

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 library LowGasSafeMath {
5     /// @notice Returns x + y, reverts if sum overflows
6     uint256
7     /// @param x The augend
8     /// @param y The addend
9     /// @return z The sum of x and y
10    function add(uint256 x, uint256 y) internal pure
11    returns (uint256 z) {
12        require((z = x + y) >= x);
13    }
14
15    function add32(uint32 x, uint32 y) internal pure
16    returns (uint32 z) {
17        require((z = x + y) >= x);
18    }
19
20    /// @notice Returns x - y, reverts if underflows
21    s
22    /// @param x The minuend
23    /// @param y The subtrahend
24    /// @return z The difference of x and y
25    function sub(uint256 x, uint256 y) internal pure
26    returns (uint256 z) {
27        require((z = x - y) <= x);
28    }
29
30    function sub32(uint32 x, uint32 y) internal pure
31    returns (uint32 z) {
32        require((z = x - y) <= x);
33    }
34
35    /// @notice Returns x * y, reverts if overflows
36    /// @param x The multiplicand
37    /// @param y The multiplier
38    /// @return z The product of x and y
39    function mul(uint256 x, uint256 y) internal pure
40    returns (uint256 z) {
41        require(x == 0 || (z = x * y) / x == y);
42    }
43
44    function mul32(uint32 x, uint32 y) internal pure
45    returns (uint32 z) {
46        require(x == 0 || (z = x * y) / x == y);
47    }
48
49    /// @notice Returns x + y, reverts if overflows
50    or underflows
51    /// @param x The augend
52    /// @param y The addend
53    /// @return z The sum of x and y
54    function add(int256 x, int256 y) internal pure
55    returns (int256 z) {
56        require((z = x + y) >= x == (y >= 0));
57    }
58
59    /// @notice Returns x - y, reverts if overflows
60    or underflows
61    /// @param x The minuend
62    /// @param y The subtrahend
63    /// @return z The difference of x and y

```

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 library LowGasSafeMath {
5     /// @notice Returns x + y, reverts if sum overflows
6     uint256
7     /// @param x The augend
8     /// @param y The addend
9     /// @return z The sum of x and y
10    function add(uint256 x, uint256 y) internal pure
11    returns (uint256 z) {
12        require((z = x + y) >= x);
13    }
14
15    function add32(uint32 x, uint32 y) internal pure
16    returns (uint32 z) {
17        require((z = x + y) >= x);
18    }
19
20    /// @notice Returns x - y, reverts if underflows
21    s
22    /// @param x The minuend
23    /// @param y The subtrahend
24    /// @return z The difference of x and y
25    function sub(uint256 x, uint256 y) internal pure
26    returns (uint256 z) {
27        require((z = x - y) <= x);
28    }
29
30    function sub32(uint32 x, uint32 y) internal pure
31    returns (uint32 z) {
32        require((z = x - y) <= x);
33    }
34
35    /// @notice Returns x * y, reverts if overflows
36    /// @param x The multiplicand
37    /// @param y The multiplier
38    /// @return z The product of x and y
39    function mul(uint256 x, uint256 y) internal pure
40    returns (uint256 z) {
41        require(x == 0 || (z = x * y) / x == y);
42    }
43
44    function mul32(uint32 x, uint32 y) internal pure
45    returns (uint32 z) {
46        require(x == 0 || (z = x * y) / x == y);
47    }
48
49    /// @notice Returns x + y, reverts if overflows
50    or underflows
51    /// @param x The augend
52    /// @param y The addend
53    /// @return z The sum of x and y
54    function add(int256 x, int256 y) internal pure
55    returns (int256 z) {
56        require((z = x + y) >= x == (y >= 0));
57    }
58
59    /// @notice Returns x - y, reverts if overflows
60    or underflows
61    /// @param x The minuend
62    /// @param y The subtrahend
63    /// @return z The difference of x and y

```

```

53     function sub(int256 x, int256 y) internal pure
        returns (int256 z) {
54         require((z = x - y) <= x == (y >= 0));
55     }
56
57     function div(uint256 x, uint256 y) internal pur
        e returns(uint256 z){
58         require(y > 0);
59         z=x/y;
60     }
61 }
62
63 library Address {
64
65     function isContract(address account) internal vie
        w returns (bool) {
66         // This method relies in extcodesize, which
        returns 0 for contracts in
67         // construction, since the code is only sto
        red at the end of the
68         // constructor execution.
69
70         uint256 size;
71         // solhint-disable-next-line no-inline-asse
        mbly
72         assembly { size := extcodesize(account) }
73         return size > 0;
74     }
75
76     function functionCall(
77         address target,
78         bytes memory data,
79         string memory errorMessage
80     ) internal returns (bytes memory) {
81         return _functionCallWithValue(target, data,
            0, errorMessage);
82     }
83
84     function _functionCallWithValue(
85         address target,
86         bytes memory data,
87         uint256 weiValue,
88         string memory errorMessage
89     ) private returns (bytes memory) {
90         require(isContract(target), "Address: call
            to non-contract");
91
92         // solhint-disable-next-line avoid-low-leve
            l-calls
93         (bool success, bytes memory returndata) = t
            arget.call{ value: weiValue }(data);
94         if (success) {
95             return returndata;
96         } else {
97             if (returndata.length > 0) {
98                 // solhint-disable-next-line no-inl
                    ine-assembly
99                 assembly {
100                     let returndata_size := mload(re
                        turndata)
101                     revert(add(32, returndata), ret
                        urndata_size)
102                 }
103             } else {
104                 revert(errorMessage);
105             }
106         }
107     }
108

```

```

53     function sub(int256 x, int256 y) internal pure
        returns (int256 z) {
54         require((z = x - y) <= x == (y >= 0));
55     }
56
57     function div(uint256 x, uint256 y) internal pur
        e returns(uint256 z){
58         require(y > 0);
59         z=x/y;
60     }
61 }
62
63 library Address {
64
65     function isContract(address account) internal vie
        w returns (bool) {
66         // This method relies in extcodesize, which
        returns 0 for contracts in
67         // construction, since the code is only sto
        red at the end of the
68         // constructor execution.
69
70         uint256 size;
71         // solhint-disable-next-line no-inline-asse
        mbly
72         assembly { size := extcodesize(account) }
73         return size > 0;
74     }
75
76     function functionCall(
77         address target,
78         bytes memory data,
79         string memory errorMessage
80     ) internal returns (bytes memory) {
81         return _functionCallWithValue(target, data,
            0, errorMessage);
82     }
83
84     function _functionCallWithValue(
85         address target,
86         bytes memory data,
87         uint256 weiValue,
88         string memory errorMessage
89     ) private returns (bytes memory) {
90         require(isContract(target), "Address: call
            to non-contract");
91
92         // solhint-disable-next-line avoid-low-leve
            l-calls
93         (bool success, bytes memory returndata) = t
            arget.call{ value: weiValue }(data);
94         if (success) {
95             return returndata;
96         } else {
97             if (returndata.length > 0) {
98                 // solhint-disable-next-line no-inl
                    ine-assembly
99                 assembly {
100                     let returndata_size := mload(re
                        turndata)
101                     revert(add(32, returndata), ret
                        urndata_size)
102                 }
103             } else {
104                 revert(errorMessage);
105             }
106         }
107     }
108

