

Audit

By OCamlPro

August 2, 2021

Contents

1	Contract DEXClient	8
1.1	Contract Inheritance	8
1.2	Type Definitions	8
1.2.1	Struct Connector	8
1.2.2	Struct Callback	9
1.2.3	Struct Pair	9
1.3	Constant Definitions	10
1.4	Static Variable Definitions	10
1.5	Variable Definitions	11
1.6	Modifier Definitions	13
1.6.1	Modifier alwaysAccept	13
1.6.2	Modifier checkOwnerAndAccept	13
1.7	Constructor Definitions	13
1.7.1	Constructor	13
1.8	Public Method Definitions	14
1.8.1	Function connectCallback	14
1.8.2	Function connectPair	15
1.8.3	Function connectRoot	15
1.8.4	Function createNewPair	16
1.8.5	Function getAllDataPreparation	17
1.8.6	Function getBalance	17
1.8.7	Function getCallback	18
1.8.8	Function getConnectorAddress	19
1.8.9	Function getPairData	19
1.8.10	Function processLiquidity	20
1.8.11	Function processSwapA	21
1.8.12	Function processSwapB	21
1.8.13	Function returnLiquidity	22
1.8.14	Function sendTokens	23
1.8.15	Function setPair	23
1.8.16	Function tokensReceivedCallback	24
1.9	Internal Method Definitions	25
1.9.1	Receive function	25
1.9.2	Function computeConnectorAddress	25

1.9.3	Function getFirstCallback	26
1.9.4	Function getQuotient	26
1.9.5	Function getRemainder	26
1.9.6	Function isReady	27
1.9.7	Function isReadyToProvide	27
1.9.8	Function thisBalance	28
2	Contract DEXConnector	29
2.1	Contract Inheritance	29
2.2	Constant Definitions	29
2.3	Static Variable Definitions	29
2.4	Variable Definitions	30
2.5	Modifier Definitions	32
2.5.1	Modifier alwaysAccept	32
2.5.2	Modifier checkOwnerAndAccept	32
2.6	Constructor Definitions	32
2.6.1	Constructor	32
2.7	Public Method Definitions	33
2.7.1	Function burn	33
2.7.2	Function deployEmptyWallet	33
2.7.3	Function expectedWalletAddressCallback	34
2.7.4	Function getBalance	34
2.7.5	Function setBouncedCallback	35
2.7.6	Function setTransferCallback	35
2.7.7	Function transfer	35
2.8	Internal Method Definitions	36
2.8.1	Receive function	36
2.8.2	Function getQuotient	36
2.8.3	Function getRemainder	36
3	Contract DEXPair	38
3.1	Contract Inheritance	38
3.2	Type Definitions	38
3.2.1	Struct Connector	38
3.2.2	Struct Callback	39
3.3	Constant Definitions	39
3.4	Static Variable Definitions	40
3.5	Variable Definitions	42
3.6	Modifier Definitions	44
3.6.1	Modifier alwaysAccept	44
3.6.2	Modifier checkOwnerAndAccept	44
3.6.3	Modifier checkPubKeyAndAccept	44
3.7	Constructor Definitions	45
3.7.1	Constructor	45
3.8	Public Method Definitions	46
3.8.1	Function burnCallback	46

3.8.2	Function connect	47
3.8.3	Function connectCallback	47
3.8.4	Function getBalance	48
3.8.5	Function getCallback	49
3.8.6	Function getConnectorAddress	50
3.8.7	Function tokensReceivedCallback	50
3.9	Internal Method Definitions	56
3.9.1	Receive function	56
3.9.2	Function acceptForProvide	57
3.9.3	Function cleanProcessing	57
3.9.4	Function computeConnectorAddress	58
3.9.5	Function connectRoot	58
3.9.6	Function getAmountOut	59
3.9.7	Function getFirstCallback	59
3.9.8	Function getQuotient	60
3.9.9	Function getRemainder	60
3.9.10	Function liquidityA	60
3.9.11	Function liquidityB	61
3.9.12	Function qtyAforB	61
3.9.13	Function qtyBforA	62
3.9.14	Function thisBalance	62
4	Contract DEXroot	63
4.1	Contract Inheritance	63
4.2	Type Definitions	63
4.2.1	Struct Pair	63
4.3	Constant Definitions	64
4.4	Static Variable Definitions	64
4.5	Variable Definitions	64
4.6	Modifier Definitions	66
4.6.1	Modifier alwaysAccept	66
4.6.2	Modifier checkOwnerAndAccept	66
4.6.3	Modifier checkCreatorAndAccept	66
4.7	Constructor Definitions	67
4.7.1	Constructor	67
4.8	Public Method Definitions	67
4.8.1	Function checkPubKey	67
4.8.2	Function createDEXclient	68
4.8.3	Function createDEXpair	69
4.8.4	Function getBalanceTONgrams	70
4.8.5	Function getClientAddress	71
4.8.6	Function getConnectorAddress	71
4.8.7	Function getPairAddress	72
4.8.8	Function getPairByRoots01	72
4.8.9	Function getPairByRoots10	73
4.8.10	Function getRootTokenAddress	73

4.8.11	Function getRootsByPair	74
4.8.12	Function sendTransfer	74
4.8.13	Function setCreator	74
4.8.14	Function setDEXclientCode	75
4.8.15	Function setDEXconnectorCode	75
4.8.16	Function setDEXpairCode	75
4.8.17	Function setRootTokenCode	76
4.8.18	Function setTONTokenWalletCode	76
4.9	Internal Method Definitions	76
4.9.1	Receive function	76
4.9.2	Function computeClientAddress	77
4.9.3	Function computeConnectorAddress	77
4.9.4	Function computePairAddress	78
4.9.5	Function computeRootTokenAddress	79
5	Contract RootTokenContract	80
5.1	Contract Inheritance	80
5.2	Static Variable Definitions	80
5.3	Variable Definitions	81
5.4	Modifier Definitions	83
5.4.1	Modifier onlyOwner	83
5.4.2	Modifier onlyInternalOwner	83
5.5	Constructor Definitions	83
5.5.1	Constructor	83
5.6	Internal Method Definitions	84
5.6.1	Fallback function	84
5.6.2	OnBounce function	84
5.6.3	Function deployEmptyWallet	85
5.6.4	Function deployWallet	86
5.6.5	Function getDetails	87
5.6.6	Function getExpectedWalletAddress	88
5.6.7	Function getTotalSupply	89
5.6.8	Function getVersion	89
5.6.9	Function getWalletAddress	89
5.6.10	Function getWalletCode	90
5.6.11	Function isExternalOwner	90
5.6.12	Function isInternalOwner	90
5.6.13	Function isOwner	91
5.6.14	Function mint	91
5.6.15	Function proxyBurn	92
5.6.16	Function sendExpectedWalletAddress	92
5.6.17	Function sendPausedCallbackTo	93
5.6.18	Function sendSurplusGas	93
5.6.19	Function setPaused	94
5.6.20	Function tokensBurned	94
5.6.21	Function transferOwner	95

6	Contract TONTOKENWallet	97
6.1	Contract Inheritance	97
6.2	Static Variable Definitions	97
6.3	Variable Definitions	99
6.4	Modifier Definitions	101
6.4.1	Modifier onlyRoot	101
6.4.2	Modifier onlyOwner	101
6.4.3	Modifier onlyInternalOwner	101
6.5	Constructor Definitions	102
6.5.1	Constructor	102
6.6	Public Method Definitions	102
6.6.1	Function destroy	102
6.7	Internal Method Definitions	103
6.7.1	Fallback function	103
6.7.2	OnBounce function	103
6.7.3	Function accept	104
6.7.4	Function allowance	104
6.7.5	Function approve	105
6.7.6	Function balance	106
6.7.7	Function burnByOwner	106
6.7.8	Function burnByRoot	107
6.7.9	Function disapprove	108
6.7.10	Function getDetails	108
6.7.11	Function getExpectedAddress	109
6.7.12	Function getVersion	109
6.7.13	Function getWalletCode	110
6.7.14	Function internalTransfer	110
6.7.15	Function internalTransferFrom	111
6.7.16	Function setBouncedCallback	113
6.7.17	Function setReceiveCallback	113
6.7.18	Function transfer	114
6.7.19	Function transferFrom	115
6.7.20	Function transferToRecipient	117

Table of Issues

Critical issue: Constructor for DEXClient	13
Critical issue: Constructor for DEXConnector	32
Critical issue: Constructor for DEXPair	45
Critical issue: Constructor for DEXroot	67
Critical issue: Constructor for RootTokenContract	83
Critical issue: Constructor for TONTokenWallet	102

To edit this document

In the `report.tex` file, choose:

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

Issues can be entered with:

- `\issueCritical{title}{text}`
- `\issueMedium{title}{text}`
- `\issueLow{title}{text}`

Chapter 1

Contract DEXClient

In file DEXClient.sol

1.1 Contract Inheritance

ITokensReceivedCallback	
IDEXClient	
IDEXConnect	

1.2 Type Definitions

Advises

Check that types have the correct integer types (Pubkey : uint256, Amount: uint128, Time: uint64).

1.2.1 Struct Connector

root_address	address	
souint	uint256	
status	bool	

```
29 struct Connector {
30     address root_address;
31     uint256 souint;
32     bool status;
33 }
```

1.2.2 Struct Callback

token_wallet	address	
token_root	address	
amount	uint128	
sender_public_key	uint256	
sender_address	address	
sender_wallet	address	
original_gas_to	address	
updated_balance	uint128	
payload_arg0	uint8	
payload_arg1	address	
payload_arg2	address	

```
43 struct Callback {
44     address token_wallet;
45     address token_root;
46     uint128 amount;
47     uint256 sender_public_key;
48     address sender_address;
49     address sender_wallet;
50     address original_gas_to;
51     uint128 updated_balance;
52     uint8 payload_arg0;
53     address payload_arg1;
54     address payload_arg2;
55 }
```

1.2.3 Struct Pair

status	bool	
rootA	address	
walletA	address	
rootB	address	
walletB	address	
rootAB	address	

```
60 struct Pair {
61     bool status;
62     address rootA;
63     address walletA;
64     address rootB;
65     address walletB;
66     address rootAB;
67 }
```

1.3 Constant Definitions

Advises

Use a naming convention to distinguish constants from other, such as all uppercase names.

Use `ton` unit instead of nanotons for cost constants to avoid numbers with too many zeroes.

uint128	GRAMS_CONNECT_PAIR	Initialized to 500000000
uint128	GRAMS_SET_CALLBACK_ADDR	Initialized to 100000000
uint128	GRAMS_SWAP	Initialized to 500000000
uint128	GRAMS_PROCESS_LIQUIDITY	Initialized to 500000000
uint128	GRAMS_RETURN_LIQUIDITY	Initialized to 500000000

```
23  uint128 constant GRAMS_CONNECT_PAIR = 500000000;
```

```
24  uint128 constant GRAMS_SET_CALLBACK_ADDR = 100000000;
```

```
25  uint128 constant GRAMS_SWAP = 500000000;
```

```
26  uint128 constant GRAMS_PROCESS_LIQUIDITY = 500000000;
```

```
27  uint128 constant GRAMS_RETURN_LIQUIDITY = 500000000;
```

1.4 Static Variable Definitions

Advises

Use a naming convention to distinguish static variables from global variables, such as `s_` prefix.

address	rootDEX	
		used in @1.DEX-Client.createNewPair
		used in @1.DEX-Client.createNewPair
		used in @1.DEX-Client.:constructor
uint256	soUINT	
TvmCell	codeDEXConnector	
		used in @1.DEX-Client.connectRoot
		used in @1.DEX-Client.computeConnectorAddress

```
18  address static public rootDEX;
```

```
19  uint256 static public soUINT;
```

```
20  TvmCell static public codeDEXConnector;
```

1.5 Variable Definitions

Advises

Use a naming convention to distinguish global variables from local variables, such as `g_` or `m_` prefix.

address []	rootKeys	
		used in @1.DEX-Client.getAllDataPreparation
		used in @1.DEX-Client.connectCallback
mapping (address => address)	rootWallet	
		used in @1.DEX-Client.sendTokens
		used in @1.DEX-Client.returnLiquidity
		used in @1.DEX-Client.returnLiquidity
		used in @1.DEX-Client.processSwapB
		used in @1.DEX-Client.processSwapA
		used in @1.DEX-Client.processLiquidity
		used in @1.DEX-Client.processLiquidity
		used in @1.DEX-Client.isReadyToProvide
		used in @1.DEX-Client.isReadyToProvide
		used in @1.DEX-Client.isReadyToProvide
		used in @1.DEXClient.isReady
		used in @1.DEXClient.isReady
		used in @1.DEX-Client.connectRoot
		assigned in @1.DEX-Client.connectCallback
		used in @1.DEX-Client.connectCallback
mapping (address => address)	rootConnector	
		used in @1.DEX-Client.sendTokens
		used in @1.DEX-Client.sendTokens
		used in @1.DEX-Client.returnLiquidity
		used in @1.DEX-Client.returnLiquidity
		used in @1.DEX-Client.processSwapB
		used in @1.DEX-Client.processSwapA
	12	used in @1.DEX-Client.processLiquidity
		used in @1.DEX-Client.processLiquidity
		used in @1.DEX-Client.isReadyToProvide
		used in @1.DEX-Client.isReadyToProvide

```

35     address[] public rootKeys;
36     mapping (address => address) public rootWallet;
37     mapping (address => address) public rootConnector;
38     mapping (address => Connector) connectors;
39
40     uint public counterCallback;
41
42     mapping (uint => Callback) callbacks;
43
44     mapping(address => Pair) public pairs;
45
46     address[] public pairKeys;

```

1.6 Modifier Definitions

Advises

Calling `tvm.accept()` without checking `pubkey` should not be allowed

1.6.1 Modifier alwaysAccept

```
73     modifier alwaysAccept {
74         tvml.accept();
75     };
76 }
```

1.6.2 Modifier checkOwnerAndAccept

```

79     modifier checkOwnerAndAccept {
80         require(msg.pubkey() == tvn.pubkey(), 102);
81         tvn.accept();
82         _;
83     }

```

1.7 Constructor Definitions

1.7.1 Constructor

Critical issue: Constructor for DEXClient

lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem
ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum
lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum
lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem
ipsum lorem ipsum lorem ipsum

Advises
Check who can call the constructor. If the constructor sets global values, only legitimate users should be allowed.
Check that every argument is protected by a require().
If external users are allowed, their pubkey should be verified (require(msg.pubkey() != 0 && msg.pubkey() == tvn.pubkey(),100) , and tvn.accept() should be called.

```

85 constructor() public {
86     require(msg.sender == rootDEX, 103);
87     tvn.accept();
88     counterCallback = 0;
89 }

```

1.8 Public Method Definitions

1.8.1 Function connectCallback

Parameters		
address	wallet	
Modifiers		
alwaysAccept	<i>no args</i>	

```

181 function connectCallback(address wallet) public override
182     alwaysAccept {
183     address connector = msg.sender;
184     if (connectors.exists(connector)) {
185         Connector cc = connectors[connector];
186         rootKeys.push(cc.root_address);
187         rootWallet[cc.root_address] = wallet;
188         rootConnector[cc.root_address] = connector;
189         TvmCell bodySTC = tvn.encodeBody(IDEXConnector(connector).
190             setTransferCallback);
191         connector.transfer({value: GRAMS_SET_CALLBACK_ADDR, bounce:
192             true, flag: 0, body:bodySTC});
193         TvmCell bodySBC = tvn.encodeBody(IDEXConnector(connector).
194             setBouncedCallback);
195         connector.transfer({value: GRAMS_SET_CALLBACK_ADDR, bounce:
196             true, flag: 0, body:bodySBC});
197         cc.status = true;

