping(uint32 => uint256));
69      }
```

10.2.7 Function decodeBorrowAddition

- TODO

```
83      function decodeBorrowAddition(TvmCell args) internal pure
            returns (uint256, address, fraction, uint32) {
84          TvmSlice ts = args.toSlice();
85          return ts.decode(uint256, address, fraction, uint32);
86      }
```

10.2.8 Function decodeBorrowOperation

- TODO

```
48      function decodeBorrowOperation(TvmCell args) internal pure
            returns(uint32, uint256, address) {
49          TvmSlice ts = args.toSlice();
50          return ts.decode(uint32, uint256, address);
51      }
```

10.2.9 Function decodeIndexUpdateRequest

- TODO

```
101     function decodeIndexUpdateRequest(TvmCell args) internal pure
            returns(address, uint32, mapping (uint32=>bool) upd,
            address userTip3Wallet, uint256 amountToBorrow) {
102         TvmSlice ts = args.toSlice();
103         return ts.decode(address, uint32, mapping (uint32=>bool),
                address, uint256);
104     }
```

10.2.10 Function decodeSupplyOperation

- TODO

```
21      function decodeSupplyOperation(TvmCell args) internal pure
            returns(uint32, uint256, uint128, address) {
22          TvmSlice ts = args.toSlice();
23          return ts.decode(uint32, uint256, uint128, address);
24      }
```

10.2.11 Function encodeBorrow

- TODO

```
53      function encodeBorrow(address tonWallet, address userTip3Wallet
            , uint256 toBorrow, mapping(uint32 => uint256) bi, mapping(
            uint32 => uint256) si) internal pure returns (TvmCell) {
54          TvmBuilder tb;
55          tb.store(MarketOperationCodes.BORROW_TOKENS);
56          TvmBuilder op;
57          op.store(tonWallet);
58          op.store(userTip3Wallet);
59          op.store(toBorrow);
60          op.store(bi);
61          op.store(si);
62          tb.store(op.toCell());
63          return tb.toCell();
64      }
```

10.2.12 Function encodeBorrowAddition

- TODO

```
71      function encodeBorrowAddition(uint256 borrowAmount, address
            tip3Wallet, fraction index, uint32 marketId_) internal pure
             returns (TvmCell) {
72          TvmBuilder tb;
73          tb.store(MarketOperationCodes.BORROW_FINALIZE);
74          TvmBuilder op;
75          op.store(borrowAmount);
76          op.store(tip3Wallet);
77          op.store(index);
78          op.store(marketId_);
79          tb.store(op.toCell());
80          return tb.toCell();
81      }
```

10.2.13 Function getOperationType

- TODO

```
41      function getOperationType(TvmCell payload) internal pure
            returns (uint8, TvmCell) {
42          TvmSlice ts = payload.toSlice();
43          uint8 op = ts.decode(uint8);
44          TvmCell opArgs = ts.loadRef();
45          return (op, opArgs);
46      }
```

Chapter 11

Library MsgFlag

Contents

11.1 Overview

In file `MsgFlag.sol`

11.2 Constant Definitions

```
4     uint8 constant SENDER_PAYS_FEES     = 1;
5     uint8 constant IGNORE_ERRORS        = 2;
6     uint8 constant DESTROY_IF_ZERO      = 32;
7     uint8 constant REMAINING_GAS        = 64;
8     uint8 constant ALL_NOT_RESERVED     = 128;
```

Chapter 12

Library OperationCodes

Contents

12.1 Overview

In file `OperationCodes.sol`

12.2 Constant Definitions

```
4     uint8 constant SUPPLY_TOKENS = 0;
5     uint8 constant REPAY_TOKENS = 1;
6     uint8 constant WITHDRAW_TOKENS = 2;
7     uint8 constant BORROW_TOKENS = 3;
8     uint8 constant LIQUIDATE_TOKENS = 4;
9     uint8 constant REQUEST_TOKEN_PAYOUT = 100;
10    uint8 constant NO_OP = 255;
```

Chapter 13

Library OracleErrorCodes

Contents

13.1 Overview

In file `OracleErrorCodes.sol`

13.2 Constant Definitions

```
4     uint8 constant ERROR_NOT_OWNER = 100;

5     uint8 constant ERROR_NOT_TRUSTED = 101;

6     uint8 constant ERROR_NOT_ROOT = 102;

8     uint8 constant ERROR_NOT_KNOWN_SWAP_PAIR = 110;

9     uint8 constant ERROR_NOT_KNOWN_TOKEN_ROOT = 111;

11    uint8 constant ERROR_INVALID_CONTRACT_TYPE = 200;
```

Chapter 14

Library RolesErrors

Contents

14.1 Overview

In file `IRoles.sol`

14.2 Constant Definitions

```
6     uint8 constant CANNOT_UPGRADE = 220;
7     uint8 constant CANNOT_CHANGE_PARAMS = 221;
8     uint8 constant IS_NOT_OWNER = 222;
```

Chapter 15

Library RootTokenContractErrors

Contents

15.1 Overview

In file RootTokenContractErrors.sol

15.2 Constant Definitions

```
4     uint8 constant error_message_sender_is_not_my_owner = 100;
5     uint8 constant error_not_enough_balance = 101;
6     uint8 constant error_message_sender_is_not_good_wallet = 102;
7     uint8 constant error_define_public_key_or_owner_address = 103;
8     uint8 constant error_paused = 104;
```

Chapter 16

Library TIP3DeployerErrorCodes

Contents

16.1 Overview

In file TIP3DeployerErrorCodes.sol

16.2 Constant Definitions

```
4     uint8 constant ERROR_MSG_SENDER_IS_NOT_OWNER = 100;

5     uint8 constant ERROR_MSG_SENDER_IS_NOT_ROOT = 101;

7     uint8 constant ERROR_MSG_VALUE_IS_TOO_LOW = 110;

9     uint8 constant ERROR_INVALID_CONTRACT_TYPE = 200;
```

Chapter 17

Library TONTokenWalletConstants

Contents

17.1 Overview

In file TONTokenWalletConstants.sol

17.2 Constant Definitions

```
4     uint128 constant target_gas_balance = 0.05 ton;
```

Chapter 18

Library TONTokenWalletErrors

Contents

18.1 Overview

In file TONTokenWalletErrors.sol

18.2 Constant Definitions

```
4     uint8 constant error_message_sender_is_not_my_owner
          = 100;

5     uint8 constant error_not_enough_balance
          = 101;

6     uint8 constant error_message_sender_is_not_my_root
          = 102;

7     uint8 constant error_message_sender_is_not_good_wallet
          = 103;

8     uint8 constant error_wrong_bounced_header
          = 104;

9     uint8 constant error_wrong_bounced_args
          = 105;
```

```
10      uint8 constant error_non_zero_remaining
            = 106;

11      uint8 constant error_no_allowance_set
            = 107;

12      uint8 constant error_wrong_spender
            = 108;

13      uint8 constant error_not_enough_allowance
            = 109;

14      uint8 constant error_low_message_value
            = 110;

15      uint8 constant error_wrong_recipient
            = 111;

16      uint8 constant error_recipient_has_disallow_non_notifiable
            = 112;
```

Chapter 19

Library TvmCellOperations

Contents

19.1 Overview

In file `TvmCellOperations.sol`

19.2 Internal Method Definitions

19.2.1 Function decodeOperation

- TODO

```
4     function decodeOperation(TvmCell input) internal pure returns(
          uint8 operationId, TvmSlice data) {
5         TvmSlice s = input.toSlice();
6         operationId = s.decode(uint8);
7         data = s.loadRefAsSlice();
8     }
```

19.2.2 Function encodeOperation

- TODO

```
10      function encodeOperation(uint8 operationId, TvmCell input)
            internal pure returns (TvmCell result) {
11          TvmBuilder builder;
12          builder.store(operationId);
13          builder.store(input);
14          result = builder.toCell();
15      }
```

Chapter 20

Library UFO

Contents

20.1 Overview

In file `FloatingPointOperations.sol`

20.2 Internal Method Definitions

20.2.1 Function numAdd

- TODO

```
114     function numAdd(uint256 a, fraction b) internal pure returns (
            uint256) {
115         return (a*b.denom + b.nom) / b.denom;
116     }
```

20.2.2 Function numFDiv

- TODO

```
110     function numFDiv(uint256 a, fraction b) internal pure returns (
            fraction) {
111         return fraction(a * b.denom, b.nom);
112     }
```

20.2.3 Function numFMul

- TODO

```
106     function numFMul(uint256 a, fraction b) internal pure returns (
            fraction) {
107         return fraction(a * b.nom, b.denom);
108     }
```

20.2.4 Function numMul

- TODO

```
102     function numMul(uint256 a, fraction b) internal pure returns (
            uint256) {
103         return a*b.nom/b.denom;
104     }
```

20.2.5 Function numSub

- TODO

```
118     function numSub(uint256 a, fraction b) internal pure returns (
            uint256) {
119         return (a * b.denom - b.nom)/b.denom;
120     }
```

20.2.6 Function toF

- TODO

```
122     function toF(uint256 num) internal pure returns(fraction) {
123         return fraction(num, 1);
124     }
```

Chapter 21

Library UserAccountCostConstants

Contents

21.1 Overview

In file CostConstants.sol

21.2 Constant Definitions

```
4     uint128 constant useForUADeploy = 1 ton;
5     uint128 constant estimatedExecCost = 0.3 ton;
6     uint128 constant updateHealthCost = 1 ton;
```

Chapter 22

Library UserAccountErrorCodes

Contents

22.1 Overview

In file `UserAccountErrorCodes.sol`

22.2 Constant Definitions

```
4     uint8 constant ERROR_NOT_ROOT = 102;

6     uint8 constant ERROR_INVALID_CONTRACT_TYPE = 200;

8     uint8 constant ERROR_NOT_APPROVED_MARKET = 104;

9     uint8 constant ERROR_NOT_ENTERED_MARKET = 105;

11    uint8 constant ERROR_NOT_MARKET = 106;

12    uint8 constant ERROR_NOT_TRUSTED = 107;

13    uint8 constant ERROR_NOT_MODULE = 108;

14    uint8 constant ERROR_NOT_EXECUTOR = 109;

15    uint8 constant ERROR_INVALID_MODULE = 110;
```

```
16      uint8 constant ERROR_INVALID_EXECUTOR = 111;
17      uint8 constant INVALID_USER_ACCOUNT = 112;
```

Chapter 23

Library Utilities

Contents

23.1 Overview

In file `IModule.sol`

23.2 Internal Method Definitions

23.2.1 Function calculateSupplyBorrow

- TODO

```
90      function calculateSupplyBorrow(
91          mapping(uint32 => uint256) supplyInfo,
92          mapping(uint32 => BorrowInfo) borrowInfo,
93          mapping(uint32 => MarketInfo) marketInfo,
94          mapping(address => fraction) tokenPrices
95      ) internal returns (fraction) {
96          fraction accountHealth = fraction(0, 0);
97          fraction tmp;
98          fraction nom = fraction(0, 1);
99          fraction denom = fraction(0, 1);
100
101         // Supply:
102         // 1. Calculate real token amount: vToken*exchangeRate
103         // 2. Calculate real token amount in USD: realTokens/
                tokenPrice
```

```
104         // 3. Multiply by collateral factor: usdValue*
                collateralFactor
105         for ((uint32 marketId, uint256 supplied): supplyInfo) {
106             tmp = supplied.numFMul(marketInfo[marketId].
                    exchangeRate);
107             tmp = tmp.fDiv(tokenPrices[marketInfo[marketId].token])
                    ;
108             tmp = tmp.fMul(marketInfo[marketId].collateralFactor);
109             nom = nom.fAdd(tmp);
110             nom = nom.simplify();
111         }
112
113         // Borrow:
114         // 1. Recalculate borrow amount according to new index
115         // 2. Calculate borrow value in USD
116         // NOTE: no conversion from vToken to real tokens required,
                as value is stored in real tokens
117         for ((uint32 marketId, BorrowInfo _bi): borrowInfo) {
118             if (_bi.tokensBorrowed != 0) {
119                 if (!_bi.index.eq(marketInfo[marketId].index)) {
120                     tmp = borrowInfo[marketId].tokensBorrowed.
                            numFMul(marketInfo[marketId].index);
121                     tmp = tmp.fDiv(borrowInfo[marketId].index);
122                 } else {
123                     tmp = borrowInfo[marketId].tokensBorrowed.toF()
                            ;
124                 }
125                 tmp = tmp.fDiv(tokenPrices[marketInfo[marketId].
                        token]);
126                 tmp = tmp.simplify();
127                 denom = denom.fAdd(tmp);
128                 denom = denom.simplify();
129             }
130         }
131
132         accountHealth = nom.fDiv(denom);
133
134         return accountHealth;
135     }
```

23.2.1.0.1 Some functions inherited by using

Chapter 24

Library WCCostConstants

Contents

24.1 Overview

In file `CostConstants.sol`

24.2 Constant Definitions

```
4     uint128 constant WALLET_DEPLOY_COST = 2 ton;
5     uint128 constant WALLET_DEPLOY_GRAMS = 1.5 ton;
6     uint128 constant GET_WALLET_ADDRESS = 1 ton;
7     uint128 constant SET_RECEIVE_CALLBACK = 0.5 ton;
```

Chapter 25

Library WalletControllerErrorCodes

Contents

25.1 Overview

In file `WalletControllerErrorCodes.sol`

25.2 Constant Definitions

```
4     uint8 constant ERROR_MSG_SENDER_IS_NOT_ROOT = 100;

5     uint8 constant ERROR_MSG_SENDER_IS_NOT_MARKET = 101;

6     uint8 constant ERROR_MSG_SENDER_IS_NOT_OWN_WALLET = 102;

7     uint8 constant ERROR_TIP3_ROOT_IS_UNKNOWN = 103;

9     uint8 constant ERROR_INVALID_CONTRACT_TYPE = 200;
```

Chapter 26

Contract BorrowModule

Contents

26.1 Overview

In file `BorrowModule.sol`

26.2 Contract Inheritance

IRoles	
IModule	
IContractStateCache	
IContractAddressSG	
IBorrowModule	
IUpgradableContract	

26.3 Event Definitions

```
16      event TokenBorrow(uint32 marketId, MarketDelta marketDelta,
            address tonWallet, uint256 tokensBorrowed);
```

26.4 Variable Definitions

address	marketAddress	
		used in @6.BorrowModule.upgradeContractCode
		assigned in @6.BorrowModule.setMarketAddress
		used in @6.BorrowModule.setMarketAddress
		assigned in @6.BorrowModule.onCodeUpgrade
		used in @6.BorrowModule.onCodeUpgrade
		used in @6.BorrowModule.getContractAddresses
		used in @6.BorrowModule.borrowTokensFromMarket
address	userAccountManager	
		used in @6.BorrowModule.upgradeContractCode
		assigned in @6.BorrowModule.setUserAccountManager
		used in @6.BorrowModule.setUserAccountManager
		used in @6.BorrowModule.resumeOperation
		used in @6.BorrowModule.performAction
		assigned in @6.BorrowModule.onCodeUpgrade
		used in @6.BorrowModule.onCodeUpgrade
		used in @6.BorrowModule.getContractAddresses
		used in @6.BorrowModule.borrowTokensFromMarket
		used in @6.BorrowModule.borrowTokensFromMarket
uint32	contractCodeVersion	
		assigned in @6.BorrowModule.onCodeUpgrade
		used in @6.BorrowModule.onCodeUpgrade
mapping (uint32 => MarketInfo)	marketInfo	
		used in @6.BorrowModule.upgradeContractCode
		assigned in @6.BorrowModule.updateCache
		used in @6.BorrowModule.updateCache
		used in @6.BorrowModule.resumeOperation
		assigned in @6.BorrowModule.resumeOperation

```
    address marketAddress;

    address userAccountManager;

    uint32 public contractCodeVersion;

    mapping (uint32 => MarketInfo) marketInfo;

    mapping (address => fraction) tokenPrices;
```

26.5 Modifier Definitions

26.5.1 Modifier onlyUserAccountManager

```
    modifier onlyUserAccountManager() {
        require(msg.sender == userAccountManager);
        _;
    }
```

26.5.2 Modifier onlyMarket

```
    modifier onlyMarket() {
        require(msg.sender == marketAddress);
        tvm.rawReserve(msg.value, 2);
        _;
    }
```

26.6 Constructor Definitions

26.6.1 Constructor

Critical issue: Constructor for BorrowModule (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

26.7 Public Method Definitions

26.7.1 Function borrowTokensFromMarket

- TODO

```
105     function borrowTokensFromMarket(
106         address tonWallet,
107         address userTip3Wallet,
108         uint256 tokensToBorrow,
109         uint32 marketId,
110         mapping (uint32 => uint256) supplyInfo,
111         mapping (uint32 => BorrowInfo) borrowInfo
112     ) external override onlyUserAccountManager {
113         tvm.rawReserve(msg.value, 2);
114         mapping(uint32 => MarketDelta) marketsDelta;
115         MarketDelta marketDelta;
116 
117         // Borrow:
118         // 1. Check that market has enough tokens for lending
119         // 2. Calculate user account health
120         // 3. Calculate USD value of tokens to borrow
121         // 4. Check if there is enough (collateral - borrowed) for
                new token borrow
122         // 5. Increase user's borrowed amount
123 
124         if (tokensToBorrow < marketInfo[marketId].realTokenBalance
                - marketInfo[marketId].totalReserve) {
125             fraction accountHealth = Utilities.
                    calculateSupplyBorrow(supplyInfo, borrowInfo,
                    marketInfo, tokenPrices);
126             if (accountHealth.nom > accountHealth.denom) {
127                 uint256 healthDelta = accountHealth.nom -
                        accountHealth.denom;
128                 fraction tmp = healthDelta.numFMul(tokenPrices[
                        marketInfo[marketId].token]);
129                 uint256 possibleTokenWithdraw = tmp.toNum();
130                 if (possibleTokenWithdraw >= tokensToBorrow) {
131                     marketDelta.totalBorrowed.delta =
                            tokensToBorrow;
132                     marketDelta.totalBorrowed.positive = true;
133                     marketDelta.realTokenBalance.delta =
                            tokensToBorrow;
134                     marketDelta.realTokenBalance.positive = false;
135 
136                     marketsDelta[marketId] = marketDelta;
137 
138                     TvmBuilder tb;
139                     tb.store(marketId);
140                     tb.store(tonWallet);
141                     tb.store(userTip3Wallet);
142                     tb.store(tokensToBorrow);
143 
144                     emit TokenBorrow(marketId, marketDelta,
                            tonWallet, tokensToBorrow);
145 
```

```
146                     IContractStateCacheRoot(marketAddress).
                            receiveCacheDelta{
147                         flag: MsgFlag.REMAINING_GAS
148                     }(marketsDelta, tb.toCell());
149                 } else {
150                     IUAMUserAccount(userAccountManager).
                            writeBorrowInformation{
151                         flag: MsgFlag.REMAINING_GAS
152                     }(tonWallet, userTip3Wallet, 0, marketId,
                            marketInfo[marketId].index);
153                 }
154             } else {
155                 IUAMUserAccount(userAccountManager).
                        writeBorrowInformation{
156                     flag: MsgFlag.REMAINING_GAS
157                 }(tonWallet, userTip3Wallet, 0, marketId,
                        marketInfo[marketId].index);
158             }
159         } else {
160             address(tonWallet).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
161         }
162     }
```

26.7.2 Function getContractAddresses

- TODO

```
77     function getContractAddresses() external override view
           responsible returns(address _owner, address _marketAddress,
            address _userAccountManager) {
78         return {flag: MsgFlag.REMAINING_GAS} (_owner, marketAddress
               , userAccountManager);
79     }
```

26.7.3 Function getModuleState

- TODO

```
61     function getModuleState() external override view returns(
           mapping(uint32 => MarketInfo), mapping(address => fraction)
           ) {
62         return(marketInfo, tokenPrices);
63     }
```

26.7.4 Function performAction

- TODO

```
87      function performAction(uint32 marketId, TvmCell args, mapping (
            uint32 => MarketInfo) _marketInfo, mapping (address =>
            fraction) _tokenPrices) external override onlyMarket {
88          tvm.rawReserve(msg.value, 2);
89          marketInfo = _marketInfo;
90          tokenPrices = _tokenPrices;
91          TvmSlice ts = args.toSlice();
92          (address tonWallet, address userTip3Wallet, uint256
                tokensToBorrow) = ts.decode(address, address, uint256);
93          mapping(uint32 => fraction) updatedIndexes =
                _createUpdatedIndexes();
94          IUAMUserAccount(userAccountManager).updateUserIndexes{
95              flag: MsgFlag.REMAINING_GAS
96          }(tonWallet, userTip3Wallet, tokensToBorrow, marketId,
                updatedIndexes);
97      }
```

26.7.5 Function resumeOperation

- TODO

```
164     function resumeOperation(TvmCell args, mapping(uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
165         tvm.rawReserve(msg.value, 2);
166         marketInfo = _marketInfo;
167         tokenPrices = _tokenPrices;
168         TvmSlice ts = args.toSlice();
169         (uint32 marketId, address tonWallet, address userTip3Wallet
                , uint256 tokensToBorrow) = ts.decode(uint32, address,
                address, uint256);
170         IUAMUserAccount(userAccountManager).writeBorrowInformation{
171             flag: MsgFlag.REMAINING_GAS
172         }(tonWallet, userTip3Wallet, tokensToBorrow, marketId,
                marketInfo[marketId].index);
173     }
```

26.7.6 Function sendActionId

- TODO

```
57      function sendActionId() external override view responsible
            returns(uint8) {
58          return {flag: MsgFlag.REMAINING_GAS} OperationCodes.
                BORROW_TOKENS;
59      }
```

26.7.7 Function setMarketAddress

- TODO

```
65      function setMarketAddress(address _marketAddress) external
            override canChangeParams {
66          tvm.rawReserve(msg.value, 2);
67          marketAddress = _marketAddress;
68          address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
69      }
```

26.7.8 Function setUserAccountManager

- TODO

```
71      function setUserAccountManager(address _userAccountManager)
            external override canChangeParams {
72          tvm.rawReserve(msg.value, 2);
73          userAccountManager = _userAccountManager;
74          address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
75      }
```

26.7.9 Function updateCache

- TODO

```
81      function updateCache(address tonWallet, mapping (uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
82          marketInfo = _marketInfo;
83          tokenPrices = _tokenPrices;
84          tonWallet.transfer({value: 0, flag: MsgFlag.REMAINING_GAS})
                ;
85      }
```

26.7.10 Function upgradeContractCode

- TODO

```
23      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) external override canUpgrade {
24          tvm.rawReserve(msg.value, 2);
25
26          tvm.setcode(code);
27          tvm.setCurrentCode(code);
28
29          onCodeUpgrade (
30              _owner,
31              marketAddress,
32              userAccountManager,
33              marketInfo,
34              tokenPrices,
```

```
35              codeVersion
36          );
37      }
```

26.8 Internal Method Definitions

26.8.1 Function _createUpdatedIndexes

- TODO

```
99      function _createUpdatedIndexes() internal view returns(mapping(
            uint32 => fraction) updatedIndexes) {
100         for ((uint32 marketId, MarketInfo mi): marketInfo) {
101             updatedIndexes[marketId] = mi.index;
102         }
103     }
```

26.8.2 Function onCodeUpgrade

- TODO

```
39      function onCodeUpgrade(
40          address owner,
41          address _marketAddress,
42          address _userAccountManager,
43          mapping(uint32 => MarketInfo) _marketInfo,
44          mapping(address => fraction) _tokenPrices,
45          uint32 _codeVersion
46      ) private {
47          tvm.accept();
48          tvm.resetStorage();
49          _owner = owner;
50          marketAddress = _marketAddress;
51          userAccountManager = _userAccountManager;
52          marketInfo = _marketInfo;
53          tokenPrices = _tokenPrices;
54          contractCodeVersion = _codeVersion;
55      }
```

26.8.2.0.1 Some functions inherited by using

Chapter 27

Contract Giver

Contents

27.1 Overview

In file `Giver.sol`

27.2 Constructor Definitions

27.2.1 Constructor

Critical issue: Constructor for Giver (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum
loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor() public {
        tvm.accept();
    }
```

27.3 Public Method Definitions

27.3.1 Function sendGrams

- TODO

```
10      function sendGrams(address dest, uint64 amount) external pure {
11          tvm.accept();
12          address(dest).transfer({value: amount, bounce: false});
13      }
```

Chapter 28

Abstract Contract IRoles

Contents

28.1 Overview

In file `IRoles.sol`

28.2 Variable Definitions

address	_owner	
		used in @15.WalletController.upgradeContractCode
		assigned in @15.WalletController.onCodeUpgrade
		used in @15.WalletController.onCodeUpgrade
		assigned in @15.WalletController.:constructor
		used in @15.WalletController.:constructor
		used in @14.UserAccountManager.upgradeContractCode
		assigned in @14.UserAccountManager.onCodeUpgrade
		used in @14.UserAccountManager.onCodeUpgrade
		assigned in @14.UserAccountManager.:constructor
		used in @14.UserAccountManager.:constructor
		used in @11.Oracle.upgradeContractCode
		assigned in @11.Oracle.onCodeUpgrade
		used in @11.Oracle.onCodeUpgrade
		used in @11.Oracle.getDetails
		assigned in @11.Oracle.:constructor
		used in @11.Oracle.:constructor
		used in @10.WithdrawModule.upgradeContractCode
		used in @10.WithdrawModule.setUserAccountManager
		used in @10.WithdrawModule.setMarketAddress
		assigned in @10.WithdrawModule.onCodeUpgrade
		used in @10.WithdrawModule.onCodeUpgrade
		assigned in @10.WithdrawModule.:constructor
		used in @10.WithdrawModule.:constructor
		used in @9.SupplyModule.upgradeContractCode
		used in @9.SupplyModule.setUserAccountManager
		used in @9.SupplyModule.setMarketAddress
		assigned in @9.SupplyModule.onCodeUpgrade

```
13      address _owner;

14      mapping(address => bool) _canUpgrade;

15      mapping(address => bool) _canChangeParams;
```