```

```

109     function _verifyCallResult(
110         bool success,
111         bytes memory returndata,
112         string memory errorMessage
113     ) private pure returns(bytes memory) {
114         if (success) {
115             return returndata;
116         } else {
117             if (returndata.length > 0) {
118                 // solhint-disable-next-line no-inl
119                 ine-assembly
120                 assembly {
121                     let returndata_size := mload(re
122                     turndata)
123                     revert(add(32, returndata), ret
124                     urndata_size)
125                 }
126             } else {
127                 revert(errorMessage);
128             }
129         }
130     }
131 }
132
133 contract OwnableData {
134     address public owner;
135     address public pendingOwner;
136 }
137
138 contract Ownable is OwnableData {
139     event OwnershipTransferred(address indexed prev
140     iousOwner, address indexed newOwner);
141
142     /// @notice `owner` defaults to msg.sender on c
143     onstruction.
144     constructor() {
145         owner = msg.sender;
146         emit OwnershipTransferred(address(0), msg.s
147         ender);
148     }
149
150     /// @notice Transfers ownership to `newOwner`.
151     Either directly or claimable by the new pending ow
152     ner.
153     /// Can only be invoked by the current `owner`.
154     /// @param newOwner Address of the new owner.
155     /// @param direct True if `newOwner` should be
156     set immediately. False if `newOwner` needs to use
157     `claimOwnership`.
158     /// @param renounce Allows the `newOwner` to be
159     `address(0)` if `direct` and `renounce` is True. Ha
160     s no effect otherwise.
161     function transferOwnership(
162         address newOwner,
163         bool direct,
164         bool renounce
165     ) public onlyOwner {
166         if (direct) {
167             // Checks
168             require(newOwner != address(0) || renou
169             nce, "Ownable: zero address");
170
171             // Effects
172             emit OwnershipTransferred(owner, newOwn
173             er);
174
175             owner = newOwner;
176             pendingOwner = address(0);
177         } else {

```

```

109     function _verifyCallResult(
110         bool success,
111         bytes memory returndata,
112         string memory errorMessage
113     ) private pure returns(bytes memory) {
114         if (success) {
115             return returndata;
116         } else {
117             if (returndata.length > 0) {
118                 // solhint-disable-next-line no-inl
119                 ine-assembly
120                 assembly {
121                     let returndata_size := mload(re
122                     turndata)
123                     revert(add(32, returndata), ret
124                     urndata_size)
125                 }
126             } else {
127                 revert(errorMessage);
128             }
129         }
130     }
131 }
132
133 contract OwnableData {
134     address public owner;
135     address public pendingOwner;
136 }
137
138 contract Ownable is OwnableData {
139     event OwnershipTransferred(address indexed prev
140     iousOwner, address indexed newOwner);
141
142     /// @notice `owner` defaults to msg.sender on c
143     onstruction.
144     constructor() {
145         owner = msg.sender;
146         emit OwnershipTransferred(address(0), msg.s
147         ender);
148     }
149
150     /// @notice Transfers ownership to `newOwner`.
151     Either directly or claimable by the new pending ow
152     ner.
153     /// Can only be invoked by the current `owner`.
154     /// @param newOwner Address of the new owner.
155     /// @param direct True if `newOwner` should be
156     set immediately. False if `newOwner` needs to use
157     `claimOwnership`.
158     /// @param renounce Allows the `newOwner` to be
159     `address(0)` if `direct` and `renounce` is True. Ha
160     s no effect otherwise.
161     function transferOwnership(
162         address newOwner,
163         bool direct,
164         bool renounce
165     ) public onlyOwner {
166         if (direct) {
167             // Checks
168             require(newOwner != address(0) || renou
169             nce, "Ownable: zero address");
170
171             // Effects
172             emit OwnershipTransferred(owner, newOwn
173             er);
174
175             owner = newOwner;
176             pendingOwner = address(0);
177         } else {

```

```

163         // Effects
164         pendingOwner = newOwner;
165     }
166 }
167
168 /// @notice Needs to be called by `pendingOwner`
169 ` to claim ownership.
170 function claimOwnership() public {
171     address _pendingOwner = pendingOwner;
172
173     // Checks
174     require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");
175
176     // Effects
177     emit OwnershipTransferred(owner, _pendingOwner);
178     owner = _pendingOwner;
179     pendingOwner = address(0);
180 }
181
182 /// @notice Only allows the `owner` to execute
183 the function.
184 modifier onlyOwner() {
185     require(msg.sender == owner, "Ownable: caller is not the owner");
186     _;
187 }
188
189 interface IERC20 {
190     function decimals() external view returns (uint8);
191     function balanceOf(address account) external view returns (uint256);
192     function transfer(address recipient, uint256 amount) external returns (bool);
193     function approve(address spender, uint256 amount) external returns (bool);
194     function totalSupply() external view returns (uint256);
195     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
196     event Transfer(address indexed from, address indexed to, uint256 value);
197     event Approval(address indexed owner, address indexed spender, uint256 value);
198 }
199
200 library SafeERC20 {
201     using LowGasSafeMath for uint256;
202     using Address for address;
203
204     function safeTransfer(IERC20 token, address to, uint256 value) internal {
205         _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
206     }
207
208     function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
209         _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
210     }
211 }

```

```

163         // Effects
164         pendingOwner = newOwner;
165     }
166 }
167
168 /// @notice Needs to be called by `pendingOwner`
169 ` to claim ownership.
170 function claimOwnership() public {
171     address _pendingOwner = pendingOwner;
172
173     // Checks
174     require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");
175
176     // Effects
177     emit OwnershipTransferred(owner, _pendingOwner);
178     owner = _pendingOwner;
179     pendingOwner = address(0);
180 }
181
182 /// @notice Only allows the `owner` to execute
183 the function.
184 modifier onlyOwner() {
185     require(msg.sender == owner, "Ownable: caller is not the owner");
186     _;
187 }
188
189 interface IERC20 {
190     function decimals() external view returns (uint8);
191     function balanceOf(address account) external view returns (uint256);
192     function transfer(address recipient, uint256 amount) external returns (bool);
193     function approve(address spender, uint256 amount) external returns (bool);
194     function totalSupply() external view returns (uint256);
195     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
196     event Transfer(address indexed from, address indexed to, uint256 value);
197     event Approval(address indexed owner, address indexed spender, uint256 value);
198 }
199
200 library SafeERC20 {
201     using LowGasSafeMath for uint256;
202     using Address for address;
203
204     function safeTransfer(IERC20 token, address to, uint256 value) internal {
205         _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
206     }
207
208     function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
209         _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
210     }
211 }