```

```

193     connectors[connector] = cc;
194 }
195 }

```

1.8.2 Function connectPair

Parameters		
address	pairAddr	
Returns		
bool	statusConnection	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

92 function connectPair(address pairAddr) public checkOwnerAndAccept
93     returns (bool statusConnection) {
94     statusConnection = false;
95     if (!pairs.exists(pairAddr)){
96         Pair cp = pairs[pairAddr];
97         cp.status = false;
98         pairs[pairAddr] = cp;
99         pairKeys.push(pairAddr);
100         TvmCell body = tvm.encodeBody(IDEXPair(pairAddr).connect);
101         pairAddr.transfer({value:GRAMS_CONNECT_PAIR, body:body});
102         statusConnection = true;
103     }
104 }

```

1.8.3 Function connectRoot

Parameters		
address	root	
uint256	souint	
uint128	gramsToConnector	
uint128	gramsToRoot	
Returns		
bool	statusConnected	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	


```

158 function connectRoot(address root, uint256 souint, uint128
    gramsToConnector, uint128 gramsToRoot) public
    checkOwnerAndAccept returns (bool statusConnected){
159     statusConnected = false;
160     if (!rootWallet.exists(root)) {
161         TvmCell stateInit = tvm.buildStateInit({
162             contr: DEXConnector,
163             varInit: { soUINT: souint, dexclient: address(this) },
164             code: codeDEXConnector,
165             pubkey: tvm.pubkey()
166         });
167         TvmCell init = tvm.encodeBody(DEXConnector);
168         address connector = tvm.deploy(stateInit, init,
            gramsToConnector, address(this).wid);
169         Connector cr = connectors[connector];
170         cr.root_address = root;
171         cr.souint = souint;
172         cr.status = false;
173         connectors[connector] = cr;
174         TvmCell body = tvm.encodeBody(IDEXConnector(connector).
            deployEmptyWallet, root);
175         connector.transfer({value:gramsToRoot, bounce:true, body:body
            });
176         statusConnected = true;
177     }
178 }

```

1.8.4 Function createNewPair

Parameters		
address	root0	
address	root1	
uint256	pairSoArg	
uint256	connectorSoArg0	
uint256	connectorSoArg1	
uint256	rootSoArg	
bytes	rootName	
bytes	rootSymbol	
uint8	rootDecimals	
uint128	grammsForPair	
uint128	grammsForRoot	
uint128	grammsForConnector	
uint128	grammsForWallet	
uint128	grammsTotal	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

356 function createNewPair(
357     address root0,
358     address root1,
359     uint256 pairSoArg,
360     uint256 connectorSoArg0,
361     uint256 connectorSoArg1,
362     uint256 rootSoArg,
363     bytes rootName,
364     bytes rootSymbol,
365     uint8 rootDecimals,
366     uint128 gramsForPair,
367     uint128 gramsForRoot,
368     uint128 gramsForConnector,
369     uint128 gramsForWallet,
370     uint128 gramsTotal
371 ) public view checkOwnerAndAccept {
372     require (!(gramsTotal < (gramsForPair+2*gramsForConnector+2*
        gramsForWallet+gramsForRoot)) && !(gramsTotal < 5 ton)
        ,106);
373     require (!(address(this).balance < gramsTotal),105);
374     TvmCell body = tvml.encodeBody(IDEXRoot(rootDEX).createDEXpair,
        root0,root1,pairSoArg,connectorSoArg0,connectorSoArg1,
        rootSoArg,rootName,rootSymbol,rootDecimals,gramsForPair,
        gramsForRoot,gramsForConnector,gramsForWallet);
375     rootDEX.transfer({value:gramsTotal, bounce:false, flag: 1,
        body:body});
376 }

```

1.8.5 Function getAllDataPreparation

Returns		
address []	pairKeysR	
address []	rootKeysR	
Modifiers		
alwaysAccept	no args	

```

215 function getAllDataPreparation() public view alwaysAccept returns
    (address[] pairKeysR, address[] rootKeysR){
216     pairKeysR = pairKeys;
217     rootKeysR = rootKeys;
218 }

```

1.8.6 Function getBalance

Returns		
uint128	no name	

```

351     function getBalance() public pure responsible returns (uint128) {
352         return { value: 0, bounce: false, flag: 64 } thisBalance();
353     }

```

1.8.7 Function getCallback

Parameters		
uint256	id	
Returns		
address	token_wallet	
address	token_root	
uint128	amount	
uint256	sender_public_key	
address	sender_address	
address	sender_wallet	
address	original_gas_to	
uint128	updated_balance	
uint8	payload_arg0	
address	payload_arg1	
address	payload_arg2	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

318     function getCallback(uint id) public view checkOwnerAndAccept
319         returns (
320             address token_wallet,
321             address token_root,
322             uint128 amount,
323             uint256 sender_public_key,
324             address sender_address,
325             address sender_wallet,
326             address original_gas_to,
327             uint128 updated_balance,
328             uint8 payload_arg0,
329             address payload_arg1,
330             address payload_arg2
331         ){
332         Callback cc = callbacks[id];
333         token_wallet = cc.token_wallet;
334         token_root = cc.token_root;
335         amount = cc.amount;
336         sender_public_key = cc.sender_public_key;
337         sender_address = cc.sender_address;
338         sender_wallet = cc.sender_wallet;
339         original_gas_to = cc.original_gas_to;

```

```

339     updated_balance = cc.updated_balance;
340     payload_arg0 = cc.payload_arg0;
341     payload_arg1 = cc.payload_arg1;
342     payload_arg2 = cc.payload_arg2;
343 }

```

1.8.8 Function getConnectorAddress

Parameters		
uint256	connectorSoArg	
Returns		
address	<i>no name</i>	

```

153 function getConnectorAddress(uint256 connectorSoArg) public view
    responsible returns (address) {
154     return { value: 0, bounce: false, flag: 64 }
        computeConnectorAddress( connectorSoArg);
155 }

```

1.8.9 Function getPairData

Parameters		
address	pairAddr	
Returns		
bool	pairStatus	
address	pairRootA	
address	pairWalletA	
address	pairRootB	
address	pairWalletB	
address	pairRootAB	
address	curPair	
Modifiers		
alwaysAccept	<i>no args</i>	

```

379 function getPairData(address pairAddr) public view alwaysAccept
    returns (
380     bool pairStatus,
381     address pairRootA,

```

```

382     address pairWalletA,
383     address pairRootB,
384     address pairWalletB,
385     address pairRootAB,
386     address curPair
387 ){
388     Pair cp = pairs[pairAddr];
389     pairStatus = cp.status;
390     pairRootA = cp.rootA;
391     pairWalletA = cp.walletA;
392     pairRootB = cp.rootB;
393     pairWalletB = cp.walletB;
394     pairRootAB = cp.rootAB;
395     curPair = pairAddr;
396 }

```

1.8.10 Function processLiquidity

Parameters		
address	pairAddr	
uint128	qtyA	
uint128	qtyB	
Returns		
bool	processLiquidityStatus	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

251 function processLiquidity(address pairAddr, uint128 qtyA, uint128
    qtyB) public view checkOwnerAndAccept returns (bool
    processLiquidityStatus) {
252     processLiquidityStatus = false;
253     if (isReadyToProvide(pairAddr)) {
254         Pair cp = pairs[pairAddr];
255         address connectorA = rootConnector[cp.rootA];
256         address connectorB = rootConnector[cp.rootB];
257         TvmBuilder builderA;
258         builderA.store(uint8(2), address(this), rootWallet[cp.rootAB
            ]);
259         TvmCell payloadA = builderA.toCell();
260         TvmBuilder builderB;
261         builderB.store(uint8(2), address(this), rootWallet[cp.rootAB
            ]);
262         TvmCell payloadB = builderB.toCell();
263         TvmCell bodyA = tvm.encodeBody(IDEXConnector(connectorA).
            transfer, cp.walletA, qtyA, payloadA);
264         TvmCell bodyB = tvm.encodeBody(IDEXConnector(connectorB).
            transfer, cp.walletB, qtyB, payloadB);

```

```

265     connectorA.transfer({value: GRAMS_PROCESS_LIQUIDITY, bounce:
266         true, body:bodyA});
266     connectorB.transfer({value: GRAMS_PROCESS_LIQUIDITY, bounce:
267         true, body:bodyB});
267     processLiquidityStatus = true;
268 }
269 }

```

1.8.11 Function processSwapA

Parameters		
address	pairAddr	
uint128	qtyA	
Returns		
bool	processSwapStatus	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

221 function processSwapA(address pairAddr, uint128 qtyA) public view
222     checkOwnerAndAccept returns (bool processSwapStatus) {
223     processSwapStatus = false;
223     if (isReady(pairAddr)) {
224         Pair cp = pairs[pairAddr];
224         address connector = rootConnector[cp.rootA];
225         TvmBuilder builder;
226         builder.store(uint8(1), cp.rootB, rootWallet[cp.rootB]);
227         TvmCell payload = builder.toCell();
228         TvmCell body = tvm.encodeBody(IDEXConnector(connector).
229             transfer, cp.walletA, qtyA, payload);
230         connector.transfer({value: GRAMS_SWAP, bounce:true, body:body
231             });
231         processSwapStatus = true;
232     }
233 }

```

1.8.12 Function processSwapB

Parameters		
address	pairAddr	
uint128	qtyB	
Returns		
bool	processSwapStatus	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

236 function processSwapB(address pairAddr, uint128 qtyB) public view
    checkOwnerAndAccept returns (bool processSwapStatus) {
237     processSwapStatus = false;
238     if (isReady(pairAddr)) {
239         Pair cp = pairs[pairAddr];
240         address connector = rootConnector[cp.rootB];
241         TvmBuilder builder;
242         builder.store(uint8(1), cp.rootA, rootWallet[cp.rootA]);
243         TvmCell payload = builder.toCell();
244         TvmCell body = tvn.encodeBody(IDEXConnector(connector).
            transfer, cp.walletB, qtyB, payload);
245         connector.transfer({value: GRAMS_SWAP, bounce:true, body:body
            });
246         processSwapStatus = true;
247     }
248 }

```

1.8.13 Function returnLiquidity

Parameters		
address	pairAddr	
uint128	tokens	
Returns		
bool	returnLiquidityStatus	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

272 function returnLiquidity(address pairAddr, uint128 tokens) public
    view checkOwnerAndAccept returns (bool returnLiquidityStatus
    ) {
273     returnLiquidityStatus = false;
274     if (isReadyToProvide(pairAddr)) {
275         Pair cp = pairs[pairAddr];
276         TvmBuilder builder;
277         builder.store(uint8(3), rootWallet[cp.rootA], rootWallet[cp.
            rootB]);
278         TvmCell callback_payload = builder.toCell();
279         TvmCell body = tvn.encodeBody(IDEXConnector(rootConnector[cp.
            rootAB]).burn, tokens, pairAddr, callback_payload);
280         rootConnector[cp.rootAB].transfer({value:GRAMS_RETURN_LIQUIDITY
            , body:body});
281         returnLiquidityStatus = true;
282     }
283 }

```

1.8.14 Function sendTokens

Parameters		
address	tokenRoot	
address	to	
uint128	tokens	
uint128	grams	
Returns		
bool	sendTokenStatus	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

399     function sendTokens(address tokenRoot, address to, uint128 tokens
        , uint128 grams) public checkOwnerAndAccept view returns (
        bool sendTokenStatus){
400     sendTokenStatus = false;
401     if (rootConnector[tokenRoot] != address(0)) {
402         address connector = rootConnector[tokenRoot];
403         TvmBuilder builder;
404         builder.store(uint8(4), address(this), rootWallet[tokenRoot])
            ;
405         TvmCell payload = builder.toCell();
406         TvmCell body = tvm.encodeBody(IDEXConnector(connector).
            transfer, to, tokens, payload);
407         connector.transfer({value: grams, bounce:true, body:body});
408         sendTokenStatus = true;
409     }
410 }

```

1.8.15 Function setPair

Parameters		
address	arg0	
address	arg1	
address	arg2	
address	arg3	
address	arg4	
Modifiers		
alwaysAccept	<i>no args</i>	


```

127 function setPair(address arg0, address arg1, address arg2,
128               address arg3, address arg4) public alwaysAccept override {
129     address dexpair = msg.sender;
130     if (pairs.exists(dexpair)){
131         Pair cp = pairs[dexpair];
132         cp.status = true;
133         cp.rootA = arg0;
134         cp.walletA = arg1;
135         cp.rootB = arg2;
136         cp.walletB = arg3;
137         cp.rootAB = arg4;
138         pairs[dexpair] = cp;
139     }
140 }

```

1.8.16 Function tokensReceivedCallback

Parameters		
address	token_wallet	
address	token_root	
uint128	amount	
uint256	sender_public_key	
address	sender_address	
address	sender_wallet	
address	original_gas_to	
uint128	updated_balance	
TvmCell	payload	
Modifiers		
alwaysAccept	<i>no args</i>	

```

286 function tokensReceivedCallback(
287     address token_wallet,
288     address token_root,
289     uint128 amount,
290     uint256 sender_public_key,
291     address sender_address,
292     address sender_wallet,
293     address original_gas_to,
294     uint128 updated_balance,
295     TvmCell payload
296 ) public override alwaysAccept {
297     Callback cc = callbacks[counterCallback];
298     cc.token_wallet = token_wallet;
299     cc.token_root = token_root;
300     cc.amount = amount;
301     cc.sender_public_key = sender_public_key;
302     cc.sender_address = sender_address;

```

```

303     cc.sender_wallet = sender_wallet;
304     cc.original_gas_to = original_gas_to;
305     cc.updated_balance = updated_balance;
306     TvmSlice slice = payload.toSlice();
307     (uint8 arg0, address arg1, address arg2) = slice.decode(uint8,
308         address, address);
309     cc.payload_arg0 = arg0;
310     cc.payload_arg1 = arg1;
311     cc.payload_arg2 = arg2;
312     callbacks[counterCallback] = cc;
313     counterCallback++;
314     if (counterCallback > 10){delete callbacks[getFirstCallback()
    ];}

```

1.9 Internal Method Definitions

1.9.1 Receive function

```

413 receive() external {
414 }

```

1.9.2 Function computeConnectorAddress

Parameters		
uint256	souint	
Returns		
address	<i>no name</i>	

```

142 function computeConnectorAddress(uint256 souint) private inline
143     view returns (address) {
144     TvmCell stateInit = tvm.buildStateInit({
145         contr: DEXConnector,
146         varInit: { soUINT: souint, dexclient: address(this) },
147         code: codeDEXConnector,
148         pubkey: tvm.pubkey()
149     });
149     return address(tvm.hash(stateInit));
150 }

```

1.9.3 Function getFirstCallback

Returns		
uint256	<i>no name</i>	

```
121 function getFirstCallback() private view returns (uint) {
122     optional(uint, Callback) rc = callbacks.min();
123     if (rc.hasValue()) {(uint number, ) = rc.get();return number;}
124     else {return 0;}
}
```

1.9.4 Function getQuotient

Parameters		
uint128	arg0	
uint128	arg1	
uint128	arg2	
Returns		
uint128	<i>no name</i>	

```
109 function getQuotient(uint128 arg0, uint128 arg1, uint128 arg2)
110     private inline pure returns (uint128) {
111     (uint128 quotient, ) = math.muldivmod(arg0, arg1, arg2);
112     return quotient;
}
```