28.3 Modifier Definitions

28.3.1 Modifier onlyOwner

```
62      modifier onlyOwner() {
63          require(
64              msg.sender == _owner,
65              RolesErrors.IS_NOT_OWNER
66          );
67          _;
68      }
```

28.3.2 Modifier canUpgrade

```
70      modifier canUpgrade() {
71          require(
72              _canUpgrade[msg.sender] ||
73              msg.sender == _owner,
74              RolesErrors.CANNOT_UPGRADE
75          );
76          _;
77      }
```

28.3.3 Modifier canChangeParams

```
79      modifier canChangeParams() {
80          require(
81              _canChangeParams[msg.sender] ||
82              msg.sender == _owner,
83              RolesErrors.CANNOT_CHANGE_PARAMS
84          );
85          _;
86      }
```

28.4 Public Method Definitions

28.4.1 Function changeOwner

- TODO

```
39      function changeOwner(address _newOwner) external onlyOwner {
40          tvm.rawReserve(msg.value, 2);
41
42          _owner = _newOwner;
```

```
43
44          address(msg.sender).transfer({
45              value: 0,
46              flag: MsgFlag.REMAINING_GAS
47          });
48      }
```

28.4.2 Function getOwner

- TODO

```
50      function getOwner() external view returns(address) {
51          return _owner;
52      }
```

28.4.3 Function getParamChangers

- TODO

```
58      function getParamChangers() external view returns(mapping(
            address => bool)) {
59          return _canChangeParams;
60      }
```

28.4.4 Function getUpgraders

- TODO

```
54      function getUpgraders() external view returns(mapping(address
            => bool)) {
55          return _canUpgrade;
56      }
```

28.4.5 Function setParamChanger

- TODO

```
28      function setParamChanger(address paramChanger, bool allowed)
            external onlyOwner {
29          tvm.rawReserve(msg.value, 2);
30
31          _canChangeParams[paramChanger] = allowed;
32
33          address(msg.sender).transfer({
34              value: 0,
35              flag: MsgFlag.REMAINING_GAS
36          });
37      }
```

28.4.6 Function setUpgrader

- TODO

```
17      function setUpgrader(address upgrader, bool allowed) external
            onlyOwner {
18          tvm.rawReserve(msg.value, 2);
19
20          _canUpgrade[upgrader] = allowed;
21
22          address(msg.sender).transfer({
23              value: 0,
24              flag: MsgFlag.REMAINING_GAS
25          });
26      }
```

Chapter 29

Contract LiquidationModule

Contents

29.1 Overview

In file `LiquidationModule.sol`

29.2 Contract Inheritance

IRoles	
IModule	
IContractStateCache	
IContractAddressSG	
ILiquidationModule	
IUpgradableContract	

29.3 Event Definitions

```
16      event TokensLiquidated(uint32 marketId, mapping(uint32 =>
            MarketDelta) marketDeltas, address liquidator, address
            targetUser, uint256 tokensLiquidated, uint256 vTokensSeized
            );
```

29.4 Variable Definitions

address	marketAddress	
		used in @7.LiquidationModule.upgradeContractCode
		assigned in @7.LiquidationModule.setMarketAddress
		used in @7.LiquidationModule.setMarketAddress
		assigned in @7.LiquidationModule.onCodeUpgrade
		used in @7.LiquidationModule.onCodeUpgrade
		used in @7.LiquidationModule.liquidate
		used in @7.LiquidationModule.getContractAddresses
address	userAccountManager	
		used in @7.LiquidationModule.upgradeContractCode
		assigned in @7.LiquidationModule.setUserAccountManager
		used in @7.LiquidationModule.setUserAccountManager
		used in @7.LiquidationModule.resumeOperation
		used in @7.LiquidationModule.performAction
		assigned in @7.LiquidationModule.onCodeUpgrade
		used in @7.LiquidationModule.onCodeUpgrade
		used in @7.LiquidationModule.liquidate
		used in @7.LiquidationModule.liquidate
		used in @7.LiquidationModule.getContractAddresses
uint32	contractCodeVersion	
		assigned in @7.LiquidationModule.onCodeUpgrade
		used in @7.LiquidationModule.onCodeUpgrade
mapping (uint32 => MarketInfo)	marketInfo	
		used in @7.LiquidationModule.upgradeContractCode
		assigned in @7.LiquidationModule.updateCache
		used in @7.LiquidationModule.updateCache
		assigned in @7.LiquidationModule.resumeOperation
		used in @7.LiquidationModule.resumeOperation

```
    address marketAddress;

    address userAccountManager;

    uint32 public contractCodeVersion;

    mapping (uint32 => MarketInfo) marketInfo;

    mapping (address => fraction) tokenPrices;
```

29.5 Modifier Definitions

29.5.1 Modifier onlyUserAccountManager

```
    modifier onlyUserAccountManager() {
        require(msg.sender == userAccountManager);
        _;
    }
```

29.5.2 Modifier onlyMarket

```
    modifier onlyMarket() {
        require(msg.sender == marketAddress);
        tvm.rawReserve(msg.value, 2);
        _;
    }
```

29.6 Constructor Definitions

29.6.1 Constructor

Critical issue: Constructor for LiquidationModule (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

29.7 Public Method Definitions

29.7.1 Function getContractAddresses

- TODO

```
77      function getContractAddresses() external override view
            responsible returns(address _owner, address _marketAddress,
             address _userAccountManager) {
78          return {flag: MsgFlag.REMAINING_GAS} (_owner, marketAddress
                , userAccountManager);
79      }
```

29.7.2 Function getModuleState

- TODO

```
61      function getModuleState() external override view returns(
            mapping(uint32 => MarketInfo), mapping(address => fraction)
            ) {
62          return(marketInfo, tokenPrices);
63      }
```

29.7.3 Function liquidate

- TODO

```
101     function liquidate(
102         address tonWallet,
103         address targetUser,
104         address tip3UserWallet,
105         uint32 marketId,
106         uint32 marketToLiquidate,
107         uint256 tokensProvided,
108         mapping(uint32 => uint256) supplyInfo,
109         mapping(uint32 => BorrowInfo) borrowInfo
110     ) external override onlyUserAccountManager {
111         tvm.rawReserve(msg.value, 2);
112         // Liquidation:
113         // 1. Calculate user account health to check if liquidation
                is required
114         // 2. Calculate max values
115         // 3. Choose minimal value of all max values
116         // 4. Based on min value calculate rest of parameters, it
                is guaranteed that:
117         // - User will not exceed tokens that he provided for
                liquidation (providingLimit)
118         // - User will not exceed tokens that are available for
                liquidation (borrowLimit)
```

```
119         // - User will not exceed vToken balance of user that is
                liquidated (vTokenLimit)
120
121         fraction health = Utilities.calculateSupplyBorrow(
                supplyInfo, borrowInfo, marketInfo, tokenPrices);
122         if (health.nom <= health.denom) {
123             uint256 tokensToLiquidate = math.min(
124                 borrowInfo[marketId].tokensBorrowed,
125                 tokensProvided
126             );
127
128             // Calculating USD value of liquidation
129             fraction ftokensToLiquidateUSD = tokensToLiquidate.
                    numFMul(marketInfo[marketId].liquidationMultiplier)
                    ;
130             ftokensToLiquidateUSD = ftokensToLiquidateUSD.fDiv(
                    tokenPrices[marketInfo[marketId].token]);
131
132             // Calculating USD value of collateral
133             fraction fvTokensCollateralUSD = supplyInfo[
                    marketToLiquidate].numFMul(marketInfo[
                    marketToLiquidate].exchangeRate);
134             fvTokensCollateralUSD = fvTokensCollateralUSD.fDiv(
                    tokenPrices[marketInfo[marketToLiquidate].token]);
135
136             uint256 tokensToSeize;
137             uint256 tokensToReturn;
138             uint256 tokensFromReserve;
139
140             // Calculating how much of collateral tokens to seize
141             fraction fvTokensCollateral = fvTokensCollateralUSD.
                    getMin(ftokensToLiquidateUSD);
142             fraction ftokensToSeize = fvTokensCollateral.fMul(
                    tokenPrices[marketInfo[marketToLiquidate].token]);
143             ftokensToSeize = ftokensToSeize.fDiv(marketInfo[
                    marketToLiquidate].exchangeRate);
144             tokensToSeize = ftokensToSeize.toNum();
145
146             tokensToReturn = tokensProvided - tokensToLiquidate;
147             mapping(uint32 => MarketDelta) marketDeltas;
148             MarketDelta collateralMarketDelta;
149             MarketDelta liquidationMarketDelta;
150
151             liquidationMarketDelta.totalBorrowed.delta =
                    tokensToLiquidate;
152             liquidationMarketDelta.totalBorrowed.positive = false;
153             liquidationMarketDelta.realTokenBalance.delta =
                    tokensToLiquidate;
154             liquidationMarketDelta.realTokenBalance.positive = true
                    ;
155
156             if (fvTokensCollateralUSD.lessThan(
                    ftokensToLiquidateUSD)) {
157                 // Using reserves from market to compensate
                        liquidity absence
158                 fraction freservesUsageUSD = ftokensToLiquidateUSD.
                        fSub(fvTokensCollateralUSD);
```

```
159                 freservesUsageUSD = freservesUsageUSD.simplify();
160                 fraction freservesUsageTokens = freservesUsageUSD.
                        fMul(tokenPrices[marketInfo[marketToLiquidate].
                        token]);
161                 uint256 reservesUsageTokens = freservesUsageTokens.
                        toNum();
162                 if (reservesUsageTokens < marketInfo[marketId].
                        totalReserve) {
163                     tokensFromReserve = reservesUsageTokens;
164                     collateralMarketDelta.totalReserve.delta =
                            tokensFromReserve;
165                     collateralMarketDelta.totalReserve.positive =
                            false;
166                 } else {
167                     // abort liquidation
168                     IUAMUserAccount(userAccountManager).
                            requestTokenPayout{
169                         flag: MsgFlag.REMAINING_GAS
170                     }(
171                         tonWallet, tip3UserWallet, marketId,
                                tokensProvided
172                     );
173                     tvm.exit();
174                 }
175             }
176
177             marketDeltas[marketId] = liquidationMarketDelta;
178             marketDeltas[marketToLiquidate] = collateralMarketDelta
                    ;
179
180             emit TokensLiquidated(marketId, marketDeltas, tonWallet
                    , targetUser, tokensToLiquidate, tokensToSeize);
181
182             BorrowInfo userBorrowInfo = BorrowInfo(borrowInfo[
                    marketId].tokensBorrowed - tokensToLiquidate,
                    marketInfo[marketId].index);
183
184             TvmBuilder tb;
185             TvmBuilder addressStorage;
186             addressStorage.store(tonWallet);
187             addressStorage.store(targetUser);
188             addressStorage.store(tip3UserWallet);
189             TvmBuilder valueStorage;
190             valueStorage.store(marketId);
191             valueStorage.store(marketToLiquidate);
192             valueStorage.store(tokensToSeize);
193             valueStorage.store(tokensToReturn);
194             valueStorage.store(tokensFromReserve);
195             TvmBuilder borrowInfoStorage;
196             borrowInfoStorage.store(userBorrowInfo);
197             tb.store(addressStorage.toCell());
198             tb.store(valueStorage.toCell());
199             tb.store(borrowInfoStorage.toCell());
200
201             IContractStateCacheRoot(marketAddress).
                    receiveCacheDelta{
202                 flag: MsgFlag.REMAINING_GAS
```

```
203             }(marketDeltas, tb.toCell());
204         } else {
205             IUAMUserAccount(userAccountManager).requestTokenPayout{
206                 flag: MsgFlag.REMAINING_GAS
207             }(
208                 tonWallet, tip3UserWallet, marketId, tokensProvided
209             );
210         }
211     }
```

29.7.4 Function performAction

- TODO

```
87      function performAction(uint32 marketId, TvmCell args, mapping (
            uint32 => MarketInfo) _marketInfo, mapping (address =>
            fraction) _tokenPrices) external override onlyMarket {
88          tvm.rawReserve(msg.value, 2);
89          marketInfo = _marketInfo;
90          tokenPrices = _tokenPrices;
91          TvmSlice ts = args.toSlice();
92          (address tonWallet, address targetUser, address
                tip3UserWallet) = ts.decode(address, address, address);
93          TvmSlice amountTS = ts.loadRefAsSlice();
94          (uint32 marketToLiquidate, uint256 tokenAmount) = amountTS.
                decode(uint32, uint256);
95          mapping(uint32 => fraction) updatedIndexes =
                _createUpdatedIndexes();
96          IUAMUserAccount(userAccountManager).
                requestLiquidationInformation{
97              flag: MsgFlag.REMAINING_GAS
98          }(tonWallet, targetUser, tip3UserWallet, marketId,
                marketToLiquidate, tokenAmount, updatedIndexes);
99      }
```

29.7.5 Function resumeOperation

- TODO

```
213     function resumeOperation(TvmCell args, mapping(uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
214         tvm.rawReserve(msg.value, 2);
215         marketInfo = _marketInfo;
216         tokenPrices = _tokenPrices;
217         TvmSlice ts = args.toSlice();
218         TvmSlice addressStorage = ts.loadRefAsSlice();
219         (address tonWallet, address targetUser, address
                tip3UserWallet) = addressStorage.decode(address,
                address, address);
220         TvmSlice valueStorage = ts.loadRefAsSlice();
```

```
221         (uint32 marketId, uint32 marketToLiquidate, uint256
                tokensToSeize, uint256 tokensToReturn, uint256
                tokensFromReserve) = valueStorage.decode(uint32, uint32
                , uint256, uint256, uint256);
222         TvmSlice borrowInfoStorage = ts.loadRefAsSlice();
223         (BorrowInfo borrowInfo) = borrowInfoStorage.decode(
                BorrowInfo);
224         IUAMUserAccount(userAccountManager).seizeTokens{
225             flag: MsgFlag.REMAINING_GAS
226         }(tonWallet, targetUser, tip3UserWallet, marketId,
                marketToLiquidate, tokensToSeize, tokensToReturn,
                tokensFromReserve, borrowInfo);
227     }
```

29.7.6 Function sendActionId

- TODO

```
57     function sendActionId() external override view responsible
           returns(uint8) {
58         return {flag: MsgFlag.REMAINING_GAS} OperationCodes.
               LIQUIDATE_TOKENS;
59     }
```

29.7.7 Function setMarketAddress

- TODO

```
65     function setMarketAddress(address _marketAddress) external
           override canChangeParams {
66         tvm.rawReserve(msg.value, 2);
67         marketAddress = _marketAddress;
68         address(_owner).transfer({value: 0, flag: MsgFlag.
               REMAINING_GAS});
69     }
```

29.7.8 Function setUserAccountManager

- TODO

```
71     function setUserAccountManager(address _userAccountManager)
           external override canChangeParams {
72         tvm.rawReserve(msg.value, 2);
73         userAccountManager = _userAccountManager;
74         address(_owner).transfer({value: 0, flag: MsgFlag.
               REMAINING_GAS});
75     }
```

29.7.9 Function updateCache

- TODO

```
81      function updateCache(address tonWallet, mapping (uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
82          marketInfo = _marketInfo;
83          tokenPrices = _tokenPrices;
84          tonWallet.transfer({value: 0, flag: MsgFlag.REMAINING_GAS})
                ;
85      }
```

29.7.10 Function upgradeContractCode

- TODO

```
23      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) external override canUpgrade {
24          tvm.rawReserve(msg.value, 2);
25
26          tvm.setcode(code);
27          tvm.setCurrentCode(code);
28
29          onCodeUpgrade (
30              _owner,
31              marketAddress,
32              userAccountManager,
33              marketInfo,
34              tokenPrices,
35              codeVersion
36          );
37      }
```

29.8 Internal Method Definitions

29.8.1 Function _createUpdatedIndexes

- TODO

```
229     function _createUpdatedIndexes() internal view returns(mapping(
            uint32 => fraction) updatedIndexes) {
230         for ((uint32 marketId, MarketInfo mi): marketInfo) {
231             updatedIndexes[marketId] = mi.index;
232         }
233     }
```

29.8.2 Function onCodeUpgrade

- TODO

```
39      function onCodeUpgrade(
40          address owner,
41          address _marketAddress,
42          address _userAccountManager,
43          mapping(uint32 => MarketInfo) _marketInfo,
44          mapping(address => fraction) _tokenPrices,
45          uint32 _codeVersion
46      ) private {
47          tvm.accept();
48          tvm.resetStorage();
49          _owner = owner;
50          marketAddress = _marketAddress;
51          userAccountManager = _userAccountManager;
52          marketInfo = _marketInfo;
53          tokenPrices = _tokenPrices;
54          contractCodeVersion = _codeVersion;
55      }
```

29.8.2.0.1 Some functions inherited by using

Chapter 30

Contract MarketAggregator

Contents

30.1 Overview

In file `MarketsAggregator.sol`

30.2 Contract Inheritance

IRoles	
IUpgradableContract	
IMarketOracle	
IMarketSetters	
IMarketOwnerFunctions	
IMarketGetters	
IMarketOperations	
IContractStateCacheRoot	

30.3 Event Definitions

```
32     event MarketCreated(uint32 marketId, MarketInfo marketState);

33     event MarketDeleted(uint32 marketId, MarketInfo marketState);

34     event LiquidationPossible(address tonWallet, fraction
           accountHealth);
```

30.4 Variable Definitions

uint32	contractCodeVersion	
		assigned in @2.MarketAggregator.onCodeUpgrade
		used in @2.MarketAggregator.onCodeUpgrade
address	userAccountManager	
		used in @2.MarketAggregator.upgradeContractCode
		assigned in @2.MarketAggregator.setUserAccountManager
		used in @2.MarketAggregator.setUserAccountManager
		assigned in @2.MarketAggregator.onCodeUpgrade
		used in @2.MarketAggregator.onCodeUpgrade
		used in @2.MarketAggregator.getServiceContractAddresses
		used in @2.MarketAggregator.calculateUserAccountHealth
address	walletController	
		used in @2.MarketAggregator.upgradeContractCode
		assigned in @2.MarketAggregator.setWalletController
		used in @2.MarketAggregator.setWalletController
		used in @2.MarketAggregator.requestTokenPayout
		assigned in @2.MarketAggregator.onCodeUpgrade
		used in @2.MarketAggregator.onCodeUpgrade
		used in @2.MarketAggregator.getServiceContractAddresses
address	oracle	
		used in @2.MarketAggregator.upgradeContractCode
		used in @2.MarketAggregator.updatePrice
		assigned in @2.MarketAggregator.setOracleAddress
		used in @2.MarketAggregator.setOracleAddress
		assigned in @2.MarketAggregator.onCodeUpgrade
		used in @2.MarketAggregator.onCodeUpgrade
		used in @2.MarketAggregator.getServiceContractAddresses
		used in @2.MarketAggregator._updateAllPrices
mapping (uint32 => bool)	createdMarkets	

```
14      uint32 public contractCodeVersion;

16      address public userAccountManager;

17      address public walletController;

18      address public oracle;

19      mapping(uint32 => bool) createdMarkets;

20      mapping(address => uint32) tokensToMarkets;

21      mapping(uint32 => MarketInfo) markets;

22      mapping(address => fraction) tokenPrices;

23      mapping(address => bool) realTokenRoots;

25      mapping(uint8 => address) public modules;

26      uint128 moduleAmount;

27      mapping(address => bool) isModule;
```

30.5 Modifier Definitions

30.5.1 Modifier onlySelf

```
510     modifier onlySelf() {
511         require(msg.sender == address(this), MarketErrorCodes.
                ERROR_MSG_SENDER_IS_NOT_SELF);
512         _;
513     }
```

30.5.2 Modifier onlyOracle

```
515     modifier onlyOracle() {
516         require(msg.sender == oracle, MarketErrorCodes.
                ERROR_MSG_SENDER_IS_NOT_ORACLE);
517         tvm.rawReserve(msg.value, 2);
518         _;
519     }
```

30.5.3 Modifier onlyUserAccountManager

```
521     modifier onlyUserAccountManager() {
522         require(msg.sender == userAccountManager, MarketErrorCodes.
                ERROR_MSG_SENDER_IS_NOT_USER_ACCOUNT_MANAGER);
523         tvm.rawReserve(msg.value, 2);
524         _;
525     }
```

30.5.4 Modifier onlyWalletController

```
    modifier onlyWalletController() {
        require(msg.sender == walletController, MarketErrorCodes.
            ERROR_MSG_SENDER_IS_NOT_TIP3_WALLET_CONTROLLER);
        tvm.rawReserve(msg.value, 2);
        _;
    }
```

30.5.5 Modifier onlyRealTokenRoot

```
    modifier onlyRealTokenRoot() {
        require(realTokenRoots.exists(msg.sender));
        _;
    }
```

30.5.6 Modifier onlyModule

```
    modifier onlyModule() {
        require(isModule.exists(msg.sender));
        _;
    }
```

30.5.7 Modifier onlyExecutor

```
    modifier onlyExecutor() {
        require(
            (msg.sender == userAccountManager) ||
            (isModule.exists(msg.sender))
        );
        _;
    }
```

30.6 Constructor Definitions

30.6.1 Constructor

Critical issue: Constructor for MarketAggregator (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

30.7 Public Method Definitions

30.7.1 Function addModule

- TODO

```
332     function addModule(uint8 operationId, address module) external
            canChangeParams {
333         tvm.rawReserve(msg.value, 2);
334         modules[operationId] = module;
335         isModule[module] = true;
336         moduleAmount = moduleAmount + 1;
337         IContractStateCache(module).updateCache{
338             flag: MsgFlag.REMAINING_GAS
339         }(_owner, markets, tokenPrices);
340     }
```

30.7.2 Function calculateUserAccountHealth

- TODO

```
376     function calculateUserAccountHealth(
377         address tonWallet,
378         address gasTo,
379         mapping(uint32 => uint256) supplyInfo,
380         mapping(uint32 => BorrowInfo) borrowInfo,
381         TvmCell dataToTransfer
382     ) external override onlyUserAccountManager {
383         tvm.rawReserve(msg.value, 2);
384 
385         _updateAllMarkets();
386 
387         mapping(uint32 => fraction) updatedIndexes =
                _createUpdatedIndexes();
388         fraction accountHealth = Utilities.calculateSupplyBorrow(
                supplyInfo, borrowInfo, markets, tokenPrices);
389 
390         if (accountHealth.nom < accountHealth.denom) {
391             emit LiquidationPossible(tonWallet, accountHealth);
392         }
393 
394         IUAMUserAccount(userAccountManager).updateUserAccountHealth
                {
395             flag: MsgFlag.REMAINING_GAS
396         }(tonWallet, gasTo, accountHealth, updatedIndexes,
                dataToTransfer);
397     }
```

30.7.3 Function createNewMarket

- TODO

```
242     function createNewMarket(
243         uint32 marketId,
244         address realToken,
245         fraction _baseRate,
246         fraction _utilizationMultiplier,
247         fraction _reserveFactor,
248         fraction _exchangeRate,
249         fraction _collateralFactor,
250         fraction _liquidationMultiplier
251     ) external canChangeParams {
252         tvm.rawReserve(msg.value, 2);
253         if (!createdMarkets[marketId]) {
254             createdMarkets[marketId] = true;
255 
256             fraction one = fraction({nom: 1, denom: 1});
257 
258             markets[marketId] = MarketInfo({
259                 token: realToken,
260                 realTokenBalance: 0,
261                 vTokenBalance: 0,
262                 totalBorrowed: 0,
263                 totalReserve: 0,
264 
265                 index: one,
266                 baseRate: _baseRate,
267                 utilizationMultiplier: _utilizationMultiplier,
268                 reserveFactor: _reserveFactor,
269                 exchangeRate: _exchangeRate,
270                 collateralFactor: _collateralFactor,
271                 liquidationMultiplier: _liquidationMultiplier,
272 
273                 lastUpdateTime: now
274             });
275 
276             tokensToMarkets[realToken] = marketId;
277 
278             emit MarketCreated(marketId, markets[marketId]);
279         } else {
280             address(msg.sender).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
281         }
282     }
```

30.7.4 Function forceUpdateAllPrices

- TODO

```
471     function forceUpdateAllPrices() external override onlyOwner {
472         tvm.rawReserve(msg.value, 2);
473         TvmBuilder tb;
474         tb.store(OperationCodes.NO_OP);
475         _updateAllPrices(tb.toCell());
476     }
```

30.7.5 Function getAllMarkets

- TODO

```
227     function getAllMarkets() external override view responsible
            returns(mapping(uint32 => MarketInfo)) {
228         return {flag: MsgFlag.REMAINING_GAS} markets;
229     }
```

30.7.6 Function getAllModules

- TODO

```
236     function getAllModules() external override view responsible
            returns(mapping(uint8 => address)) {
237         return {flag: MsgFlag.REMAINING_GAS} modules;
238     }
```

30.7.7 Function getMarketInformation

- TODO

```
223     function getMarketInformation(uint32 marketId) external
            override view responsible returns(MarketInfo) {
224         return {flag: MsgFlag.REMAINING_GAS} markets[marketId];
225     }
```

30.7.8 Function getServiceContractAddresses

- TODO

```
215     function getServiceContractAddresses() external override view
            responsible returns(address _userAccountManager, address
            _tip3WalletController, address _oracle) {
216         return {flag: MsgFlag.REMAINING_GAS} (userAccountManager,
                walletController, oracle);
217     }
```