```

```

216     }
217
218     function _callOptionalReturn(IERC20 token, byte
s memory data) private {
219         bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
220         if (returndata.length > 0) { // Return data
is optional
221             // solhint-disable-next-line max-line-l
ength
222             require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
223         }
224     }
225 }
226
227 interface IERC20Mintable {
228     function mint( uint256 amount_ ) external;
229
230     function mint( address account_, uint256 amount_
) external;
231 }
232
233 interface ITIMEERC20 is IERC20Mintable, IERC20 {
234     function burnFrom(address account_, uint256 amo
unt_) external;
235 }
236
237 interface IBondCalculator {
238     function valuation( address pair_, uint amount_ )
external view returns ( uint _value );
239 }
240
241 contract TimeTreasury is Ownable {
242
243     using LowGasSafeMath for uint;
244     using LowGasSafeMath for uint32;
245     using SafeERC20 for IERC20;
246
247     event Deposit( address indexed token, uint amou
nt, uint value );
248     event Withdrawal( address indexed token, uint a
mount, uint value );
249     event CreateDebt( address indexed debtor, addre
ss indexed token, uint amount, uint value );
250     event RepayDebt( address indexed debtor, addres
s indexed token, uint amount, uint value );
251     event ReservesManaged( address indexed token, u
int amount );
252     event ReservesUpdated( uint indexed totalReserv
es );
253     event ReservesAudited( uint indexed totalReserv
es );
254     event RewardsMinted( address indexed caller, ad
dress indexed recipient, uint amount );
255     event ChangeQueued( MANAGING indexed managing,
address queued );
256     event ChangeActivated( MANAGING indexed managin
g, address activated, bool result );
257     event ChangeLimitAmount( uint256 amount );
258
259     enum MANAGING {
260         RESERVEDEPOSITOR,
261         RESERVESPENDER,
262         RESERVETOKEN,
263         RESERVEMANAGER,
264         LIQUIDITYDEPOSITOR,
265         LIQUIDITYTOKEN,

```

```

216     }
217
218     function _callOptionalReturn(IERC20 token, byte
s memory data) private {
219         bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
220         if (returndata.length > 0) { // Return data
is optional
221             // solhint-disable-next-line max-line-l
ength
222             require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
223         }
224     }
225 }
226
227 interface IERC20Mintable {
228     function mint( uint256 amount_ ) external;
229
230     function mint( address account_, uint256 amount_
) external;
231 }
232
233 interface ITIMEERC20 is IERC20Mintable, IERC20 {
234     function burnFrom(address account_, uint256 amo
unt_) external;
235 }
236
237 interface IBondCalculator {
238     function valuation( address pair_, uint amount_ )
external view returns ( uint _value );
239 }
240
241 contract TimeTreasury is Ownable {
242
243     using LowGasSafeMath for uint;
244     using LowGasSafeMath for uint32;
245     using SafeERC20 for IERC20;
246
247     event Deposit( address indexed token, uint amou
nt, uint value );
248     event Withdrawal( address indexed token, uint a
mount, uint value );
249     event CreateDebt( address indexed debtor, addre
ss indexed token, uint amount, uint value );
250     event RepayDebt( address indexed debtor, addres
s indexed token, uint amount, uint value );
251     event ReservesManaged( address indexed token, u
int amount );
252     event ReservesUpdated( uint indexed totalReserv
es );
253     event ReservesAudited( uint indexed totalReserv
es );
254     event RewardsMinted( address indexed caller, ad
dress indexed recipient, uint amount );
255     event ChangeQueued( MANAGING indexed managing,
address queued );
256     event ChangeActivated( MANAGING indexed managin
g, address activated, bool result );
257     event ChangeLimitAmount( uint256 amount );
258
259     enum MANAGING {
260         RESERVEDEPOSITOR,
261         RESERVESPENDER,
262         RESERVETOKEN,
263         RESERVEMANAGER,
264         LIQUIDITYDEPOSITOR,
265         LIQUIDITYTOKEN,

```

```

266     LIQUIDITYMANAGER,
267     DEBTOR,
268     REWARDMANAGER,
269     SOHM
270 }
271
272     ITIMEERC20 public immutable Time;
273     uint32 public immutable secondsNeededForQueue;
274
275     address[] public reserveTokens; // Push only, beware false-positives.
276     mapping( address => bool ) public isReserveToken;
277     mapping( address => uint32 ) public reserveTokenQueue; // Delays changes to mapping.
278
279     address[] public reserveDepositors; // Push only, beware false-positives. Only for viewing.
280     mapping( address => bool ) public isReserveDepositor;
281     mapping( address => uint32 ) public reserveDepositorQueue; // Delays changes to mapping.
282
283     address[] public reserveSpenders; // Push only, beware false-positives. Only for viewing.
284     mapping( address => bool ) public isReserveSpender;
285     mapping( address => uint32 ) public reserveSpenderQueue; // Delays changes to mapping.
286
287     address[] public liquidityTokens; // Push only, beware false-positives.
288     mapping( address => bool ) public isLiquidityToken;
289     mapping( address => uint32 ) public liquidityTokenQueue; // Delays changes to mapping.
290
291     address[] public liquidityDepositors; // Push only, beware false-positives. Only for viewing.
292     mapping( address => bool ) public isLiquidityDepositor;
293     mapping( address => uint32 ) public liquidityDepositorQueue; // Delays changes to mapping.
294
295     mapping( address => address ) public bondCalculator; // bond calculator for liquidity token
296
297     address[] public reserveManagers; // Push only, beware false-positives. Only for viewing.
298     mapping( address => bool ) public isReserveManager;
299     mapping( address => uint32 ) public ReserveManagerQueue; // Delays changes to mapping.
300
301     address[] public liquidityManagers; // Push only, beware false-positives. Only for viewing.
302     mapping( address => bool ) public isLiquidityManager;
303     mapping( address => uint32 ) public liquidityManagerQueue; // Delays changes to mapping.
304
305     address[] public debtors; // Push only, beware false-positives. Only for viewing.
306     mapping( address => bool ) public isDebtor;
307     mapping( address => uint32 ) public debtorQueue; // Delays changes to mapping.
308     mapping( address => uint ) public debtorBalance;
309

```

```

266     LIQUIDITYMANAGER,
267     DEBTOR,
268     REWARDMANAGER,
269     SOHM
270 }
271
272     ITIMEERC20 public immutable Time;
273     uint32 public immutable secondsNeededForQueue;
274
275     address[] public reserveTokens; // Push only, beware false-positives.
276     mapping( address => bool ) public isReserveToken;
277     mapping( address => uint32 ) public reserveTokenQueue; // Delays changes to mapping.
278
279     address[] public reserveDepositors; // Push only, beware false-positives. Only for viewing.
280     mapping( address => bool ) public isReserveDepositor;
281     mapping( address => uint32 ) public reserveDepositorQueue; // Delays changes to mapping.
282
283     address[] public reserveSpenders; // Push only, beware false-positives. Only for viewing.
284     mapping( address => bool ) public isReserveSpender;
285     mapping( address => uint32 ) public reserveSpenderQueue; // Delays changes to mapping.
286
287     address[] public liquidityTokens; // Push only, beware false-positives.
288     mapping( address => bool ) public isLiquidityToken;
289     mapping( address => uint32 ) public liquidityTokenQueue; // Delays changes to mapping.
290
291     address[] public liquidityDepositors; // Push only, beware false-positives. Only for viewing.
292     mapping( address => bool ) public isLiquidityDepositor;
293     mapping( address => uint32 ) public liquidityDepositorQueue; // Delays changes to mapping.
294
295     mapping( address => address ) public bondCalculator; // bond calculator for liquidity token
296
297     address[] public reserveManagers; // Push only, beware false-positives. Only for viewing.
298     mapping( address => bool ) public isReserveManager;
299     mapping( address => uint32 ) public ReserveManagerQueue; // Delays changes to mapping.
300
301     address[] public liquidityManagers; // Push only, beware false-positives. Only for viewing.
302     mapping( address => bool ) public isLiquidityManager;
303     mapping( address => uint32 ) public liquidityManagerQueue; // Delays changes to mapping.
304
305     address[] public debtors; // Push only, beware false-positives. Only for viewing.
306     mapping( address => bool ) public isDebtor;
307     mapping( address => uint32 ) public debtorQueue; // Delays changes to mapping.
308     mapping( address => uint ) public debtorBalance;
309