1.9.5 Function getRemainder

Parameters		
uint128	arg0	
uint128	arg1	
uint128	arg2	
Returns		
uint128	<i>no name</i>	

```

115     function getRemainder(uint128 arg0, uint128 arg1, uint128 arg2)
116         private inline pure returns (uint128) {
117             (, uint128 remainder) = math.muldivmod(arg0, arg1, arg2);
118             return remainder;
119         }

```

1.9.6 Function isReady

Parameters		
address	pair	
Returns		
bool	<i>no name</i>	

```

198     function isReady(address pair) private inline view returns (bool)
199     {
200         Pair cp = pairs[pair];
201         Connector ccA = connectors[rootConnector[cp.rootA]];
202         Connector ccB = connectors[rootConnector[cp.rootB]];
203         return cp.status && rootWallet.exists(cp.rootA) && rootWallet.
                exists(cp.rootB) && rootConnector.exists(cp.rootA) &&
                rootConnector.exists(cp.rootB) && ccA.status && ccB.status;
204     }

```

1.9.7 Function isReadyToProvide

Parameters		
address	pair	
Returns		
bool	<i>no name</i>	

```

206     function isReadyToProvide(address pair) private inline view
207         returns (bool) {
208         Pair cp = pairs[pair];
209         Connector ccA = connectors[rootConnector[cp.rootA]];
210         Connector ccB = connectors[rootConnector[cp.rootB]];
211         return cp.status && rootWallet.exists(cp.rootA) && rootWallet.
                exists(cp.rootB) && rootWallet.exists(cp.rootAB) &&
                rootConnector.exists(cp.rootA) && rootConnector.exists(cp.
                rootB) && ccA.status && ccB.status;
212     }

```

1.9.8 Function thisBalance

Returns		
uint128	<i>no name</i>	

```
346     function thisBalance() private inline pure returns (uint128) {  
347         return address(this).balance;  
348     }
```

Chapter 2

Contract DEXConnector

In file `DEXConnector.sol`

2.1 Contract Inheritance

IEpectedWalletAddressCallback	
IDEXConnector	

2.2 Constant Definitions

Advises		
Use a naming convention to distinguish constants from other, such as all uppercase names.		
Use <code>ton</code> unit instead of nanotons for cost constants to avoid numbers with too many zeroes.		
uint128	GRAMS_TO_ROOT	Initialized to 500000000
uint128	GRAMS_TO_NEW_WALLET	Initialized to 250000000

```
19  uint128 constant GRAMS_TO_ROOT = 500000000;
```

```
20  uint128 constant GRAMS_TO_NEW_WALLET = 250000000;
```

2.3 Static Variable Definitions

Advises	
Use a naming convention to distinguish static variables from global variables, such as <code>s_</code> prefix.	

uint256	soUINT	
address	dexclient	
		used in @2.DEXConnector.transfer
		used in @2.DEXConnector.transfer
		used in @2.DEXConnector.setTransferCallback
		used in @2.DEXConnector.setTransferCallback
		used in @2.DEXConnector.setBouncedCallback
		used in @2.DEXConnector.setBouncedCallback
		used in @2.DEXConnector.expectedWalletAddressCallback
		used in @2.DEXConnector.expectedWalletAddressCallback
		used in @2.DEXConnector.deployEmptyWallet
		used in @2.DEXConnector.deployEmptyWallet
		used in @2.DEXConnector.deployEmptyWallet
		used in @2.DEXConnector.deployEmptyWallet
		used in @2.DEXConnector.burn
		used in @2.DEXConnector.burn

```
15  uint256 static public soUINT;
```

```
16  address static public dexclient;
```

2.4 Variable Definitions

Advises

Use a naming convention to distinguish global variables from local variables, such as `g_` or `m_` prefix.

address	drivenRoot	
		used in @2.DEXConnector.expectedWalletAddressCallback
		assigned in @2.DEXConnector.deployEmptyWallet
		used in @2.DEXConnector.deployEmptyWallet
address	driven	
		used in @2.DEXConnector.transfer
		used in @2.DEXConnector.transfer
		used in @2.DEXConnector.setTransferCallback
		used in @2.DEXConnector.setTransferCallback
		used in @2.DEXConnector.setBouncedCallback
		used in @2.DEXConnector.setBouncedCallback
		assigned in @2.DEXConnector.expectedWalletAddressCallback
		used in @2.DEXConnector.expectedWalletAddressCallback
		used in @2.DEXConnector.burn
		used in @2.DEXConnector.burn
bool	statusConnected	
		assigned in @2.DEXConnector.expectedWalletAddressCallback
		used in @2.DEXConnector.expectedWalletAddressCallback
		used in @2.DEXConnector.deployEmptyWallet
		assigned in @2.DEXConnector.:constructor
		used in @2.DEXConnector.:constructor

22 `address public drivenRoot;`

23 `address public driven;`

24 `bool public statusConnected;`

2.5 Modifier Definitions

Advises

Calling `tvm.accept()` without checking pubkey should not be allowed

2.5.1 Modifier `alwaysAccept`

```
27  modifier alwaysAccept {
28      tvm.accept();
29      -;
30  }
```

2.5.2 Modifier `checkOwnerAndAccept`

```
32  modifier checkOwnerAndAccept {
33      // Check that message from contract owner.
34      require(msg.sender == dexclient, 101);
35      tvm.accept();
36      -;
37  }
```

2.6 Constructor Definitions

2.6.1 Constructor

Critical issue: Constructor for `DEXConnector`

loren ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum
loren ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum
loren ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum
loren ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum
loren ipsum lorem ipsum

Advises

Check who can call the constructor. If the constructor sets global values, only legitimate users should be allowed.

Check that every argument is protected by a `require()`.

If external users are allowed, their pubkey should be verified (`require(msg.pubkey() != 0 && msg.pubkey() == tvm.pubkey(),100)`), and `tvm.accept()` should be called.

Modifiers

<code>checkOwnerAndAccept</code>	<i>no args</i>
----------------------------------	----------------

```
39  constructor() public checkOwnerAndAccept {
40      statusConnected = false;
```

41 }

2.7 Public Method Definitions

2.7.1 Function burn

Parameters		
uint128	tokens	
address	callback_address	
TvmCell	callback_payload	

```
116     function burn(uint128 tokens, address callback_address, TvmCell
117         callback_payload) public override {
118         require(msg.sender == dexclient, 101);
119         tvm.rawReserve(address(this).balance - msg.value, 2);
120         TvmCell body = tvm.encodeBody(IBurnableByOwnerTokenWallet(
121             driven).burnByOwner, tokens, 0, dexclient, callback_address
122             , callback_payload);
123         driven.transfer({value: 0, bounce:true, flag: 128, body:body});
124     }
```

2.7.2 Function deployEmptyWallet

Parameters		
address	root	

```
60     function deployEmptyWallet(address root) public override {
61         require(msg.sender == dexclient, 101);
62         require(!(msg.value < GRAMS_TO_ROOT * 2), 103);
63         tvm.rawReserve(address(this).balance - msg.value, 2);
64         if (!statusConnected) {
65             drivenRoot = root;
66             TvmCell bodyD = tvm.encodeBody(IRootTokenContract(root).
67                 deployEmptyWallet, GRAMS_TO_NEW_WALLET, 0, address(this),
68                 dexclient);
69             root.transfer({value:GRAMS_TO_ROOT, bounce:true, body:bodyD})
70             ;
71             TvmCell bodyA = tvm.encodeBody(IRootTokenContract(root).
72                 sendExpectedWalletAddress, 0, address(this), address(this)
73                 );
74         }
```

```

69     root.transfer({value: GRAMS_TO_ROOT, bounce: true, body: bodyA})
70     ;
71     dexclient.transfer({value: 0, bounce: true, flag: 128});
72   } else {
73     dexclient.transfer({value: 0, bounce: true, flag: 128});
74   }
75 }

```

2.7.3 Function expectedWalletAddressCallback

Parameters		
address	wallet	
uint256	wallet_public_key	
address	owner_address	

```

77 function expectedWalletAddressCallback(address wallet, uint256
78   wallet_public_key, address owner_address) public override {
79   require(msg.sender == drivenRoot && wallet_public_key == 0 &&
80     owner_address == address(this), 102);
81   tvn.rawReserve(address(this).balance - msg.value, 2);
82   statusConnected = true;
83   driven = wallet;
84   TvmCell body = tvn.encodeBody(IDEXConnect(dexclient).
85     connectCallback, wallet);
86   dexclient.transfer({value: 0, bounce: true, flag: 128, body: body
87     });
88 }

```

2.7.4 Function getBalance

Returns		
uint128	balance	
Modifiers		
checkOwnerAndAccept	no args	

```

124 function getBalance() public pure checkOwnerAndAccept returns (
125   uint128 balance){
126   balance = address(this).balance;
127 }

```

2.7.5 Function setBouncedCallback

```
95 function setBouncedCallback() public override {
96     require(msg.sender == dexclient, 101);
97     tvvm.rawReserve(address(this).balance - msg.value, 2);
98     TvmCell body = tvvm.encodeBody(ITONTTokenWallet(driven).
        setBouncedCallback, dexclient);
99     driven.transfer({value: 0, bounce:true, flag: 128, body:body});
100 }
```

2.7.6 Function setTransferCallback

```
87 function setTransferCallback() public override {
88     require(msg.sender == dexclient, 101);
89     tvvm.rawReserve(address(this).balance - msg.value, 2);
90     TvmCell body = tvvm.encodeBody(ITONTTokenWallet(driven).
        setReceiveCallback, dexclient, true);
91     driven.transfer({value: 0, bounce:true, flag: 128, body:body});
92 }
```

2.7.7 Function transfer

Parameters		
address	to	
uint128	tokens	
TvmCell	payload	

```
108 function transfer(address to, uint128 tokens, TvmCell payload)
109     public override {
110     require(msg.sender == dexclient, 101);
110     tvvm.rawReserve(address(this).balance - msg.value, 2);
111     TvmCell body = tvvm.encodeBody(ITONTTokenWallet(driven).transfer,
        to, tokens, 0, dexclient, true, payload);
112     driven.transfer({value: 0, bounce:true, flag: 128, body:body});
113 }
```

2.8 Internal Method Definitions

2.8.1 Receive function

```
129 receive() external {  
130 }
```

2.8.2 Function getQuotient

Parameters		
uint128	arg0	
uint128	arg1	
uint128	arg2	
Returns		
uint128	<i>no name</i>	

```
48 function getQuotient(uint128 arg0, uint128 arg1, uint128 arg2)  
    private inline pure returns (uint128) {  
49     (uint128 quotient, ) = math.muldivmod(arg0, arg1, arg2);  
50     return quotient;  
51 }
```

2.8.3 Function getRemainder

Parameters		
uint128	arg0	
uint128	arg1	
uint128	arg2	
Returns		
uint128	<i>no name</i>	

```
54 function getRemainder(uint128 arg0, uint128 arg1, uint128 arg2)  
    private inline pure returns (uint128) {
```

```
55     (, uint128 remainder) = math.muldivmod(arg0, arg1, arg2);  
56     return remainder;  
57 }
```

Chapter 3

Contract DEXPair

In file DEXPair.sol

3.1 Contract Inheritance

IDEXPair	
IDEXConnect	
ITokensReceivedCallback	
IBurnTokensCallback	

3.2 Type Definitions

Advises

Check that types have the correct integer types (Pubkey : uint256, Amount: uint128, Time: uint64).

3.2.1 Struct Connector

root_address	address	
souint	uint256	
status	bool	

```
37 struct Connector {
38     address root_address;
39     uint256 souint;
40     bool status;
41 }
```

3.2.2 Struct Callback

token_wallet	address	
token_root	address	
amount	uint128	
sender_public_key	uint256	
sender_address	address	
sender_wallet	address	
original_gas_to	address	
updated_balance	uint128	
payload_arg0	uint8	
payload_arg1	address	
payload_arg2	address	

```

48 struct Callback {
49     address token_wallet;
50     address token_root;
51     uint128 amount;
52     uint256 sender_public_key;
53     address sender_address;
54     address sender_wallet;
55     address original_gas_to;
56     uint128 updated_balance;
57     uint8 payload_arg0;
58     address payload_arg1;
59     address payload_arg2;
60 }

```

3.3 Constant Definitions

Advises		
Use a naming convention to distinguish constants from other, such as all uppercase names.		
Use ton unit instead of nanotons for cost constants to avoid numbers with too many zeroes.		
uint128	GRAMS_SET_CALLBACK_ADDR	Initialized to 500000000
uint128	GRAMS_SEND_UNUSED_RETURN	Initialized to 100000000
uint128	GRAMS_MINT	Initialized to 50000000
uint128	GRAMS_RETURN	Initialized to 200000000

```

65 uint128 constant GRAMS_SET_CALLBACK_ADDR = 500000000;

```

```

66 uint128 constant GRAMS_SEND_UNUSED_RETURN = 100000000;

```

```

67 uint128 constant GRAMS_MINT = 50000000;

```

```

68 uint128 constant GRAMS_RETURN = 200000000;

```


3.4 Static Variable Definitions

Advises

Use a naming convention to distinguish static variables from global variables, such as `s_` prefix.

[illegible]

```
19  address static public rootDEX;
20  uint256 static public soUINT;
21  address static public creator;
22  TvmCell static public codeDEXConnector;
23  address static public rootA;
24  address static public rootB;
25  address static public rootAB;
```

3.5 Variable Definitions

Advises

Use a naming convention to distinguish global variables from local variables, such as `g_` or `m_` prefix.

mapping (address => address)	walletReserve	
		used in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		assigned in @3.DEX Pair.connectCallback
		used in @3.DEX Pair.connectCallback
		used in @3.DEXPair.connect
		used in @3.DEXPair.connect
mapping (address => bool)	syncStatus	
		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		assigned in @3.DEX Pair.connectCallback
		used in @3.DEX Pair.connectCallback
mapping (address => uint128)	balanceReserve	
		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
43		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback
		assigned in @3.DEX Pair.tokensReceivedCallback
		used in @3.DEX Pair.tokensReceivedCallback

```

27 mapping(address => address) public walletReserve;
28 mapping(address => bool) public syncStatus;
29 mapping(address => uint128) public balanceReserve;
31 uint128 public totalSupply;
33 mapping(address => mapping(address => bool)) public
    processingStatus;
34 mapping(address => mapping(address => uint128)) public
    processingData;
35 mapping(address => mapping(address => address)) public
    processingDest;
43 mapping (address => address) public rootConnector;
44 mapping (address => Connector) public connectors;
46 uint public counterCallback;
62 mapping (uint => Callback) callbacks;

```

3.6 Modifier Definitions

Advises

Calling tvn.accept() without checking pubkey should not be allowed

3.6.1 Modifier alwaysAccept

```

71 modifier alwaysAccept {
72     tvn.accept();
73     -;
74 }

```

3.6.2 Modifier checkOwnerAndAccept

```

76 modifier checkOwnerAndAccept {
77     require(msg.sender == rootDEX, 102);
78     tvn.accept();
79     -;
80 }

```

3.6.3 Modifier checkPubKeyAndAccept

```
82     modifier checkPubKeyAndAccept {
83         require(msg.pubkey() == tvn.pubkey(), 103);
84         tvn.accept();
85         -;
86     }
```