30.7.9 Function getTokenPrices

- TODO

```
219     function getTokenPrices() external override view responsible
            returns(mapping(address => fraction)) {
220         return {flag: MsgFlag.REMAINING_GAS} tokenPrices;
221     }
```

30.7.10 Function performOperationUserAccountManager

- TODO

```
356     function performOperationUserAccountManager(uint8 operationId,
            uint32 marketId, TvmCell args) external override view
            onlyUserAccountManager {
357         TvmCell payload = _createOperationUpdatePayload(operationId
                , marketId, args);
358         _updateAllPrices(payload);
359     }
```

30.7.11 Function performOperationWalletController

- TODO

```
350     function performOperationWalletController(uint8 operationId,
            address tokenRoot, TvmCell args) external override view
            onlyWalletController {
351         uint32 marketId = tokensToMarkets[tokenRoot];
352         TvmCell payload = _createOperationUpdatePayload(operationId
                , marketId, args);
353         _updateAllPrices(payload);
354     }
```

30.7.12 Function receiveAllUpdatedPrices

- TODO

```
457     function receiveAllUpdatedPrices(mapping(address =>
            MarketPriceInfo) updatedPrices, TvmCell payload) external
            override onlyOracle {
458         for((address t, MarketPriceInfo mpi): updatedPrices) {
459             tokenPrices[t] = fraction(mpi.tokens, mpi.usd);
460             tokenPrices[t] = tokenPrices[t].simplify();
461
462             _updateAllMarkets();
463         }
464
465         performOperation(payload);
466     }
```

30.7.13 Function receiveCacheDelta

- TODO

```
100     function receiveCacheDelta(mapping(uint32 => MarketDelta)
            marketsDelta, TvmCell args) external override onlyModule {
101         tvm.rawReserve(msg.value, 2);
102         for ((uint32 marketId, MarketDelta marketDelta):
                marketsDelta) {
103             _acquireInterest(marketId);
104             _updateMarketDelta(marketId, marketDelta);
105             _updateExchangeRate(marketId);
106         }
107
108         IModule(msg.sender).resumeOperation{
109             flag: MsgFlag.REMAINING_GAS
110         }(args, markets, tokenPrices);
111     }
```

30.7.14 Function receiveUpdatedPrice

- TODO

```
440     function receiveUpdatedPrice(address tokenRoot, uint128 nom,
            uint128 denom, TvmCell) external override onlyOracle {
441         tokenPrices[tokenRoot] = fraction(nom, denom);
442     }
```

30.7.15 Function removeMarket

- TODO

```
315     function removeMarket(
316         uint32 marketId
317     ) external canChangeParams {
318         tvm.rawReserve(msg.value, 2);
319
320         emit MarketDeleted(marketId, markets[marketId]);
321
322         delete tokensToMarkets[markets[marketId].token];
323         delete createdMarkets[marketId];
324         delete markets[marketId];
325
326         address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
327     }
```

30.7.16 Function removeModule

- TODO

```
342     function removeModule(uint8 operationId) external
            canChangeParams {
343         tvm.rawReserve(msg.value, 2);
344         delete isModule[modules[operationId]];
345         delete modules[operationId];
346         moduleAmount = moduleAmount - 1;
347         address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
348     }
```

30.7.17 Function requestTokenPayout

- TODO

```
413     function requestTokenPayout(address tonWallet, address
            userTip3Wallet, uint32 marketId, uint256 toPayout) external
             override view onlyUserAccountManager {
414         tvm.rawReserve(msg.value, 2);
415
416         IWCMInteractions(walletController).transferTokensToWallet{
417             flag: MsgFlag.REMAINING_GAS
418         }(tonWallet, markets[marketId].token, userTip3Wallet,
                toPayout);
419     }
```

30.7.18 Function setOracleAddress

- TODO

```
501     function setOracleAddress(address _oracle) external override
            canChangeParams {
502         tvm.rawReserve(msg.value, 2);
503         oracle = _oracle;
504         address(msg.sender).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
505     }
```

30.7.19 Function setUserAccountManager

- TODO

```
483     function setUserAccountManager(address _userAccountManager)
            external override canChangeParams {
484         tvm.rawReserve(msg.value, 2);
485         userAccountManager = _userAccountManager;
486         address(msg.sender).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
487     }
```

30.7.20 Function setWalletController

- TODO

```
492     function setWalletController(address _tip3WalletController)
            external override canChangeParams {
493         tvm.rawReserve(msg.value, 2);
494         walletController = _tip3WalletController;
495         address(msg.sender).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
496     }
```

30.7.21 Function updateMarketParameters

- TODO

```
284     function updateMarketParameters(
285         uint32 marketId,
286         fraction _baseRate,
287         fraction _utilizationMultiplier,
288         fraction _reserveFactor,
289         fraction _exchangeRate,
290         fraction _collateralFactor,
291         fraction _liquidationMultiplier
292     ) external canChangeParams {
293         tvm.rawReserve(msg.value, 2);
294 
295         MarketInfo mi = markets[marketId];
296         mi.baseRate = _baseRate;
297         mi.utilizationMultiplier = _utilizationMultiplier;
298         mi.reserveFactor = _reserveFactor;
299         mi.collateralFactor = _collateralFactor;
300         mi.liquidationMultiplier = _liquidationMultiplier;
301         if (mi.vTokenBalance == 0) {
302             mi.exchangeRate = _exchangeRate;
303         }
304 
305         markets[marketId] = mi;
306         MarketDelta marketDelta;
307 
308         _acquireInterest(marketId);
309         _updateMarketDelta(marketId, marketDelta);
310         _updateExchangeRate(marketId);
311 
312         address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
313     }
```

30.7.22 Function updateModulesCache

- TODO

```
203     function updateModulesCache() external view override onlyOwner
            {
204         tvm.rawReserve(msg.value, 2);
205         uint128 valueToTransfer = msg.value / (moduleAmount + 1);
206         for ((, address module) : modules) {
207             IContractStateCache(module).updateCache{
208                 value: valueToTransfer
209             }(_owner, markets, tokenPrices);
210         }
211     }
```

30.7.23 Function upgradeContractCode

- TODO

```
44      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) override external canUpgrade {
45          tvm.accept();
46
47          tvm.setcode(code);
48          tvm.setCurrentCode(code);
49
50          onCodeUpgrade(
51              _owner,
52              userAccountManager,
53              walletController,
54              oracle,
55              markets,
56              tokenPrices,
57              modules,
58              updateParams,
59              codeVersion
60          );
61      }
```

30.7.24 Function withdrawExtraTons

- TODO

```
231     function withdrawExtraTons(uint128 amount) external override
            onlyOwner {
232         tvm.accept();
233         address(_owner).transfer({flag: 1, value: amount});
234     }
```

30.8 Internal Method Definitions

30.8.1 Function _acquireInterest

- TODO

```
126     function _acquireInterest(uint32 marketId) internal {
127         MarketInfo mi = markets[marketId];
128         uint256 dt = now - mi.lastUpdateTime;
129         if (
130             (markets[marketId].realTokenBalance != 0 ) ||
131             (markets[marketId].totalBorrowed != 0)
132         ) {
133             fraction borrowRate = MarketOperations.
                    calculateBorrowInterestRate(mi.baseRate, mi.
                    realTokenBalance, mi.totalBorrowed, mi.
                    utilizationMultiplier);
134             borrowRate = borrowRate.simplify();
135             fraction simpleInterestFactor = borrowRate.fNumMul(dt);
136             fraction newIndex = simpleInterestFactor.fNumAdd(1);
137             newIndex = mi.index.fMul(newIndex);
138             newIndex = newIndex.simplify();
139             fraction finterestAccumulated = mi.totalBorrowed.
                    numFMul(simpleInterestFactor);
140             uint256 interestAccumulated = finterestAccumulated.
                    toNum();
141             fraction freservesDelta = interestAccumulated.numFMul(
                    mi.reserveFactor);
142             uint256 totalBorrowNew = mi.totalBorrowed +
                    interestAccumulated;
143             uint256 totalReservesNew = mi.totalReserve +
                    freservesDelta.toNum();
144             mi.index = newIndex;
145             mi.totalBorrowed = totalBorrowNew;
146             mi.totalReserve = totalReservesNew;
147             markets[marketId] = mi;
148         }
149         markets[marketId].lastUpdateTime = now;
150     }
```

30.8.2 Function _createOperationUpdatePayload

- TODO

```
405     function _createOperationUpdatePayload(uint8 operationId,
            uint32 marketId, TvmCell args) internal pure returns (
            TvmCell payload) {
406         TvmBuilder tb;
407         tb.store(operationId);
408         tb.store(marketId);
409         tb.storeRef(args);
410         return tb.toCell();
411     }
```

30.8.3 Function _createUpdatedIndexes

- TODO

```
399     function _createUpdatedIndexes() internal view returns(mapping(
            uint32 => fraction) updatedIndexes) {
400         for ((uint32 marketId, MarketInfo mi) : markets) {
401             updatedIndexes[marketId] = mi.index;
402         }
403     }
```

30.8.4 Function _updateAllMarkets

- TODO

```
118     function _updateAllMarkets() internal {
119         for ((uint32 marketId,) : markets) {
120             _acquireInterest(marketId);
121             _updateExchangeRate(marketId);
122         }
123     }
```

30.8.5 Function _updateAllPrices

- TODO

```
447     function _updateAllPrices(TvmCell payload) internal view {
448         IOracleReturnPrices(oracle).getAllTokenPrices{
449             flag: MsgFlag.REMAINING_GAS,
450             callback: this.receiveAllUpdatedPrices
451         }(payload);
452     }
```

30.8.6 Function _updateExchangeRate

- TODO

```
190     function _updateExchangeRate(uint32 marketId) internal {
191         if (markets[marketId].vTokenBalance != 0) {
192             fraction exchangeRate = MarketOperations.
                    calculateExchangeRate(
193                 markets[marketId].realTokenBalance,
194                 markets[marketId].totalBorrowed,
195                 markets[marketId].totalReserve,
196                 markets[marketId].vTokenBalance
197             );
198             markets[marketId].exchangeRate = exchangeRate;
199         }
200     }
```

30.8.7 Function _updateMarketDelta

- TODO

```
152     function _updateMarketDelta(uint32 marketId, MarketDelta
            marketDelta) internal {
153         if (
154             marketDelta.realTokenBalance.delta != 0 &&
155             marketDelta.realTokenBalance.positive
156         ) {
157             markets[marketId].realTokenBalance += marketDelta.
                    realTokenBalance.delta;
158         } else {
159             markets[marketId].realTokenBalance -= marketDelta.
                    realTokenBalance.delta;
160         }
161 
162         if (
163             marketDelta.totalBorrowed.delta != 0 &&
164             marketDelta.totalBorrowed.positive
165         ) {
166             markets[marketId].totalBorrowed += marketDelta.
                    totalBorrowed.delta;
167         } else {
168             markets[marketId].totalBorrowed -= marketDelta.
                    totalBorrowed.delta;
169         }
170 
171         if (
172             marketDelta.vTokenBalance.delta != 0 &&
173             marketDelta.vTokenBalance.positive
174         ) {
175             markets[marketId].vTokenBalance += marketDelta.
                    vTokenBalance.delta;
176         } else {
177             markets[marketId].vTokenBalance -= marketDelta.
                    vTokenBalance.delta;
178         }
179 
180         if (
181             marketDelta.totalReserve.delta != 0 &&
182             marketDelta.totalReserve.positive
183         ) {
184             markets[marketId].totalReserve += marketDelta.
                    totalReserve.delta;
185         } else {
186             markets[marketId].totalReserve -= marketDelta.
                    totalReserve.delta;
187         }
188     }
```

30.8.8 Function onCodeUpgrade

- TODO

```
64      function onCodeUpgrade(
65          address owner,
66          address _userAccountManager,
67          address _walletController,
68          address _oracle,
69          mapping(uint32 => MarketInfo) _markets,
70          mapping(address => fraction) _tokenPrices,
71          mapping(uint8 => address) _modules,
72          TvmCell,
73          uint32 _codeVersion
74      ) private {
75          tvm.resetStorage();
76          contractCodeVersion = _codeVersion;
77          _owner = owner;
78          userAccountManager = _userAccountManager;
79          walletController = _walletController;
80          oracle = _oracle;
81          markets = _markets;
82          tokenPrices = _tokenPrices;
83          modules = _modules;
84          moduleAmount = 0;
85          for ((, address module): modules) {
86              moduleAmount += 1;
87              isModule[module] = true;
88          }
89
90          for ((uint32 marketId, MarketInfo market): markets) {
91              createdMarkets[marketId] = true;
92              tokensToMarkets[market.token] = marketId;
93              realTokenRoots[market.token] = true;
94          }
95      }
```

30.8.9 Function performOperation

- TODO

```
361     function performOperation(TvmCell args) internal view {
362         TvmSlice ts = args.toSlice();
363
364         uint8 operationId = ts.decode(uint8);
365         if (operationId != OperationCodes.NO_OP) {
366             uint32 marketId = ts.decode(uint32);
367             TvmCell moduleArgs = ts.loadRef();
368             IModule(modules[operationId]).performAction{
369                 flag: MsgFlag.REMAINING_GAS
370             }(marketId, moduleArgs, markets, tokenPrices);
371         } else {
372             address(_owner).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
373         }
374     }
```

30.8.10 Function updatePrice

- TODO

```
428     function updatePrice(address tokenRoot, TvmCell payload)
            internal view {
429         IOracleReturnPrices(oracle).getTokenPrice{
430             flag: MsgFlag.REMAINING_GAS,
431             callback: this.receiveUpdatedPrice
432         }(tokenRoot, payload);
433     }
```

30.8.10.0.1 Some functions inherited by using

Chapter 31

Contract Oracle

Contents

31.1 Overview

In file `Oracle.sol`

31.2 Contract Inheritance

IRoles	
IOracleService	
IOracleUpdatePrices	
IOracleReturnPrices	
IOracleManageTokens	
IUpgradableContract	

31.3 Static Variable Definitions

uint256	nonce	
		used in @11.Oracle.upgradeContractCode
		assigned in @11.Oracle.onCodeUpgrade
		used in @11.Oracle.onCodeUpgrade

```
22     uint256 static nonce;
```

31.4 Variable Definitions

mapping (address => MarketPriceInfo)	prices	
		used in @11.Oracle.upgradeContractCode
		assigned in @11.Oracle.removeToken
		used in @11.Oracle.removeToken
		used in @11.Oracle.removeToken
		assigned in @11.Oracle.onCodeUpgrade
		used in @11.Oracle.onCodeUpgrade
		used in @11.Oracle.internalUpdatePrice
		assigned in @11.Oracle.internalGetUpdatedPrice
		used in @11.Oracle.internalGetUpdatedPrice
		used in @11.Oracle.internalGetUpdatedPrice
		assigned in @11.Oracle.internalGetUpdatedPrice
		used in @11.Oracle.internalGetUpdatedPrice
		used in @11.Oracle.internalGetUpdatedPrice
		used in @11.Oracle.getTokenPrice
		used in @11.Oracle.getTokenPrice
		used in @11.Oracle.getAllTokenPrices
		assigned in @11.Oracle.externalUpdatePrice
		used in @11.Oracle.externalUpdatePrice
		assigned in @11.Oracle.externalUpdatePrice
		used in @11.Oracle.externalUpdatePrice
		assigned in @11.Oracle.addToken
		used in @11.Oracle.addToken
mapping (address => address)	swapPairToTokenRoot	
		used in @11.Oracle.upgradeContractCode
		assigned in @11.Oracle.removeToken
		used in @11.Oracle.removeToken
		assigned in @11.Oracle.onCodeUpgrade
		used in @11.Oracle.onCodeUpgrade

```
    mapping(address => MarketPriceInfo) prices;

    mapping(address => address) swapPairToTokenRoot;

    uint32 contractCodeVersion;
```

31.5 Modifier Definitions

31.5.1 Modifier onlyTrustedSwapPair

```
    modifier onlyTrustedSwapPair() {
        require(swapPairToTokenRoot.exists(msg.sender),
            OracleErrorCodes.ERROR_NOT_KNOWN_SWAP_PAIR);
        _;
    }
```

31.5.2 Modifier onlyKnownTokenRoot

```
    modifier onlyKnownTokenRoot(address _tokenRoot) {
        require(prices.exists(_tokenRoot), OracleErrorCodes.
            ERROR_NOT_KNOWN_TOKEN_ROOT);
        _;
    }
```

31.6 Constructor Definitions

31.6.1 Constructor

Critical issue: Constructor for Oracle (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

31.7 Public Method Definitions

31.7.1 Function addToken

- TODO

```
179     function addToken(address tokenRoot, address swapPairAddress,
            bool isLeft) override external canChangeParams {
180         tvm.accept();
181         swapPairToTokenRoot[swapPairAddress] = tokenRoot;
182         prices[tokenRoot] = MarketPriceInfo(swapPairAddress, isLeft
                , 0, 0);
183         this.internalUpdatePrice{value: CostConstants.
                TOKEN_INITIAL_UPDATE_PRICE, bounce: false}(tokenRoot);
184     }
```

31.7.2 Function externalUpdatePrice

- TODO

```
117     function externalUpdatePrice(address tokenRoot, uint128 tokens,
             uint128 usd) override external canChangeParams
            onlyKnownTokenRoot(tokenRoot) {
118         if (msg.sender.value == 0) {
119             tvm.accept();
120         } else {
121             tvm.rawReserve(msg.value, 2);
122         }
123
124         prices[tokenRoot].tokens = tokens;
125         prices[tokenRoot].usd = usd;
126
127         address(msg.sender).transfer({value: 0, flag: 64});
128     }
```

31.7.3 Function getAllTokenPrices

- TODO

```
167     function getAllTokenPrices(TvmCell payload) override external
            responsible view returns (mapping(address =>
            MarketPriceInfo), TvmCell) {
168         tvm.rawReserve(msg.value, 2);
169         return { value: 0, bounce: false, flag: MsgFlag.
                REMAINING_GAS } (prices, payload);
170     }
```

31.7.4 Function getDetails

- TODO

```
104     function getDetails() override external responsible view
            returns (OracleServiceInformation) {
105         tvm.rawReserve(msg.value, 2);
106         return { value: 0, bounce: false, flag: MsgFlag.
                REMAINING_GAS } OracleServiceInformation(
                contractCodeVersion, _owner);
107     }
```

31.7.5 Function getTokenPrice

- TODO

```
159     function getTokenPrice(address tokenRoot, TvmCell payload)
            override external responsible view returns(address, uint128
            , uint128, TvmCell) {
160         tvm.rawReserve(msg.value, 2);
161         return { value: 0, bounce: false, flag: MsgFlag.
                REMAINING_GAS } (tokenRoot, prices[tokenRoot].tokens,
                prices[tokenRoot].usd, payload);
162     }
```

31.7.6 Function getVersion

- TODO

```
 99     function getVersion() override external responsible view
            returns (uint32) {
100         tvm.rawReserve(msg.value, 2);
101         return { value: 0, bounce: false, flag: MsgFlag.
                REMAINING_GAS } contractCodeVersion;
102     }
```

31.7.7 Function internalGetUpdatedPrice

- TODO

```
146     function internalGetUpdatedPrice(IDexPairBalances updatedPrice)
             override external onlyTrustedSwapPair {
147         tvm.rawReserve(msg.value, 2);
148         address affectedToken = swapPairToTokenRoot[msg.sender];
149         prices[affectedToken].tokens = prices[affectedToken].isLeft
                 ? updatedPrice.left_balance : updatedPrice.
                right_balance;
150         prices[affectedToken].usd = prices[affectedToken].isLeft ?
                updatedPrice.right_balance : updatedPrice.left_balance;
151     }
```

31.7.8 Function internalUpdatePrice

- TODO

```
133     function internalUpdatePrice(address tokenRoot) override
            external onlyKnownTokenRoot(tokenRoot) {
134         tvm.rawReserve(msg.value, 2);
135         IDexPair(prices[tokenRoot].swapPair).getBalances{
136             value: 0,
```

```
137             bounce: true,
138             flag: MsgFlag.REMAINING_GAS,
139             callback: this.internalGetUpdatedPrice
140         }();
141     }
```

31.7.9 Function removeToken

- TODO

```
189     function removeToken(address tokenRoot) override external
            canChangeParams {
190         tvm.accept();
191         delete swapPairToTokenRoot[prices[tokenRoot].swapPair];
192         delete prices[tokenRoot];
193     }
```

31.7.10 Function upgradeContractCode

- TODO

```
62      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) override external canUpgrade {
63          tvm.accept();
64
65          tvm.setcode(code);
66          tvm.setCurrentCode(code);
67
68          onCodeUpgrade(
69              nonce,
70              prices,
71              swapPairToTokenRoot,
72              0,
73              _owner,
74              updateParams,
75              codeVersion
76          );
77      }
```

31.8 Internal Method Definitions

31.8.1 Function onCodeUpgrade

- TODO

```
79      function onCodeUpgrade(
80          uint256 _nonce,
81          mapping(address => MarketPriceInfo) _prices,
82          mapping(address => address) _swapPairToTokenRoot,
83          uint256,
84          address _ownerAddress,
85          TvmCell,
86          uint32 _codeVersion
87      ) private {
88          tvm.accept();
89          tvm.resetStorage();
90          nonce = _nonce;
91          prices = _prices;
92          swapPairToTokenRoot = _swapPairToTokenRoot;
93          _owner = _ownerAddress;
94          contractCodeVersion = _codeVersion;
95      }
```

Chapter 32

Contract Platform

Contents

32.1 Overview

In file `Platform.sol`

32.2 Static Variable Definitions

address	root	
		used in @16.Platform.initializeContract
uint8	platformType	
		used in @16.Platform.initializeContract
TvmCell	initialData	
		used in @16.Platform.initializeContract
TvmCell	platformCode	
		used in @16.Platform.initializeContract

```
    address static root;

    uint8 static platformType;

    TvmCell static initialData;

    TvmCell static platformCode;
```

32.3 Constructor Definitions

32.3.1 Constructor

Critical issue: Constructor for Platform (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum
loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(TvmCell contractCode, TvmCell params) public {
        tvm.accept();
        initializeContract(contractCode, params);
    }
```

32.4 Internal Method Definitions

32.4.1 Function initializeContract

- TODO

```
    function initializeContract(TvmCell contractCode, TvmCell
        params) private {
        tvm.accept();
        TvmBuilder builder;

        builder.store(root);
        builder.store(platformType);

        builder.store(platformCode); // ref 1
        builder.store(initialData);  // ref 2
        builder.store(params);       // ref 3

        tvm.setcode(contractCode);
        tvm.setCurrentCode(contractCode);

        onCodeUpgrade(builder.toCell());
    }
```

32.4.2 Function onCodeUpgrade

- TODO

```
35     function onCodeUpgrade(TvmCell data) private {}
```

Chapter 33

Contract RepayModule

Contents

33.1 Overview

In file `RepayModule.sol`

33.2 Contract Inheritance

IRoles	
IModule	
IContractStateCache	
IContractAddressSG	
IRepayModule	
IUpgradableContract	

33.3 Event Definitions

```
18      event RepayBorrow(uint32 marketId, MarketDelta marketDelta,
            address tonWallet, uint256 tokenDelta);
```

33.4 Variable Definitions

address	marketAddress	
		used in @8.RepayModule.upgradeContractCode
		assigned in @8.RepayModule.setMarketAddress
		used in @8.RepayModule.setMarketAddress
		used in @8.RepayModule.repayLoan
		assigned in @8.RepayModule.onCodeUpgrade
		used in @8.RepayModule.onCodeUpgrade
		used in @8.RepayModule.getContractAddresses
address	userAccountManager	
		used in @8.RepayModule.upgradeContractCode
		assigned in @8.RepayModule.setUserAccountManager
		used in @8.RepayModule.setUserAccountManager
		used in @8.RepayModule.resumeOperation
		used in @8.RepayModule.performAction
		assigned in @8.RepayModule.onCodeUpgrade
		used in @8.RepayModule.onCodeUpgrade
		used in @8.RepayModule.getContractAddresses
uint32	contractCodeVersion	
		assigned in @8.RepayModule.onCodeUpgrade
		used in @8.RepayModule.onCodeUpgrade
mapping (uint32 => MarketInfo)	marketInfo	
		used in @8.RepayModule.upgradeContractCode
		assigned in @8.RepayModule.updateCache
		used in @8.RepayModule.updateCache
		assigned in @8.RepayModule.resumeOperation
		used in @8.RepayModule.resumeOperation
		used in @8.RepayModule.repayLoan
		used in @8.RepayModule.repayLoan

```
    address marketAddress;

    address userAccountManager;

    uint32 public contractCodeVersion;

    mapping (uint32 => MarketInfo) marketInfo;

    mapping (address => fraction) tokenPrices;
```

33.5 Modifier Definitions

33.5.1 Modifier onlyMarket

```
    modifier onlyMarket() {
        require(msg.sender == marketAddress);
        _;
    }
```

33.5.2 Modifier onlyUserAccountManager

```
    modifier onlyUserAccountManager() {
        require(msg.sender == userAccountManager);
        _;
    }
```

33.6 Constructor Definitions

33.6.1 Constructor

Critical issue: Constructor for RepayModule (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

33.7 Public Method Definitions

33.7.1 Function getContractAddresses

- TODO

```
79      function getContractAddresses() external override view
            responsible returns(address _owner, address _marketAddress,
             address _userAccountManager) {
80          return {flag: MsgFlag.REMAINING_GAS} (_owner, marketAddress
                , userAccountManager);
81      }
```

33.7.2 Function getModuleState

- TODO