```

```

310     address[] public rewardManagers; // Push only,
        beware false-positives. Only for viewing.
311     mapping( address => bool ) public isRewardManag
er;
312     mapping( address => uint32 ) public rewardManag
erQueue; // Delays changes to mapping.
313
314     mapping( address => uint256 ) public hourlyLimi
tAmounts; // tracks amounts
315     mapping( address => uint32 ) public hourlyLimit
Queue; // Delays changes to mapping.
316
317     uint256 public limitAmount;
318
319     IERC20 public MEMORies;
320     uint public sOHMQueue; // Delays change to sOHM
address
321
322     uint public totalReserves; // Risk-free value o
f all assets
323     uint public totalDebt;
324
325     constructor (
326         address _Time,
327         address _MIM,
328         uint32 _secondsNeededForQueue,
329         uint256 _limitAmount
330     ) {
331         require( _Time != address(0) );
332         Time = ITIMEERC20(_Time);
333
334         isReserveToken[ _MIM ] = true;
335         reserveTokens.push( _MIM );
336
337         // isLiquidityToken[ _OHMDAI ] = true;
338         // liquidityTokens.push( _OHMDAI );
339
340         secondsNeededForQueue = _secondsNeededForQu
eue;
341         limitAmount = _limitAmount;
342     }
343
344     function setLimitAmount(uint amount) external o
nlyOwner {
345         limitAmount = amount;
346         emit ChangeLimitAmount(limitAmount);
347     }
348
349     /**
350     @notice allow approved address to deposit a
n asset for Time
351     @param _amount uint
352     @param _token address
353     @param _profit uint
354     @return send_ uint
355     */
356     function deposit( uint _amount, address _token,
uint _profit ) external returns ( uint send_ ) {
357         require( isReserveToken[ _token ] || isLiqu
idityToken[ _token ], "Not accepted" );
358         IERC20( _token ).safeTransferFrom( msg.send
er, address(this), _amount );
359
360         if ( isReserveToken[ _token ] ) {
361             require( isReserveDepositor[ msg.sender
], "Not approved" );
362         } else {
363             require( isLiquidityDepositor[ msg.send
er ], "Not approved" );
364         }

```

```

310     address[] public rewardManagers; // Push only,
        beware false-positives. Only for viewing.
311     mapping( address => bool ) public isRewardManag
er;
312     mapping( address => uint32 ) public rewardManag
erQueue; // Delays changes to mapping.
313
314
315     IERC20 public MEMORies;
316     uint public sOHMQueue; // Delays change to sOHM
address
317
318     uint public totalReserves; // Risk-free value o
f all assets
319     uint public totalDebt;
320
321     constructor (
322         address _Time,
323         address _MIM,
324         uint32 _secondsNeededForQueue
325     ) {
326         require( _Time != address(0) );
327         Time = ITIMEERC20(_Time);
328
329         isReserveToken[ _MIM ] = true;
330         reserveTokens.push( _MIM );
331
332         // isLiquidityToken[ _OHMDAI ] = true;
333         // liquidityTokens.push( _OHMDAI );
334
335         secondsNeededForQueue = _secondsNeededForQu
eue;
336
337
338     @notice allow approved address to deposit a
n asset for Time
339     @param _amount uint
340     @param _token address
341     @param _profit uint
342     @return send_ uint
343
344     */
345     function deposit( uint _amount, address _token,
uint _profit ) external returns ( uint send_ ) {
346         require( isReserveToken[ _token ] || isLiqu
idityToken[ _token ], "Not accepted" );
347         IERC20( _token ).safeTransferFrom( msg.send
er, address(this), _amount );
348
349         if ( isReserveToken[ _token ] ) {
350             require( isReserveDepositor[ msg.sender
], "Not approved" );
351         } else {
352             require( isLiquidityDepositor[ msg.send
er ], "Not approved" );
353         }

```



```

365
366     uint value = valueOf(_token, _amount);
367     // mint Time needed and store amount of rewa
ards for distribution
368     send_ = value.sub( _profit );
369     limitRequirements(msg.sender, send_);
370     Time.mint( msg.sender, send_ );
371
372     totalReserves = totalReserves.add( value );
373     emit ReservesUpdated( totalReserves );
374
375     emit Deposit( _token, _amount, value );
376 }
377
378 /**
379     @notice allow approved address to burn Time
for reserves
380     @param _amount uint
381     @param _token address
382     */
383     function withdraw( uint _amount, address _token
) external {
384         require( isReserveToken[ _token ], "Not acc
epted" ); // Only reserves can be used for redempti
ons
385         require( isReserveSpender[ msg.sender ], "N
ot approved" );
386
387         uint value = valueOf( _token, _amount );
388         Time.burnFrom( msg.sender, value );
389
390         totalReserves = totalReserves.sub( value );
391         emit ReservesUpdated( totalReserves );
392
393         IERC20( _token ).safeTransfer( msg.sender,
_amount );
394
395         emit Withdrawal( _token, _amount, value );
396     }
397
398 /**
399     @notice allow approved address to borrow re
serves
400     @param _amount uint
401     @param _token address
402     */
403     function incurDebt( uint _amount, address _toke
n ) external {
404         require( isDebtor[ msg.sender ], "Not appro
ved" );
405         require( isReserveToken[ _token ], "Not acc
epted" );
406
407         uint value = valueOf( _token, _amount );
408
409         uint maximumDebt = MEMORies.balanceOf( msg.
sender ); // Can only borrow against sOHM held
410         uint balance = debtorBalance[ msg.sender ];
411         uint availableDebt = maximumDebt.sub( balan
ce );
412         require( value <= availableDebt, "Exceeds d
ebt limit" );
413         limitRequirements(msg.sender, value);
414         debtorBalance[ msg.sender ] = balance.add(
value );
415         totalDebt = totalDebt.add( value );
416
417         totalReserves = totalReserves.sub( value );