3.7 Constructor Definitions

3.7.1 Constructor

Critical issue: Constructor for DEXPair		
<p> lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum ipsum lorem ipsum </p>		
Advises		
Check who can call the constructor. If the constructor sets global values, only legitimate users should be allowed.		
Check that every argument is protected by a require().		
If external users are allowed, their pubkey should be verified (require(msg.pubkey() != 0 && msg.pubkey() == tvn.pubkey(),100) , and tvn.accept() should be called.		
Parameters		
uint256	souintA	
uint256	souintB	
uint128	gramsDeployConnector	
uint128	gramsDeployWallet	
Modifiers		
checkOwnerAndAccept	no args	

```

88     constructor(uint256 souintA, uint256 souintB, uint128
        gramsDeployConnector, uint128 gramsDeployWallet) public
        checkOwnerAndAccept {
89         counterCallback = 0;
90         connectRoot(rootA, souintA, gramsDeployConnector,
            gramsDeployWallet);
91         connectRoot(rootB, souintB, gramsDeployConnector,
            gramsDeployWallet);
92     }

```

3.8 Public Method Definitions

3.8.1 Function burnCallback

Parameters		
uint128	tokens	
TvmCell	payload	
uint256	sender_public_key	
address	sender_address	
address	wallet_address	
address	send_gas_to	
Modifiers		
alwaysAccept	<i>no args</i>	

```
522 function burnCallback(  
523     uint128 tokens,  
524     TvmCell payload,  
525     uint256 sender_public_key,  
526     address sender_address,  
527     address wallet_address,  
528     address send_gas_to  
529 ) public override alwaysAccept {  
530     if (msg.sender == rootAB) {  
531         tvn.rawReserve(address(this).balance - msg.value, 2);  
532         TvmSlice slice = payload.toSlice();  
533         (uint8 arg0, address arg1, address arg2) = slice.decode(uint8,  
534             address, address);  
535         counterCallback++;  
536         Callback cc = callbacks[counterCallback];  
537         cc.token_wallet = wallet_address;  
538         cc.token_root = msg.sender;  
539         cc.amount = tokens;  
540         cc.sender_public_key = sender_public_key;  
541         cc.sender_address = sender_address;  
542         cc.sender_wallet = wallet_address;  
543         cc.original_gas_to = address(0);  
544         cc.updated_balance = 0;  
545         cc.payload_arg0 = arg0;  
546         cc.payload_arg1 = arg1;  
547         cc.payload_arg2 = arg2;  
548         callbacks[counterCallback] = cc;  
549         if (arg0 == 3 && arg1 != address(0) && arg2 != address(0)) {  
550             uint128 returnA = math.muldiv(balanceReserve[rootA], tokens,  
551                 totalSupply);  
552             uint128 returnB = math.muldiv(balanceReserve[rootB], tokens,  
553                 totalSupply);  
554             if (!(returnA > balanceReserve[rootA]) && !(returnB >  
555                 balanceReserve[rootB])) {
```

```

552     totalSupply -= tokens;
553     balanceReserve[rootA] -= returnA;
554     balanceReserve[rootB] -= returnB;
555     TvmBuilder builder;
556     builder.store(uint8(6), address(0), address(0));
557     TvmCell new_payload = builder.toCell();
558     TvmCell bodyA = tvml.encodeBody(IDEXConnector(rootConnector[
559         rootA]).transfer, arg1, returnA, new_payload);
560     TvmCell bodyB = tvml.encodeBody(IDEXConnector(rootConnector[
561         rootB]).transfer, arg2, returnB, new_payload);
562     rootConnector[rootA].transfer({value: GRAMS_RETURN, bounce:
563         true, body:bodyA});
564     rootConnector[rootB].transfer({value: GRAMS_RETURN, bounce:
565         true, body:bodyB});
566     if (counterCallback > 10){delete callbacks[getFirstCallback
567         ()];}
568     send_gas_to.transfer({value: 0, bounce:true, flag: 128});
569     }
570     if (counterCallback > 10){delete callbacks[getFirstCallback()
571         ];}
572     }
573     if (counterCallback > 10){delete callbacks[getFirstCallback()
574         ];}
575 }
576 }

```

3.8.2 Function connect

```

149 function connect() public override {
150     address dexclient = msg.sender;
151     tvml.rawReserve(address(this).balance - msg.value, 2);
152     TvmCell body = tvml.encodeBody(IDEXClient(dexclient).setPair,
153         rootA, walletReserve[rootA], rootB, walletReserve[rootB],
154         rootAB);
155     dexclient.transfer({ value: 0, flag: 128, body:body});
156 }

```

3.8.3 Function connectCallback

Parameters		
address	wallet	
Modifiers		
alwaysAccept	no args	


```

132 function connectCallback(address wallet) public override
    alwaysAccept {
133     address connector = msg.sender;
134     if (connectors.exists(connector)) {
135         Connector cr = connectors[connector];
136         walletReserve[cr.root_address] = wallet;
137         syncStatus[cr.root_address] = true;
138         rootConnector[cr.root_address] = connector;
139         TvmCell bodySTC = tvvm.encodeBody(IDEXConnector(connector).
            setTransferCallback());
140         connector.transfer({value: GRAMS_SET_CALLBACK_ADDR, bounce:
            true, flag: 0, body:bodySTC});
141         TvmCell bodySBC = tvvm.encodeBody(IDEXConnector(connector).
            setBouncedCallback());
142         connector.transfer({value: GRAMS_SET_CALLBACK_ADDR, bounce:
            true, flag: 0, body:bodySBC});
143         cr.status = true;
144         connectors[connector] = cr;
145     }
146 }

```

3.8.4 Function getBalance

Returns		
uint128	<i>no name</i>	

```

604 function getBalance() public pure responsible returns (uint128) {
605     return { value: 0, bounce: false, flag: 64 } thisBalance();
606 }

```

3.8.5 Function getCallback

Parameters		
uint256	id	
Returns		
address	token_wallet	
address	token_root	
uint128	amount	
uint256	sender_public_key	
address	sender_address	
address	sender_wallet	
address	original_gas_to	
uint128	updated_balance	
uint8	payload_arg0	
address	payload_arg1	
address	payload_arg2	
Modifiers		
checkPubKeyAndAccept	<i>no args</i>	

```

571 function getCallback(uint id) public view checkPubKeyAndAccept
    returns (
572     address token_wallet,
573     address token_root,
574     uint128 amount,
575     uint256 sender_public_key,
576     address sender_address,
577     address sender_wallet,
578     address original_gas_to,
579     uint128 updated_balance,
580     uint8 payload_arg0,
581     address payload_arg1,
582     address payload_arg2
583 ){
584     Callback cc = callbacks[id];
585     token_wallet = cc.token_wallet;
586     token_root = cc.token_root;
587     amount = cc.amount;
588     sender_public_key = cc.sender_public_key;
589     sender_address = cc.sender_address;
590     sender_wallet = cc.sender_wallet;
591     original_gas_to = cc.original_gas_to;
592     updated_balance = cc.updated_balance;
593     payload_arg0 = cc.payload_arg0;
594     payload_arg1 = cc.payload_arg1;
595     payload_arg2 = cc.payload_arg2;
596 }

```

3.8.6 Function getConnectorAddress

Parameters		
uint256	connectorSoArg	
Returns		
address	<i>no name</i>	

```

108     function getConnectorAddress(uint256 connectorSoArg) public view
109         responsible returns (address) {
110             return { value: 0, bounce: false, flag: 64 }
111                 computeConnectorAddress( connectorSoArg);
112         }

```

3.8.7 Function tokensReceivedCallback

Parameters		
address	token_wallet	
address	token_root	
uint128	amount	
uint256	sender_public_key	
address	sender_address	
address	sender_wallet	
address	original_gas_to	
uint128	updated_balance	
TvmCell	payload	
Modifiers		
alwaysAccept	<i>no args</i>	

```

248     function tokensReceivedCallback(
249         address token_wallet,
250         address token_root,
251         uint128 amount,
252         uint256 sender_public_key,
253         address sender_address,
254         address sender_wallet,
255         address original_gas_to,
256         uint128 updated_balance,
257         TvmCell payload
258     ) public override alwaysAccept {

```

```

259     if (msg.sender == walletReserve[rootA] || msg.sender ==
        walletReserve[rootB]) {
260         if (counterCallback > 10) {
261             Callback cc = callbacks[counterCallback];
262             cc.token_wallet = token_wallet;
263             cc.token_root = token_root;
264             cc.amount = amount;
265             cc.sender_public_key = sender_public_key;
266             cc.sender_address = sender_address;
267             cc.sender_wallet = sender_wallet;
268             cc.original_gas_to = original_gas_to;
269             cc.updated_balance = updated_balance;
270             TvmSlice slice = payload.toSlice();
271             (uint8 arg0, address arg1, address arg2) = slice.decode(
                uint8, address, address);
272             cc.payload_arg0 = arg0;
273             cc.payload_arg1 = arg1;
274             cc.payload_arg2 = arg2;
275             callbacks[counterCallback] = cc;
276             counterCallback++;
277             delete callbacks[getFirstCallback()];
278             if (arg0 == 1) {
279                 tvn.rawReserve(address(this).balance - msg.value, 2);
280                 uint128 amountOut = getAmountOut(amount, token_root, arg1
                    );
281                 if (!(amountOut > balanceReserve[arg1])){
282                     balanceReserve[token_root] += amount;
283                     balanceReserve[arg1] -= amountOut;
284                     syncStatus[token_root] = balanceReserve[token_root] ==
                        updated_balance ? true : false;
285                     TvmBuilder builder;
286                     builder.store(uint8(0), address(0), address(0));
287                     TvmCell new_payload = builder.toCell();
288                     TvmCell body = tvn.encodeBody(IDEXConnector(
                        rootConnector[arg1]).transfer, arg2, amountOut,
                        new_payload);
289                     rootConnector[arg1].transfer({value: 0, bounce:true,
                        flag: 128, body:body});
290                 } else {
291                     TvmBuilder builder;
292                     builder.store(uint8(8), address(0), address(0));
293                     TvmCell new_payload = builder.toCell();
294                     TvmCell body = tvn.encodeBody(IDEXConnector(
                        rootConnector[token_root]).transfer, token_wallet,
                        amount, new_payload);
295                     rootConnector[token_root].transfer({value: 0, bounce:
                        true, flag: 128, body:body});
296                 }
297             }
298             if (arg0 == 2) {
299                 tvn.rawReserve(address(this).balance - msg.value, 2);
300                 processingStatus[token_root][arg1] = true;
301                 processingData[token_root][arg1] += amount;
302                 processingDest[token_root][arg1] = sender_wallet;
303                 if (processingStatus[rootA][arg1] == true &&
                    processingStatus[rootB][arg1] == true) {
304                     uint128 amountA = processingData[rootA][arg1];

```

```

305     uint128 amountB = processingData[rootB][arg1];
306     if (totalSupply == 0 && balanceReserve[rootA] == 0 &&
        balanceReserve[rootB] == 0) {
307         uint128 liquidity = math.min(amountA, amountB);
308         balanceReserve[rootA] += amountA;
309         balanceReserve[rootB] += amountB;
310         totalSupply += liquidity;
311         TvmCell body = tvm.encodeBody(IRootTokenContract(
            rootAB).mint, liquidity, arg2);
312         rootAB.transfer({value: GRAMS_MINT, bounce:true, body
            :body});
313         cleanProcessing(arg1);
314         arg1.transfer({ value: 0, flag: 128});
315     } else {
316         (uint128 provideA, uint128 provideB) =
            acceptForProvide(amountA, amountB);
317         if (provideA > 0 && provideB > 0) {
318             uint128 liquidity = math.min(liquidityA(provideA),
                liquidityB(provideB));
319             uint128 unusedReturnA = amountA - provideA;
320             uint128 unusedReturnB = amountB - provideB;
321             balanceReserve[rootA] += provideA;
322             balanceReserve[rootB] += provideB;
323             totalSupply += liquidity;
324             TvmCell body = tvm.encodeBody(IRootTokenContract(
                rootAB).mint, liquidity, arg2);
325             rootAB.transfer({value: GRAMS_MINT, bounce:true,
                body:body});
326             if (unusedReturnA > 0 && unusedReturnB > 0) {
327                 TvmBuilder builder;
328                 builder.store(uint8(7), address(0), address(0));
329                 TvmCell new_payload = builder.toCell();
330                 TvmCell bodyA = tvm.encodeBody(IDEXConnector(
                    rootConnector[rootA]).transfer,
                    processingDest[rootA][arg1], unusedReturnA,
                    new_payload);
331                 TvmCell bodyB = tvm.encodeBody(IDEXConnector(
                    rootConnector[rootB]).transfer,
                    processingDest[rootB][arg1], unusedReturnB,
                    new_payload);
332                 rootConnector[rootA].transfer({value:
                    GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
                    bodyA});
333                 rootConnector[rootB].transfer({value:
                    GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
                    bodyB});
334                 cleanProcessing(arg1);
335                 arg1.transfer({ value: 0, flag: 128});
336             } else if (unusedReturnA > 0) {
337                 TvmBuilder builder;
338                 builder.store(uint8(7), address(0), address(0));
339                 TvmCell new_payload = builder.toCell();
340                 TvmCell bodyA = tvm.encodeBody(IDEXConnector(
                    rootConnector[rootA]).transfer,
                    processingDest[rootA][arg1], unusedReturnA,
                    new_payload);

```

```

341         rootConnector[rootA].transfer({value:
            GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
            bodyA});
342         cleanProcessing(arg1);
343         arg1.transfer({ value: 0, flag: 128});
344     } else if (unusedReturnB > 0) {
345         TvmBuilder builder;
346         builder.store(uint8(7), address(0), address(0));
347         TvmCell new_payload = builder.toCell();
348         TvmCell bodyB = tvn.encodeBody(IDEXConnector(
            rootConnector[rootB]).transfer,
            processingDest[rootB][arg1], unusedReturnB,
            new_payload);
349         rootConnector[rootB].transfer({value:
            GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
            bodyB});
350         cleanProcessing(arg1);
351         arg1.transfer({ value: 0, flag: 128});
352     } else {
353         cleanProcessing(arg1);
354         arg1.transfer({ value: 0, flag: 128});
355     }
356 } else {
357     TvmBuilder builder;
358     builder.store(uint8(9), address(0), address(0));
359     TvmCell new_payload = builder.toCell();
360     TvmCell bodyA = tvn.encodeBody(IDEXConnector(
        rootConnector[rootA]).transfer, processingDest[
        rootA][arg1], amountA, new_payload);
361     TvmCell bodyB = tvn.encodeBody(IDEXConnector(
        rootConnector[rootB]).transfer, processingDest[
        rootB][arg1], amountB, new_payload);
362     rootConnector[rootA].transfer({value:
        GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
        bodyA});
363     rootConnector[rootB].transfer({value:
        GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
        bodyB});
364     cleanProcessing(arg1);
365     arg1.transfer({ value: 0, flag: 128});
366 }
367 }
368 } else {
369     arg1.transfer({ value: 0, flag: 128});
370 }
371 }
372 } else {
373     Callback cc = callbacks[counterCallback];
374     cc.token_wallet = token_wallet;
375     cc.token_root = token_root;
376     cc.amount = amount;
377     cc.sender_public_key = sender_public_key;
378     cc.sender_address = sender_address;
379     cc.sender_wallet = sender_wallet;
380     cc.original_gas_to = original_gas_to;
381     cc.updated_balance = updated_balance;
382     TvmSlice slice = payload.toSlice();