```
63      function getModuleState() external override view returns(
            mapping(uint32 => MarketInfo), mapping(address => fraction)
            ) {
64          return(marketInfo, tokenPrices);
65      }
```

33.7.3 Function performAction

- TODO

```
90      function performAction(uint32 marketId, TvmCell args, mapping (
            uint32 => MarketInfo) _marketInfo, mapping (address =>
            fraction) _tokenPrices) external override onlyMarket {
91          tvm.rawReserve(msg.value, 2);
92          marketInfo = _marketInfo;
93          tokenPrices = _tokenPrices;
94          TvmSlice ts = args.toSlice();
95          (address tonWallet, address userTip3Wallet, uint256
                tokensReceived) = ts.decode(address, address, uint256);
96          mapping(uint32 => fraction) updatedIndexes =
                _createUpdatedIndexes();
97
98          IUAMUserAccount(userAccountManager).requestRepayInfo{
99              flag: MsgFlag.REMAINING_GAS
100         }(tonWallet, userTip3Wallet, tokensReceived, marketId,
                updatedIndexes);
101     }
```

33.7.4 Function repayLoan

- TODO

```
109     function repayLoan(
110         address tonWallet,
111         address userTip3Wallet,
112         uint256 tokensForRepay,
113         uint32 marketId,
114         BorrowInfo borrowInfo
115     ) external override onlyUserAccountManager {
116         tvm.rawReserve(msg.value, 0);
117         mapping(uint32 => MarketDelta) marketsDelta;
118         MarketDelta marketDelta;
119 
120         uint256 tokensToRepay = borrowInfo.tokensBorrowed;
121         uint256 tokensToReturn;
122         uint256 tokenDelta;
123 
124         fraction ftokensToRepay = borrowInfo.tokensBorrowed.numFMul
                (marketInfo[marketId].index);
125         ftokensToRepay = ftokensToRepay.fDiv(borrowInfo.index);
126         tokensToRepay = ftokensToRepay.toNum();
127 
128         if (tokensToRepay <= tokensForRepay) {
129             tokensToReturn = tokensForRepay - tokensToRepay;
130             borrowInfo.tokensBorrowed = 0;
131             borrowInfo.index = marketInfo[marketId].index;
132             tokenDelta = tokensToRepay;
133         } else {
134             tokensToReturn = 0;
135             borrowInfo.tokensBorrowed = tokensToRepay -
                    tokensForRepay;
136             borrowInfo.index = marketInfo[marketId].index;
137             tokenDelta = tokensForRepay;
138         }
139 
140         marketDelta.totalBorrowed.delta = tokenDelta;
141         marketDelta.totalBorrowed.positive = false;
142         marketDelta.realTokenBalance.delta = tokenDelta;
143         marketDelta.realTokenBalance.positive = true;
144 
145         marketsDelta[marketId] = marketDelta;
146 
147         emit RepayBorrow(marketId, marketDelta, tonWallet,
                tokenDelta);
148 
149         TvmBuilder tb;
150         tb.store(marketId);
151         tb.store(tonWallet);
152         tb.store(userTip3Wallet);
153         tb.store(tokensToReturn);
154         TvmBuilder borrowInfoStorage;
155         borrowInfoStorage.store(borrowInfo);
156         tb.store(borrowInfoStorage.toCell());
157 
158         IContractStateCacheRoot(marketAddress).receiveCacheDelta{
```

```
159             flag: MsgFlag.REMAINING_GAS
160         }(marketsDelta, tb.toCell());
161     }
```

33.7.5 Function resumeOperation

- TODO

```
163     function resumeOperation(TvmCell args, mapping(uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
164         tvm.rawReserve(msg.value, 2);
165         marketInfo = _marketInfo;
166         tokenPrices = _tokenPrices;
167         TvmSlice ts = args.toSlice();
168         (uint32 marketId, address tonWallet, address userTip3Wallet
                , uint256 tokensToReturn) = ts.decode(uint32, address,
                address, uint256);
169         TvmSlice borrowInfoStorage = ts.loadRefAsSlice();
170         (BorrowInfo borrowInfo) = borrowInfoStorage.decode(
                BorrowInfo);
171         IUAMUserAccount(userAccountManager).writeRepayInformation{
172             flag: MsgFlag.REMAINING_GAS
173         }(tonWallet, userTip3Wallet, marketId, tokensToReturn,
                borrowInfo);
174     }
```

33.7.6 Function sendActionId

- TODO

```
59     function sendActionId() external override view responsible
           returns(uint8) {
60         return {flag: MsgFlag.REMAINING_GAS} OperationCodes.
               REPAY_TOKENS;
61     }
```

33.7.7 Function setMarketAddress

- TODO

```
67     function setMarketAddress(address _marketAddress) external
           override canChangeParams {
68         tvm.rawReserve(msg.value, 2);
69         marketAddress = _marketAddress;
70         address(_owner).transfer({value: 0, flag: MsgFlag.
               REMAINING_GAS});
71     }
```

33.7.8 Function setUserAccountManager

- TODO

```
73      function setUserAccountManager(address _userAccountManager)
            external override canChangeParams {
74          tvm.rawReserve(msg.value, 2);
75          userAccountManager = _userAccountManager;
76          address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
77      }
```

33.7.9 Function updateCache

- TODO

```
83      function updateCache(address tonWallet, mapping(uint32 =>
            MarketInfo) _marketInfo, mapping(address => fraction)
            _tokenPrices) external override onlyMarket {
84          tvm.rawReserve(msg.value , 2);
85          marketInfo = _marketInfo;
86          tokenPrices = _tokenPrices;
87          tonWallet.transfer({value: 0, flag: MsgFlag.REMAINING_GAS})
                ;
88      }
```

33.7.10 Function upgradeContractCode

- TODO

```
25      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) external override canUpgrade {
26          tvm.rawReserve(msg.value, 2);
27
28          tvm.setcode(code);
29          tvm.setCurrentCode(code);
30
31          onCodeUpgrade (
32              _owner,
33              marketAddress,
34              userAccountManager,
35              marketInfo,
36              tokenPrices,
37              codeVersion
38          );
39      }
```

33.8 Internal Method Definitions

33.8.1 Function _createUpdatedIndexes

- TODO

```
103     function _createUpdatedIndexes() internal view returns(mapping(
            uint32 => fraction) updatedIndexes) {
104         for ((uint32 marketId, MarketInfo mi): marketInfo) {
105             updatedIndexes[marketId] = mi.index;
106         }
107     }
```

33.8.2 Function onCodeUpgrade

- TODO

```
41      function onCodeUpgrade(
42          address owner,
43          address _marketAddress,
44          address _userAccountManager,
45          mapping(uint32 => MarketInfo) _marketInfo,
46          mapping(address => fraction) _tokenPrices,
47          uint32 _codeVersion
48      ) private {
49          tvm.accept();
50          tvm.resetStorage();
51          _owner = owner;
52          marketAddress = _marketAddress;
53          userAccountManager = _userAccountManager;
54          marketInfo = _marketInfo;
55          tokenPrices = _tokenPrices;
56          contractCodeVersion = _codeVersion;
57      }
```

33.8.2.0.1 Some functions inherited by using

Chapter 34

Contract RootTokenContract

Contents

34.1 Overview

In file `RootTokenContract.sol`

34.2 Contract Inheritance

IRootTokenContract	
IBurnableTokenRootContract	
IBurnableByRootTokenRootContract	
IPausable	
ITransferOwner	
ISendSurplusGas	
IVersioned	

34.3 Static Variable Definitions

uint256	_randomNonce	
bytes	name	
		used in @17.RootTokenContract.getDetails
bytes	symbol	
		used in @17.RootTokenContract.getDetails
uint8	decimals	
		used in @17.RootTokenContract.getDetails
TvmCell	wallet_code	
		used in @17.RootTokenContract.getWalletCode
		used in @17.RootTokenContract.getExpectedWalletAddress
		used in @17.RootTokenContract.getExpectedWalletAddress
		used in @17.RootTokenContract.deployWallet
		used in @17.RootTokenContract.deployWallet
		used in @17.RootTokenContract.deployEmptyWallet

```
29     uint256 static _randomNonce;

31     bytes public static name;

32     bytes public static symbol;

33     uint8 public static decimals;

35     TvmCell static wallet_code;
```

34.4 Variable Definitions

uint128	total_supply	
		assigned in @17.RootTokenContract.tokensBurned
		used in @17.RootTokenContract.tokensBurned
		assigned in @17.RootTokenContract.mint
		used in @17.RootTokenContract.mint
		used in @17.RootTokenContract.getTotalSupply
		used in @17.RootTokenContract.getDetails
		assigned in @17.RootTokenContract.deployWallet
		used in @17.RootTokenContract.deployWallet
		assigned in @17.RootTokenContract.:onBounce
		used in @17.RootTokenContract.:onBounce
		assigned in @17.RootTokenContract.:constructor
		used in @17.RootTokenContract.:constructor
uint256	root_public_key	
		assigned in @17.RootTokenContract.transferOwner
		used in @17.RootTokenContract.transferOwner
		used in @17.RootTokenContract.isExternalOwner
		used in @17.RootTokenContract.isExternalOwner
		used in @17.RootTokenContract.getDetails
		assigned in @17.RootTokenContract.:constructor
		used in @17.RootTokenContract.:constructor
address	root_owner_address	
		assigned in @17.RootTokenContract.transferOwner
		used in @17.RootTokenContract.transferOwner
		used in @17.RootTokenContract.isInternalOwner
		used in @17.RootTokenContract.isInternalOwner
		used in @17.RootTokenContract.getDetails
		used in @17.RootTokenContract.deployWallet

```
    uint128 total_supply;

    uint256 root_public_key;

    address root_owner_address;

    uint128 public start_gas_balance;

    bool public paused;
```

34.5 Modifier Definitions

34.5.1 Modifier onlyOwner

```
    modifier onlyOwner() {
        require(isOwner(), RootTokenContractErrors.
            error_message_sender_is_not_my_owner);
        _;
    }
```

34.5.2 Modifier onlyInternalOwner

```
    modifier onlyInternalOwner() {
        require(isInternalOwner(), RootTokenContractErrors.
            error_message_sender_is_not_my_owner);
        _;
    }
```

34.6 Constructor Definitions

34.6.1 Constructor

Critical issue: Constructor for RootTokenContract (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(uint256 root_public_key_, address
        root_owner_address_) public {
        require((root_public_key_ != 0 && root_owner_address_.value
             == 0) ||
                (root_public_key_ == 0 && root_owner_address_.value
                     != 0),
                RootTokenContractErrors.
                    error_define_public_key_or_owner_address);
        tvm.accept();
```

```
54
55          root_public_key = root_public_key_;
56          root_owner_address = root_owner_address_;
57
58          total_supply = 0;
59          paused = false;
60
61          start_gas_balance = address(this).balance;
62      }
```

34.7 Public Method Definitions

34.7.1 Fallback function

- TODO

```
524     fallback() external {
525     }
```

34.7.2 OnBounce function

- TODO

```
515     onBounce(TvmSlice slice) external {
516         tvm.accept();
517         uint32 functionId = slice.decode(uint32);
518         if (functionId == tvm.functionId(ITONTokenWallet.accept)) {
519             uint128 latest_bounced_tokens = slice.decode(uint128);
520             total_supply -= latest_bounced_tokens;
521         }
522     }
```

34.7.3 Function deployEmptyWallet

- TODO

```
238     function deployEmptyWallet(
239         uint128 deploy_grams,
240         uint256 wallet_public_key_,
241         address owner_address_,
242         address gas_back_address
243     )
244         override
245         external
246     returns (
247         address
248     ) {
249         require((owner_address_.value != 0 && wallet_public_key_ ==
                0) ||
```

```
250                 (owner_address_.value == 0 && wallet_public_key_ !=
                        0),
251                 RootTokenContractErrors.
                        error_define_public_key_or_owner_address);
252
253         tvm.rawReserve(address(this).balance - msg.value, 2);
254
255         address wallet = new TONTokenWallet{
256             value: deploy_grams,
257             flag: 1,
258             code: wallet_code,
259             pubkey: wallet_public_key_,
260             varInit: {
261                 root_address: address(this),
262                 code: wallet_code,
263                 wallet_public_key: wallet_public_key_,
264                 owner_address: owner_address_
265             }
266         }();
267
268         if (gas_back_address.value != 0) {
269             gas_back_address.transfer({ value: 0, flag: 128 });
270         } else {
271             msg.sender.transfer({ value: 0, flag: 128 });
272         }
273
274         return wallet;
275     }
```

34.7.4 Function deployWallet

- TODO

```
165     function deployWallet(
166         uint128 tokens,
167         uint128 deploy_grams,
168         uint256 wallet_public_key_,
169         address owner_address_,
170         address gas_back_address
171     )
172         override
173         external
174         onlyOwner
175     returns(
176         address
177     ) {
178         require(tokens >= 0);
179         require((owner_address_.value != 0 && wallet_public_key_ ==
                    0) ||
180                 (owner_address_.value == 0 && wallet_public_key_ !=
                        0),
181                 RootTokenContractErrors.
                        error_define_public_key_or_owner_address);
182
183         if(root_owner_address.value == 0) {
```

```
184             tvm.accept();
185         } else {
186             tvm.rawReserve(math.max(start_gas_balance, address(this
                    ).balance - msg.value), 2);
187         }
188 
189         TvmCell stateInit = tvm.buildStateInit({
190             contr: TONTokenWallet,
191             varInit: {
192                 root_address: address(this),
193                 code: wallet_code,
194                 wallet_public_key: wallet_public_key_,
195                 owner_address: owner_address_
196             },
197             pubkey: wallet_public_key_,
198             code: wallet_code
199         });
200 
201         address wallet;
202 
203         if(deploy_grams > 0) {
204             wallet = new TONTokenWallet{
205                 stateInit: stateInit,
206                 value: deploy_grams,
207                 wid: address(this).wid,
208                 flag: 1
209             }();
210         } else {
211             wallet = address(tvm.hash(stateInit));
212         }
213 
214         ITONTokenWallet(wallet).accept(tokens);
215 
216         total_supply += tokens;
217 
218         if (root_owner_address.value != 0) {
219             if (gas_back_address.value != 0) {
220                 gas_back_address.transfer({ value: 0, flag: 128 });
221             } else {
222                 msg.sender.transfer({ value: 0, flag: 128 });
223             }
224         }
225 
226         return wallet;
227     }
```

34.7.5 Function getDetails

- TODO

```
78     function getDetails() override external view responsible
           returns (IRootTokenContractDetails) {
79         return { value: 0, bounce: false, flag: 64 }
               IRootTokenContractDetails(
80             name,
```

```
81              symbol,
82              decimals,
83              root_public_key,
84              root_owner_address,
85              total_supply
86          );
87      }
```

34.7.6 Function getTotalSupply

- TODO

```
93      function getTotalSupply() override external view responsible
            returns (uint128) {
94          return { value: 0, bounce: false, flag: 64 } total_supply;
95      }
```

34.7.7 Function getVersion

- TODO

```
64      function getVersion() override external pure responsible
            returns (uint32) {
65          return 4;
66      }
```

34.7.8 Function getWalletAddress

- TODO

```
112     function getWalletAddress(
113         uint256 wallet_public_key_,
114         address owner_address_
115     )
116         override
117         external
118         view
119         responsible
120     returns (
121         address
122     ) {
123         require((owner_address_.value != 0 && wallet_public_key_ ==
                0) ||
124                 (owner_address_.value == 0 && wallet_public_key_ !=
                        0),
125                 RootTokenContractErrors.
                        error_define_public_key_or_owner_address);
126         return { value: 0, bounce: false, flag: 64 }
                getExpectedWalletAddress(wallet_public_key_,
                owner_address_);
127     }
```

34.7.9 Function getWalletCode

- TODO

```
101     function getWalletCode() override external view responsible
            returns (TvmCell) {
102         return { value: 0, bounce: false, flag: 64 } wallet_code;
103     }
```

34.7.10 Function mint

- TODO

```
283     function mint(
284         uint128 tokens,
285         address to
286     )
287         override
288         external
289         onlyOwner
290     {
291         tvm.accept();
292 
293         ITONTokenWallet(to).accept(tokens);
294 
295         total_supply += tokens;
296     }
```

34.7.11 Function proxyBurn

- TODO

```
308     function proxyBurn(
309         uint128 tokens,
310         address sender_address,
311         address send_gas_to,
312         address callback_address,
313         TvmCell callback_payload
314     )
315         override
316         external
317         onlyInternalOwner
318     {
319         tvm.rawReserve(address(this).balance - msg.value, 2);
320 
321         address send_gas_to_ = send_gas_to;
322         address expectedWalletAddress = getExpectedWalletAddress(0,
                sender_address);
323 
324         if (send_gas_to.value == 0) {
325             send_gas_to_ = sender_address;
```

```
326         }
327
328         IBurnableByRootTokenWallet(expectedWalletAddress).
                burnByRoot{value: 0, flag: 128}(
329             tokens,
330             send_gas_to_,
331             callback_address,
332             callback_payload
333         );
334     }
```

34.7.12 Function sendExpectedWalletAddress

- TODO

```
135     function sendExpectedWalletAddress(
136         uint256 wallet_public_key_,
137         address owner_address_,
138         address to
139     )
140         override
141         external
142     {
143         tvm.rawReserve(address(this).balance - msg.value, 2);
144
145         address wallet = getExpectedWalletAddress(
                wallet_public_key_, owner_address_);
146         IExpectedWalletAddressCallback(to).
                expectedWalletAddressCallback{value: 0, flag: 128}(
147             wallet,
148             wallet_public_key_,
149             owner_address_
150         );
151     }
```

34.7.13 Function sendPausedCallbackTo

- TODO

```
424     function sendPausedCallbackTo(
425         uint64 callback_id,
426         address callback_addr
427     )
428         override
429         external
430     {
431         tvm.rawReserve(address(this).balance - msg.value, 2);
432         IPausedCallback(callback_addr).pausedCallback{ value: 0,
                flag: 128 }(callback_id, paused);
433     }
```

34.7.14 Function sendSurplusGas

- TODO

```
387     function sendSurplusGas(
388         address to
389     )
390         override
391         external
392         onlyInternalOwner
393     {
394         tvm.rawReserve(start_gas_balance, 2);
395         IReceiveSurplusGas(to).receiveSurplusGas{ value: 0, flag:
                128 }();
396     }
```

34.7.15 Function setPaused

- TODO

```
408     function setPaused(
409         bool value
410     )
411         override
412         external
413         onlyOwner
414     {
415         tvm.accept();
416         paused = value;
417     }
```

34.7.16 Function tokensBurned

- TODO

```
348     function tokensBurned(
349         uint128 tokens,
350         uint256 sender_public_key,
351         address sender_address,
352         address send_gas_to,
353         address callback_address,
354         TvmCell callback_payload
355     ) override external {
356
357         require(!paused, RootTokenContractErrors.error_paused);
358
359         address expectedWalletAddress = getExpectedWalletAddress(
                sender_public_key, sender_address);
360
361         require(msg.sender == expectedWalletAddress,
                RootTokenContractErrors.
                error_message_sender_is_not_good_wallet);
```

```
362
363         tvm.rawReserve(address(this).balance - msg.value, 2);
364
365         total_supply -= tokens;
366
367         if (callback_address.value == 0) {
368             send_gas_to.transfer({ value: 0, flag: 128 });
369         } else {
370             IBurnTokensCallback(callback_address).burnCallback{
                    value: 0, flag: 128}(
371                 tokens,
372                 callback_payload,
373                 sender_public_key,
374                 sender_address,
375                 expectedWalletAddress,
376                 send_gas_to
377             );
378         }
379
380     }
```

34.7.17 Function transferOwner

- TODO

```
441     function transferOwner(
442         uint256 root_public_key_,
443         address root_owner_address_
444     )
445         override
446         external
447         onlyOwner
448     {
449         require((root_public_key_ != 0 && root_owner_address_.value
                    == 0) ||
450                 (root_public_key_ == 0 && root_owner_address_.value
                        != 0),
451                 RootTokenContractErrors.
                        error_define_public_key_or_owner_address);
452         tvm.accept();
453         root_public_key = root_public_key_;
454         root_owner_address = root_owner_address_;
455     }
```

34.8 Internal Method Definitions

34.8.1 Function getExpectedWalletAddress

- TODO

```
486     function getExpectedWalletAddress(
487         uint256 wallet_public_key_,
488         address owner_address_
489     )
490         private
491         inline
492         view
493     returns (
494         address
495     ) {
496         TvmCell stateInit = tvm.buildStateInit({
497             contr: TONTokenWallet,
498             varInit: {
499                 root_address: address(this),
500                 code: wallet_code,
501                 wallet_public_key: wallet_public_key_,
502                 owner_address: owner_address_
503             },
504             pubkey: wallet_public_key_,
505             code: wallet_code
506         });
507 
508         return address(tvm.hash(stateInit));
509     }
```

34.8.2 Function isExternalOwner

- TODO

```
477     function isExternalOwner() private inline view returns (bool) {
478         return root_public_key != 0 && root_public_key == msg.
                pubkey();
479     }
```

34.8.3 Function isInternalOwner

- TODO

```
473     function isInternalOwner() private inline view returns (bool) {
474         return root_owner_address.value != 0 && root_owner_address
                == msg.sender;
475     }
```

34.8.4 Function isOwner

- TODO

```
469     function isOwner() private inline view returns (bool) {
470         return isInternalOwner() || isExternalOwner();
471     }
```

Chapter 35

Contract SupplyModule

Contents

35.1 Overview

In file `SupplyModule.sol`

35.2 Contract Inheritance

IRoles	
IModule	
IContractStateCache	
IContractAddressSG	
IUpgradableContract	

35.3 Event Definitions

```
19      event TokensSupplied(uint32 marketId, MarketDelta marketDelta,
            address tonWallet, uint256 tokensSupplied);
```

35.4 Variable Definitions

address	marketAddress	
		used in @9.SupplyModule.upgradeContractCode
		assigned in @9.SupplyModule.setMarketAddress
		used in @9.SupplyModule.setMarketAddress
		used in @9.SupplyModule.performAction
		assigned in @9.SupplyModule.onCodeUpgrade
		used in @9.SupplyModule.onCodeUpgrade
		used in @9.SupplyModule.getContractAddresses
address	userAccountManager	
		used in @9.SupplyModule.upgradeContractCode
		assigned in @9.SupplyModule.setUserAccountManager
		used in @9.SupplyModule.setUserAccountManager
		used in @9.SupplyModule.resumeOperation
		assigned in @9.SupplyModule.onCodeUpgrade
		used in @9.SupplyModule.onCodeUpgrade
		used in @9.SupplyModule.getContractAddresses
uint32	contractCodeVersion	
		assigned in @9.SupplyModule.onCodeUpgrade
		used in @9.SupplyModule.onCodeUpgrade
mapping (uint32 => MarketInfo)	marketInfo	
		used in @9.SupplyModule.upgradeContractCode
		assigned in @9.SupplyModule.updateCache
		used in @9.SupplyModule.updateCache
		used in @9.SupplyModule.resumeOperation
		assigned in @9.SupplyModule.resumeOperation
		used in @9.SupplyModule.resumeOperation
		used in @9.SupplyModule.performAction
		assigned in @9.SupplyModule.performAction

```
    address marketAddress;
```

```
    address userAccountManager;
```

```
    uint32 public contractCodeVersion;
```

```
    mapping (uint32 => MarketInfo) marketInfo;
```

```
    mapping (address => fraction) tokenPrices;
```

35.5 Modifier Definitions

35.5.1 Modifier onlyMarket

```
    modifier onlyMarket() {
        require(msg.sender == marketAddress);
        _;
    }
```

35.5.2 Modifier onlyUserAccountManager

```
    modifier onlyUserAccountManager() {
        require(msg.sender == userAccountManager);
        tvm.rawReserve(msg.value, 2);
        _;
    }
```

35.6 Constructor Definitions

35.6.1 Constructor

Critical issue: Constructor for SupplyModule (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum
loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

35.7 Public Method Definitions

35.7.1 Function getContractAddresses

- TODO

```
80      function getContractAddresses() external override view
            responsible returns(address _owner, address _marketAddress,
             address _userAccountManager) {
81          return {flag: MsgFlag.REMAINING_GAS} (_owner, marketAddress
                , userAccountManager);
82      }
```

35.7.2 Function getModuleState

- TODO

```
64      function getModuleState() external override view returns(
            mapping(uint32 => MarketInfo), mapping(address => fraction)
            ) {
65          return(marketInfo, tokenPrices);
66      }
```

35.7.3 Function performAction

- TODO

```
91      function performAction(uint32 marketId, TvmCell args, mapping (
            uint32 => MarketInfo) _marketInfo, mapping (address =>
            fraction) _tokenPrices) external override onlyMarket {
92          tvm.rawReserve(msg.value, 2);
93          marketInfo = _marketInfo;
94          tokenPrices = _tokenPrices;
95          TvmSlice ts = args.toSlice();
96          (address tonWallet, uint256 tokenAmount) = ts.decode(
                address, uint256);
97
98          // Supply process:
99          // 1. Convert real tokens to vTokens by dividing real token
                 amount by exchange rate
100         fraction vTokensToProvide = tokenAmount.numFDiv(marketInfo[
                marketId].exchangeRate);
101
102         MarketDelta marketDelta;
103         mapping(uint32 => MarketDelta) marketsDelta;
104         marketDelta.realTokenBalance.delta = tokenAmount;
105         marketDelta.realTokenBalance.positive = true;
106         marketDelta.vTokenBalance.delta = vTokensToProvide.toNum();
107         marketDelta.vTokenBalance.positive = true;
108         marketsDelta[marketId] = marketDelta;
```

```
109
110         TvmBuilder tb;
111         tb.store(marketId);
112         tb.store(tonWallet);
113         tb.store(vTokensToProvide.toNum());
114
115         emit TokensSupplied(marketId, marketDelta, tonWallet,
                tokenAmount);
116
117         IContractStateCacheRoot(marketAddress).receiveCacheDelta{
118             flag: MsgFlag.REMAINING_GAS
119         }(marketsDelta, tb.toCell());
120     }
```