```

```

353
354     uint value = valueOfToken(_token, _amount);
355     // mint Time needed and store amount of rew
ards for distribution
356     send_ = value.sub( _profit );
357
358     Time.mint( msg.sender, send_ );
359
360     totalReserves = totalReserves.add( value );
361     emit ReservesUpdated( totalReserves );
362
363     emit Deposit( _token, _amount, value );
364 }
365
366 /**
367     @notice allow approved address to burn Time
for reserves
368     @param _amount uint
369     @param _token address
370     */
371     function withdraw( uint _amount, address _token
) external {
372         require( isReserveToken[ _token ], "Not acc
epted" ); // Only reserves can be used for redempti
ons
373         require( isReserveSpender[ msg.sender ], "N
ot approved" );
374
375         uint value = valueOfToken( _token, _amount
);
376         Time.burnFrom( msg.sender, value );
377
378         totalReserves = totalReserves.sub( value );
379         emit ReservesUpdated( totalReserves );
380
381         IERC20( _token ).safeTransfer( msg.sender,
_amount );
382
383         emit Withdrawal( _token, _amount, value );
384     }
385
386 /**
387     @notice allow approved address to borrow re
serves
388     @param _amount uint
389     @param _token address
390     */
391     function incurDebt( uint _amount, address _toke
n ) external {
392         require( isDebtor[ msg.sender ], "Not appro
ved" );
393         require( isReserveToken[ _token ], "Not acc
epted" );
394
395         uint value = valueOfToken( _token, _amount
);
396         uint maximumDebt = MEMORies.balanceOf( msg.
sender ); // Can only borrow against sOHM held
397         uint balance = debtorBalance[ msg.sender ];
398         uint availableDebt = maximumDebt.sub( balan
ce );
399         require( value <= availableDebt, "Exceeds d
ebt limit" );
400         debtorBalance[ msg.sender ] = balance.add(
value );
401         totalDebt = totalDebt.add( value );
402
403         totalReserves = totalReserves.sub( value );

```



```

418         emit ReservesUpdated( totalReserves );
419
420         IERC20( _token ).safeTransfer( msg.sender,
         _amount );
421
422         emit CreateDebt( msg.sender, _token, _amount,
         value );
423     }
424
425     /**
426     @notice allow approved address to repay borrowed
         reserves with reserves
427     @param _amount uint
428     @param _token address
429     */
430     function repayDebtWithReserve( uint _amount, address
         _token ) external {
431         require( isDebtor[ msg.sender ], "Not approved" );
432         require( isReserveToken[ _token ], "Not accepted" );
433
434         IERC20( _token ).safeTransferFrom( msg.sender,
         address(this), _amount );
435
436         uint value = valueOf( _token, _amount );
437
438         debtorBalance[ msg.sender ] = debtorBalance[
         msg.sender ].sub( value );
439         totalDebt = totalDebt.sub( value );
440
441         totalReserves = totalReserves.add( value );
442         emit ReservesUpdated( totalReserves );
443         emit RepayDebt( msg.sender, _token, _amount,
         value );
444     }
445
446     /**
447     @notice allow approved address to repay borrowed
         reserves with Time
448     @param _amount uint
449     */
450     function repayDebtWithTime( uint _amount ) external {
451         require( isDebtor[ msg.sender ], "Not approved as
         debtor" );
452         require( isReserveSpender[ msg.sender ], "Not
         approved as spender" );
453
454         Time.burnFrom( msg.sender, _amount );
455
456         debtorBalance[ msg.sender ] = debtorBalance[
         msg.sender ].sub( _amount );
457         totalDebt = totalDebt.sub( _amount );
458
459         emit RepayDebt( msg.sender, address(Time),
         _amount, _amount );
460     }
461
462     /**
463     @notice allow approved address to withdraw assets
464     @param _token address
465     @param _amount uint
466     */
467     function manage( address _token, uint _amount )
         external {
468         uint value = valueOf( _token, _amount );
469         if( isLiquidityToken[ _token ] ) {

```

```

404         emit ReservesUpdated( totalReserves );
405
406         IERC20( _token ).safeTransfer( msg.sender,
         _amount );
407
408         emit CreateDebt( msg.sender, _token, _amount,
         value );
409     }
410
411     /**
412     @notice allow approved address to repay borrowed
         reserves with reserves
413     @param _amount uint
414     @param _token address
415     */
416     function repayDebtWithReserve( uint _amount, address
         _token ) external {
417         require( isDebtor[ msg.sender ], "Not approved" );
418         require( isReserveToken[ _token ], "Not accepted" );
419
420         IERC20( _token ).safeTransferFrom( msg.sender,
         address(this), _amount );
421
422         uint value = valueOfToken( _token, _amount );
423
424         debtorBalance[ msg.sender ] = debtorBalance[
         msg.sender ].sub( value );
425         totalDebt = totalDebt.sub( value );
426
427         totalReserves = totalReserves.add( value );
428         emit ReservesUpdated( totalReserves );
429         emit RepayDebt( msg.sender, _token, _amount,
         value );
430     }
431
432     /**
433     @notice allow approved address to repay borrowed
         reserves with Time
434     @param _amount uint
435     */
436     function repayDebtWithTime( uint _amount ) external {
437         require( isDebtor[ msg.sender ], "Not approved as
         debtor" );
438         require( isReserveSpender[ msg.sender ], "Not
         approved as spender" );
439
440         Time.burnFrom( msg.sender, _amount );
441
442         debtorBalance[ msg.sender ] = debtorBalance[
         msg.sender ].sub( _amount );
443         totalDebt = totalDebt.sub( _amount );
444
445         emit RepayDebt( msg.sender, address(Time),
         _amount, _amount );
446     }
447
448     /**
449     @notice allow approved address to withdraw assets
450     @param _token address
451     @param _amount uint
452     */
453     function manage( address _token, uint _amount )
         external {
454         uint value = valueOfToken( _token, _amount );
455         if( isLiquidityToken[ _token ] ) {

```

```

470         require( isLiquidityManager[ msg.sender
], "Not approved" );
471         require(value <= excessReserves());
472     } else {
473         if (isReserveToken[ _token ]) require(v
alue <= excessReserves());
474         require( isReserveManager[ msg.sender
], "Not approved" );
475     }
476
477     limitRequirements(msg.sender, value);
478     totalReserves = totalReserves.sub( value );
479     emit ReservesUpdated( totalReserves );
480
481     IERC20( _token ).safeTransfer( msg.sender,
_amount );
482
483     emit ReservesManaged( _token, _amount );
484 }
485
486 /**
487  @notice send epoch reward to staking contra
ct
488  */
489  function mintRewards( address _recipient, uint
_amount ) external {
490      require( isRewardManager[ msg.sender ], "No
t approved" );
491      require( _amount <= excessReserves(), "Insu
fficient reserves" );
492      limitRequirements(msg.sender, _amount);
493      Time.mint( _recipient, _amount );
494
495      emit RewardsMinted( msg.sender, _recipient,
_amount );
496  }
497
498  /**
499  @notice returns excess reserves not backing
tokens
500  @return uint
501  */
502  function excessReserves() public view returns (
uint ) {
503      return totalReserves.sub( Time.totalSupply
().sub( totalDebt ) );
504  }
505
506  /**
507  @notice takes inventory of all tracked asse
ts
508  @notice always consolidate to recognized re
serves before audit
509  */
510  function auditReserves() external onlyOwner {
511      uint reserves;
512      for( uint i = 0; i < reserveTokens.length;
i++ ) {
513          reserves = reserves.add (
514              valueOf( reserveTokens[ i ], IERC20
( reserveTokens[ i ] ).balanceOf( address(this) ) )
515          );
516      }
517      for( uint i = 0; i < liquidityTokens.lengt
h; i++ ) {
518          reserves = reserves.add (