```

```

383     (uint8 arg0, address arg1, address arg2) = slice.decode(
384         uint8, address, address);
385     cc.payload_arg0 = arg0;
386     cc.payload_arg1 = arg1;
387     cc.payload_arg2 = arg2;
388     callbacks[counterCallback] = cc;
389     counterCallback++;
390     if (arg0 == 1) {
391         tvn.rawReserve(address(this).balance - msg.value, 2);
392         uint128 amountOut = getAmountOut(amount, token_root, arg1
393             );
394         if (!(amountOut > balanceReserve[arg1])){
395             balanceReserve[token_root] += amount;
396             balanceReserve[arg1] -= amountOut;
397             syncStatus[token_root] = balanceReserve[token_root] ==
398                 updated_balance ? true : false;
399             TvmBuilder builder;
400             builder.store(uint8(0), address(0), address(0));
401             TvmCell new_payload = builder.toCell();
402             TvmCell body = tvn.encodeBody(IDEXConnector(
403                 rootConnector[arg1]).transfer, arg2, amountOut,
404                 new_payload);
405             rootConnector[arg1].transfer({value: 0, bounce:true,
406                 flag: 128, body:body});
407         } else {
408             TvmBuilder builder;
409             builder.store(uint8(8), address(0), address(0));
410             TvmCell new_payload = builder.toCell();
411             TvmCell body = tvn.encodeBody(IDEXConnector(
412                 rootConnector[token_root]).transfer, token_wallet,
413                 amount, new_payload);
414             rootConnector[token_root].transfer({value: 0, bounce:
415                 true, flag: 128, body:body});
416         }
417     }
418     if (arg0 == 2) {
419         tvn.rawReserve(address(this).balance - msg.value, 2);
420         processingStatus[token_root][arg1] = true;
421         processingData[token_root][arg1] += amount;
422         processingDest[token_root][arg1] = sender_wallet;
423         if (processingStatus[rootA][arg1] == true &&
424             processingStatus[rootB][arg1] == true) {
425             uint128 amountA = processingData[rootA][arg1];
426             uint128 amountB = processingData[rootB][arg1];
427             if (totalSupply == 0 && balanceReserve[rootA] == 0 &&
428                 balanceReserve[rootB] == 0) {
429                 uint128 liquidity = math.min(amountA, amountB);
430                 balanceReserve[rootA] += amountA;
431                 balanceReserve[rootB] += amountB;
432                 totalSupply += liquidity;
433                 TvmCell body = tvn.encodeBody(IRootTokenContract(
434                     rootAB).mint, liquidity, arg2);
435                 rootAB.transfer({value: GRAMS_MINT, bounce:true, body
436                     :body});
437                 cleanProcessing(arg1);
438                 arg1.transfer({ value: 0, flag: 128});
439             } else {

```

```

427         (uint128 provideA, uint128 provideB) =
428             acceptForProvide(amountA, amountB);
429         if (provideA > 0 && provideB > 0) {
430             uint128 liquidity = math.min(liquidityA(provideA),
431                 liquidityB(provideB));
432             uint128 unusedReturnA = amountA - provideA;
433             uint128 unusedReturnB = amountB - provideB;
434             balanceReserve[rootA] += provideA;
435             balanceReserve[rootB] += provideB;
436             totalSupply += liquidity;
437             TvmCell body = tvm.encodeBody(IRootTokenContract(
438                 rootAB).mint, liquidity, arg2);
439             rootAB.transfer({value: GRAMS_MINT, bounce:true,
440                 body:body});
441             if (unusedReturnA > 0 && unusedReturnB > 0) {
442                 TvmBuilder builder;
443                 builder.store(uint8(7), address(0), address(0));
444                 TvmCell new_payload = builder.toCell();
445                 TvmCell bodyA = tvm.encodeBody(IDEXConnector(
446                     rootConnector[rootA]).transfer,
447                     processingDest[rootA][arg1], unusedReturnA,
448                     new_payload);
449                 TvmCell bodyB = tvm.encodeBody(IDEXConnector(
450                     rootConnector[rootB]).transfer,
451                     processingDest[rootB][arg1], unusedReturnB,
452                     new_payload);
453                 rootConnector[rootA].transfer({value:
454                     GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
455                     bodyA});
456                 rootConnector[rootB].transfer({value:
457                     GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
458                     bodyB});
459                 cleanProcessing(arg1);
460                 arg1.transfer({ value: 0, flag: 128});
461             } else if (unusedReturnA > 0) {
462                 TvmBuilder builder;
463                 builder.store(uint8(7), address(0), address(0));
464                 TvmCell new_payload = builder.toCell();
465                 TvmCell bodyA = tvm.encodeBody(IDEXConnector(
466                     rootConnector[rootA]).transfer,
467                     processingDest[rootA][arg1], unusedReturnA,
468                     new_payload);
469                 rootConnector[rootA].transfer({value:
470                     GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
471                     bodyA});
472                 cleanProcessing(arg1);
473                 arg1.transfer({ value: 0, flag: 128});
474             } else if (unusedReturnB > 0) {
475                 TvmBuilder builder;
476                 builder.store(uint8(7), address(0), address(0));
477                 TvmCell new_payload = builder.toCell();
478                 TvmCell bodyB = tvm.encodeBody(IDEXConnector(
479                     rootConnector[rootB]).transfer,
480                     processingDest[rootB][arg1], unusedReturnB,
481                     new_payload);
482                 rootConnector[rootB].transfer({value:
483                     GRAMS_SEND_UNUSED_RETURN, bounce:true, body:

```



```

461         bodyB});
462         cleanProcessing(arg1);
463         arg1.transfer({ value: 0, flag: 128});
464     } else {
465         cleanProcessing(arg1);
466         arg1.transfer({ value: 0, flag: 128});
467     }
468     } else {
469         TvmBuilder builder;
470         builder.store(uint8(9), address(0), address(0));
471         TvmCell new_payload = builder.toCell();
472         TvmCell bodyA = tvm.encodeBody(IDEXConnector(
473             rootConnector[rootA]).transfer, processingDest[
474                 rootA][arg1], amountA, new_payload);
475         TvmCell bodyB = tvm.encodeBody(IDEXConnector(
476             rootConnector[rootB]).transfer, processingDest[
477                 rootB][arg1], amountB, new_payload);
478         rootConnector[rootA].transfer({value:
479             GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
480                 bodyA});
481         rootConnector[rootB].transfer({value:
482             GRAMS_SEND_UNUSED_RETURN, bounce:true, body:
483                 bodyB});
484         cleanProcessing(arg1);
485         arg1.transfer({ value: 0, flag: 128});
486     }
487 }
488 }
489 } else {
490     arg1.transfer({ value: 0, flag: 128});
491 }
492 }
493 }
494 }
495 }
496 }

```

3.9 Internal Method Definitions

3.9.1 Receive function

```

609 receive() external {
610 }

```

3.9.2 Function acceptForProvide

Parameters		
uint128	arg0	
uint128	arg1	
Returns		
uint128	<i>no name</i>	
uint128	<i>no name</i>	

```

214 function acceptForProvide(uint128 arg0, uint128 arg1) private
      inline view returns (uint128, uint128) {
215     require(balanceReserve[rootA] > 0 && balanceReserve[rootB] > 0,
        106);
216     uint128 qtyB = qtyBforA(arg0);
217     uint128 qtyA = qtyAforB(arg1);
218     uint128 minAmountA = math.min(arg0, qtyA);
219     uint128 minAmountB = math.min(arg1, qtyB);
220     uint128 crmin = math.min(balanceReserve[rootA], balanceReserve[
        rootB]);
221     uint128 crmax = math.max(balanceReserve[rootA], balanceReserve[
        rootB]);
222     uint128 crquotient = getQuotient(crmin, crmax);
223     uint128 crremainder = getRemainder(crmin, crmax);
224     uint128 amountMin = math.min(minAmountA, minAmountB);
225     uint128 amountOther = amountMin * crquotient + math.muldiv(
        amountMin, crremainder, crmin);
226     uint128 acceptForProvideA = minAmountA < minAmountB ? amountMin
        : amountOther;
227     uint128 acceptForProvideB = minAmountB < minAmountA ? amountMin
        : amountOther;
228     return (acceptForProvideA, acceptForProvideB);
229 }

```

3.9.3 Function cleanProcessing

Parameters		
address	dexclient	

```

232 function cleanProcessing(address dexclient) private inline {
233     delete processingStatus[rootA][dexclient];
234     delete processingStatus[rootB][dexclient];
235     delete processingData[rootA][dexclient];

```

```

236     delete processingData[rootB][dexclient];
237     delete processingDest[rootA][dexclient];
238     delete processingDest[rootB][dexclient];
239 }

```

3.9.4 Function computeConnectorAddress

Parameters		
uint256	souint	
Returns		
address	<i>no name</i>	

```

95     function computeConnectorAddress(uint256 souint) private inline
96         view returns (address) {
97         TvmCell stateInit = tvm.buildStateInit({
98             contr: DEXConnector,
99             varInit: { soUINT: souint, dexclient: address(this) },
100             code: codeDEXConnector,
101             pubkey: tvm.pubkey()
102         });
103         return address(tvm.hash(stateInit));

```

3.9.5 Function connectRoot

Parameters		
address	root	
uint256	souint	
uint128	gramsToConnector	
uint128	gramsToRoot	

```

113     function connectRoot(address root, uint256 souint, uint128
114         gramsToConnector, uint128 gramsToRoot) private inline {
115         TvmCell stateInit = tvm.buildStateInit({
116             contr: DEXConnector,
117             varInit: { soUINT: souint, dexclient: address(this) },
118             code: codeDEXConnector,
119             pubkey: tvm.pubkey()
120         });
121         TvmCell init = tvm.encodeBody(DEXConnector);

```

```

121     address connector = tvn.deploy(stateInit, init,
122         gramsToConnector, address(this).wid);
123     Connector cr = connectors[connector];
124     cr.root_address = root;
125     cr.souint = souint;
126     cr.status = false;
127     connectors[connector] = cr;
128     TvmCell body = tvn.encodeBody(IDEXConnector(connector).
129         deployEmptyWallet, root);
129     connector.transfer({value:gramsToRoot, bounce:true, body:body})
    ;
}

```

3.9.6 Function getAmountOut

Parameters		
uint128	amountIn	
address	rootIn	
address	rootOut	
Returns		
uint128	<i>no name</i>	

```

157 function getAmountOut(uint128 amountIn, address rootIn, address
158     rootOut) private inline view returns (uint128){
159     uint128 amountInWithFee = math.muldiv(amountIn,997,1);
160     uint128 numerator = math.muldiv(amountInWithFee,balanceReserve[
161         rootOut],1);
162     uint128 denominator = amountInWithFee + math.muldiv(
163         balanceReserve[rootIn],1000,1);
164     return math.muldiv(1,numerator,denominator);
165 }

```

3.9.7 Function getFirstCallback

Returns		
uint256	<i>no name</i>	

```

242 function getFirstCallback() private view returns (uint) {
243     optional(uint, Callback) rc = callbacks.min();
244     if (rc.hasValue()) {(uint number, ) = rc.get();return number;}
245     else {return 0;}
}

```

245 }

3.9.8 Function getQuotient

Parameters		
uint128	min	
uint128	max	
Returns		
uint128	<i>no name</i>	

```
165     function getQuotient(uint128 min, uint128 max) private inline
166         pure returns (uint128) {
167         (uint128 quotient, ) = math.muldivmod(1, max, min);
168         return quotient;
169     }
```

3.9.9 Function getRemainder

Parameters		
uint128	min	
uint128	max	
Returns		
uint128	<i>no name</i>	

```
171     function getRemainder(uint128 min, uint128 max) private inline
172         pure returns (uint128) {
173         (, uint128 remainder) = math.muldivmod(1, max, min);
174         return remainder;
175     }
```

3.9.10 Function liquidityA

Parameters		
uint128	arg0	
Returns		
uint128	<i>no name</i>	

```

191     function liquidityA(uint128 arg0) private inline view returns (
192         uint128) {
193         require(arg0 > 0, 105);
194         require(totalSupply > 0, 110);
195         require(balanceReserve[rootA] > 0, 108);
196         return math.muldiv(arg0, totalSupply, balanceReserve[rootA]);

```

3.9.11 Function liquidityB

Parameters		
uint128	arg1	
Returns		
uint128	<i>no name</i>	

```

199     function liquidityB(uint128 arg1) private inline view returns (
200         uint128) {
201         require(arg1 > 0, 105);
202         require(totalSupply > 0, 110);
203         require(balanceReserve[rootB] > 0, 109);
204         return math.muldiv(arg1, totalSupply, balanceReserve[rootB]);

```

3.9.12 Function qtyAforB

Parameters		
uint128	arg1	
Returns		
uint128	<i>no name</i>	

```

184     function qtyAforB(uint128 arg1) private inline view returns (
185         uint128) {
186         require(arg1 > 0, 107);
187         require(balanceReserve[rootA] > 0 && balanceReserve[rootB] > 0,
188             106);
189         return math.muldiv(arg1, balanceReserve[rootA], balanceReserve[
190             rootB]);

```

3.9.13 Function qtyBforA

Parameters		
uint128	arg0	
Returns		
uint128	<i>no name</i>	

```
177     function qtyBforA(uint128 arg0) private inline view returns (
178         uint128) {
179         require(arg0 > 0, 105);
179         require(balanceReserve[rootA] > 0 && balanceReserve[rootB] > 0,
180             106);
180         return math.muldiv(arg0, balanceReserve[rootB], balanceReserve[
181             rootA]);
181     }
```

3.9.14 Function thisBalance

Returns		
uint128	<i>no name</i>	

```
599     function thisBalance() private inline pure returns (uint128) {
600         return address(this).balance;
601     }
```

Chapter 4

Contract DEXroot

In file `DEXroot.sol`

4.1 Contract Inheritance

IDEXRoot	
----------	--

4.2 Type Definitions

Advises

Check that types have the correct integer types (Pubkey : uint256, Amount: uint128, Time: uint64).

4.2.1 Struct Pair

root0	address	
root1	address	
rootLP	address	

```
24 struct Pair {
25     address root0;
26     address root1;
27     address rootLP;
28 }
```


4.3 Constant Definitions

Advises

Use a naming convention to distinguish constants from other, such as all uppercase names.

Use `ton` unit instead of nanotons for cost constants to avoid numbers with too many zeroes.

uint128	GRAMS_CREATE_DEX_CLIENT	Initialized to 1 ton
---------	-------------------------	----------------------

```
42  uint128 constant public GRAMS_CREATE_DEX_CLIENT = 1 ton;
```

4.4 Static Variable Definitions

Advises

Use a naming convention to distinguish static variables from global variables, such as `s_` prefix.

uint256	soUINT	
---------	--------	--

```
13  uint256 static public soUINT;
```

4.5 Variable Definitions

Advises

Use a naming convention to distinguish global variables from local variables, such as `g_` or `m_` prefix.