35.7.4 Function resumeOperation

- TODO

```
122     function resumeOperation(TvmCell args, mapping(uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
123         tvm.rawReserve(msg.value, 2);
124         marketInfo = _marketInfo;
125         tokenPrices = _tokenPrices;
126
127         TvmSlice ts = args.toSlice();
128         (uint32 marketId, address tonWallet, uint256
                vTokensToProvide) = ts.decode(uint32, address, uint256)
                ;
129
130         IUAMUserAccount(userAccountManager).writeSupplyInfo{
131             flag: MsgFlag.REMAINING_GAS
132         }(tonWallet, marketId, vTokensToProvide, marketInfo[
                marketId].index);
133     }
```

35.7.5 Function sendActionId

- TODO

```
60     function sendActionId() external override view responsible
           returns(uint8) {
61         return {flag: MsgFlag.REMAINING_GAS} OperationCodes.
               SUPPLY_TOKENS;
62     }
```

35.7.6 Function setMarketAddress

- TODO

```
68      function setMarketAddress(address _marketAddress) external
            override canChangeParams {
69          tvm.rawReserve(msg.value, 2);
70          marketAddress = _marketAddress;
71          address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
72      }
```

35.7.7 Function setUserAccountManager

- TODO

```
74      function setUserAccountManager(address _userAccountManager)
            external override canChangeParams {
75          tvm.rawReserve(msg.value, 2);
76          userAccountManager = _userAccountManager;
77          address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
78      }
```

35.7.8 Function updateCache

- TODO

```
84      function updateCache(address tonWallet, mapping (uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
85          tvm.rawReserve(msg.value, 2);
86          marketInfo = _marketInfo;
87          tokenPrices = _tokenPrices;
88          tonWallet.transfer({value: 0, flag: MsgFlag.REMAINING_GAS})
                ;
89      }
```

35.7.9 Function upgradeContractCode

- TODO

```
26      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) external override canUpgrade {
27          tvm.rawReserve(msg.value, 2);
28
29          tvm.setcode(code);
30          tvm.setCurrentCode(code);
31
32          onCodeUpgrade (
33              _owner,
34              marketAddress,
35              userAccountManager,
36              marketInfo,
```

```
            tokenPrices,
            codeVersion
        );
    }
```

35.8 Internal Method Definitions

35.8.1 Function onCodeUpgrade

- TODO

```
    function onCodeUpgrade(
        address owner,
        address _marketAddress,
        address _userAccountManager,
        mapping(uint32 => MarketInfo) _marketInfo,
        mapping(address => fraction) _tokenPrices,
        uint32 _codeVersion
    ) private {
        tvm.accept();
        tvm.resetStorage();
        _owner = owner;
        marketAddress = _marketAddress;
        userAccountManager = _userAccountManager;
        marketInfo = _marketInfo;
        tokenPrices = _tokenPrices;
        contractCodeVersion = _codeVersion;
    }
```

35.8.1.0.1 Some functions inherited by using

Chapter 36

Contract TIP3TokenDeployer

Contents

36.1 Overview

In file `TIP3Deployer.sol`

36.2 Contract Inheritance

ITIP3Deployer	
ITIP3DeployerManageCode	
ITIP3DeployerServiceInfo	
IUpgradableContract	

36.3 Variable Definitions

TvmCell	rootContractCode	
		used in @12.TIP3TokenDeployer.upgradeContractCode
		assigned in @12.TIP3TokenDeployer.setTIP3RootContractCode
		used in @12.TIP3TokenDeployer.setTIP3RootContractCode
		used in @12.TIP3TokenDeployer.getServiceInfo
		used in @12.TIP3TokenDeployer.getFutureTIP3Address
		used in @12.TIP3TokenDeployer.deployTIP3
TvmCell	walletContractCode	
		used in @12.TIP3TokenDeployer.upgradeContractCode
		assigned in @12.TIP3TokenDeployer.setTIP3WalletContractCode
		used in @12.TIP3TokenDeployer.setTIP3WalletContractCode
		used in @12.TIP3TokenDeployer.getServiceInfo
		used in @12.TIP3TokenDeployer.getFutureTIP3Address
address	ownerAddress	
		used in @12.TIP3TokenDeployer.upgradeContractCode
		assigned in @12.TIP3TokenDeployer.:constructor
		used in @12.TIP3TokenDeployer.:constructor
uint32	contractCodeVersion	

```
18     TvmCell rootContractCode;
19     TvmCell walletContractCode;
```

```
    address ownerAddress;

    uint32 contractCodeVersion;
```

36.4 Modifier Definitions

36.4.1 Modifier onlyOwner

```
    modifier onlyOwner() {
        require(msg.sender == ownerAddress, TIP3DeployerErrorCodes.
            ERROR_MSG_SENDER_IS_NOT_OWNER);
        _;
    }
```

36.4.2 Modifier checkMsgValue

```
    modifier checkMsgValue(uint128 gramsRequired) {
        require(msg.value > gramsRequired, TIP3DeployerErrorCodes.
            ERROR_MSG_VALUE_IS_TOO_LOW);
        _;
    }
```

36.5 Constructor Definitions

36.5.1 Constructor

Critical issue: Constructor for TIP3TokenDeployer (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _owner) public {
        tvm.accept();
        ownerAddress = _owner;
    }
```

36.6 Public Method Definitions

36.6.1 Function deployTIP3

- TODO

```
66      function deployTIP3(IRootTokenContract.
            IRootTokenContractDetails rootInfo, uint128 deployGrams,
            uint256 pubkeyToInsert, TvmCell payloadToReturn)
67          external
68          responsible
69          override
70          checkMsgValue(deployGrams)
71          returns (address, TvmCell)
72      {
73          tvm.rawReserve(msg.value, 2);
74          address tip3TokenAddress = new RootTokenContract{
75              value: deployGrams,
76              flag: 0,
77              code: rootContractCode,
78              pubkey: pubkeyToInsert,
79              varInit: {
80                  _randomNonce: 0,
81                  name: rootInfo.name,
82                  symbol: rootInfo.symbol,
83                  decimals: rootInfo.decimals,
84                  wallet_code: walletContractCode
85              }
86          }(rootInfo.root_public_key, rootInfo.root_owner_address);
87
88          return { value: 0, bounce: false, flag: MsgFlag.
                REMAINING_GAS } (tip3TokenAddress, payloadToReturn);
89      }
```

36.6.2 Function getFutureTIP3Address

- TODO

```
95      function getFutureTIP3Address(IRootTokenContract.
            IRootTokenContractDetails rootInfo, uint256 pubkeyToInsert)
             external override responsible returns (address) {
96          tvm.accept();
97          TvmCell stateInit = tvm.buildStateInit({
98              contr: RootTokenContract,
99              code: rootContractCode,
100             pubkey: pubkeyToInsert,
101             varInit: {
102                 _randomNonce: 0,
103                 name: rootInfo.name,
104                 symbol: rootInfo.symbol,
105                 decimals: rootInfo.decimals,
106                 wallet_code: walletContractCode
107             }
108         });
109
110         return address.makeAddrStd(0, tvm.hash(stateInit));
111     }
```

36.6.3 Function getServiceInfo

- TODO

```
131     function getServiceInfo() external override responsible view
            returns (ServiceInfo) {
132         return ServiceInfo(rootContractCode, walletContractCode);
133     }
```

36.6.4 Function setTIP3RootContractCode

- TODO

```
118     function setTIP3RootContractCode(TvmCell _rootContractCode)
            external override onlyOwner {
119         tvm.accept();
120         rootContractCode = _rootContractCode;
121     }
```

36.6.5 Function setTIP3WalletContractCode

- TODO

```
126     function setTIP3WalletContractCode(TvmCell _walletContractCode)
             external override onlyOwner {
127         tvm.accept();
128         walletContractCode = _walletContractCode;
129     }
```

36.6.6 Function upgradeContractCode

- TODO

```
33      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) override external onlyOwner {
34          tvm.accept();
35
36          tvm.setcode(code);
37          tvm.setCurrentCode(code);
38
39          onCodeUpgrade(
40              ownerAddress,
41              rootContractCode,
42              walletContractCode,
43              updateParams,
44              codeVersion
45          );
46      }
```

36.7 Internal Method Definitions

36.7.1 Function onCodeUpgrade

- TODO

```
48      function onCodeUpgrade(
49          address,
50          TvmCell,
51          TvmCell,
52          TvmCell,
53          uint32
54      ) private {
55
56      }
```

Chapter 37

Contract TONTokenWallet

Contents

37.1 Overview

In file `TONTokenWallet.sol`

37.2 Contract Inheritance

ITONTokenWallet	
IDestroyable	
IBurnableByOwnerTokenWallet	
IBurnableByRootTokenWallet	
IVersioned	

37.3 Static Variable Definitions

address	root_address	
		used in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.internalTransfer
		used in @18.TONTokenWallet.getExpectedAddress
		used in @18.TONTokenWallet.getDetails
		used in @18.TONTokenWallet.burnByRoot
		used in @18.TONTokenWallet.burnByOwner
		used in @18.TONTokenWallet.burnByOwner
		used in @18.TONTokenWallet.:onBounce
		used in @18.TONTokenWallet.:constructor
TvmCell	code	
		used in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.getWalletCode
		used in @18.TONTokenWallet.getExpectedAddress
		used in @18.TONTokenWallet.getExpectedAddress
uint256	wallet_public_key	
		used in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.transfer
		used in @18.TONTokenWallet.transfer
		used in @18.TONTokenWallet.internalTransferFrom
		used in @18.TONTokenWallet.internalTransfer
		used in @18.TONTokenWallet.getDetails
		used in @18.TONTokenWallet.burnByRoot
		used in @18.TONTokenWallet.burnByOwner
		used in @18.TONTokenWallet.burnByOwner

```
24      address static root_address;
25      TvmCell static code;
27      uint256 static wallet_public_key;
29      address static owner_address;
```

37.4 Variable Definitions

uint128	balance_	
		assigned in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.transferToRecipient
		assigned in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.transferToRecipient
		used in @18.TONTokenWallet.transferToRecipient
		assigned in @18.TONTokenWallet.transfer
		used in @18.TONTokenWallet.transfer
		assigned in @18.TONTokenWallet.transfer
		used in @18.TONTokenWallet.transfer
		used in @18.TONTokenWallet.transfer
		assigned in @18.TONTokenWallet.internalTransferFrom
		used in @18.TONTokenWallet.internalTransferFrom
		used in @18.TONTokenWallet.internalTransferFrom
		used in @18.TONTokenWallet.internalTransfer
		assigned in @18.TONTokenWallet.internalTransfer
		used in @18.TONTokenWallet.internalTransfer
		used in @18.TONTokenWallet.getDetails
		used in @18.TONTokenWallet.destroy
		assigned in @18.TONTokenWallet.burnByRoot
		used in @18.TONTokenWallet.burnByRoot
		used in @18.TONTokenWallet.burnByRoot
		assigned in @18.TONTokenWallet.burnByOwner
		used in @18.TONTokenWallet.burnByOwner
		assigned in @18.TONTokenWallet.burnByOwner
		used in @18.TONTokenWallet.burnByOwner
		used in @18.TONTokenWallet.burnByOwner

```
    uint128 balance_;
```

```
    optional(AllowanceInfo) allowance_;
```

```
    address receive_callback;
```

```
    address bounced_callback;
```

```
    bool allow_non_notifiable;
```

37.5 Modifier Definitions

37.5.1 Modifier onlyRoot

```
    modifier onlyRoot() {
        require(root_address == msg.sender, TONTokenWalletErrors.
            error_message_sender_is_not_my_root);
        _;
    }
```

37.5.2 Modifier onlyOwner

```
    modifier onlyOwner() {
        require((owner_address.value != 0 && owner_address == msg.
            sender) ||
                (wallet_public_key != 0 && wallet_public_key == msg
                    .pubkey()),
                TONTokenWalletErrors.
                    error_message_sender_is_not_my_owner);
        _;
    }
```

37.5.3 Modifier onlyInternalOwner

```
    modifier onlyInternalOwner() {
        require(owner_address.value != 0 && owner_address == msg.
            sender);
        _;
    }
```

37.6 Constructor Definitions

37.6.1 Constructor

Critical issue: Constructor for TONTokenWallet (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
43      constructor() public {
44          require(wallet_public_key == tvm.pubkey() && (owner_address
                .value == 0 || wallet_public_key == 0));
45          tvm.accept();
46
47          allow_non_notifiable = true;
48
49          if (owner_address.value != 0) {
50              ITokenWalletDeployedCallback(owner_address).
                    notifyWalletDeployed{value: 0.00001 ton, flag: 1}(
                    root_address);
51          }
52      }
```

37.7 Public Method Definitions

37.7.1 Fallback function

- TODO

```
683     fallback() external {
684     }
```

37.7.2 OnBounce function

- TODO

```
653     onBounce(TvmSlice body) external {
654         tvm.accept();
655
656         uint32 functionId = body.decode(uint32);
657         if (functionId == tvm.functionId(ITONTokenWallet.
                internalTransfer)) {
658             uint128 tokens = body.decode(uint128);
659             balance_ += tokens;
660
661             if (bounced_callback.value != 0) {
662                 tvm.rawReserve(address(this).balance - msg.value,
                        2);
663                 ITokensBouncedCallback(bounced_callback).
                        tokensBouncedCallback{ value: 0, flag: 128 }(
664                     address(this),
665                     root_address,
666                     tokens,
667                     msg.sender,
668                     balance_
669                 );
670             } else if (owner_address.value != 0) {
```

```
671                 tvm.rawReserve(math.max(TONTokenWalletConstants.
                        target_gas_balance, address(this).balance - msg
                        .value), 2);
672                 owner_address.transfer({ value: 0, flag: 128 });
673             }
674         } else if (functionId == tvm.functionId(
                IBurnableTokenRootContract.tokensBurned)) {
675             balance_ += body.decode(uint128);
676             if (owner_address.value != 0) {
677                 tvm.rawReserve(math.max(TONTokenWalletConstants.
                        target_gas_balance, address(this).balance - msg
                        .value), 2);
678                 owner_address.transfer({ value: 0, flag: 128 });
679             }
680         }
681     }
```

37.7.3 Function accept

- TODO

```
96      function accept(
97          uint128 tokens
98      )
99          override
100         external
101         onlyRoot
102     {
103         tvm.accept();
104         balance_ += tokens;
105     }
```

37.7.4 Function allowance

- TODO

```
107     function allowance() override external view responsible returns
            (AllowanceInfo) {
108         return { value: 0, bounce: false, flag: 64 } (allowance_.
                hasValue() ? allowance_.get() : AllowanceInfo(0,
                address.makeAddrStd(0, 0)));
109     }
```

37.7.5 Function approve

- TODO

```
119     function approve(
120         address spender,
121         uint128 remaining_tokens,
122         uint128 tokens
123     )
124         override
125         external
126         onlyOwner
127     {
128         require(remaining_tokens == 0 || !allowance_.hasValue(),
                TONTokenWalletErrors.error_non_zero_remaining);
129         if (owner_address.value != 0 ) {
130             tvm.rawReserve(math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value), 2);
131         } else {
132             tvm.accept();
133         }
134 
135         if (allowance_.hasValue()) {
136             if (allowance_.get().remaining_tokens ==
                    remaining_tokens) {
137                 allowance_.set(AllowanceInfo(tokens, spender));
138             }
139         } else {
140             allowance_.set(AllowanceInfo(tokens, spender));
141         }
142 
143         if (owner_address.value != 0 ) {
144             msg.sender.transfer({ value: 0, flag: 128 });
145         }
146     }
```

37.7.6 Function balance

- TODO

```
58     function balance() override external view responsible returns (
           uint128) {
59         return { value: 0, bounce: false, flag: 64 } balance_;
60     }
```

37.7.7 Function burnByOwner

- TODO

```
473     function burnByOwner(
474         uint128 tokens,
475         uint128 grams,
476         address send_gas_to,
477         address callback_address,
478         TvmCell callback_payload
```

```
479     ) override external onlyOwner {
480         require(tokens > 0);
481         require(tokens <= balance_, TONTokenWalletErrors.
                error_not_enough_balance);
482         require((owner_address.value != 0 && msg.value > 0) ||
483                 (owner_address.value == 0 && grams <= address(this)
                        .balance && grams > 0), TONTokenWalletErrors.
                        error_low_message_value);
484
485         if (owner_address.value != 0 ) {
486             tvm.rawReserve(math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value), 2);
487             balance_ -= tokens;
488             IBurnableTokenRootContract(root_address)
489                 .tokensBurned{ value: 0, flag: 128, bounce: true }(
490                     tokens,
491                     wallet_public_key,
492                     owner_address,
493                     send_gas_to.value != 0 ? send_gas_to :
                            owner_address,
494                     callback_address,
495                     callback_payload
496                 );
497         } else {
498             tvm.accept();
499             balance_ -= tokens;
500             IBurnableTokenRootContract(root_address)
501                 .tokensBurned{ value: grams, bounce: true }(
502                     tokens,
503                     wallet_public_key,
504                     owner_address,
505                     send_gas_to.value != 0 ? send_gas_to : address(
                            this),
506                     callback_address,
507                     callback_payload
508                 );
509         }
510     }
```

37.7.8 Function burnByRoot

- TODO

```
520     function burnByRoot(
521         uint128 tokens,
522         address send_gas_to,
523         address callback_address,
524         TvmCell callback_payload
525     ) override external onlyRoot {
526         require(tokens > 0);
527         require(tokens <= balance_, TONTokenWalletErrors.
                error_not_enough_balance);
528
529         tvm.rawReserve(address(this).balance - msg.value, 2);
```

```
530
531         balance_ -= tokens;
532
533         IBurnableTokenRootContract(root_address)
534             .tokensBurned{ value: 0, flag: 128, bounce: true }(
535                 tokens,
536                 wallet_public_key,
537                 owner_address,
538                 send_gas_to,
539                 callback_address,
540                 callback_payload
541             );
542     }
```

37.7.9 Function destroy

- TODO

```
584     function destroy(
585         address gas_dest
586     )
587         override
588         public
589         onlyOwner
590     {
591         require(balance_ == 0);
592         tvm.accept();
593         selfdestruct(gas_dest);
594     }
```

37.7.10 Function disapprove

- TODO

```
148     function disapprove() override external onlyOwner {
149         if (owner_address.value != 0 ) {
150             tvm.rawReserve(math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value), 2);
151         } else {
152             tvm.accept();
153         }
154
155         allowance_.reset();
156
157         if (owner_address.value != 0 ) {
158             msg.sender.transfer({ value: 0, flag: 128 });
159         }
160     }
```

37.7.11 Function getDetails

- TODO

```
72      function getDetails() override external view responsible
            returns (ITONTokenWalletDetails) {
73          return { value: 0, bounce: false, flag: 64 }
                ITONTokenWalletDetails(
74              root_address,
75              wallet_public_key,
76              owner_address,
77              balance_,
78              receive_callback,
79              bounced_callback,
80              allow_non_notifiable
81          );
82      }
```

37.7.12 Function getVersion

- TODO

```
54      function getVersion() override external pure responsible
            returns (uint32) {
55          return 4;
56      }
```

37.7.13 Function getWalletCode

- TODO

```
87      function getWalletCode() override external view responsible
            returns (TvmCell) {
88          return { value: 0, bounce: false, flag: 64 } code;
89      }
```

37.7.14 Function internalTransfer

- TODO

```
370     function internalTransfer(
371         uint128 tokens,
372         uint256 sender_public_key,
373         address sender_address,
374         address send_gas_to,
375         bool notify_receiver,
376         TvmCell payload
377     )
```

```
378         override
379         external
380     {
381         require(notify_receiver || allow_non_notifiable ||
                receive_callback.value == 0,
382                     TONTokenWalletErrors.
                            error_recipient_has_disallow_non_notifiable);
383         address expectedSenderAddress = getExpectedAddress(
                sender_public_key, sender_address);
384         require(msg.sender == expectedSenderAddress,
                TONTokenWalletErrors.
                error_message_sender_is_not_good_wallet);
385         require(sender_address != owner_address ||
                sender_public_key != wallet_public_key,
                TONTokenWalletErrors.error_wrong_recipient);
386
387         if (owner_address.value != 0 ) {
388             uint128 reserve = math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value);
389             require(address(this).balance > reserve,
                    TONTokenWalletErrors.error_low_message_value);
390             tvm.rawReserve(reserve, 2);
391         } else {
392             tvm.rawReserve(address(this).balance - msg.value, 2);
393         }
394
395         balance_ += tokens;
396
397         if (notify_receiver && receive_callback.value != 0) {
398             ITokensReceivedCallback(receive_callback).
                    tokensReceivedCallback{ value: 0, flag: 128 }(
399                 address(this),
400                 root_address,
401                 tokens,
402                 sender_public_key,
403                 sender_address,
404                 msg.sender,
405                 send_gas_to,
406                 balance_,
407                 payload
408             );
409         } else {
410             send_gas_to.transfer({ value: 0, flag: 128 });
411         }
412     }
```

37.7.15 Function internalTransferFrom

- TODO

```
423     function internalTransferFrom(
424         address to,
425         uint128 tokens,
426         address send_gas_to,
```

```
427         bool notify_receiver,
428         TvmCell payload
429     )
430         override
431         external
432     {
433         require(allowance_.hasValue(), TONTokenWalletErrors.
                error_no_allowance_set);
434         require(msg.sender == allowance_.get().spender,
                TONTokenWalletErrors.error_wrong_spender);
435         require(tokens <= allowance_.get().remaining_tokens,
                TONTokenWalletErrors.error_not_enough_allowance);
436         require(tokens <= balance_, TONTokenWalletErrors.
                error_not_enough_balance);
437         require(tokens > 0);
438         require(to != address(this), TONTokenWalletErrors.
                error_wrong_recipient);
439
440         if (owner_address.value != 0 ) {
441             uint128 reserve = math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value);
442             require(address(this).balance > reserve +
                    TONTokenWalletConstants.target_gas_balance,
                    TONTokenWalletErrors.error_low_message_value);
443             tvm.rawReserve(reserve, 2);
444             tvm.rawReserve(math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value), 2);
445         } else {
446             require(msg.value > TONTokenWalletConstants.
                    target_gas_balance, TONTokenWalletErrors.
                    error_low_message_value);
447             tvm.rawReserve(address(this).balance - msg.value, 2);
448         }
449
450         balance_ -= tokens;
451
452         allowance_.set(AllowanceInfo(allowance_.get().
                remaining_tokens - tokens, allowance_.get().spender));
453
454         ITONTokenWallet(to).internalTransfer{ value: 0, bounce:
                true, flag: 129 }(
455             tokens,
456             wallet_public_key,
457             owner_address,
458             send_gas_to,
459             notify_receiver,
460             payload
461         );
462     }
```

37.7.16 Function setBouncedCallback

- TODO

```
568     function setBouncedCallback(
569         address bounced_callback_
570     )
571         override
572         external
573         onlyOwner
574     {
575         tvm.accept();
576         bounced_callback = bounced_callback_;
577     }
```

37.7.17 Function setReceiveCallback

- TODO

```
550     function setReceiveCallback(
551         address receive_callback_,
552         bool allow_non_notifiable_
553     )
554         override
555         external
556         onlyOwner
557     {
558         tvm.accept();
559         receive_callback = receive_callback_;
560         allow_non_notifiable = allow_non_notifiable_;
561     }
```

37.7.18 Function transfer

- TODO

```
262     function transfer(
263         address to,
264         uint128 tokens,
265         uint128 grams,
266         address send_gas_to,
267         bool notify_receiver,
268         TvmCell payload
269     ) override external onlyOwner {
270         require(tokens > 0);
271         require(tokens <= balance_, TONTokenWalletErrors.
                error_not_enough_balance);
272         require(to.value != 0, TONTokenWalletErrors.
                error_wrong_recipient);
273         require(to != address(this), TONTokenWalletErrors.
                error_wrong_recipient);
274
275         if (owner_address.value != 0 ) {
276             uint128 reserve = math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value);
```

```
277             require(address(this).balance > reserve +
                    TONTokenWalletConstants.target_gas_balance,
                    TONTokenWalletErrors.error_low_message_value);
278             tvm.rawReserve(reserve, 2);
279             balance_ -= tokens;
280
281             ITONTokenWallet(to).internalTransfer{ value: 0, flag:
                    129, bounce: true }(
282                 tokens,
283                 wallet_public_key,
284                 owner_address,
285                 send_gas_to.value != 0 ? send_gas_to :
                        owner_address,
286                 notify_receiver,
287                 payload
288             );
289         } else {
290             require(address(this).balance > grams,
                    TONTokenWalletErrors.error_low_message_value);
291             require(grams > TONTokenWalletConstants.
                    target_gas_balance, TONTokenWalletErrors.
                    error_low_message_value);
292             tvm.accept();
293             balance_ -= tokens;
294
295             ITONTokenWallet(to).internalTransfer{ value: grams,
                    bounce: true, flag: 1 }(
296                 tokens,
297                 wallet_public_key,
298                 owner_address,
299                 send_gas_to.value != 0 ? send_gas_to : address(this
                        ),
300                 notify_receiver,
301                 payload
302             );
303         }
304     }
```