```

```

456         require( isLiquidityManager[ msg.sender
], "Not approved" );
457         require(value <= excessReserves());
458     } else {
459         if (isReserveToken[ _token ]) require(v
alue <= excessReserves());
460         require( isReserveManager[ msg.sender
], "Not approved" );
461     }
462
463     totalReserves = totalReserves.sub( value );
464     emit ReservesUpdated( totalReserves );
465
466     IERC20( _token ).safeTransfer( msg.sender,
_amount );
467
468     emit ReservesManaged( _token, _amount );
469 }
470
471 /**
472  @notice send epoch reward to staking contra
ct
473  */
474  function mintRewards( address _recipient, uint
_amount ) external {
475      require( isRewardManager[ msg.sender ], "No
t approved" );
476      require( _amount <= excessReserves(), "Insu
fficient reserves" );
477      Time.mint( _recipient, _amount );
478
479      emit RewardsMinted( msg.sender, _recipient,
_amount );
480  }
481
482  /**
483  @notice returns excess reserves not backing
tokens
484  @return uint
485  */
486  function excessReserves() public view returns (
uint ) {
487      return totalReserves.sub( Time.totalSupply
().sub( totalDebt ) );
488  }
489
490  /**
491  @notice takes inventory of all tracked asse
ts
492  @notice always consolidate to recognized re
serves before audit
493  */
494  function auditReserves() external onlyOwner {
495      uint reserves;
496      for( uint i = 0; i < reserveTokens.length;
i++ ) {
497          reserves = reserves.add (
498              valueOfToken( reserveTokens[ i ], I
ERC20( reserveTokens[ i ] ).balanceOf( address(thi
s) ) )
499          );
500      }
501      for( uint i = 0; i < liquidityTokens.lengt
h; i++ ) {
502          reserves = reserves.add (

```

```

519         valueOf( liquidityTokens[ i ], IERC
20( liquidityTokens[ i ] ).balanceOf( address(this)
) )
520     );
521 }
522 totalReserves = reserves;
523 emit ReservesUpdated( reserves );
524 emit ReservesAudited( reserves );
525 }
526
527 /**
528  @notice returns Time valuation of asset
529  @param _token address
530  @param _amount uint
531  @return value_ uint
532  */
533 function valueOf( address _token, uint _amount
) public view returns ( uint value_ ) {
534     if ( isReserveToken[ _token ] ) {
535         // convert amount to match Time decimal
536         value_ = _amount.mul( 10 ** Time.decima
ls() ).div( 10 ** IERC20( _token ).decimals() );
537     } else if ( isLiquidityToken[ _token ] ) {
538         value_ = IBondCalculator( bondCalculato
r[ _token ] ).valuation( _token, _amount );
539     }
540 }
541
542 /**
543  @notice queue address to change boolean in
mapping
544  @param _managing MANAGING
545  @param _address address
546  @return bool
547  */
548 function queue( MANAGING _managing, address _ad
dress ) external onlyOwner returns ( bool ) {
549     require( _address != address(0), "IA" );
550     if ( _managing == MANAGING.RESERVEDEPOSITOR
) { // 0
551         reserveDepositorQueue[ _address ] = uin
t32(block.timestamp).add32( secondsNeededForQueue
);
552     } else if ( _managing == MANAGING.RESERVESP
ENDER ) { // 1
553         reserveSpenderQueue[ _address ] = uint3
2(block.timestamp).add32( secondsNeededForQueue );
554     } else if ( _managing == MANAGING.RESERVETO
KEN ) { // 2
555         reserveTokenQueue[ _address ] = uint32
(block.timestamp).add32( secondsNeededForQueue );
556     } else if ( _managing == MANAGING.RESERVEMA
NAGER ) { // 3
557         ReserveManagerQueue[ _address ] = uint3
2(block.timestamp).add32( secondsNeededForQueue.mul
32( 2 ) );
558     } else if ( _managing == MANAGING.LIQUIDITY
DEPOSITOR ) { // 4
559         LiquidityDepositorQueue[ _address ] = u
int32(block.timestamp).add32( secondsNeededForQueue
);
560     } else if ( _managing == MANAGING.LIQUIDITY
TOKEN ) { // 5
561         LiquidityTokenQueue[ _address ] = uint3
2(block.timestamp).add32( secondsNeededForQueue );
562     } else if ( _managing == MANAGING.LIQUIDITY
MANAGER ) { // 6

```

```

503         valueOfToken( liquidityTokens[ i ],
IERC20( liquidityTokens[ i ] ).balanceOf( address(t
his) ) )
504     );
505 }
506 totalReserves = reserves;
507 emit ReservesUpdated( reserves );
508 emit ReservesAudited( reserves );
509 }
510
511 /**
512  @notice returns Time valuation of asset
513  @param _token address
514  @param _amount uint
515  @return value_ uint
516  */
517 function valueOfToken( address _token, uint _am
ount ) public view returns ( uint value_ ) {
518     if ( isReserveToken[ _token ] ) {
519         // convert amount to match Time decimal
520         value_ = _amount.mul( 10 ** Time.decima
ls() ).div( 10 ** IERC20( _token ).decimals() );
521     } else if ( isLiquidityToken[ _token ] ) {
522         value_ = IBondCalculator( bondCalculato
r[ _token ] ).valuation( _token, _amount );
523     }
524 }
525
526 /**
527  @notice queue address to change boolean in
mapping
528  @param _managing MANAGING
529  @param _address address
530  @return bool
531  */
532 function queue( MANAGING _managing, address _ad
dress ) external onlyOwner returns ( bool ) {
533     require( _address != address(0), "IA" );
534     if ( _managing == MANAGING.RESERVEDEPOSITOR
) { // 0
535         reserveDepositorQueue[ _address ] = uin
t32(block.timestamp).add32( secondsNeededForQueue
);
536     } else if ( _managing == MANAGING.RESERVESP
ENDER ) { // 1
537         reserveSpenderQueue[ _address ] = uint3
2(block.timestamp).add32( secondsNeededForQueue );
538     } else if ( _managing == MANAGING.RESERVETO
KEN ) { // 2
539         reserveTokenQueue[ _address ] = uint32
(block.timestamp).add32( secondsNeededForQueue );
540     } else if ( _managing == MANAGING.RESERVEMA
NAGER ) { // 3
541         ReserveManagerQueue[ _address ] = uint3
2(block.timestamp).add32( secondsNeededForQueue.mul
32( 2 ) );
542     } else if ( _managing == MANAGING.LIQUIDITY
DEPOSITOR ) { // 4
543         LiquidityDepositorQueue[ _address ] = u
int32(block.timestamp).add32( secondsNeededForQueue
);
544     } else if ( _managing == MANAGING.LIQUIDITY
TOKEN ) { // 5
545         LiquidityTokenQueue[ _address ] = uint3
2(block.timestamp).add32( secondsNeededForQueue );
546     } else if ( _managing == MANAGING.LIQUIDITY
MANAGER ) { // 6