TvmCell	codeDEXclient	
		assigned in @4 root.setDEXclientCode
		used in @4 root.setDEXclientCode
		used in @4 root.createDEXclient
		used in @4 root.computeClientAddress
TvmCell	codeDEXpair	
		assigned in @4 root.setDEXpairCode
		used in @4 root.setDEXpairCode
		used in @4 root.createDEXpair
		used in @4 root.computePairAddress
TvmCell	codeDEXconnector	
		assigned in @4 root.setDEXconnectorCode
		used in @4 root.setDEXconnectorCode
		used in @4 root.createDEXpair
		used in @4 root.createDEXclient
		used in @4 root.computePairAddress
		used in @4 root.computeConnectorAddress
		used in @4 root.computeClientAddress
TvmCell	codeRootToken	
		assigned in @4 root.setRootTokenCode
		used in @4 root.setRootTokenCode
		used in @4 root.createDEXpair
		used in @4 root.computeRootTokenAddress
TvmCell	codeTONTTokenWallet	
		assigned in @4 root.setTONTTokenWalletCode
		used in @4 root.setTONTTokenWalletCode
65		used in @4 root.createDEXpair
		used in @4 root.computeRootTokenAddress
mapping (address => mapping (address => address))	roots	
		used in @4 root.getPairByRoots10
		used in @4

```

15  TvmCell public codeDEXclient;
16  TvmCell public codeDEXpair;
17  TvmCell public codeDEXconnector;
18  TvmCell public codeRootToken;
19  TvmCell public codeTONTTokenWallet;
21  mapping(address => mapping(address => address)) roots;
30  mapping(address => Pair) public pairs;
31  address[] public pairKeys;
33  mapping(uint256 => address) public pubkeys;
34  mapping(address => uint256) public clients;
35  address[] public clientKeys;
37  mapping(address => uint128) public balanceOf;
38  mapping(uint256 => address) public creators;

```

4.6 Modifier Definitions

Advises

Calling `tvm.accept()` without checking pubkey should not be allowed

4.6.1 Modifier `alwaysAccept`

```

45  modifier alwaysAccept {
46      tvm.accept();
47      _;
48  }

```

4.6.2 Modifier `checkOwnerAndAccept`

```

51  modifier checkOwnerAndAccept {
52      require(msg.pubkey() == tvm.pubkey(), 101);
53      tvm.accept();
54      _;
55  }

```

4.6.3 Modifier `checkCreatorAndAccept`

```
modifier checkCreatorAndAccept {
    require(msg.pubkey() != 0, 103);
    tvmm.accept();
    -;
}
```

4.7 Constructor Definitions

4.7.1 Constructor

Critical issue: Constructor for DEXroot

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

Advises

Check who can call the constructor. If the constructor sets global values, only legitimate users should be allowed.

Check that every argument is protected by a `require()`.

If external users are allowed, their pubkey should be verified (`require(msg.pubkey() != 0 && msg.pubkey() == tvn.pubkey(),100)`), and `tvn.accept()` should be called.

```
constructor() public {
    require(tvm.pubkey() == msg.pubkey(), 102);
    tvm.accept();
}
```

4.8 Public Method Definitions

4.8.1 Function checkPubKey

Parameters		
uint256	pubkey	
Returns		
bool	status	
address	dexclient	
Modifiers		
alwaysAccept	<i>no args</i>	

```

328 function checkPubKey(uint256 pubkey) public view alwaysAccept
329     returns (bool status, address dexclient) {
330     status = pubkeys.exists(pubkey);
331     dexclient = pubkeys[pubkey];
332 }

```

4.8.2 Function createDEXclient

Parameters		
uint256	pubkey	
uint256	souint	
Returns		
address	deployedAddress	
bool	statusCreate	
Modifiers		
alwaysAccept	<i>no args</i>	

```

122 function createDEXclient(uint256 pubkey, uint256 souint) public
123     alwaysAccept returns (address deployedAddress, bool
124     statusCreate){
125     statusCreate = false;
126     deployedAddress = address(0);
127     uint128 prepay = balanceOf[creators[pubkey]];
128     require (!pubkeys.exists(pubkey) && !(prepay <
129     GRAMS_CREATE_DEX_CLIENT), 106);
130     delete balanceOf[creators[pubkey]];
131     TvmCell stateInit = tvml.buildStateInit({
132     contr: DEXClient,
133     varInit: {rootDEX: address(this), soUINT: souint,
134     codeDEXConnector: codeDEXconnector},
135     code: codeDEXclient,
136     pubkey: pubkey
137     });
138     deployedAddress = new DEXClient{
139     stateInit: stateInit,
140     flag: 0,
141     bounce : false,
142     value : (prepay - 3100000)
143     }();
144     pubkeys[pubkey] = deployedAddress;
145     clients[deployedAddress] = pubkey;
146     clientKeys.push(deployedAddress);
147     statusCreate = true;
148 }

```

4.8.3 Function createDEXpair

Parameters		
address	root0	
address	root1	
uint256	pairSoArg	
uint256	connectorSoArg0	
uint256	connectorSoArg1	
uint256	rootSoArg	
bytes	rootName	
bytes	rootSymbol	
uint8	rootDecimals	
uint128	grammsForPair	
uint128	grammsForRoot	
uint128	grammsForConnector	
uint128	grammsForWallet	

```

237 function createDEXpair(
238     address root0,
239     address root1,
240     uint256 pairSoArg,
241     uint256 connectorSoArg0,
242     uint256 connectorSoArg1,
243     uint256 rootSoArg,
244     bytes rootName,
245     bytes rootSymbol,
246     uint8 rootDecimals,
247     uint128 grammsForPair,
248     uint128 grammsForRoot,
249     uint128 grammsForConnector,
250     uint128 grammsForWallet
251 ) public override {
252     require(root0 != address(0) && root1 != address(0), 104);
253     require(!(grammsForPair < 5000000000) && !(grammsForRoot <
        5000000000) && !(grammsForConnector < 5000000000) && !(
        grammsForWallet < 5000000000), 105);
254     tvn.rawReserve(address(this).balance - msg.value, 2);
255     uint128 grammsNeeded = grammsForPair + (2 * grammsForConnector)
        + (2 * grammsForWallet) + grammsForRoot;
256     if (clients.exists(msg.sender) && !(msg.value < grammsNeeded)
        && !(root0 == root1) && !roots[root0].exists(root1) && !
        roots[root1].exists(root0)) {
257         TvmCell stateInitR = tvn.buildStateInit({
258             contr: RootTokenContract,
259             varInit: {
260                 _randomNonce: rootSoArg,
261                 name: rootName,
262                 symbol: rootSymbol,

```

```

263         decimals:rootDecimals,
264         wallet_code:codeTONTOKENWallet
265     },
266     code: codeRootToken,
267     pubkey : clients[msg.sender]
268 });
269 address root01 = address(tvm.hash(stateInitR));
270 TvmCell stateInitP = tvM.buildStateInit({
271     contr: DEXPair,
272     varInit: {
273         rootDEX:address(this),
274         soUINT:pairSoArg,
275         creator:msg.sender,
276         codeDEXConnector:codeDEXconnector,
277         rootA:root0,
278         rootB:root1,
279         rootAB:root01
280     },
281     code: codeDEXpair,
282     pubkey : clients[msg.sender]
283 });
284 address pairAddress = new DEXPair{
285     stateInit: stateInitP,
286     flag: 0,
287     bounce : false,
288     value : gramsForPair + (2 * gramsForConnector) + (2 *
289         gramsForWallet)
290 }(connectorSoArg0, connectorSoArg1, gramsForConnector,
291     gramsForWallet);
292 address rootAddress = new RootTokenContract{
293     stateInit: stateInitR,
294     flag: 0,
295     bounce : false,
296     value : gramsForRoot
297 }(0, pairAddress);
298 roots[root0][root1] = pairAddress;
299 roots[root1][root0] = pairAddress;
300 Pair cp = pairs[pairAddress];
301 cp.root0 = root0;
302 cp.root1 = root1;
303 cp.rootLP = rootAddress;
304 pairs[pairAddress] = cp;
305 pairKeys.push(pairAddress);
306 msg.sender.transfer({ value: 0, flag: 128});
307 } else {
308     msg.sender.transfer({ value: 0, flag: 128});
309 }
310 }

```

4.8.4 Function getBalanceTONgrams

Returns		
uint128	balanceTONgrams	
Modifiers		
alwaysAccept	no args	

```

334 function getBalanceTONgrams() public pure alwaysAccept returns (
      uint128 balanceTONgrams){
335     return address(this).balance;
336 }

```

4.8.5 Function getClientAddress

Parameters		
uint256	clientPubKey	
uint256	clientSoArg	
Returns		
address	<i>no name</i>	

```

118 function getClientAddress(uint256 clientPubKey, uint256
      clientSoArg) public view responsible returns (address) {
119     return { value: 0, bounce: false, flag: 64 }
      computeClientAddress(clientPubKey,clientSoArg);
120 }

```

4.8.6 Function getConnectorAddress

Parameters		
uint256	connectorPubKey	
uint256	connectorSoArg	
address	connectorCommander	
Returns		
address	<i>no name</i>	

```

233 function getConnectorAddress(uint256 connectorPubKey, uint256
      connectorSoArg, address connectorCommander) public view
      responsible returns (address) {
234     return { value: 0, bounce: false, flag: 64 }
      computeConnectorAddress(connectorPubKey, connectorSoArg,
      connectorCommander);
235 }

```


4.8.7 Function getPairAddress

Parameters		
uint256	pairPubKey	
uint256	pairSoArg	
address	pairCreator	
address	pairRootA	
address	pairRootB	
address	pairRootAB	
Returns		
address	<i>no name</i>	

```

171 function getPairAddress(
172     uint256 pairPubKey,
173     uint256 pairSoArg,
174     address pairCreator,
175     address pairRootA,
176     address pairRootB,
177     address pairRootAB
178 ) public view responsible returns (address) {
179     return { value: 0, bounce: false, flag: 64 } computePairAddress
        (pairPubKey,pairSoArg,pairCreator,pairRootA,pairRootB,
        pairRootAB);
180 }

```

4.8.8 Function getPairByRoots01

Parameters		
address	root0	
address	root1	
Returns		
address	pairAddr	
Modifiers		
alwaysAccept	<i>no args</i>	

```

314 function getPairByRoots01(address root0, address root1) public
    view alwaysAccept returns (address pairAddr) {
315     pairAddr = roots[root0][root1];
316 }

```

4.8.9 Function getPairByRoots10

Parameters		
address	root1	
address	root0	
Returns		
address	pairAddr	
Modifiers		
alwaysAccept	<i>no args</i>	

```

318     function getPairByRoots10(address root1, address root0) public
319         view alwaysAccept returns (address pairAddr) {
320             pairAddr = roots[root1][root0];
321         }

```

4.8.10 Function getRootTokenAddress

Parameters		
uint256	rootPubKey	
uint256	rootSoArg	
bytes	rootName	
bytes	rootSymbol	
uint8	rootDecimals	
Returns		
address	<i>no name</i>	

```

213     function getRootTokenAddress(
214         uint256 rootPubKey,
215         uint256 rootSoArg,
216         bytes rootName,
217         bytes rootSymbol,
218         uint8 rootDecimals
219     ) public view responsible returns (address) {
220         return { value: 0, bounce: false, flag: 64 }
221             computeRootTokenAddress(rootPubKey, rootSoArg, rootName,
222                                     rootSymbol, rootDecimals);
223     }

```

4.8.11 Function getRootsByPair

Parameters		
address	pairAddr	
Returns		
address	root0	
address	root1	
Modifiers		
alwaysAccept	<i>no args</i>	

```
322 function getRootsByPair(address pairAddr) public view
    alwaysAccept returns (address root0, address root1) {
323     Pair cp = pairs[pairAddr];
324     root0 = cp.root0;
325     root1 = cp.root1;
326 }
```

4.8.12 Function sendTransfer

Parameters		
address	dest	
uint128	value	
bool	bounce	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```
71 function sendTransfer(address dest, uint128 value, bool bounce)
    public pure checkOwnerAndAccept {
72     dest.transfer(value, bounce, 0);
73 }
```

4.8.13 Function setCreator

Parameters		
address	giverAddr	
Modifiers		
checkCreatorAndAccept	<i>no args</i>	

```

100  function setCreator(address giverAddr) public
      checkCreatorAndAccept {
101      uint256 pubkey = msg.pubkey();
102      creators[pubkey] = giverAddr;
103  }

```

4.8.14 Function setDEXclientCode

Parameters		
TvmCell	code	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

80  function setDEXclientCode(TvmCell code) public
      checkOwnerAndAccept {
81      codeDEXclient = code;
82  }

```

4.8.15 Function setDEXconnectorCode

Parameters		
TvmCell	code	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

88  function setDEXconnectorCode(TvmCell code) public
      checkOwnerAndAccept {
89      codeDEXconnector = code;
90  }

```

4.8.16 Function setDEXpairCode

Parameters		
TvmCell	code	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

84     function setDEXpairCode(TvmCell code) public checkOwnerAndAccept
85         {
86             codeDEXpair = code;
87         }

```

4.8.17 Function setRootTokenCode

Parameters		
TvmCell	code	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

92     function setRootTokenCode(TvmCell code) public
93         checkOwnerAndAccept {
94             codeRootToken = code;
95         }

```

4.8.18 Function setTONTokenWalletCode

Parameters		
TvmCell	code	
Modifiers		
checkOwnerAndAccept	<i>no args</i>	

```

96     function setTONTokenWalletCode(TvmCell code) public
97         checkOwnerAndAccept {
98             codeTONTTokenWallet = code;
99         }

```

4.9 Internal Method Definitions

4.9.1 Receive function

```

76     receive() external {
77         balanceOf[msg.sender] += msg.value;
78     }

```

4.9.2 Function computeClientAddress

Parameters		
uint256	pubkey	
uint256	souint	
Returns		
address	<i>no name</i>	

```

108     function computeClientAddress(uint256 pubkey, uint256 souint)
109         private inline view returns (address) {
110         TvmCell stateInit = tvm.buildStateInit({
111             contr: DEXClient,
112             varInit: {rootDEX: address(this), soUINT: souint,
113                 codeDEXConnector: codeDEXconnector},
114             code: codeDEXclient,
115             pubkey: pubkey
116         });
117         return address(tvm.hash(stateInit));
118     }

```

4.9.3 Function computeConnectorAddress

Parameters		
uint256	pubkey	
uint256	souint	
address	commander	
Returns		
address	<i>no name</i>	

```

223     function computeConnectorAddress(uint256 pubkey, uint256 souint,
224         address commander) private inline view returns (address) {
225     TvmCell stateInit = tvm.buildStateInit({
226         contr: DEXConnector,
227         varInit: { soUINT: souint, dexclient: commander },
228         code: codeDEXconnector,
229         pubkey: pubkey
230     });

```

```

230     return address(tvm.hash(stateInit));
231 }

```

4.9.4 Function computePairAddress

Parameters		
uint256	pubkey	
uint256	souint	
address	creator	
address	rootA	
address	rootB	
address	rootAB	
Returns		
address	<i>no name</i>	

```

146 function computePairAddress(
147     uint256 pubkey,
148     uint256 souint,
149     address creator,
150     address rootA,
151     address rootB,
152     address rootAB
153 ) private inline view returns (address){
154     TvmCell stateInit = tvm.buildStateInit({
155         contr: DEXPair,
156         varInit: {
157             rootDEX: address(this),
158             soUINT: souint,
159             creator: creator,
160             codeDEXConnector: codeDEXconnector,
161             rootA: rootA,
162             rootB: rootB,
163             rootAB: rootAB
164         },
165         code: codeDEXpair,
166         pubkey : pubkey
167     });
168     return address(tvm.hash(stateInit));
169 }

```

4.9.5 Function computeRootTokenAddress

Parameters		
uint256	pubkey	
uint256	souint	
bytes	name	
bytes	symbol	
uint8	decimals	
Returns		
address	<i>no name</i>	