37.7.19 Function transferFrom

- TODO

```
317     function transferFrom(
318         address from,
319         address to,
320         uint128 tokens,
321         uint128 grams,
322         address send_gas_to,
323         bool notify_receiver,
324         TvmCell payload
325     )
326         override
327         external
328         onlyOwner
329     {
```

```
330         require(to.value != 0, TONTokenWalletErrors.
                error_wrong_recipient);
331         require(tokens > 0);
332         require(from != to, TONTokenWalletErrors.
                error_wrong_recipient);
333
334         if (owner_address.value != 0 ) {
335             uint128 reserve = math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value);
336             require(address(this).balance > reserve + (
                    TONTokenWalletConstants.target_gas_balance * 2),
                    TONTokenWalletErrors.error_low_message_value);
337             tvm.rawReserve(reserve, 2);
338
339             ITONTokenWallet(from).internalTransferFrom{ value: 0,
                    flag: 129 }(
340                 to,
341                 tokens,
342                 send_gas_to.value != 0 ? send_gas_to :
                        owner_address,
343                 notify_receiver,
344                 payload
345             );
346         } else {
347             require(address(this).balance > grams,
                    TONTokenWalletErrors.error_low_message_value);
348             require(grams > TONTokenWalletConstants.
                    target_gas_balance * 2, TONTokenWalletErrors.
                    error_low_message_value);
349             tvm.accept();
350             ITONTokenWallet(from).internalTransferFrom{ value:
                    grams, flag: 1 }(
351                 to,
352                 tokens,
353                 send_gas_to.value != 0 ? send_gas_to : address(this
                        ),
354                 notify_receiver,
355                 payload
356             );
357         }
358     }
```

37.7.20 Function transferToRecipient

- TODO

```
177     function transferToRecipient(
178         uint256 recipient_public_key,
179         address recipient_address,
180         uint128 tokens,
181         uint128 deploy_grams,
182         uint128 transfer_grams,
183         address send_gas_to,
184         bool notify_receiver,
```

```
185         TvmCell payload
186     ) override external onlyOwner {
187         require(tokens > 0);
188         require(tokens <= balance_, TONTokenWalletErrors.
                error_not_enough_balance);
189         require(recipient_address.value == 0 ||
                recipient_public_key == 0, TONTokenWalletErrors.
                error_wrong_recipient);
190
191         if (owner_address.value != 0 ) {
192             uint128 reserve = math.max(TONTokenWalletConstants.
                    target_gas_balance, address(this).balance - msg.
                    value);
193             require(address(this).balance > reserve +
                    TONTokenWalletConstants.target_gas_balance +
                    deploy_grams, TONTokenWalletErrors.
                    error_low_message_value);
194             require(recipient_address != owner_address,
                    TONTokenWalletErrors.error_wrong_recipient);
195             tvm.rawReserve(reserve, 2);
196         } else {
197             require(address(this).balance > deploy_grams +
                    transfer_grams, TONTokenWalletErrors.
                    error_low_message_value);
198             require(transfer_grams > TONTokenWalletConstants.
                    target_gas_balance, TONTokenWalletErrors.
                    error_low_message_value);
199             require(recipient_public_key != wallet_public_key);
200             tvm.accept();
201         }
202
203         TvmCell stateInit = tvm.buildStateInit({
204             contr: TONTokenWallet,
205             varInit: {
206                 root_address: root_address,
207                 code: code,
208                 wallet_public_key: recipient_public_key,
209                 owner_address: recipient_address
210             },
211             pubkey: recipient_public_key,
212             code: code
213         });
214
215         address to;
216
217         if(deploy_grams > 0) {
218             to = new TONTokenWallet{
219                 stateInit: stateInit,
220                 value: deploy_grams,
221                 wid: address(this).wid,
222                 flag: 1
223             }();
224         } else {
225             to = address(tvm.hash(stateInit));
226         }
227
228         if (owner_address.value != 0 ) {
```

```
229             balance_ -= tokens;
230             ITONTokenWallet(to).internalTransfer{ value: 0, flag:
                    129, bounce: true }(
231                 tokens,
232                 wallet_public_key,
233                 owner_address,
234                 send_gas_to.value != 0 ? send_gas_to :
                        owner_address,
235                 notify_receiver,
236                 payload
237             );
238         } else {
239             balance_ -= tokens;
240             ITONTokenWallet(to).internalTransfer{ value:
                    transfer_grams, flag: 1, bounce: true }(
241                 tokens,
242                 wallet_public_key,
243                 owner_address,
244                 send_gas_to.value != 0 ? send_gas_to : address(this
                        ),
245                 notify_receiver,
246                 payload
247             );
248         }
249     }
```

37.8 Internal Method Definitions

37.8.1 Function getExpectedAddress

- TODO

```
620     function getExpectedAddress(
621         uint256 wallet_public_key_,
622         address owner_address_
623     )
624         private
625         inline
626         view
627     returns (
628         address
629     ) {
630         TvmCell stateInit = tvm.buildStateInit({
631             contr: TONTokenWallet,
632             varInit: {
633                 root_address: root_address,
634                 code: code,
635                 wallet_public_key: wallet_public_key_,
636                 owner_address: owner_address_
637             },
638             pubkey: wallet_public_key_,
639             code: code
640         });
641
```

```
642         return address(tvm.hash(stateInit));
643     }
```

Chapter 38

Contract UserAccount

Contents

38.1 Overview

In file `UserAccount.sol`

38.2 Contract Inheritance

IUserAccount	
IUserAccountData	
IUpgradableContract	
IUserAccountGetters	

38.3 Static Variable Definitions

address	owner	
		used in @13.UserAccount.writeWithdrawInfo
		used in @13.UserAccount.writeWithdrawInfo
		used in @13.UserAccount.writeSupplyInfo
		used in @13.UserAccount.writeRepayInformation
		used in @13.UserAccount.writeRepayInformation
		used in @13.UserAccount.writeRepayInformation
		used in @13.UserAccount.writeBorrowInformation
		used in @13.UserAccount.writeBorrowInformation
		used in @13.UserAccount.writeBorrowInformation
		used in @13.UserAccount.withdrawExtraTons
		used in @13.UserAccount.withdraw
		used in @13.UserAccount.withdraw
		used in @13.UserAccount.upgradeContractCode
		used in @13.UserAccount.sendRepayInfo
		used in @13.UserAccount.requestWithdrawInfo
		used in @13.UserAccount.requestWithdrawInfo
		used in @13.UserAccount.requestLiquidationInformation
		assigned in @13.UserAccount.onCodeUpgrade
		used in @13.UserAccount.onCodeUpgrade
		used in @13.UserAccount.liquidateVTokens
		used in @13.UserAccount.grantVTokens
		used in @13.UserAccount.grantVTokens
		used in @13.UserAccount.grantVTokens
		used in @13.UserAccount.grantVTokens
		used in @13.UserAccount.getOwner
		used in @13.UserAccount.enterMarket

```
23     address static public owner;
```

38.4 Variable Definitions

bool	borrowLock	
		assigned in @13.UserAccount.writeBorrowInformation
		used in @13.UserAccount.writeBorrowInformation
		used in @13.UserAccount.upgradeContractCode
		used in @13.UserAccount.upgradeContractCode
		assigned in @13.UserAccount.updateUserAccountHealth
		used in @13.UserAccount.updateUserAccountHealth
		assigned in @13.UserAccount.onCodeUpgrade
		used in @13.UserAccount.onCodeUpgrade
		assigned in @13.UserAccount.disableBorrowLock
		used in @13.UserAccount.disableBorrowLock
		assigned in @13.UserAccount.borrow
		used in @13.UserAccount.borrow
		used in @13.UserAccount.borrow
bool	liquidationLock	
		used in @13.UserAccount.withdraw
		used in @13.UserAccount.upgradeContractCode
		assigned in @13.UserAccount.updateUserAccountHealth
		used in @13.UserAccount.updateUserAccountHealth
		assigned in @13.UserAccount.onCodeUpgrade
		used in @13.UserAccount.onCodeUpgrade
		used in @13.UserAccount.borrow
address	userAccountManager	
		used in @13.UserAccount.withdraw
		used in @13.UserAccount.upgradeContractCode
		used in @13.UserAccount.updateUserAccountHealth
		used in @13.UserAccount.sendRepayInfo
		used in @13.UserAccount.requestWithdrawInfo
		used in @13.UserAccount.requestLiquidationInformation

```
20      bool public borrowLock;

21      bool public liquidationLock;

26      address public userAccountManager;

29      uint32 public contractCodeVersion;

31      fraction public accountHealth;

33      mapping(uint32 => bool) knownMarkets;

34      mapping(uint32 => UserMarketInfo) markets;
```

38.5 Modifier Definitions

38.5.1 Modifier onlyOwner

```
412     modifier onlyOwner() {
413         require(msg.sender == owner);
414         _;
415     }
```

38.5.2 Modifier onlyUserAccountManager

```
417     modifier onlyUserAccountManager() {
418         require(msg.sender == userAccountManager);
419         _;
420     }
```

38.5.3 Modifier onlySelf

```
422     modifier onlySelf() {
423         require(msg.sender == address(this));
424         _;
425     }
```

38.5.4 Modifier onlyExecutor

```
427     modifier onlyExecutor() {
428         require(
429             msg.sender == userAccountManager ||
430             msg.sender == owner ||
431             msg.sender == address(this)
432         );
433         _;
434     }
```

38.6 Constructor Definitions

38.6.1 Constructor

Critical issue: Constructor for UserAccount (fake)

loren ipsum loren ipsum

- TODO

```
49      constructor() public {
50          tvm.accept();
51          userAccountManager = msg.sender;
52      }
```

38.7 Public Method Definitions

38.7.1 Function abortLiquidation

- TODO

```
327     function abortLiquidation(address tonWallet, address
            tip3UserWallet, uint32 marketId, uint256 tokensProvided)
            external override onlyUserAccountManager {
328         if (tokensProvided != 0) {
329             _checkUserAccountHealth(owner,
                    _createTokenPayoutPayload(tonWallet, tip3UserWallet
                    , marketId, tokensProvided));
330         } else {
331             _checkUserAccountHealth(owner, _createNoOpPayload());
332         }
333     }
```

38.7.2 Function borrow

- TODO

```
169     function borrow(uint32 marketId, uint256 amountToBorrow,
            address userTip3Wallet) external override onlyOwner {
170         tvm.rawReserve(msg.value, 2);
171         if (
172             (!borrowLock) &&
173             (accountHealth.nom > accountHealth.denom) &&
174             !liquidationLock
175         ) {
```

```
176             borrowLock = true;
177             TvmBuilder tb;
178             tb.store(owner);
179             tb.store(userTip3Wallet);
180             tb.store(amountToBorrow);
181             IUAMUserAccount(userAccountManager).requestIndexUpdate{
182                 flag: MsgFlag.REMAINING_GAS
183             }(owner, marketId, tb.toCell());
184         } else {
185             address(msg.sender).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
186         }
187     }
```

38.7.3 Function borrowUpdateIndexes

- TODO

```
189     function borrowUpdateIndexes(uint32 marketId, mapping(uint32 =>
            fraction) updatedIndexes, address userTip3Wallet, uint256
        toBorrow) external override onlyUserAccountManager {
190         tvm.rawReserve(msg.value, 2);
191
192         _updateIndexes(updatedIndexes);
193
194         mapping(uint32 => BorrowInfo) borrowInfo;
195         mapping(uint32 => uint256) supplyInfo;
196
197         (borrowInfo, supplyInfo) = _getBorrowSupplyInfo();
198
199         IUAMUserAccount(userAccountManager).passBorrowInformation{
200             flag: MsgFlag.REMAINING_GAS
201         }(owner, userTip3Wallet, marketId, toBorrow, supplyInfo,
                borrowInfo);
202     }
```

38.7.4 Function checkUserAccountHealth

- TODO

```
249     function checkUserAccountHealth(address gasTo) external
            override onlyExecutor {
250         tvm.rawReserve(msg.value, 2);
251         TvmBuilder no_op;
252         no_op.store(OperationCodes.NO_OP);
253
254         _checkUserAccountHealth(gasTo, no_op.toCell());
255     }
```

38.7.5 Function disableBorrowLock

- TODO

```
379     function disableBorrowLock() external override
            onlyUserAccountManager {
380         tvm.rawReserve(msg.value, 2);
381         borrowLock = false;
382         address(userAccountManager).transfer({value: 0, flag:
                MsgFlag.REMAINING_GAS});
383     }
```

38.7.6 Function enterMarket

- TODO

```
391     function enterMarket(uint32 marketId) external override
            onlyOwner {
392         tvm.rawReserve(msg.value, 2);
393         if (!knownMarkets[marketId]) {
394             knownMarkets[marketId] = true;
395
396             markets[marketId].exists = true;
397             markets[marketId]._marketId = marketId;
398             markets[marketId].suppliedTokens = 0;
399         }
400         address(owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
401     }
```

38.7.7 Function getAllMarketsInfo

- TODO

```
40      function getAllMarketsInfo() external override view responsible
             returns(mapping(uint32 => UserMarketInfo)) {
41          return {flag: MsgFlag.REMAINING_GAS} markets;
42      }
```

38.7.8 Function getKnownMarkets

- TODO

```
36      function getKnownMarkets() external override view responsible
            returns(mapping(uint32 => bool)) {
37          return {flag: MsgFlag.REMAINING_GAS} knownMarkets;
38      }
```

38.7.9 Function getMarketInfo

- TODO

```
44      function getMarketInfo(uint32 marketId) external override view
            responsible returns(UserMarketInfo) {
45          return {flag: MsgFlag.REMAINING_GAS} markets[marketId];
46      }
```

38.7.10 Function getOwner

- TODO

```
105     function getOwner() external override responsible view returns(
            address) {
106         return { value: 0, bounce: false, flag: MsgFlag.
                REMAINING_GAS } owner;
107     }
```

38.7.11 Function grantVTokens

- TODO

```
312     function grantVTokens(address tip3UserWallet, uint32 marketId,
            uint32 marketToLiquidate, uint256 tokensToSeize, uint256
            tokensToReturn, uint256 tokensFromReserve) external
            override onlyUserAccountManager {
313         markets[marketToLiquidate].suppliedTokens += tokensToSeize;
314         if (tokensFromReserve != 0) {
315             IUAMUserAccount(userAccountManager).returnAndSupply{
316                 flag: MsgFlag.REMAINING_GAS
317             }(owner, tip3UserWallet, marketId, marketToLiquidate,
                    tokensToReturn, tokensFromReserve);
318         } else {
319             if (tokensToReturn != 0) {
320                 _checkUserAccountHealth(owner,
                        _createTokenPayoutPayload(owner, tip3UserWallet
                        , marketId, tokensToReturn));
321             } else {
322                 _checkUserAccountHealth(owner, _createNoOpPayload()
                        );
323             }
324         }
325     }
```

38.7.12 Function liquidateVTokens

- TODO

```
303     function liquidateVTokens(address tonWallet, address
            tip3UserWallet, uint32 marketId, uint32 marketToLiquidate,
            uint256 tokensToSeize, uint256 tokensToReturn, uint256
            tokensFromReserve, BorrowInfo borrowInfo) external override
             onlyUserAccountManager {
304         markets[marketToLiquidate].suppliedTokens -= tokensToSeize;
305         markets[marketId].borrowInfo = borrowInfo;
306
307         IUAMUserAccount(userAccountManager).grantVTokens{
308             flag: MsgFlag.REMAINING_GAS
309         }(tonWallet, owner, tip3UserWallet, marketId,
                marketToLiquidate, tokensToSeize, tokensToReturn,
                tokensFromReserve);
310     }
```

38.7.13 Function removeMarket

- TODO

```
283     function removeMarket(uint32 marketId) external override
            onlyUserAccountManager {
284         tvm.rawReserve(msg.value, 2);
285         delete markets[marketId];
286         delete knownMarkets[marketId];
287         address(userAccountManager).transfer({value: 0, flag:
                MsgFlag.REMAINING_GAS});
288     }
```

38.7.14 Function requestLiquidationInformation

- TODO

```
293     function requestLiquidationInformation(address tonWallet,
            address tip3UserWallet, uint32 marketId, uint32
            marketToLiquidate, uint256 tokensProvided, mapping(uint32
            => fraction) updatedIndexes) external override
            onlyUserAccountManager {
294         _updateIndexes(updatedIndexes);
295
296         (mapping(uint32 => BorrowInfo) borrowInfo, mapping(uint32
                => uint256) supplyInfo) = _getBorrowSupplyInfo();
297
298         IUAMUserAccount(userAccountManager).
                receiveLiquidationInformation{
299             flag: MsgFlag.REMAINING_GAS
300         }(tonWallet, owner, tip3UserWallet, marketId,
                marketToLiquidate, tokensProvided, supplyInfo,
                borrowInfo);
301     }
```

38.7.15 Function requestWithdrawInfo

- TODO

```
138     function requestWithdrawInfo(address userTip3Wallet, uint32
            marketId, uint256 tokensToWithdraw, mapping(uint32 =>
            fraction) updatedIndexes) external override
            onlyUserAccountManager {
139         tvm.rawReserve(msg.value, 2);
140         if (
141             accountHealth.nom > accountHealth.denom
142         ) {
143             for ((uint32 marketId_, fraction index): updatedIndexes
                    ) {
144                 _updateMarketInfo(marketId_, index);
145             }
146
147             mapping(uint32 => BorrowInfo) borrowInfo;
148             mapping(uint32 => uint256) supplyInfo;
149
150             (borrowInfo, supplyInfo) = _getBorrowSupplyInfo();
151
152             IUAMUserAccount(userAccountManager).receiveWithdrawInfo
                    {
153                 flag: MsgFlag.REMAINING_GAS
154             }(owner, userTip3Wallet, tokensToWithdraw, marketId,
                    supplyInfo, borrowInfo);
155         } else {
156             address(owner).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
157         }
158     }
```

38.7.16 Function sendRepayInfo

- TODO

```
223     function sendRepayInfo(address userTip3Wallet, uint32 marketId,
             uint256 tokensForRepay, mapping(uint32 => fraction)
            updatedIndexes) external override onlyUserAccountManager {
224         tvm.rawReserve(msg.value, 2);
225         for ((uint32 marketId_, fraction index): updatedIndexes) {
226             _updateMarketInfo(marketId_, index);
227         }
228
229         IUAMUserAccount(userAccountManager).receiveRepayInfo{
230             flag: MsgFlag.REMAINING_GAS
231         }(owner, userTip3Wallet, tokensForRepay, marketId, markets[
                marketId].borrowInfo);
232     }
```

38.7.17 Function updateUserAccountHealth

- TODO

```
266     function updateUserAccountHealth(address gasTo, fraction
            _accountHealth, mapping(uint32 => fraction) updatedIndexes,
             TvmCell dataToTransfer) external override
            onlyUserAccountManager {
267         accountHealth = _accountHealth;
268         liquidationLock = accountHealth.denom > accountHealth.nom;
269         borrowLock = accountHealth.denom > accountHealth.nom;
270         _updateIndexes(updatedIndexes);
271         TvmSlice ts = dataToTransfer.toSlice();
272         (uint8 operation) = ts.decode(uint8);
273         if (operation == OperationCodes.REQUEST_TOKEN_PAYOUT) {
274             (address tonWallet, address userTip3Wallet, uint32
                    marketId, uint256 tokensToPayout) = ts.decode(
                    address, address, uint32, uint256);
275             IUAMUserAccount(userAccountManager).requestTokenPayout{
276                 flag: MsgFlag.REMAINING_GAS
277             }(tonWallet, userTip3Wallet, marketId, tokensToPayout);
278         } else {
279             address(gasTo).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
280         }
281     }
```

38.7.18 Function upgradeContractCode

- TODO

```
54      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) override external
            onlyUserAccountManager {
55          require(!borrowLock);
56          tvm.accept();
57
58          bool _borrowLock = borrowLock;
59          bool _liquidationLock = liquidationLock;
60          address _owner = owner;
61          address _userAccountManager = userAccountManager;
62          mapping (uint32 => bool) _knownMarkets = knownMarkets;
63          mapping (uint32 => UserMarketInfo) _markets = markets;
64          fraction _accountHealth = accountHealth;
65
66          tvm.setcode(code);
67          tvm.setCurrentCode(code);
68
69          onCodeUpgrade(
70              _borrowLock,
71              _liquidationLock,
72              _owner,
73              _userAccountManager,
74              _knownMarkets,
```

```
75              _markets,
76              _accountHealth,
77              updateParams,
78              codeVersion
79          );
80      }
```

38.7.19 Function withdraw

- TODO

```
123     function withdraw(address userTip3Wallet, uint32 marketId,
            uint256 tokensToWithdraw) external override view onlyOwner
            {
124         if (
125             !liquidationLock &&
126             tokensToWithdraw <= markets[marketId].suppliedTokens
127         ) {
128             tvm.rawReserve(msg.value, 2);
129 
130             IUAMUserAccount(userAccountManager).requestWithdraw{
131                 flag: MsgFlag.REMAINING_GAS
132             }(owner, userTip3Wallet, marketId, tokensToWithdraw);
133         } else {
134             address(owner).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
135         }
136     }
```

38.7.20 Function withdrawExtraTons

- TODO

```
406     function withdrawExtraTons() external override view onlyOwner {
407         address(owner).transfer({ value: 0, bounce: false, flag:
                MsgFlag.ALL_NOT_RESERVED });
408     }
```

38.7.21 Function writeBorrowInformation

- TODO

```
204     function writeBorrowInformation(uint32 marketId, uint256
            toBorrow, address userTip3Wallet, fraction marketIndex)
            external override onlyUserAccountManager {
205         tvm.rawReserve(msg.value, 2);
206         if (toBorrow > 0) {
207             _updateMarketInfo(marketId, marketIndex);
208             markets[marketId].borrowInfo.tokensBorrowed += toBorrow
                    ;
```

```
209         }
210
211         borrowLock = false;
212
213         if (toBorrow > 0) {
214             _checkUserAccountHealth(owner,
                    _createTokenPayoutPayload(owner, userTip3Wallet,
                    marketId, toBorrow));
215         } else {
216             _checkUserAccountHealth(owner, _createNoOpPayload());
217         }
218     }
```

38.7.22 Function writeRepayInformation

- TODO

```
234     function writeRepayInformation(address userTip3Wallet, uint32
            marketId, uint256 tokensToReturn, BorrowInfo bi) external
            override onlyUserAccountManager {
235         tvm.rawReserve(msg.value, 2);
236
237         markets[marketId].borrowInfo = bi;
238
239         if (tokensToReturn != 0) {
240             _checkUserAccountHealth(owner,
                    _createTokenPayoutPayload(owner, userTip3Wallet,
                    marketId, tokensToReturn));
241         } else {
242             _checkUserAccountHealth(owner, _createNoOpPayload());
243         }
244     }
```

38.7.23 Function writeSupplyInfo

- TODO

```
112     function writeSupplyInfo(uint32 marketId, uint256
            tokensToSupply, fraction index) external override
            onlyUserAccountManager {
113         tvm.rawReserve(msg.value, 2);
114         markets[marketId].suppliedTokens += tokensToSupply;
115         _updateMarketInfo(marketId, index);
116
117         _checkUserAccountHealth(owner, _createNoOpPayload());
118     }
```

38.7.24 Function writeWithdrawInfo

- TODO

```
160     function writeWithdrawInfo(address userTip3Wallet, uint32
            marketId, uint256 tokensToWithdraw, uint256 tokensToSend)
            external override onlyUserAccountManager{
161         tvm.rawReserve(msg.value, 2);
162         markets[marketId].suppliedTokens -= tokensToWithdraw;
163         _checkUserAccountHealth(owner, _createTokenPayoutPayload(
                owner, userTip3Wallet, marketId, tokensToSend));
164     }
```

38.8 Internal Method Definitions

38.8.1 Function _checkUserAccountHealth

- TODO

```
257     function _checkUserAccountHealth(address gasTo, TvmCell
            dataToTransfer) internal view {
258         mapping(uint32 => uint256) supplyInfo;
259         mapping(uint32 => BorrowInfo) borrowInfo;
260         (borrowInfo, supplyInfo) = _getBorrowSupplyInfo();
261         IUAMUserAccount(userAccountManager).
                calculateUserAccountHealth{
262             flag: MsgFlag.REMAINING_GAS
263         }(owner, gasTo, supplyInfo, borrowInfo, dataToTransfer);
264     }
```

38.8.2 Function _createNoOpPayload

- TODO

```
373     function _createNoOpPayload() internal pure returns (TvmCell) {
374         TvmBuilder no_op;
375         no_op.store(OperationCodes.NO_OP);
376         return no_op.toCell();
377     }
```

38.8.3 Function _createTokenPayoutPayload

- TODO

```
363     function _createTokenPayoutPayload(address tonWallet, address
            userTip3Wallet, uint32 marketId, uint256 tokensToSend)
            internal pure returns (TvmCell) {
364         TvmBuilder op;
365         op.store(OperationCodes.REQUEST_TOKEN_PAYOUT);
366         op.store(tonWallet);
367         op.store(userTip3Wallet);
368         op.store(marketId);
369         op.store(tokensToSend);
370         return op.toCell();
371     }
```

38.8.4 Function _getBorrowSupplyInfo

- TODO

```
356     function _getBorrowSupplyInfo() internal view returns(mapping(
            uint32 => BorrowInfo) borrowInfo, mapping(uint32 => uint256
            ) supplyInfo) {
357         for ((uint32 marketId, UserMarketInfo umi) : markets) {
358             supplyInfo[marketId] = umi.suppliedTokens;
359             borrowInfo[marketId] = umi.borrowInfo;
360         }
361     }
```