```

```

563         LiquidityManagerQueue[ _address ] = uint32(block.timestamp).add32( secondsNeededForQueue.mul32( 2 ) );
564     } else if ( _managing == MANAGING.DEBTOR )
565     { // 7
566         debtorQueue[ _address ] = uint32(block.timestamp).add32( secondsNeededForQueue );
567     } else if ( _managing == MANAGING.REWARDMANAGER ) { // 8
568         rewardManagerQueue[ _address ] = uint32(block.timestamp).add32( secondsNeededForQueue );
569     } else if ( _managing == MANAGING.SOHM ) {
570         // 9
571         SOHMQueue = uint32(block.timestamp).add32( secondsNeededForQueue );
572     } else return false;
573     emit ChangeQueued( _managing, _address );
574     return true;
575 }
576 /**
577  @notice verify queue then set boolean in mapping
578  @param _managing MANAGING
579  @param _address address
580  @param _calculator address
581  @return bool
582  */
583 function toggle(
584     MANAGING _managing,
585     address _address,
586     address _calculator
587 ) external onlyOwner returns ( bool ) {
588     require( _address != address(0), "IA" );
589     bool result;
590     if ( _managing == MANAGING.RESERVEDEPOSITOR ) { // 0
591         if ( requirements( reserveDepositorQueue, isReserveDepositor, _address ) ) {
592             reserveDepositorQueue[ _address ] = 0;
593             if( !listContains( reserveDepositors, _address ) ) {
594                 reserveDepositors.push( _address );
595             }
596         }
597         result = !isReserveDepositor[ _address ];
598         isReserveDepositor[ _address ] = result;
599     } else if ( _managing == MANAGING.RESERVESPENDER ) { // 1
600         if ( requirements( reserveSpenderQueue, isReserveSpender, _address ) ) {
601             reserveSpenderQueue[ _address ] = 0;
602             if( !listContains( reserveSpenders, _address ) ) {
603                 reserveSpenders.push( _address );
604             }
605         }
606     }
607     result = !isReserveSpender[ _address ];
608     isReserveSpender[ _address ] = result;
609 }

```

```

547         LiquidityManagerQueue[ _address ] = uint32(block.timestamp).add32( secondsNeededForQueue.mul32( 2 ) );
548     } else if ( _managing == MANAGING.DEBTOR )
549     { // 7
550         debtorQueue[ _address ] = uint32(block.timestamp).add32( secondsNeededForQueue );
551     } else if ( _managing == MANAGING.REWARDMANAGER ) { // 8
552         rewardManagerQueue[ _address ] = uint32(block.timestamp).add32( secondsNeededForQueue );
553     } else if ( _managing == MANAGING.SOHM ) {
554         // 9
555         SOHMQueue = uint32(block.timestamp).add32( secondsNeededForQueue );
556     } else return false;
557     emit ChangeQueued( _managing, _address );
558     return true;
559 }
560 /**
561  @notice verify queue then set boolean in mapping
562  @param _managing MANAGING
563  @param _address address
564  @param _calculator address
565  @return bool
566  */
567 function toggle(
568     MANAGING _managing,
569     address _address,
570     address _calculator
571 ) external onlyOwner returns ( bool ) {
572     require( _address != address(0), "IA" );
573     bool result;
574     if ( _managing == MANAGING.RESERVEDEPOSITOR ) { // 0
575         if ( requirements( reserveDepositorQueue, isReserveDepositor, _address ) ) {
576             reserveDepositorQueue[ _address ] = 0;
577             if( !listContains( reserveDepositors, _address ) ) {
578                 reserveDepositors.push( _address );
579             }
580         }
581         result = !isReserveDepositor[ _address ];
582         isReserveDepositor[ _address ] = result;
583     } else if ( _managing == MANAGING.RESERVESPENDER ) { // 1
584         if ( requirements( reserveSpenderQueue, isReserveSpender, _address ) ) {
585             reserveSpenderQueue[ _address ] = 0;
586             if( !listContains( reserveSpenders, _address ) ) {
587                 reserveSpenders.push( _address );
588             }
589         }
590     }
591     result = !isReserveSpender[ _address ];
592     isReserveSpender[ _address ] = result;
593 }

```

```

610         } else if ( _managing == MANAGING.RESERVETO
KEN ) { // 2
611             if ( requirements( reserveTokenQueue, i
sReserveToken, _address ) ) {
612                 reserveTokenQueue[ _address ] = 0;
613                 if( !listContains( reserveTokens, _
address ) && !listContains( liquidityTokens, _addre
ss ) ) {
614                     reserveTokens.push( _address );
615                 }
616             }
617             result = !isReserveToken[ _address ];
618             require(!result || !isLiquidityToken[_a
ddress], "Do not add to both types of token");
619             isReserveToken[ _address ] = result;
620
621         } else if ( _managing == MANAGING.RESERVEMA
NAGER ) { // 3
622             if ( requirements( ReserveManagerQueue,
isReserveManager, _address ) ) {
623                 reserveManagers.push( _address );
624                 ReserveManagerQueue[ _address ] =
0;
625                 if( !listContains( reserveManagers,
_address ) ) {
626                     reserveManagers.push( _address
);
627                 }
628             }
629             result = !isReserveManager[ _address ];
630             isReserveManager[ _address ] = result;
631
632         } else if ( _managing == MANAGING.LIQUIDITY
DEPOSITOR ) { // 4
633             if ( requirements( LiquidityDepositorQu
eue, isLiquidityDepositor, _address ) ) {
634                 liquidityDepositors.push( _address
);
635                 LiquidityDepositorQueue[ _address ]
= 0;
636                 if( !listContains( liquidityDeposit
ors, _address ) ) {
637                     liquidityDepositors.push( _addr
ess );
638                 }
639             }
640             result = !isLiquidityDepositor[ _adres
s ];
641             isLiquidityDepositor[ _address ] = resu
lt;
642
643         } else if ( _managing == MANAGING.LIQUIDITY
TOKEN ) { // 5
644             if ( requirements( LiquidityTokenQueue,
isLiquidityToken, _address ) ) {
645                 LiquidityTokenQueue[ _address ] =
0;
646                 if( !listContains( liquidityTokens,
_address ) && !listContains( reserveTokens, _addres
s ) ) {
647                     liquidityTokens.push( _address
);
648                 }
649             }
650             result = !isLiquidityToken[ _address ];
651             require(!result || !isReserveToken[_add
ress], "Do not add to both types of token");
652             isLiquidityToken[ _address ] = result;

```

```

594         } else if ( _managing == MANAGING.RESERVETO
KEN ) { // 2
595             if ( requirements( reserveTokenQueue, i
sReserveToken, _address ) ) {
596                 reserveTokenQueue[ _address ] = 0;
597                 if( !listContains( reserveTokens, _
address ) && !listContains( liquidityTokens, _addre
ss ) ) {
598                     reserveTokens.push( _address );
599                 }
600             }
601             result = !isReserveToken[ _address ];
602             require(!result || !isLiquidityToken[_a
ddress], "Do not add to both types of token");
603             isReserveToken[ _address ] = result;
604
605         } else if ( _managing == MANAGING.RESERVEMA
NAGER ) { // 3
606             if ( requirements( ReserveManagerQueue,
isReserveManager, _address ) ) {
607                 reserveManagers.push( _address );
608                 ReserveManagerQueue[ _address ] =
0;
609                 if( !listContains( reserveManagers,
_address ) ) {
610                     reserveManagers.push( _address
);
611                 }
612             }
613             result = !isReserveManager[ _address ];
614             isReserveManager[ _address ] = result;
615
616         } else if ( _managing == MANAGING.LIQUIDITY
DEPOSITOR ) { // 4
617             if ( requirements( LiquidityDepositorQu
eue, isLiquidityDepositor, _address ) ) {
618                 liquidityDepositors.push( _address
);
619                 LiquidityDepositorQueue[ _address ]
= 0;
620                 if( !listContains( liquidityDeposit
ors, _address ) ) {
621                     liquidityDepositors.push( _addr
ess );
622                 }
623             }
624             result = !isLiquidityDepositor[ _adres
s ];
625             isLiquidityDepositor[ _address ] = resu
lt;
626
627         } else if ( _managing == MANAGING.LIQUIDITY
TOKEN ) { // 5
628             if ( requirements( LiquidityTokenQueue,
isLiquidityToken, _address ) ) {
629                 LiquidityTokenQueue[ _address ] =
0;
630                 if( !listContains( liquidityTokens,
_address ) && !listContains( reserveTokens, _addres
s ) ) {
631                     liquidityTokens.push( _address
);
632                 }
633             }
634             result = !isLiquidityToken[ _address ];
635             require(!result || !isReserveToken[_add
ress], "Do not add to both types of token");
636             isLiquidityToken[ _address ] = result;