```
182 function computeRootTokenAddress(  
183     uint256 pubkey,  
184     uint256 souint,  
185     bytes name,  
186     bytes symbol,  
187     uint8 decimals  
188 ) private inline view returns (address){  
189     TvmCell stateInit = tvm.buildStateInit({  
190         contr: RootTokenContract,  
191         varInit: {  
192             _randomNonce:souint,  
193             name:name,  
194             symbol:symbol,  
195             decimals:decimals,  
196             wallet_code:codeTONTTokenWallet  
197         },  
198         code: codeRootToken,  
199         pubkey : pubkey  
200     });  
201     return address(tvm.hash(stateInit));  
202 }
```


Chapter 5

Contract RootTokenContract

In file `RootTokenContract.sol`

5.1 Contract Inheritance

IRootTokenContract	
IBurnableTokenRootContract	
IBurnableByRootTokenRootContract	
IPausable	
ITransferOwner	
ISendSurplusGas	
IVersioned	

5.2 Static Variable Definitions

Advises
Use a naming convention to distinguish static variables from global variables, such as <code>s_</code> prefix.

uint256	_randomNonce	
bytes	name	
		used in @5.RootTokenContract.getDetails
bytes	symbol	
		used in @5.RootTokenContract.getDetails
uint8	decimals	
		used in @5.RootTokenContract.getDetails
TvmCell	wallet_code	
		used in @5.RootTokenContract.getWalletCode
		used in @5.RootTokenContract.getExpectedWalletAddress
		used in @5.RootTokenContract.getExpectedWalletAddress
		used in @5.RootTokenContract.deployWallet
		used in @5.RootTokenContract.deployWallet
		used in @5.RootTokenContract.deployEmptyWallet

```
28     uint256 static _randomNonce;
```

```
30     bytes public static name;
```

```
31     bytes public static symbol;
```

```
32     uint8 public static decimals;
```

```
34     TvmCell static wallet_code;
```

5.3 Variable Definitions

Advises

Use a naming convention to distinguish global variables from local variables, such as `g_` or `m_` prefix.

uint128	total_supply	
		assigned in @5.RootTokenContract.tokensBurned
		used in @5.RootTokenContract.tokensBurned
		assigned in @5.RootTokenContract.mint
		used in @5.RootTokenContract.mint
		used in @5.RootTokenContract.getTotalSupply
		used in @5.RootTokenContract.getDetails
		assigned in @5.RootTokenContract.deployWallet
		used in @5.RootTokenContract.deployWallet
		assigned in @5.RootTokenContract.onBounce
		used in @5.RootTokenContract.onBounce
		assigned in @5.RootTokenContract.constructor
		used in @5.RootTokenContract.constructor
uint256	root_public_key	
		assigned in @5.RootTokenContract.transferOwner
		used in @5.RootTokenContract.transferOwner
		used in @5.RootTokenContract.isExternalOwner
		used in @5.RootTokenContract.isExternalOwner
		used in @5.RootTokenContract.getDetails
		assigned in @5.RootTokenContract.constructor
		used in @5.RootTokenContract.constructor
address	root_owner_address	
		assigned in @5.RootTokenContract.transferOwner
		used in @5.RootTokenContract.transferOwner
		used in @5.RootTokenContract.isInternalOwner
		used ⁸² in @5.RootTokenContract.isInternalOwner
		used in @5.RootTokenContract.getDetails
		used in @5.RootTokenContract.deployWallet
		used in @5.RootTokenContract.deployWallet

```
36     uint128 total_supply;
38     uint256 root_public_key;
39     address root_owner_address;
40     uint128 public start_gas_balance;
42     bool public paused;
```

5.4 Modifier Definitions

Advises

Calling `tvm.accept()` without checking pubkey should not be allowed

5.4.1 Modifier onlyOwner

```
458     modifier onlyOwner() {
459         require(isOwner(), RootTokenContractErrors.
            error_message_sender_is_not_my_owner);
460     };
461 }
```

5.4.2 Modifier onlyInternalOwner

```
463     modifier onlyInternalOwner() {
464         require(isInternalOwner(), RootTokenContractErrors.
            error_message_sender_is_not_my_owner);
465     -;
466 }
```

5.5 Constructor Definitions

5.5.1 Constructor

Critical issue: Constructor for RootTokenContract

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

Advises

Check who can call the constructor. If the constructor sets global values, only legitimate users should be allowed.

Check that every argument is protected by a `require()`.

If external users are allowed, their pubkey should be verified (`require(msg.pubkey() != 0 && msg.pubkey() == tvn.pubkey(),100)`), and `tvn.accept()` should be called.

Parameters		
uint256	root_public_key_	
address	root_owner_address_	

```

48     constructor(uint256 root_public_key_, address
        root_owner_address_) public {
49         require((root_public_key_ != 0 && root_owner_address_.value
            == 0) ||
50             (root_public_key_ == 0 && root_owner_address_.value
                != 0),
51             RootTokenContractErrors.
                error_define_public_key_or_owner_address);
52         tvn.accept();
53
54         root_public_key = root_public_key_;
55         root_owner_address = root_owner_address_;
56
57         total_supply = 0;
58         paused = false;
59
60         start_gas_balance = address(this).balance;
61     }

```

5.6 Internal Method Definitions

5.6.1 Fallback function

```

523     fallback() external {
524     }

```

5.6.2 OnBounce function

Parameters		
TvmSlice	slice	

```

514     onBounce(TvmSlice slice) external {
515         tvml.accept();
516         uint32 functionId = slice.decode(uint32);
517         if (functionId == tvml.functionId(ITONTOKENWallet.accept)) {
518             uint128 latest_bounced_tokens = slice.decode(uint128);
519             total_supply -= latest_bounced_tokens;
520         }
521     }

```

5.6.3 Function deployEmptyWallet

Parameters		
uint128	deploy_grams	
uint256	wallet_public_key_	
address	owner_address_	
address	gas_back_address	
Returns		
address	<i>no name</i>	

```

237     function deployEmptyWallet(
238         uint128 deploy_grams,
239         uint256 wallet_public_key_,
240         address owner_address_,
241         address gas_back_address
242     )
243     override
244     external
245     returns (
246         address
247     ) {
248         require((owner_address_.value != 0 && wallet_public_key_ ==
249             0) ||
250             (owner_address_.value == 0 && wallet_public_key_ !=
251             0),
252             RootTokenContractErrors.
253             error_define_public_key_or_owner_address);
254
255         tvml.rawReserve(address(this).balance - msg.value, 2);
256
257         address wallet = new TONTOKENWallet{
258             value: deploy_grams,
259             flag: 1,
260             code: wallet_code,
261             pubkey: wallet_public_key_,
262             varInit: {
263                 root_address: address(this),
264                 code: wallet_code,
265                 wallet_public_key: wallet_public_key_,

```

```

263         owner_address: owner_address_
264     }
265     }();
266
267     if (gas_back_address.value != 0) {
268         gas_back_address.transfer({ value: 0, flag: 128 });
269     } else {
270         msg.sender.transfer({ value: 0, flag: 128 });
271     }
272
273     return wallet;
274 }

```

5.6.4 Function deployWallet

Parameters		
uint128	tokens	
uint128	deploy_grams	
uint256	wallet_public_key_	
address	owner_address_	
address	gas_back_address	
Returns		
address	<i>no name</i>	
Modifiers		
onlyOwner	<i>no args</i>	

```

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

```

```

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

```

5.6.5 Function getDetails

Returns		
IRootTokenContractDetails	<i>no name</i>	


```

77     function getDetails() override external view responsible
78         returns (IRootTokenContractDetails) {
79             return { value: 0, bounce: false, flag: 64 }
80                 IRootTokenContractDetails(
81                     name,
82                     symbol,
83                     decimals,
84                     root_public_key,
85                     root_owner_address,
86                     total_supply
87                 );
88         }

```

5.6.6 Function getExpectedWalletAddress

Parameters		
uint256	wallet_public_key_	
address	owner_address_	
Returns		
address	<i>no name</i>	

```

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

```

5.6.7 Function getTotalSupply

Returns		
uint128	<i>no name</i>	

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

5.6.8 Function getVersion

Returns		
uint32	<i>no name</i>	

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

5.6.9 Function getWalletAddress

Parameters		
uint256	wallet_public_key_	
address	owner_address_	
Returns		
address	<i>no name</i>	

```
111     function getWalletAddress(
112         uint256 wallet_public_key_,
113         address owner_address_
114     )
115     override
116     external
117     view
```

```

118         responsible
119     returns (
120         address
121     ) {
122         require((owner_address_.value != 0 && wallet_public_key_ ==
123                 0) ||
124                 (owner_address_.value == 0 && wallet_public_key_ !=
125                 0),
126                 RootTokenContractErrors.
127                 error_define_public_key_or_owner_address);
128         return { value: 0, bounce: false, flag: 64 }
129         getExpectedWalletAddress(wallet_public_key_,
130                                 owner_address_);
131     }

```

5.6.10 Function getWalletCode

Returns		
TvmCell	<i>no name</i>	

```

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

```

5.6.11 Function isExternalOwner

Returns		
bool	<i>no name</i>	

```

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

```

5.6.12 Function isInternalOwner

Returns		
bool	<i>no name</i>	

```

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

```

5.6.13 Function isOwner

Returns		
bool	<i>no name</i>	

```

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

```

5.6.14 Function mint

Parameters		
uint128	tokens	
address	to	
Modifiers		
onlyOwner	<i>no args</i>	

```

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

```

5.6.15 Function proxyBurn

Parameters		
uint128	tokens	
address	sender_address	
address	send_gas_to	
address	callback_address	
TvmCell	callback_payload	
Modifiers		
onlyInternalOwner	<i>no args</i>	

```

307     function proxyBurn(
308         uint128 tokens,
309         address sender_address,
310         address send_gas_to,
311         address callback_address,
312         TvmCell callback_payload
313     )
314     override
315     external
316     onlyInternalOwner
317     {
318         tvn.rawReserve(address(this).balance - msg.value, 2);
319
320         address send_gas_to_ = send_gas_to;
321         address expectedWalletAddress = getExpectedWalletAddress(0,
            sender_address);
322
323         if (send_gas_to.value == 0) {
324             send_gas_to_ = sender_address;
325         }
326
327         IBurnableByRootTokenWallet(expectedWalletAddress).
            burnByRoot{value: 0, flag: 128}(
328             tokens,
329             send_gas_to_,
330             callback_address,
331             callback_payload
332         );
333     }

```

5.6.16 Function sendExpectedWalletAddress

Parameters		
uint256	wallet_public_key_	
address	owner_address_	
address	to	

```

134     function sendExpectedWalletAddress(
135         uint256 wallet_public_key_,
136         address owner_address_,
137         address to
138     )
139     override
140     external
141     {
142         tvm.rawReserve(address(this).balance - msg.value, 2);
143
144         address wallet = getExpectedWalletAddress(
145             wallet_public_key_, owner_address_);
146         IExpectedWalletAddressCallback(to).
147             expectedWalletAddressCallback{value: 0, flag: 128}(
148             wallet,
149             wallet_public_key_,
150             owner_address_
151         );
152     }

```

5.6.17 Function sendPausedCallbackTo

Parameters		
uint64	callback_id	
address	callback_addr	

```

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

```

5.6.18 Function sendSurplusGas

Parameters		
address	to	
Modifiers		
onlyInternalOwner	no args	

```

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

```

5.6.19 Function setPaused

Parameters		
bool	value	
Modifiers		
onlyOwner	<i>no args</i>	

```

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

```

5.6.20 Function tokensBurned

Parameters		
uint128	tokens	
uint256	sender_public_key	
address	sender_address	
address	send_gas_to	
address	callback_address	
TvmCell	callback_payload	

```

347     function tokensBurned(
348         uint128 tokens,
349         uint256 sender_public_key,
350         address sender_address,
351         address send_gas_to,
352         address callback_address,
353         TvmCell callback_payload
354     ) override external {
355
356         require(!paused, RootTokenContractErrors.error_paused);
357
358         address expectedWalletAddress = getExpectedWalletAddress(
359             sender_public_key, sender_address);
360
361         require(msg.sender == expectedWalletAddress,
362             RootTokenContractErrors.
363             error_message_sender_is_not_good_wallet);
364
365         tvmm.rawReserve(address(this).balance - msg.value, 2);
366
367         total_supply -= tokens;
368
369         if (callback_address.value == 0) {
370             send_gas_to.transfer({ value: 0, flag: 128 });
371         } else {
372             IBurnTokensCallback(callback_address).burnCallback({
373                 value: 0, flag: 128}{
374                     tokens,
375                     callback_payload,
376                     sender_public_key,
377                     sender_address,
378                     expectedWalletAddress,
379                     send_gas_to
380                 });
381         }
382     }
383 }

```

5.6.21 Function transferOwner

Parameters		
uint256	root_public_key_	
address	root_owner_address_	
Modifiers		
onlyOwner	<i>no args</i>	

```

440     function transferOwner(
441         uint256 root_public_key_,
442         address root_owner_address_

```



```

443     )
444     override
445     external
446     onlyOwner
447     {
448         require((root_public_key_ != 0 && root_owner_address_.value
449                 == 0) ||
449                 (root_public_key_ == 0 && root_owner_address_.value
450                 != 0),
450                 RootTokenContractErrors.
451                 error_define_public_key_or_owner_address);
451         tvn.accept();
452         root_public_key = root_public_key_;
453         root_owner_address = root_owner_address_;
454     }

```

Chapter 6

Contract TONTokenWallet

In file `TONTokenWallet.sol`

6.1 Contract Inheritance

ITONTokenWallet	
IDestroyable	
IBurnableByOwnerTokenWallet	
IBurnableByRootTokenWallet	
IVersioned	

6.2 Static Variable Definitions

Advises
Use a naming convention to distinguish static variables from global variables, such as <code>s_</code> prefix.

address	root_address	
		used in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.internalTransfer
		used in @6.TONTokenWallet.getExpectedAddress
		used in @6.TONTokenWallet.getDetails
		used in @6.TONTokenWallet.burnByRoot
		used in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.:onBounce
		used in @6.TONTokenWallet.:constructor
TvmCell	code	
		used in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.getWalletCode
		used in @6.TONTokenWallet.getExpectedAddress
		used in @6.TONTokenWallet.getExpectedAddress
uint256	wallet_public_key	
		used in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.transfer
		used in @6.TONTokenWallet.transfer
		used in @6.TONTokenWallet.internalTransferFrom
		used in @6.TONTokenWallet.internalTransfer
		used in @6.TONTokenWallet.getDetails
		used ⁹⁸ in @6.TONTokenWallet.burnByRoot
		used in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.:constructor

```
24     address static root_address;

25     TvmCell static code;

27     uint256 static wallet_public_key;

29     address static owner_address;
```

6.3 Variable Definitions

Advises

Use a naming convention to distinguish global variables from local variables, such as `g_` or `m_` prefix.

uint128	balance_	
		assigned in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.transferToRecipient
		assigned in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.transferToRecipient
		used in @6.TONTokenWallet.transferToRecipient
		assigned in @6.TONTokenWallet.transfer
		used in @6.TONTokenWallet.transfer
		assigned in @6.TONTokenWallet.transfer
		used in @6.TONTokenWallet.transfer
		used in @6.TONTokenWallet.transfer
		assigned in @6.TONTokenWallet.internalTransferFrom
		used in @6.TONTokenWallet.internalTransferFrom
		used in @6.TONTokenWallet.internalTransferFrom
		used in @6.TONTokenWallet.internalTransfer
		assigned in @6.TONTokenWallet.internalTransfer
		used in @6.TONTokenWallet.internalTransfer
		used in @6.TONTokenWallet.getDetails
		used in @6.TONTokenWallet.destroy
		assigned in @6.TONTokenWallet.burnByRoot
		used in @6.TONTokenWallet.burnByRoot
		used in @6.TONTokenWallet.burnByRoot
		assigned in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.burnByOwner
	100	assigned in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.burnByOwner
		used in @6.TONTokenWallet.balance