38.8.5 Function _updateIndexes

- TODO

```
338     function _updateIndexes(mapping(uint32 => fraction)
            updatedIndexes) internal {
339         for ((uint32 marketId_, fraction index): updatedIndexes) {
340             _updateMarketInfo(marketId_, index);
341         }
342     }
```

38.8.6 Function _updateMarketInfo

- TODO

```
344     function _updateMarketInfo(uint32 marketId, fraction index)
            internal {
345         fraction tmpf;
346         BorrowInfo bi = markets[marketId].borrowInfo;
347         if (markets[marketId].borrowInfo.tokensBorrowed != 0) {
348             tmpf = bi.tokensBorrowed.numFMul(index);
349             tmpf = tmpf.fDiv(bi.index);
350         } else {
351             tmpf = fraction(0, 1);
352         }
353         markets[marketId].borrowInfo = BorrowInfo(tmpf.toNum(),
                index);
354     }
```

38.8.7 Function onCodeUpgrade

- TODO

```
82      function onCodeUpgrade(
83          bool _borrowLock,
84          bool _liquidationLock,
85          address _owner,
86          address _userAccountManager,
87          mapping(uint32 => bool) _knownMarkets,
88          mapping(uint32 => UserMarketInfo) _markets,
89          fraction _accountHealth,
90          TvmCell,
91          uint32 codeVersion
92      ) private {
93          tvm.resetStorage();
94          borrowLock = _borrowLock;
95          liquidationLock = _liquidationLock;
96          owner = _owner;
97          userAccountManager = _userAccountManager;
98          knownMarkets = _knownMarkets;
99          markets = _markets;
100         accountHealth = _accountHealth;
101
102         contractCodeVersion = codeVersion;
103     }
```

38.8.7.0.1 Some functions inherited by using

Chapter 39

Contract UserAccountManager

Contents

39.1 Overview

In file `UserAccountManager.sol`

39.2 Contract Inheritance

IRoles	
IUpgradableContract	
IUserAccountManager	
IUAMUserAccount	
IUAMMarket	

39.3 Event Definitions

```
33     event AccountCreated(address tonWallet, address userAddress);
```

39.4 Variable Definitions

uint32	contractCodeVersion	
		assigned in @14.UserAccountManager.onCodeUpgrade
		used in @14.UserAccountManager.onCodeUpgrade
address	marketAddress	
		used in @14.UserAccountManager.upgradeContractCode
		assigned in @14.UserAccountManager.setMarketAddress
		used in @14.UserAccountManager.setMarketAddress
		used in @14.UserAccountManager.returnAndSupply
		used in @14.UserAccountManager.returnAndSupply
		used in @14.UserAccountManager.returnAndSupply
		used in @14.UserAccountManager.requestWithdraw
		used in @14.UserAccountManager.requestTokenPayout
		used in @14.UserAccountManager.requestIndexUpdate
		assigned in @14.UserAccountManager.onCodeUpgrade
		used in @14.UserAccountManager.onCodeUpgrade
		used in @14.UserAccountManager.calculateUserAccountHealth
mapping (uint8 => address)	modules	
		used in @14.UserAccountManager.upgradeContractCode
		assigned in @14.UserAccountManager.removeModule
		used in @14.UserAccountManager.removeModule
		used in @14.UserAccountManager.removeModule
		used in @14.UserAccountManager.receiveWithdrawInfo
		used in @14.UserAccountManager.receiveRepayInfo
		used in @14.UserAccountManager.receiveLiquidationInformation
		used in @14.UserAccountManager.passBorrowInformation
		assigned in @14.UserAccountManager.onCodeUpgrade
		used in @14.UserAccountManager.onCodeUpgrade
		assigned in @14.UserAccountManager.addModule

```
26      uint32 public contractCodeVersion;

28      address public marketAddress;

29      mapping(uint8 => address) public modules;

30      mapping(address => bool) public existingModules;

31      mapping(uint32 => TvmCell) public userAccountCodes;
```

39.5 Modifier Definitions

39.5.1 Modifier onlyMarket

```
557     modifier onlyMarket() {
558         require(
559             msg.sender == marketAddress,
560             UserAccountErrorCodes.ERROR_NOT_MARKET
561         );
562         tvm.rawReserve(msg.value, 2);
563         _;
564     }
```

39.5.2 Modifier onlyTrusted

```
566     modifier onlyTrusted() {
567         require(
568             msg.sender == _owner ||
569             msg.sender == marketAddress ||
570             _canChangeParams[msg.sender],
571             UserAccountErrorCodes.ERROR_NOT_TRUSTED
572         );
573         _;
574     }
```

39.5.3 Modifier onlyModules

```
576     modifier onlyModules() {
577         require(
578             existingModules.exists(msg.sender),
579             UserAccountErrorCodes.ERROR_NOT_MODULE
580         );
581         _;
582     }
```

39.5.4 Modifier executor

```
584     modifier executor() {
585         require(
586             msg.sender == _owner ||
587             msg.sender == marketAddress ||
```

```
588             existingModules.exists(msg.sender),
589             UserAccountErrorCodes.ERROR_NOT_EXECUTOR
590         );
591         _;
592     }
```

39.5.5 Modifier onlyModule

```
594     modifier onlyModule(uint8 operationId) {
595         require(
596             msg.sender == modules[operationId],
597             UserAccountErrorCodes.ERROR_INVALID_MODULE
598         );
599         tvm.rawReserve(msg.value, 2);
600         _;
601     }
```

39.5.6 Modifier onlySelectedExecutors

```
603     modifier onlySelectedExecutors(uint8 operationId, address
            tonWallet) {
604         require(
605             (msg.sender == modules[operationId]) ||
606             (msg.sender == _calculateUserAccountAddress(tonWallet))
                    ,
607             UserAccountErrorCodes.ERROR_INVALID_EXECUTOR
608         );
609         _;
610     }
```

39.5.7 Modifier onlyValidUserAccount

```
615     modifier onlyValidUserAccount(address tonWallet) {
616         require(
617             msg.sender == _calculateUserAccountAddress(tonWallet),
618             UserAccountErrorCodes.INVALID_USER_ACCOUNT
619         );
620         tvm.rawReserve(msg.value, 2);
621         _;
622     }
```

39.5.8 Modifier onlyValidUserAccountNoReserve

```
624     modifier onlyValidUserAccountNoReserve(address tonWallet) {
625         require(
626             msg.sender == _calculateUserAccountAddress(tonWallet),
627             UserAccountErrorCodes.INVALID_USER_ACCOUNT
628         );
629         _;
630     }
```

39.6 Constructor Definitions

39.6.1 Constructor

Critical issue: Constructor for UserAccountManager (fake)
loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

39.7 Public Method Definitions

39.7.1 Function abortLiquidation

- TODO

```
    function abortLiquidation(
        address tonWallet,
        address targetUser,
        address tip3UserWallet,
        uint32 marketId,
        uint256 tokensProvided
    ) external override view onlyModule(OperationCodes.
        LIQUIDATE_TOKENS) {
        address userAccount = _calculateUserAccountAddress(
            targetUser);
        IUserAccountData(userAccount).abortLiquidation{
            flag: MsgFlag.REMAINING_GAS
        }(tonWallet, tip3UserWallet, marketId, tokensProvided);
    }
```

39.7.2 Function addModule

- TODO

```
    function addModule(uint8 operationId, address module) external
        override onlyTrusted {
        delete existingModules[module];
        modules[operationId] = module;
        existingModules[module] = true;
    }
```

39.7.3 Function calculateUserAccountAddress

- TODO

```
123     function calculateUserAccountAddress(address tonWallet)
            external override responsible view returns (address) {
124         return { value: 0, bounce: false, flag: MsgFlag.
                REMAINING_GAS } _calculateUserAccountAddress(tonWallet)
                ;
125     }
```

39.7.4 Function calculateUserAccountHealth

- TODO

```
446     function calculateUserAccountHealth(
447         address tonWallet,
448         address gasTo,
449         mapping(uint32 => uint256) supplyInfo,
450         mapping(uint32 => BorrowInfo) borrowInfo,
451         TvmCell dataToTransfer
452     ) external override view onlyValidUserAccount(tonWallet) {
453         tvm.rawReserve(msg.value, 2);
454         IMarketOperations(marketAddress).calculateUserAccountHealth
                {
455             flag: MsgFlag.REMAINING_GAS
456         }(tonWallet, gasTo, supplyInfo, borrowInfo, dataToTransfer)
                ;
457     }
```

39.7.5 Function createUserAccount

- TODO

```
97      function createUserAccount(address tonWallet) external override
             view {
98          tvm.rawReserve(msg.value, 2);
99
100         TvmSlice ts = userAccountCodes[0].toSlice();
101         require(!ts.empty());
102
103         address userAccount = new UserAccount{
104             value: UserAccountCostConstants.useForUADeploy,
105             code: userAccountCodes[0],
106             pubkey: 0,
107             varInit: {
108                 owner: tonWallet
109             }
110         }();
111
112         emit AccountCreated(tonWallet, userAccount);
```

```
113
114         IUserAccountManager(this).updateUserAccount{
115             value: msg.value - UserAccountCostConstants.
                    useForUADeploy - UserAccountCostConstants.
                    estimatedExecCost
116         }(tonWallet);
117     }
```

39.7.6 Function disableUserAccountLock

- TODO

```
525     function disableUserAccountLock(address tonWallet) external
            view onlyOwner {
526         tvm.rawReserve(msg.value, 2);
527         address userAccount = _calculateUserAccountAddress(
                tonWallet);
528         IUserAccountData(userAccount).disableBorrowLock{
529             flag: MsgFlag.REMAINING_GAS
530         }();
531     }
```

39.7.7 Function getUserAccountCode

- TODO

```
521     function getUserAccountCode(uint32 version) external override
            view responsible returns(TvmCell) {
522         return {flag: MsgFlag.REMAINING_GAS} userAccountCodes[
                version];
523     }
```

39.7.8 Function grantVTokens

- TODO

```
363     function grantVTokens(
364         address tonWallet,
365         address targetUser,
366         address tip3UserWallet,
367         uint32 marketId,
368         uint32 marketToLiquidate,
369         uint256 vTokensToGrant,
370         uint256 tokensToReturn,
371         uint256 tokensFromReserve
372     ) external override view onlyValidUserAccountNoReserve(
            targetUser) {
373         tvm.rawReserve(msg.value - UserAccountCostConstants.
                updateHealthCost, 2);
374
```

```
375         address targetAccount = _calculateUserAccountAddress(
                targetUser);
376         IUserAccountData(targetAccount).checkUserAccountHealth{
377             value: UserAccountCostConstants.updateHealthCost
378         }(tonWallet);
379
380         address userAccount = _calculateUserAccountAddress(
                tonWallet);
381         IUserAccountData(userAccount).grantVTokens{
382             flag: MsgFlag.REMAINING_GAS
383         }(tip3UserWallet, marketId, marketToLiquidate,
                vTokensToGrant, tokensToReturn, tokensFromReserve);
384     }
```

39.7.9 Function passBorrowInformation

- TODO

```
246     function passBorrowInformation(
247         address tonWallet,
248         address userTip3Wallet,
249         uint32 marketId,
250         uint256 tokensToBorrow,
251         mapping(uint32 => uint256) supplyInfo,
252         mapping(uint32 => BorrowInfo) borrowInfo
253     ) external override view onlyValidUserAccount(tonWallet) {
254         IBorrowModule(modules[OperationCodes.BORROW_TOKENS]).
                borrowTokensFromMarket{
255             flag: MsgFlag.REMAINING_GAS
256         }(tonWallet, userTip3Wallet, tokensToBorrow, marketId,
                supplyInfo, borrowInfo);
257     }
```

39.7.10 Function receiveLiquidationInformation

- TODO

```
331     function receiveLiquidationInformation(
332         address tonWallet,
333         address targetUser,
334         address tip3UserWallet,
335         uint32 marketId,
336         uint32 marketToLiquidate,
337         uint256 tokensProvided,
338         mapping(uint32 => uint256) supplyInfo,
339         mapping(uint32 => BorrowInfo) borrowInfo
340     ) external override view onlyValidUserAccount(targetUser) {
341         ILiquidationModule(modules[OperationCodes.LIQUIDATE_TOKENS
                ]).liquidate{
342             flag: MsgFlag.REMAINING_GAS
343         }(tonWallet, targetUser, tip3UserWallet, marketId,
                marketToLiquidate, tokensProvided, supplyInfo,
                borrowInfo);
344     }
```

39.7.11 Function receiveRepayInfo

- TODO

```
288     function receiveRepayInfo(
289         address tonWallet,
290         address userTip3Wallet,
291         uint256 tokensForRepay,
292         uint32 marketId,
293         BorrowInfo borrowInfo
294     ) external override view onlyValidUserAccount(tonWallet) {
295         IRepayModule(modules[OperationCodes.REPAY_TOKENS]).
                repayLoan{
296             flag: MsgFlag.REMAINING_GAS
297         }(tonWallet, userTip3Wallet, tokensForRepay, marketId,
                borrowInfo);
298     }
```

39.7.12 Function receiveWithdrawInfo

- TODO

```
194     function receiveWithdrawInfo(
195         address tonWallet,
196         address userTip3Wallet,
197         uint256 tokensToWithdraw,
198         uint32 marketId,
199         mapping(uint32 => uint256) supplyInfo,
200         mapping(uint32 => BorrowInfo) borrowInfo
201     ) external override view onlyValidUserAccount(tonWallet) {
202         IWithdrawModule(modules[OperationCodes.WITHDRAW_TOKENS]).
                withdrawTokensFromMarket{
203             flag: MsgFlag.REMAINING_GAS
204         }(tonWallet, userTip3Wallet, tokensToWithdraw, marketId,
                supplyInfo, borrowInfo);
205     }
```

39.7.13 Function removeMarket

- TODO

```
533     function removeMarket(address tonWallet, uint32 marketId)
            external view canChangeParams {
534         tvm.rawReserve(msg.value, 2);
535         address userAccount = _calculateUserAccountAddress(
                tonWallet);
536         IUserAccountData(userAccount).removeMarket{
537             flag: MsgFlag.REMAINING_GAS
538         }(marketId);
539     }
```

39.7.14 Function removeModule

- TODO

```
549     function removeModule(uint8 operationId) external override
            onlyTrusted {
550         delete existingModules[modules[operationId]];
551         delete modules[operationId];
552     }
```

39.7.15 Function requestIndexUpdate

- TODO

```
223     function requestIndexUpdate(
224         address tonWallet,
225         uint32 marketId,
226         TvmCell args
227     ) external override view onlyValidUserAccount(tonWallet) {
228         IMarketOperations(marketAddress).
                performOperationUserAccountManager{
229             flag: MsgFlag.REMAINING_GAS
230         }(OperationCodes.BORROW_TOKENS, marketId, args);
231     }
```

39.7.16 Function requestLiquidationInformation

- TODO

```
316     function requestLiquidationInformation(
317         address tonWallet,
318         address targetUser,
319         address tip3UserWallet,
320         uint32 marketId,
321         uint32 marketToLiquidate,
322         uint256 tokensProvided,
323         mapping(uint32 => fraction) updatedIndexes
324     ) external override view onlyModule(OperationCodes.
            LIQUIDATE_TOKENS) {
325         address userAccount = _calculateUserAccountAddress(
                targetUser);
326         IUserAccountData(userAccount).requestLiquidationInformation
                {
327             flag: MsgFlag.REMAINING_GAS
328         }(tonWallet, tip3UserWallet, marketId, marketToLiquidate,
                tokensProvided, updatedIndexes);
329     }
```

39.7.17 Function requestRepayInfo

- TODO

```
275     function requestRepayInfo(
276         address tonWallet,
277         address userTip3Wallet,
278         uint256 tokensForRepay,
279         uint32 marketId,
280         mapping(uint32 => fraction) updatedIndexes
281     ) external override view onlyModule(OperationCodes.REPAY_TOKENS
            ) {
282         address userAccount = _calculateUserAccountAddress(
                tonWallet);
283         IUserAccountData(userAccount).sendRepayInfo{
284             flag: MsgFlag.REMAINING_GAS
285         }(userTip3Wallet, marketId, tokensForRepay, updatedIndexes)
                ;
286     }
```

39.7.18 Function requestTokenPayout

- TODO

```
476     function requestTokenPayout(address tonWallet, address
            userTip3Wallet, uint32 marketId, uint256 toPayout) external
             override view onlySelectedExecutors(OperationCodes.
            LIQUIDATE_TOKENS, tonWallet) {
477         IMarketOperations(marketAddress).requestTokenPayout{
478             flag: MsgFlag.REMAINING_GAS
479         }(tonWallet, userTip3Wallet, marketId, toPayout);
480     }
```

39.7.19 Function requestUserAccountHealthCalculation

- TODO

```
438     function requestUserAccountHealthCalculation(address tonWallet)
             external override view executor {
439         tvm.rawReserve(msg.value, 2);
440         address userAccount = _calculateUserAccountAddress(
                tonWallet);
441         IUserAccountData(userAccount).checkUserAccountHealth{
442             flag: MsgFlag.REMAINING_GAS
443         }(tonWallet);
444     }
```

39.7.20 Function requestWithdraw

- TODO

```
166     function requestWithdraw(
167         address tonWallet,
168         address userTip3Wallet,
169         uint32 marketId,
170         uint256 tokensToWithdraw
171     ) external override view onlyValidUserAccount(tonWallet) {
172         TvmBuilder tb;
173         tb.store(tonWallet);
174         tb.store(userTip3Wallet);
175         tb.store(tokensToWithdraw);
176         IMarketOperations(marketAddress).
                performOperationUserAccountManager{
177             flag: MsgFlag.REMAINING_GAS
178         }(OperationCodes.WITHDRAW_TOKENS, marketId, tb.toCell());
179     }
```

39.7.21 Function requestWithdrawInfo

- TODO

```
181     function requestWithdrawInfo(
182         address tonWallet,
183         address userTip3Wallet,
184         uint256 tokensToWithdraw,
185         uint32 marketId,
186         mapping(uint32 => fraction) updatedIndexes
187     ) external override view onlyModule(OperationCodes.
            WITHDRAW_TOKENS) {
188         address userAccount = _calculateUserAccountAddress(
                tonWallet);
189         IUserAccountData(userAccount).requestWithdrawInfo{
190             flag: MsgFlag.REMAINING_GAS
191         }(userTip3Wallet, marketId, tokensToWithdraw,
                updatedIndexes);
192     }
```

39.7.22 Function returnAndSupply

- TODO

```
399     function returnAndSupply(
400         address tonWallet,
401         address tip3UserWallet,
402         uint32 marketId,
403         uint32 marketToLiquidate,
404         uint256 tokensToReturn,
405         uint256 tokensFromReserve
```

```
406     ) external override view onlyValidUserAccountNoReserve(
            tonWallet) {
407         if (tokensToReturn != 0) {
408             uint128 tonsToUse = msg.value / 4;
409             tvm.rawReserve(tonsToUse, 2);
410 
411             TvmBuilder tb;
412             tb.store(tonWallet);
413             tb.store(tokensFromReserve);
414 
415             IMarketOperations(marketAddress).
                    performOperationUserAccountManager{
416                 value: msg.value - tonsToUse
417             }(OperationCodes.SUPPLY_TOKENS, marketToLiquidate, tb.
                    toCell());
418 
419             IMarketOperations(marketAddress).requestTokenPayout{
420                 flag: MsgFlag.REMAINING_GAS
421             }(tonWallet, tip3UserWallet, marketId, tokensToReturn);
422         } else {
423             tvm.rawReserve(msg.value, 2);
424 
425             TvmBuilder tb;
426             tb.store(tonWallet);
427             tb.store(tokensFromReserve);
428 
429             IMarketOperations(marketAddress).
                    performOperationUserAccountManager{
430                 flag: MsgFlag.REMAINING_GAS
431             }(OperationCodes.SUPPLY_TOKENS, marketToLiquidate, tb.
                    toCell());
432         }
433     }
```

39.7.23 Function seizeTokens

- TODO

```
346     function seizeTokens(
347         address tonWallet,
348         address targetUser,
349         address tip3UserWallet,
350         uint32 marketId,
351         uint32 marketToLiquidate,
352         uint256 tokensToSeize,
353         uint256 tokensToReturn,
354         uint256 tokensFromReserve,
355         BorrowInfo borrowInfo
356     ) external override view onlyModule(OperationCodes.
            LIQUIDATE_TOKENS) {
357         address userAccount = _calculateUserAccountAddress(
                targetUser);
358         IUserAccountData(userAccount).liquidateVTokens{
359             flag: MsgFlag.REMAINING_GAS
```

```
360         }(tonWallet, tip3UserWallet, marketId, marketToLiquidate,
                tokensToSeize, tokensToReturn, tokensFromReserve,
                borrowInfo);
361     }
```

39.7.24 Function setMarketAddress

- TODO

```
493     function setMarketAddress(address _market) external override
            canChangeParams {
494         tvm.accept();
495         marketAddress = _market;
496     }
```

39.7.25 Function updateUserAccount

- TODO

```
506     function updateUserAccount(address tonWallet) external override
             {
507         tvm.rawReserve(msg.value, 2);
508         address userAccount = _calculateUserAccountAddress(
                tonWallet);
509         optional(uint32, TvmCell) latestVersion = userAccountCodes.
                max();
510         if (latestVersion.hasValue()) {
511             TvmCell empty;
512             (uint32 codeVersion, TvmCell code) = latestVersion.get
                    ();
513             IUpgradableContract(userAccount).upgradeContractCode{
514                 flag: MsgFlag.REMAINING_GAS
515             }(code, empty, codeVersion);
516         } else {
517             address(msg.sender).transfer({value: 0, flag: MsgFlag.
                    REMAINING_GAS});
518         }
519     }
```

39.7.26 Function updateUserAccountHealth

- TODO

```
459     function updateUserAccountHealth(
460         address tonWallet,
461         address gasTo,
462         fraction accountHealth,
463         mapping(uint32 => fraction) updatedIndexes,
464         TvmCell dataToTransfer
465     ) external override view onlyMarket {
```

```
466         tvm.rawReserve(msg.value, 2);
467         address userAccount = _calculateUserAccountAddress(
                tonWallet);
468         IUserAccountData(userAccount).updateUserAccountHealth{
469             flag: MsgFlag.REMAINING_GAS
470         }(gasTo, accountHealth, updatedIndexes, dataToTransfer);
471     }
```

39.7.27 Function updateUserIndexes

- TODO

```
233     function updateUserIndexes(
234         address tonWallet,
235         address userTip3Wallet,
236         uint256 tokensToBorrow,
237         uint32 marketId,
238         mapping(uint32 => fraction) updatedIndexes
239     ) external override view onlyModule(OperationCodes.
            BORROW_TOKENS) {
240         address userAccount = _calculateUserAccountAddress(
                tonWallet);
241         IUserAccountData(userAccount).borrowUpdateIndexes{
242             flag: MsgFlag.REMAINING_GAS
243         }(marketId, updatedIndexes, userTip3Wallet, tokensToBorrow)
                ;
244     }
```

39.7.28 Function upgradeContractCode

- TODO

```
56      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) override external canUpgrade {
57          tvm.accept();
58
59          tvm.setcode(code);
60          tvm.setCurrentCode(code);
61
62          onCodeUpgrade(
63              _owner,
64              marketAddress,
65              modules,
66              existingModules,
67              userAccountCodes,
68              updateParams,
69              codeVersion
70          );
71      }
```

39.7.29 Function uploadUserAccountCode

- TODO

```
500     function uploadUserAccountCode(uint32 version, TvmCell code)
            external override canChangeParams {
501         userAccountCodes[version] = code;
502
503         address(msg.sender).transfer({flag: MsgFlag.REMAINING_GAS,
                value: 0});
504     }
```

39.7.30 Function withdrawExtraTons

- TODO

```
482     function withdrawExtraTons(address tonWallet) external
            onlyOwner {
483         tvm.accept();
484         address(tonWallet).transfer({value: 0, flag: 160});
485     }
```

39.7.31 Function writeBorrowInformation

- TODO

```
259     function writeBorrowInformation(
260         address tonWallet,
261         address userTip3Wallet,
262         uint256 tokensToBorrow,
263         uint32 marketId,
264         fraction index
265     ) external override view onlyModule(OperationCodes.
            BORROW_TOKENS) {
266         address userAccount = _calculateUserAccountAddress(
                tonWallet);
267         IUserAccountData(userAccount).writeBorrowInformation{
268             flag: MsgFlag.REMAINING_GAS
269         }(marketId, tokensToBorrow, userTip3Wallet, index);
270     }
```

39.7.32 Function writeRepayInformation

- TODO

```
300     function writeRepayInformation(
301         address tonWallet,
302         address userTip3Wallet,
303         uint32 marketId,
304         uint256 tokensToReturn,
305         BorrowInfo bi
306     ) external override view onlyModule(OperationCodes.REPAY_TOKENS
            ) {
307         address userAccount = _calculateUserAccountAddress(
                tonWallet);
308         IUserAccountData(userAccount).writeRepayInformation{
309             flag: MsgFlag.REMAINING_GAS
310         }(userTip3Wallet, marketId, tokensToReturn, bi);
311     }
```

39.7.33 Function writeSupplyInfo

- TODO

```
151     function writeSupplyInfo(
152         address tonWallet,
153         uint32 marketId,
154         uint256 tokensToSupply,
155         fraction index
156     ) external override view onlyModule(OperationCodes.
            SUPPLY_TOKENS) {
157         address userAccount = _calculateUserAccountAddress(
                tonWallet);
158         IUserAccountData(userAccount).writeSupplyInfo{
159             flag: MsgFlag.REMAINING_GAS
160         }(marketId, tokensToSupply, index);
161     }
```