```

```

653         bondCalculator[ _address ] = _calculato
r;
654
655     } else if ( _managing == MANAGING.LIQUIDITY
MANAGER ) { // 6
656         if ( requirements( LiquidityManagerQueu
e, isLiquidityManager, _address ) ) {
657             LiquidityManagerQueue[ _address ] =
0;
658             if( !listContains( liquidityManager
s, _address ) ) {
659                 liquidityManagers.push( _adres
s );
660             }
661         }
662         result = !isLiquidityManager[ _address
];
663         isLiquidityManager[ _address ] = resul
t;
664
665     } else if ( _managing == MANAGING.DEBTOR )
{ // 7
666         if ( requirements( debtorQueue, isDebto
r, _address ) ) {
667             debtorQueue[ _address ] = 0;
668             if( !listContains( debtors, _addres
s ) ) {
669                 debtors.push( _address );
670             }
671         }
672         result = !isDebtor[ _address ];
673         isDebtor[ _address ] = result;
674
675     } else if ( _managing == MANAGING.REWARDMAN
AGER ) { // 8
676         if ( requirements( rewardManagerQueue,
isRewardManager, _address ) ) {
677             rewardManagerQueue[ _address ] = 0;
678             if( !listContains( rewardManagers,
_address ) ) {
679                 rewardManagers.push( _address
);
680             }
681         }
682         result = !isRewardManager[ _address ];
683         isRewardManager[ _address ] = result;
684
685     } else if ( _managing == MANAGING.SOHM ) {
// 9
686         soHMQueue = 0;
687         MEMOries = IERC20(_address);
688         result = true;
689
690     } else return false;
691
692     emit ChangeActivated( _managing, _address,
result );
693     return true;
694 }
695
696 /**
697     @notice checks requirements and returns alt
ered structs
698     @param queue_ mapping( address => uint )
699     @param status_ mapping( address => bool )
700     @param _address address
701     @return bool
702 */
703 function requirements(

```

```

637         bondCalculator[ _address ] = _calculato
r;
638
639     } else if ( _managing == MANAGING.LIQUIDITY
MANAGER ) { // 6
640         if ( requirements( LiquidityManagerQueu
e, isLiquidityManager, _address ) ) {
641             LiquidityManagerQueue[ _address ] =
0;
642             if( !listContains( liquidityManager
s, _address ) ) {
643                 liquidityManagers.push( _addres
s );
644             }
645         }
646         result = !isLiquidityManager[ _address
];
647         isLiquidityManager[ _address ] = resul
t;
648
649     } else if ( _managing == MANAGING.DEBTOR )
{ // 7
650         if ( requirements( debtorQueue, isDebto
r, _address ) ) {
651             debtorQueue[ _address ] = 0;
652             if( !listContains( debtors, _addres
s ) ) {
653                 debtors.push( _address );
654             }
655         }
656         result = !isDebtor[ _address ];
657         isDebtor[ _address ] = result;
658
659     } else if ( _managing == MANAGING.REWARDMAN
AGER ) { // 8
660         if ( requirements( rewardManagerQueue,
isRewardManager, _address ) ) {
661             rewardManagerQueue[ _address ] = 0;
662             if( !listContains( rewardManagers,
_address ) ) {
663                 rewardManagers.push( _address
);
664             }
665         }
666         result = !isRewardManager[ _address ];
667         isRewardManager[ _address ] = result;
668
669     } else if ( _managing == MANAGING.SOHM ) {
// 9
670         soHMQueue = 0;
671         MEMOries = IERC20(_address);
672         result = true;
673
674     } else return false;
675
676     emit ChangeActivated( _managing, _address,
result );
677     return true;
678 }
679
680 /**
681     @notice checks requirements and returns alt
ered structs
682     @param queue_ mapping( address => uint )
683     @param status_ mapping( address => bool )
684     @param _address address
685     @return bool
686 */
687 function requirements(

```

```

704         mapping( address => uint32 ) storage queue
705     -/
706         mapping( address => bool ) storage status_,
707         address _address
708     ) internal view returns ( bool ) {
709         if ( !status_[ _address ] ) {
710             require( queue_[ _address ] != 0, "Must
queue" );
711             require( queue_[ _address ] <= uint32(b
lock.timestamp), "Queue not expired" );
712             return true;
713         } return false;
714     }
715     /**
716     @notice checks LimitRequirements
717     @param _address address
718     @param _address value
719     */
720     function limitRequirements(
721         address _address,
722         uint256 value
723     ) internal {
724         if (block.timestamp.sub(hourlyLimitQueue[_a
ddress]) >= 1 hours)
725         {
726             hourlyLimitAmounts[_address] = limitAmo
unt;
727             hourlyLimitQueue[_address] = uint32(blo
ck.timestamp);
728         }
729         hourlyLimitAmounts[_address] = hourlyLimitA
mounts[_address].sub(value);
730     }
731
732     /**
733     @notice checks array to ensure against dupl
icate
734     @param _list address[]
735     @param _token address
736     @return bool
737     */
738     function listContains( address[] storage _list,
address _token ) internal view returns ( bool ) {
739         for( uint i = 0; i < _list.length; i++ ) {
740             if( _list[ i ] == _token ) {
741                 return true;
742             }
743         }
744         return false;
745     }
746 }

```

```

688         mapping( address => uint32 ) storage queue
689     -/
690         mapping( address => bool ) storage status_,
691         address _address
692     ) internal view returns ( bool ) {
693         if ( !status_[ _address ] ) {
694             require( queue_[ _address ] != 0, "Must
queue" );
695             require( queue_[ _address ] <= uint32(b
lock.timestamp), "Queue not expired" );
696             return true;
697         } return false;
698     }
699     /**
700     @notice checks array to ensure against dupl
icate
701     @param _list address[]
702     @param _token address
703     @return bool
704     */
705     function listContains( address[] storage _list,
address _token ) internal view returns ( bool ) {
706         for( uint i = 0; i < _list.length; i++ ) {
707             if( _list[ i ] == _token ) {
708                 return true;
709             }
710         }
711         return false;
712     }
713 }

```