```

31     uint128 balance_;
32     optional(AllowanceInfo) allowance_;
34     address receive_callback;
35     address bounced_callback;
36     bool allow_non_notifiable;

```

6.4 Modifier Definitions

Advises

Calling `tvm.accept()` without checking pubkey should not be allowed

6.4.1 Modifier onlyRoot

```

598     modifier onlyRoot() {
599         require(root_address == msg.sender, TONTokenWalletErrors.
600             error_message_sender_is_not_my_root);
601     }

```

6.4.2 Modifier onlyOwner

```

603     modifier onlyOwner() {
604         require((owner_address.value != 0 && owner_address == msg.
605             sender) ||
606             (wallet_public_key != 0 && wallet_public_key == msg.
607                 .pubkey()),
608             TONTokenWalletErrors.
609                 error_message_sender_is_not_my_owner);
610     }

```

6.4.3 Modifier onlyInternalOwner

```

610     modifier onlyInternalOwner() {
611         require(owner_address.value != 0 && owner_address == msg.
612             sender);
613     }

```

6.5 Constructor Definitions

6.5.1 Constructor

Critical issue: Constructor for TONTokenWallet
<pre> loren ipsum </pre>
Advises
<p>Check who can call the constructor. If the constructor sets global values, only legitimate users should be allowed.</p> <p>Check that every argument is protected by a require().</p> <p>If external users are allowed, their pubkey should be verified (<code>require(msg.pubkey() != 0 && msg.pubkey() == tvn.pubkey(),100)</code>), and <code>tnv.accept()</code> should be called.</p>

```

43     constructor() public {
44         require(wallet_public_key == tvmm.pubkey() && (owner_address
          .value == 0 || wallet_public_key == 0));
45         tvmm.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     }

```

6.6 Public Method Definitions

6.6.1 Function destroy

Parameters		
address	gas_dest	
Modifiers		
onlyOwner	<i>no args</i>	

```

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

```

6.7 Internal Method Definitions

6.7.1 Fallback function

```

683     fallback() external {
684     }

```

6.7.2 OnBounce function

Parameters		
TvmSlice	body	

```

653     onBounce(TvmSlice body) external {
654         tvn.accept();
655
656         uint32 functionId = body.decode(uint32);
657         if (functionId == tvn.functionId(ITONTokenWallet.
            internalTransfer)) {
658             uint128 tokens = body.decode(uint128);
659             balance_ += tokens;
660
661             if (bounced_callback.value != 0) {
662                 tvn.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         tvn.rawReserve(math.max(TONTTokenWalletConstants.
            target_gas_balance, address(this).balance - msg
            .value), 2);
672         owner_address.transfer({ value: 0, flag: 128 });
673     }
674     } else if (functionId == tvn.functionId(
        IBurnableTokenRootContract.tokensBurned)) {
675         balance_ += body.decode(uint128);
676         if (owner_address.value != 0) {
677             tvn.rawReserve(math.max(TONTTokenWalletConstants.
                target_gas_balance, address(this).balance - msg
                .value), 2);
678             owner_address.transfer({ value: 0, flag: 128 });
679         }
680     }
681 }

```

6.7.3 Function accept

Parameters		
uint128	tokens	
Modifiers		
onlyRoot	<i>no args</i>	

```

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

```

6.7.4 Function allowance

Returns		
AllowanceInfo	<i>no name</i>	

```

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     }

```

6.7.5 Function approve

Parameters		
address	spender	
uint128	remaining_tokens	
uint128	tokens	
Modifiers		
onlyOwner	<i>no args</i>	

```

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(),
            TONTTokenWalletErrors.error_non_zero_remaining);
129         if (owner_address.value != 0 ) {
130             tvn.rawReserve(math.max(TONTTokenWalletConstants.
                target_gas_balance, address(this).balance - msg.
                    value), 2);
131         } else {
132             tvn.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     }

```

6.7.6 Function balance

Returns		
uint128	<i>no name</i>	

```

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

```

6.7.7 Function burnByOwner

Parameters		
uint128	tokens	
uint128	grams	
address	send_gas_to	
address	callback_address	
TvmCell	callback_payload	
Modifiers		
onlyOwner	<i>no args</i>	

```

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.
482             error_not_enough_balance);
483         require((owner_address.value != 0 && msg.value > 0) ||
484             (owner_address.value == 0 && grams <= address(this)
485                 .balance && grams > 0), TONTOKENWalletErrors.
486             error_low_message_value);
487
488         if (owner_address.value != 0) {
489             tvn.rawReserve(math.max(TONTOKENWalletConstants.
490                 target_gas_balance, address(this).balance - msg.
491                     value), 2);
492             balance_ -= tokens;
493             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 :
494                 owner_address,
495             callback_address,
496             callback_payload
497         );
498     } else {
499         tvml.accept();
500         balance_ -= tokens;
501         IBurnableTokenRootContract(root_address)
502             .tokensBurned{ value: grams, bounce: true }(
503                 tokens,
504                 wallet_public_key,
505                 owner_address,
506                 send_gas_to.value != 0 ? send_gas_to : address(
507                     this),
508                 callback_address,
509                 callback_payload
510             );
511     }
512 }

```

6.7.8 Function burnByRoot

Parameters		
uint128	tokens	
address	send_gas_to	
address	callback_address	
TvmCell	callback_payload	
Modifiers		
onlyRoot	<i>no args</i>	

```

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_, TONTTokenWalletErrors.
528             error_not_enough_balance);
529
530         tvml.rawReserve(address(this).balance - msg.value, 2);
531
532         balance_ -= tokens;
533     }
534 }

```

```

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

```

6.7.9 Function disapprove

Modifiers		
onlyOwner	<i>no args</i>	

```

148     function disapprove() override external onlyOwner {
149         if (owner_address.value != 0 ) {
150             tvn.rawReserve(math.max(TONTokenWalletConstants.
151                 target_gas_balance, address(this).balance - msg.
152                 value), 2);
153         } else {
154             tvn.accept();
155         }
156         allowance_.reset();
157
158         if (owner_address.value != 0 ) {
159             msg.sender.transfer({ value: 0, flag: 128 });
160         }
161     }
162

```

6.7.10 Function getDetails

Returns		
ITONTokenWalletDetails	<i>no name</i>	

```

72     function getDetails() override external view responsible
73     returns (ITONTokenWalletDetails) {
74         return { value: 0, bounce: false, flag: 64 }
75         ITONTokenWalletDetails(
76

```

```

74         root_address,
75         wallet_public_key,
76         owner_address,
77         balance_,
78         receive_callback,
79         bounced_callback,
80         allow_non_notifiable
81     );
82 }

```

6.7.11 Function getExpectedAddress

Parameters		
uint256	wallet_public_key_	
address	owner_address_	
Returns		
address	<i>no name</i>	

```

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     }

```

6.7.12 Function getVersion

Returns		
uint32	<i>no name</i>	

```

54     function getVersion() override external pure responsible
55         returns (uint32) {
56             return 4;
57         }

```

6.7.13 Function getWalletCode

Returns		
TvmCell	<i>no name</i>	

```

87     function getWalletCode() override external view responsible
88         returns (TvmCell) {
89             return { value: 0, bounce: false, flag: 64 } code;
90         }

```

6.7.14 Function internalTransfer

Parameters		
uint128	tokens	
uint256	sender_public_key	
address	sender_address	
address	send_gas_to	
bool	notify_receiver	
TvmCell	payload	

```

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 ||
382             receive_callback.value == 0,
383             TONTTokenWalletErrors.
384                 error_recipient_has_disallow_non_notifiable);

```

```

383     address expectedSenderAddress = getExpectedAddress(
384         sender_public_key, sender_address);
385     require(msg.sender == expectedSenderAddress,
386         TONTokenWalletErrors.
387         error_message_sender_is_not_good_wallet);
388     require(sender_address != owner_address ||
389         sender_public_key != wallet_public_key,
390         TONTokenWalletErrors.error_wrong_recipient);
391
392     if (owner_address.value != 0 ) {
393         uint128 reserve = math.max(TONTokenWalletConstants.
394             target_gas_balance, address(this).balance - msg.
395             value);
396         require(address(this).balance > reserve,
397             TONTokenWalletErrors.error_low_message_value);
398         tvvm.rawReserve(reserve, 2);
399     } else {
400         tvvm.rawReserve(address(this).balance - msg.value, 2);
401     }
402
403     balance_ += tokens;
404
405     if (notify_receiver && receive_callback.value != 0) {
406         ITokensReceivedCallback(receive_callback).
407             tokensReceivedCallback({ value: 0, flag: 128 })(
408                 address(this),
409                 root_address,
410                 tokens,
411                 sender_public_key,
412                 sender_address,
413                 msg.sender,
414                 send_gas_to,
415                 balance_,
416                 payload
417             );
418     } else {
419         send_gas_to.transfer({ value: 0, flag: 128 });
420     }
421 }

```

6.7.15 Function internalTransferFrom

Parameters		
address	to	
uint128	tokens	
address	send_gas_to	
bool	notify_receiver	
TvmCell	payload	


```

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             tvvm.rawReserve(reserve, 2);
444             tvvm.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             tvvm.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     }

```

6.7.16 Function setBouncedCallback

Parameters		
address	bounced_callback_	
Modifiers		
onlyOwner	<i>no args</i>	

```

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

```

6.7.17 Function setReceiveCallback

Parameters		
address	receive_callback_	
bool	allow_non_notifiable_	
Modifiers		
onlyOwner	<i>no args</i>	

```

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     }

```

6.7.18 Function transfer

Parameters		
address	to	
uint128	tokens	
uint128	grams	
address	send_gas_to	
bool	notify_receiver	
TvmCell	payload	
Modifiers		
onlyOwner	<i>no args</i>	

```

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.
272             error_not_enough_balance);
273         require(to.value != 0, TONTokenWalletErrors.
274             error_wrong_recipient);
275         require(to != address(this), TONTokenWalletErrors.
276             error_wrong_recipient);
277
278         if (owner_address.value != 0 ) {
279             uint128 reserve = math.max(TONTokenWalletConstants.
280                 target_gas_balance, address(this).balance - msg.
281                 value);
282             require(address(this).balance > reserve +
283                 TONTokenWalletConstants.target_gas_balance,
284                 TONTokenWalletErrors.error_low_message_value);
285             tvn.rawReserve(reserve, 2);
286             balance_ -= tokens;
287
288             ITONTokenWallet(to).internalTransfer{ value: 0, flag:
289                 129, bounce: true }(
290                 tokens,
291                 wallet_public_key,
292                 owner_address,
293                 send_gas_to.value != 0 ? send_gas_to :
294                     owner_address,
295                 notify_receiver,
296                 payload
297             );
298         } else {

```

```

290         require(address(this).balance > grams,
291                 TONTOKENWalletErrors.error_low_message_value);
292         require(grams > TONTOKENWalletConstants.
293                 target_gas_balance, TONTOKENWalletErrors.
294                 error_low_message_value);
295         tvM.accept();
296         balance_ -= tokens;
297
298         ITONTOKENWallet(to).internalTransfer{ value: grams,
299             bounce: true, flag: 1 }(
300             tokens,
301             wallet_public_key,
302             owner_address,
303             send_gas_to.value != 0 ? send_gas_to : address(this
304             ),
305             notify_receiver,
306             payload
307         );
308     }
309 }

```

6.7.19 Function transferFrom

Parameters		
address	from	
address	to	
uint128	tokens	
uint128	grams	
address	send_gas_to	
bool	notify_receiver	
TvmCell	payload	
Modifiers		
onlyOwner	<i>no args</i>	

```

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.
331             error_wrong_recipient);
332     require(tokens > 0);
333     require(from != to, TONTokenWalletErrors.
334             error_wrong_recipient);
335
336     if (owner_address.value != 0 ) {
337         uint128 reserve = math.max(TONTokenWalletConstants.
338             target_gas_balance, address(this).balance - msg.
339             value);
340         require(address(this).balance > reserve + (
341             TONTokenWalletConstants.target_gas_balance * 2),
342             TONTokenWalletErrors.error_low_message_value);
343         tvn.rawReserve(reserve, 2);
344
345         ITONTokenWallet(from).internalTransferFrom{ value: 0,
346             flag: 129 }(
347             to,
348             tokens,
349             send_gas_to.value != 0 ? send_gas_to :
350                 owner_address,
351             notify_receiver,
352             payload
353         );
354     } else {
355         require(address(this).balance > grams,
356             TONTokenWalletErrors.error_low_message_value);
357         require(grams > TONTokenWalletConstants.
358             target_gas_balance * 2, TONTokenWalletErrors.
359             error_low_message_value);
360         tvn.accept();
361         ITONTokenWallet(from).internalTransferFrom{ value:
362             grams, flag: 1 }(
363             to,
364             tokens,
365             send_gas_to.value != 0 ? send_gas_to : address(this
366             ),
367             notify_receiver,
368             payload
369         );
370     }
371 }

```

6.7.20 Function transferToRecipient

Parameters		
uint256	recipient_public_key	
address	recipient_address	
uint128	tokens	
uint128	deploy_grams	
uint128	transfer_grams	
address	send_gas_to	
bool	notify_receiver	
TvmCell	payload	
Modifiers		
onlyOwner	<i>no args</i>	

```

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.
189             error_not_enough_balance);
190         require(recipient_address.value == 0 ||
191             recipient_public_key == 0, TONTokenWalletErrors.
192             error_wrong_recipient);
193
194         if (owner_address.value != 0 ) {
195             uint128 reserve = math.max(TONTokenWalletConstants.
196                 target_gas_balance, address(this).balance - msg.
197                 value);
198             require(address(this).balance > reserve +
199                 TONTokenWalletConstants.target_gas_balance +
200                 deploy_grams, TONTokenWalletErrors.
201                 error_low_message_value);
202             require(recipient_address != owner_address,
203                 TONTokenWalletErrors.error_wrong_recipient);
204             tvn.rawReserve(reserve, 2);
205         } else {
206             require(address(this).balance > deploy_grams +
207                 transfer_grams, TONTokenWalletErrors.
208                 error_low_message_value);
209             require(transfer_grams > TONTokenWalletConstants.
210                 target_gas_balance, TONTokenWalletErrors.

```

```

219         error_low_message_value);
220         require(recipient_public_key != wallet_public_key);
221         tvvm.accept();
222     }
223
224     TvmCell stateInit = tvvm.buildStateInit({
225         contr: TONTOKENWallet,
226         varInit: {
227             root_address: root_address,
228             code: code,
229             wallet_public_key: recipient_public_key,
230             owner_address: recipient_address
231         },
232         pubkey: recipient_public_key,
233         code: code
234     });
235
236     address to;
237
238     if(deploy_grams > 0) {
239         to = new TONTOKENWallet{
240             stateInit: stateInit,
241             value: deploy_grams,
242             wid: address(this).wid,
243             flag: 1
244         }();
245     } else {
246         to = address(tvm.hash(stateInit));
247     }
248
249     if (owner_address.value != 0 ) {
250         balance_ -= tokens;
251         ITONTOKENWallet(to).internalTransfer{ value: 0, flag:
252             129, bounce: true }(
253             tokens,
254             wallet_public_key,
255             owner_address,
256             send_gas_to.value != 0 ? send_gas_to :
257                 owner_address,
258             notify_receiver,
259             payload
260         );
261     } else {
262         balance_ -= tokens;
263         ITONTOKENWallet(to).internalTransfer{ value:
264             transfer_grams, flag: 1, bounce: true }(
265             tokens,
266             wallet_public_key,
267             owner_address,
268             send_gas_to.value != 0 ? send_gas_to : address(this
269                 ),
270             notify_receiver,
271             payload
272         );
273     }
274 }

```