39.7.34 Function writeWithdrawInfo

- TODO

```
207     function writeWithdrawInfo(
208         address tonWallet,
209         address userTip3Wallet,
210         uint32 marketId,
211         uint256 tokensToWithdraw,
212         uint256 tokensToSend
213     ) external override view onlyModule(OperationCodes.
            WITHDRAW_TOKENS) {
214         address userAccount = _calculateUserAccountAddress(
                tonWallet);
215         IUserAccountData(userAccount).writeWithdrawInfo{
216             flag: MsgFlag.REMAINING_GAS
217         }(userTip3Wallet, marketId, tokensToWithdraw, tokensToSend)
                ;
218     }
```

39.8 Internal Method Definitions

39.8.1 Function _buildUserAccountData

- TODO

```
137     function _buildUserAccountData(address tonWallet) private view
            returns (TvmCell data) {
138         return tvm.buildStateInit({
139             contr: UserAccount,
140             varInit: {
141                 owner: tonWallet
142             },
143             pubkey: 0,
144             code: userAccountCodes[0]
145         });
146     }
```

39.8.2 Function _calculateUserAccountAddress

- TODO

```
130     function _calculateUserAccountAddress(address tonWallet)
            internal view returns(address) {
131         return address(tvm.hash(_buildUserAccountData(tonWallet)));
132     }
```

39.8.3 Function onCodeUpgrade

- TODO

```
73      function onCodeUpgrade(
74          address owner,
75          address _marketAddress,
76          mapping(uint8 => address) _modules,
77          mapping(address => bool) _existingModules,
78          mapping(uint32 => TvmCell) _userAccountCodes,
79          TvmCell,
80          uint32 _codeVersion
81      ) private {
82          tvm.accept();
83          tvm.resetStorage();
84          contractCodeVersion = _codeVersion;
85          _owner = owner;
86          marketAddress = _marketAddress;
87          modules = _modules;
88          existingModules = _existingModules;
89          userAccountCodes = _userAccountCodes;
90      }
```

Chapter 40

Contract WalletController

Contents

40.1 Overview

In file `WalletController.sol`

40.2 Contract Inheritance

IRoles	
IWCMInteractions	
IWalletControllerMarketManagement	
IWalletControllerGetters	
IUpgradableContract	
ITokensReceivedCallback	

40.3 Variable Definitions

uint32	contractCodeVersion	
		assigned in @15.WalletController.onCodeUpgrade
		used in @15.WalletController.onCodeUpgrade
address	marketAddress	
		used in @15.WalletController.upgradeContractCode
		used in @15.WalletController.tokensReceivedCallback
		used in @15.WalletController.tokensReceivedCallback
		used in @15.WalletController.tokensReceivedCallback
		assigned in @15.WalletController.setMarketAddress
		used in @15.WalletController.setMarketAddress
		assigned in @15.WalletController.onCodeUpgrade
		used in @15.WalletController.onCodeUpgrade
mapping (address => address)	wallets	
		used in @15.WalletController.upgradeContractCode
		assigned in @15.WalletController.removeMarket
		used in @15.WalletController.removeMarket
		assigned in @15.WalletController.receiveTIP3WalletAddress
		used in @15.WalletController.receiveTIP3WalletAddress
		assigned in @15.WalletController.onCodeUpgrade
		used in @15.WalletController.onCodeUpgrade
		used in @15.WalletController.getWallets
		assigned in @15.WalletController.addMarket
		used in @15.WalletController.addMarket
		used in @15.WalletController._transferTokensToWallet
mapping (address => bool)	realTokenRoots	
		used in @15.WalletController.upgradeContractCode
		assigned in @15.WalletController.removeMarket
		used in @15.WalletController.removeMarket
		assigned in @15.WalletCon-

```
29      uint32 public contractCodeVersion;

31      address public marketAddress;

34      mapping (address => address) public wallets;

35      mapping (address => bool) public realTokenRoots;

36      mapping (address => bool) public vTokenRoots;

37      mapping (address => uint32) public tokensToMarkets;

39      mapping (uint32 => MarketTokenAddresses) public marketTIP3Info;
```

40.4 Modifier Definitions

40.4.1 Modifier onlyMarket

```
295     modifier onlyMarket() {
296         require(msg.sender == marketAddress,
                WalletControllerErrorCodes.
                ERROR_MSG_SENDER_IS_NOT_MARKET);
297         _;
298     }
```

40.4.2 Modifier onlyTrusted

```
300     modifier onlyTrusted() {
301         require(
302             (msg.sender == marketAddress)
303         );
304         _;
305     }
```

40.4.3 Modifier onlyOwnWallet

```
311     modifier onlyOwnWallet(address tokenRoot, address tokenWallet)
            {
312         require(wallets[tokenRoot] == tokenWallet,
                WalletControllerErrorCodes.
                ERROR_MSG_SENDER_IS_NOT_OWN_WALLET);
313         _;
314     }
```

40.4.4 Modifier onlyExisingTIP3Root

```
319     modifier onlyExisingTIP3Root(address rootAddress) {
320         require(wallets.exists(rootAddress),
                WalletControllerErrorCodes.ERROR_TIP3_ROOT_IS_UNKNOWN);
321         _;
322     }
```

40.5 Constructor Definitions

40.5.1 Constructor

Critical issue: Constructor for WalletController (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum
loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
     } // Contract will be deployed using platform
```

40.6 Public Method Definitions

40.6.1 Function addMarket

- TODO

```
    function addMarket(uint32 marketId, address realTokenRoot)
        external override canChangeParams {
        tvm.accept();
        marketTIP3Info[marketId] = MarketTokenAddresses({
            realToken: realTokenRoot,
            realTokenWallet: address.makeAddrStd(0, 0)
        });

        realTokenRoots[realTokenRoot] = true;

        wallets[realTokenRoot] = address.makeAddrStd(0, 1);

        tokensToMarkets[realTokenRoot] = marketId;

        addWallet(realTokenRoot);
    }
```

40.6.2 Function createLiquidationPayload

- TODO

```
345     function createLiquidationPayload(address targetUser, uint32
            marketId) external override pure returns(TvmCell) {
346         TvmBuilder tb;
347         tb.store(OperationCodes.LIQUIDATE_TOKENS);
348         TvmBuilder op;
349         op.store(targetUser);
350         op.store(marketId);
351         tb.store(op.toCell());
352 
353         return tb.toCell();
354     }
```

40.6.3 Function createRepayPayload

- TODO

```
336     function createRepayPayload() external override pure returns(
            TvmCell) {
337         TvmBuilder tb;
338         tb.store(OperationCodes.REPAY_TOKENS);
339         TvmBuilder op;
340         tb.store(op.toCell());
341 
342         return tb.toCell();
343     }
```

40.6.4 Function createSupplyPayload

- TODO

```
327     function createSupplyPayload() external override pure returns(
            TvmCell) {
328         TvmBuilder tb;
329         tb.store(OperationCodes.SUPPLY_TOKENS);
330         TvmBuilder op;
331         tb.store(op.toCell());
332 
333         return tb.toCell();
334     }
```

40.6.5 Function getAllMarkets

- TODO

```
288     function getAllMarkets() external override view responsible
            returns(mapping(uint32 => MarketTokenAddresses)) {
289         return {flag: MsgFlag.REMAINING_GAS} marketTIP3Info;
290     }
```

40.6.6 Function getMarketAddresses

- TODO

```
284     function getMarketAddresses(uint32 marketId) external override
            view responsible returns(MarketTokenAddresses) {
285         return {flag: MsgFlag.REMAINING_GAS} marketTIP3Info[
                marketId];
286     }
```

40.6.7 Function getRealTokenRoots

- TODO

```
276     function getRealTokenRoots() external override view responsible
             returns(mapping(address => bool)) {
277         return {flag: MsgFlag.REMAINING_GAS} realTokenRoots;
278     }
```

40.6.8 Function getWallets

- TODO

```
280     function getWallets() external override view responsible
            returns(mapping(address => address)) {
281         return {flag: MsgFlag.REMAINING_GAS} wallets;
282     }
```

40.6.9 Function receiveTIP3WalletAddress

- TODO

```
197     function receiveTIP3WalletAddress(address _wallet) external
            onlyExisingTIP3Root(msg.sender) {
198         tvm.accept();
199
200         wallets[msg.sender] = _wallet;
201         uint32 marketId = tokensToMarkets[msg.sender];
202         marketTIP3Info[marketId].realTokenWallet = _wallet;
203         this.setReceiveCallback(_wallet);
204     }
```

40.6.10 Function removeMarket

- TODO

```
142     function removeMarket(uint32 marketId) external override
            canChangeParams {
143         tvm.accept();
144         MarketTokenAddresses marketTokenAddresses = marketTIP3Info[
                marketId];
145 
146         delete wallets[marketTokenAddresses.realToken];
147         delete realTokenRoots[marketTokenAddresses.realToken];
148         delete tokensToMarkets[marketTokenAddresses.realToken];
149         delete marketTIP3Info[marketId];
150     }
```

40.6.11 Function setMarketAddress

- TODO

```
116     function setMarketAddress(address _market) external override
            canChangeParams {
117         tvm.rawReserve(msg.value, 2);
118         marketAddress = _market;
119 
120         address(msg.sender).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
121     }
```

40.6.12 Function setReceiveCallback

- TODO

```
206     function setReceiveCallback(address _wallet) external {
207         require(msg.sender == address(this));
208         tvm.accept();
209 
210         ITONTokenWallet(_wallet).setReceiveCallback{
211             value: WCCostConstants.SET_RECEIVE_CALLBACK
212         }(
213             address(this),
214             true
215         );
216     }
```

40.6.13 Function tokensReceivedCallback

- TODO

```
218     function tokensReceivedCallback(
219         address token_wallet,
220         address token_root,
221         uint128 amount,
222         uint256, // sender_public_key,
223         address sender_address,
224         address sender_wallet,
225         address, // original_gas_to,
226         uint128, // updated_balance,
227         TvmCell payload
228     ) external override onlyOwnWallet(token_root, msg.sender)
229     {
230         tvm.rawReserve(msg.value, 2);
231             TvmSlice ts = payload.toSlice();
232         if (
233             ts.bits() == 8 &&
234             ts.refs() == 1
235         ) {
236             uint8 operation = ts.decode(uint8);
237             TvmSlice args = ts.loadRefAsSlice();
238             if (operation == OperationCodes.SUPPLY_TOKENS) {
239                 TvmBuilder tb;
240                 tb.store(sender_address);
241                 tb.store(uint256(amount));
242                 MarketAggregator(marketAddress).
                        performOperationWalletController{
243                     flag: MsgFlag.REMAINING_GAS
244                 }(operation, token_root, tb.toCell());
245             } else if (operation == OperationCodes.REPAY_TOKENS) {
246                 TvmBuilder tb;
247                 tb.store(sender_address);
248                 tb.store(sender_wallet);
249                 tb.store(uint256(amount));
250                 MarketAggregator(marketAddress).
                        performOperationWalletController{
251                     flag: MsgFlag.REMAINING_GAS
252                 }(operation, token_root, tb.toCell());
253             } else if (operation == OperationCodes.LIQUIDATE_TOKENS
                    ) {
254                 (address targetUser, uint32 marketToLiquidate) =
                        args.decode(address, uint32);
255                 TvmBuilder tb;
256                 TvmBuilder amountStorage;
257                 tb.store(sender_address);
258                 tb.store(targetUser);
259                 tb.store(sender_wallet);
260                 amountStorage.store(marketToLiquidate);
261                 amountStorage.store(uint256(amount));
262                 tb.store(amountStorage.toCell());
263                 MarketAggregator(marketAddress).
                        performOperationWalletController{
264                     flag: MsgFlag.REMAINING_GAS
265                 }(operation, token_root, tb.toCell());
```

```
266             } else {
267                 _transferTokensToWallet(sender_address, token_root,
                        sender_wallet, amount, payload);
268             }
269         } else {
270             _transferTokensToWallet(sender_address, token_root,
                    sender_wallet, amount, payload);
271         }
272     }
```

40.6.14 Function transferTokensToWallet

- TODO

```
152     function transferTokensToWallet(address tonWallet, address
            tokenRoot, address userTip3Wallet, uint256 toPayout)
            external override view onlyTrusted {
153         TvmCell empty;
154         _transferTokensToWallet(tonWallet, tokenRoot,
                userTip3Wallet, uint128(toPayout), empty);
155     }
```

40.6.15 Function upgradeContractCode

- TODO

```
64      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) override external canUpgrade {
65          tvm.accept();
66
67          tvm.setcode(code);
68          tvm.setCurrentCode(code);
69
70          onCodeUpgrade(
71              _owner,
72              marketAddress,
73              wallets,
74              realTokenRoots,
75              vTokenRoots,
76              marketTIP3Info,
77              updateParams,
78              codeVersion
79          );
80      }
```

40.7 Internal Method Definitions

40.7.1 Function _transferTokensToWallet

- TODO

```
157     function _transferTokensToWallet(address tonWallet, address
            tokenRoot, address userTip3Wallet, uint128 toTransfer,
            TvmCell payload) internal view {
158         ITONTokenWallet(wallets[tokenRoot]).transfer{
159             flag: MsgFlag.REMAINING_GAS
160         }(
161             userTip3Wallet,
162             toTransfer,
163             0,
164             tonWallet,
165             true,
166             payload
167         );
168     }
```

40.7.2 Function addWallet

- TODO

```
175     function addWallet(address tokenRoot) private pure {
176         IRootTokenContract(tokenRoot).deployEmptyWallet{
177             value: WCCostConstants.WALLET_DEPLOY_COST
178         }(
179             WCCostConstants.WALLET_DEPLOY_GRAMS,
180             0,
181             address(this),
182             address(this)
183         );
184 
185         IRootTokenContract(tokenRoot).getWalletAddress{
186             value: WCCostConstants.GET_WALLET_ADDRESS,
187             callback: this.receiveTIP3WalletAddress
188         }(
189             0,
190             address(this)
191         );
192     }
```

40.7.3 Function onCodeUpgrade

- TODO

```
93      function onCodeUpgrade(
94          address owner,
95          address _marketAddress,
96          mapping(address => address) _wallets,
97          mapping(address => bool) _realTokensRoots,
98          mapping(address => bool) _vTokenRoots,
99          mapping(uint32 => MarketTokenAddresses) _marketTIP3Info,
100         TvmCell,
```

```
101         uint32 _codeVersion
102     ) private {
103         tvm.accept();
104         tvm.resetStorage();
105         _owner = owner;
106         marketAddress = _marketAddress;
107         wallets = _wallets;
108         realTokenRoots = _realTokensRoots;
109         vTokenRoots = _vTokenRoots;
110         marketTIP3Info = _marketTIP3Info;
111         contractCodeVersion = _codeVersion;
112     }
```

Chapter 41

Contract WithdrawModule

Contents

41.1 Overview

In file `WithdrawModule.sol`

41.2 Contract Inheritance

IRoles	
IModule	
IContractStateCache	
IContractAddressSG	
IWithdrawModule	
IUpgradableContract	

41.3 Event Definitions

```
18      event TokenWithdraw(uint32 marketId, MarketDelta marketDelta,
            address tonWallet, uint256 vTokensWithdrawn, uint256
            realTokensWithdrawn);
```

41.4 Variable Definitions

address	marketAddress	
		used in @10.WithdrawModule.withdrawTokensFromMarket
		used in @10.WithdrawModule.upgradeContractCode
		assigned in @10.WithdrawModule.setMarketAddress
		used in @10.WithdrawModule.setMarketAddress
		assigned in @10.WithdrawModule.onCodeUpgrade
		used in @10.WithdrawModule.onCodeUpgrade
		used in @10.WithdrawModule.getContractAddresses
address	userAccountManager	
		used in @10.WithdrawModule.withdrawTokensFromMarket
		used in @10.WithdrawModule.withdrawTokensFromMarket
		used in @10.WithdrawModule.upgradeContractCode
		assigned in @10.WithdrawModule.setUserAccountManager
		used in @10.WithdrawModule.setUserAccountManager
		used in @10.WithdrawModule.resumeOperation
		used in @10.WithdrawModule.performAction
		assigned in @10.WithdrawModule.onCodeUpgrade
		used in @10.WithdrawModule.onCodeUpgrade
		used in @10.WithdrawModule.getContractAddresses
uint32	contractCodeVersion	
		assigned in @10.WithdrawModule.onCodeUpgrade
		used in @10.WithdrawModule.onCodeUpgrade
mapping (uint32 => MarketInfo)	marketInfo	
		used in @10.WithdrawModule.withdrawTokensFromMarket
		used in @10.WithdrawModule.withdrawTokensFromMarket
		used in @10.WithdrawModule.withdrawTokensFromMarket
		used in @10.WithdrawModule.upgradeContractCode
		assigned in @10.WithdrawModule.updateCache

```
    address marketAddress;

    address userAccountManager;

    uint32 public contractCodeVersion;

    mapping (uint32 => MarketInfo) marketInfo;

    mapping (address => fraction) tokenPrices;
```

41.5 Modifier Definitions

41.5.1 Modifier onlyMarket

```
    modifier onlyMarket() {
        require(msg.sender == marketAddress);
        tvm.rawReserve(msg.value, 2);
        _;
    }
```

41.5.2 Modifier onlyUserAccountManager

```
    modifier onlyUserAccountManager() {
        require(msg.sender == userAccountManager);
        _;
    }
```

41.6 Constructor Definitions

41.6.1 Constructor

Critical issue: Constructor for WithdrawModule (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address _newOwner) public {
        tvm.accept();
        _owner = _newOwner;
    }
```

41.7 Public Method Definitions

41.7.1 Function getContractAddresses

- TODO

```
79      function getContractAddresses() external override view
            responsible returns(address _owner, address _marketAddress,
             address _userAccountManager) {
80          return {flag: MsgFlag.REMAINING_GAS} (_owner, marketAddress
                , userAccountManager);
81      }
```

41.7.2 Function getModuleState

- TODO

```
63      function getModuleState() external override view returns(
            mapping(uint32 => MarketInfo), mapping(address => fraction)
            ) {
64          return(marketInfo, tokenPrices);
65      }
```

41.7.3 Function performAction

- TODO

```
89      function performAction(uint32 marketId, TvmCell args, mapping (
            uint32 => MarketInfo) _marketInfo, mapping (address =>
            fraction) _tokenPrices) external override onlyMarket {
90          TvmSlice ts = args.toSlice();
91          marketInfo = _marketInfo;
92          tokenPrices = _tokenPrices;
93          (address tonWallet, address userTip3Wallet, uint256
                tokensToWithdraw) = ts.decode(address, address, uint256
                );
94          mapping(uint32 => fraction) updatedIndexes =
                _createUpdatedIndexes();
95          IUAMUserAccount(userAccountManager).requestWithdrawInfo{
96              flag: MsgFlag.REMAINING_GAS
97          }(tonWallet, userTip3Wallet, tokensToWithdraw, marketId,
                updatedIndexes);
98      }
```

41.7.4 Function resumeOperation

- TODO

```
177     function resumeOperation(TvmCell args, mapping(uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
178         tvm.rawReserve(msg.value, 2);
179         marketInfo = _marketInfo;
180         tokenPrices = _tokenPrices;
181         TvmSlice ts = args.toSlice();
182         (uint32 marketId, address tonWallet, address userTip3Wallet
                ) = ts.decode(uint32, address, address);
183         TvmSlice values = ts.loadRefAsSlice();
184         (uint256 tokensToWithdraw, uint256 tokensToSend) = values.
                decode(uint256, uint256);
185         IUAMUserAccount(userAccountManager).writeWithdrawInfo{
186             flag: MsgFlag.REMAINING_GAS
187         }(tonWallet, userTip3Wallet, marketId, tokensToWithdraw,
                tokensToSend);
188     }
```

41.7.5 Function sendActionId

- TODO

```
59      function sendActionId() external override view responsible
            returns(uint8) {
60          return {flag: MsgFlag.REMAINING_GAS} OperationCodes.
                WITHDRAW_TOKENS;
61      }
```

41.7.6 Function setMarketAddress

- TODO

```
67      function setMarketAddress(address _marketAddress) external
            override canChangeParams {
68          tvm.rawReserve(msg.value, 2);
69          marketAddress = _marketAddress;
70          address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
71      }
```

41.7.7 Function setUserAccountManager

- TODO

```
73      function setUserAccountManager(address _userAccountManager)
            external override canChangeParams {
74          tvm.rawReserve(msg.value, 2);
75          userAccountManager = _userAccountManager;
76          address(_owner).transfer({value: 0, flag: MsgFlag.
                REMAINING_GAS});
77      }
```

41.7.8 Function updateCache

- TODO

```
83      function updateCache(address tonWallet, mapping (uint32 =>
            MarketInfo) _marketInfo, mapping (address => fraction)
            _tokenPrices) external override onlyMarket {
84          marketInfo = _marketInfo;
85          tokenPrices = _tokenPrices;
86          tonWallet.transfer({value: 0, flag: MsgFlag.REMAINING_GAS})
                ;
87      }
```

41.7.9 Function upgradeContractCode

- TODO

```
25      function upgradeContractCode(TvmCell code, TvmCell updateParams
            , uint32 codeVersion) external override canUpgrade {
26          tvm.rawReserve(msg.value, 2);
27
28          tvm.setcode(code);
29          tvm.setCurrentCode(code);
30
31          onCodeUpgrade (
32              _owner,
33              marketAddress,
34              userAccountManager,
35              marketInfo,
36              tokenPrices,
37              codeVersion
38          );
39      }
```

41.7.10 Function withdrawTokensFromMarket

- TODO

```
106     function withdrawTokensFromMarket(
107         address tonWallet,
108         address userTip3Wallet,
109         uint256 tokensToWithdraw,
```

```
110         uint32 marketId,
111         mapping(uint32 => uint256) supplyInfo,
112         mapping(uint32 => BorrowInfo) borrowInfo
113     ) external override onlyUserAccountManager {
114         tvm.rawReserve(msg.value, 2);
115         MarketDelta marketDelta;
116         mapping(uint32 => MarketDelta) marketsDelta;
117 
118         MarketInfo mi = marketInfo[marketId];
119 
120         // For token withdraw:
121         // 1. Calculate account health
122         // 2. Calculate USD amount for withdraw token
123         // 3. Check if user can afford to withdraw required amount
                of real tokens
124 
125         fraction accountHealth = Utilities.calculateSupplyBorrow(
                supplyInfo, borrowInfo, marketInfo, tokenPrices);
126 
127         fraction fTokensToSend = tokensToWithdraw.numFMul(mi.
                exchangeRate);
128         fraction fTokensToSendUSD = fTokensToSend.fDiv(tokenPrices[
                marketInfo[marketId].token]);
129 
130         // Check user balance in tokens just in case
131         // There will be lock at user account for operation,
                unified for all operations
132         // As all operations are finished with account health check
                , account will unlock after
133         // Updating indexes
134         if (
135             (accountHealth.nom > accountHealth.denom) &&
136             (supplyInfo[marketId] >= tokensToWithdraw)
137         ) {
138             if (
139                 accountHealth.nom - accountHealth.denom >=
                        fTokensToSendUSD.toNum() &&
140                 fTokensToSend.toNum() <= mi.realTokenBalance - mi.
                        totalReserve
141             ) {
142                 uint256 tokensToSend = fTokensToSend.toNum();
143 
144                 marketDelta.realTokenBalance.delta = tokensToSend;
145                 marketDelta.realTokenBalance.positive = false;
146                 marketDelta.vTokenBalance.delta = tokensToWithdraw;
147                 marketDelta.vTokenBalance.positive = false;
148 
149                 marketsDelta[marketId] = marketDelta;
150 
151                 emit TokenWithdraw(marketId, marketDelta, tonWallet
                        , tokensToWithdraw, tokensToSend);
152 
153                 TvmBuilder tb;
154                 tb.store(marketId);
155                 tb.store(tonWallet);
156                 tb.store(userTip3Wallet);
157                 TvmBuilder valueStorate;
```

```
158                 valueStorate.store(tokensToWithdraw);
159                 valueStorate.store(tokensToSend);
160                 tb.store(valueStorate.toCell());
161
162                 IContractStateCacheRoot(marketAddress).
                        receiveCacheDelta{
163                     flag: MsgFlag.REMAINING_GAS
164                 }(marketsDelta, tb.toCell());
165             } else {
166                 IUAMUserAccount(userAccountManager).
                        requestUserAccountHealthCalculation{
167                     flag: MsgFlag.REMAINING_GAS
168                 }(tonWallet);
169             }
170         } else {
171             IUAMUserAccount(userAccountManager).
                    requestUserAccountHealthCalculation{
172                 flag: MsgFlag.REMAINING_GAS
173             }(tonWallet);
174         }
175     }
```

41.8 Internal Method Definitions

41.8.1 Function _createUpdatedIndexes

- TODO

```
100     function _createUpdatedIndexes() internal view returns(mapping(
            uint32 => fraction) updatedIndexes) {
101         for ((uint32 marketId, MarketInfo mi): marketInfo) {
102             updatedIndexes[marketId] = mi.index;
103         }
104     }
```

41.8.2 Function onCodeUpgrade

- TODO

```
41      function onCodeUpgrade(
42          address owner,
43          address _marketAddress,
44          address _userAccountManager,
45          mapping(uint32 => MarketInfo) _marketInfo,
46          mapping(address => fraction) _tokenPrices,
47          uint32 _codeVersion
48      ) private {
49          tvm.accept();
50          tvm.resetStorage();
51          _owner = owner;
52          marketAddress = _marketAddress;
```

```
53          userAccountManager = _userAccountManager;
54          marketInfo = _marketInfo;
55          tokenPrices = _tokenPrices;
56          contractCodeVersion = _codeVersion;
57      }
```

41.8.2.0.1 Some functions inherited by using