

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3 pragma abicoder v2;
4
5 interface IOwnable {
6     function policy() external view returns (address);
7
8     function renounceManagement() external;
9
10    function pushManagement( address newOwner_ ) external;
11
12    function pullManagement() external;
13 }
14
15 contract OwnableData {
16     address public owner;
17     address public pendingOwner;
18 }
19
20 contract Ownable is OwnableData {
21     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
22
23     /// @notice `owner` defaults to msg.sender on construction.
24     constructor() {
25         owner = msg.sender;
26         emit OwnershipTransferred(address(0), msg.sender);
27     }
28
29     /// @notice Transfers ownership to `newOwner`.
30     /// Either directly or claimable by the new pending owner.
31     /// Can only be invoked by the current `owner`.
32     /// @param newOwner Address of the new owner.
33     /// @param direct True if `newOwner` should be set immediately. False if `newOwner` needs to use `claimOwnership`.
34     /// @param renounce Allows the `newOwner` to be `address(0)` if `direct` and `renounce` is True. Has no effect otherwise.
35     function transferOwnership(
36         address newOwner,
37         bool direct,
38         bool renounce
39     ) public onlyOwner {
40         if (direct) {
41             // Checks
42             require(newOwner != address(0) || renounce, "Ownable: zero address");
43
44             // Effects
45             emit OwnershipTransferred(owner, newOwner);
46
47             owner = newOwner;
48             pendingOwner = address(0);
49         } else {
50             // Effects
51             pendingOwner = newOwner;
52         }
53     }

```

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3 pragma abicoder v2;
4
5 interface IOwnable {
6     function policy() external view returns (address);
7
8     function renounceManagement() external;
9
10    function pushManagement( address newOwner_ ) external;
11
12    function pullManagement() external;
13 }
14
15 contract OwnableData {
16     address public owner;
17     address public pendingOwner;
18 }
19
20 contract Ownable is OwnableData {
21     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
22
23     /// @notice `owner` defaults to msg.sender on construction.
24     constructor() {
25         owner = msg.sender;
26         emit OwnershipTransferred(address(0), msg.sender);
27     }
28
29     /// @notice Transfers ownership to `newOwner`.
30     /// Either directly or claimable by the new pending owner.
31     /// Can only be invoked by the current `owner`.
32     /// @param newOwner Address of the new owner.
33     /// @param direct True if `newOwner` should be set immediately. False if `newOwner` needs to use `claimOwnership`.
34     /// @param renounce Allows the `newOwner` to be `address(0)` if `direct` and `renounce` is True. Has no effect otherwise.
35     function transferOwnership(
36         address newOwner,
37         bool direct,
38         bool renounce
39     ) public onlyOwner {
40         if (direct) {
41             // Checks
42             require(newOwner != address(0) || renounce, "Ownable: zero address");
43
44             // Effects
45             emit OwnershipTransferred(owner, newOwner);
46
47             owner = newOwner;
48             pendingOwner = address(0);
49         } else {
50             // Effects
51             pendingOwner = newOwner;
52         }
53     }

```

```

51     }
52
53     /// @notice Needs to be called by `pendingOwner`
    ` to claim ownership.
54     function claimOwnership() public {
55         address _pendingOwner = pendingOwner;
56
57         // Checks
58         require(msg.sender == _pendingOwner, "Ownab
le: caller != pending owner");
59
60         // Effects
61         emit OwnershipTransferred(owner, _pendingOw
ner);
62         owner = _pendingOwner;
63         pendingOwner = address(0);
64     }
65
66     /// @notice Only allows the `owner` to execute
    the function.
67     modifier onlyOwner() {
68         require(msg.sender == owner, "Ownable: call
er is not the owner");
69         _;
70     }
71 }
72
73 library LowGasSafeMath {
74     /// @notice Returns x + y, reverts if sum overf
    lows uint256
75     /// @param x The augend
76     /// @param y The addend
77     /// @return z The sum of x and y
78     function add(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
79         require((z = x + y) >= x);
80     }
81
82     function add32(uint32 x, uint32 y) internal pur
    e returns (uint32 z) {
83         require((z = x + y) >= x);
84     }
85
86     /// @notice Returns x - y, reverts if underflow
    s
87     /// @param x The minuend
88     /// @param y The subtrahend
89     /// @return z The difference of x and y
90     function sub(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
91         require((z = x - y) <= x);
92     }
93
94     function sub32(uint32 x, uint32 y) internal pur
    e returns (uint32 z) {
95         require((z = x - y) <= x);
96     }
97
98     /// @notice Returns x * y, reverts if overflows
99     /// @param x The multiplicand
100    /// @param y The multiplier
101    /// @return z The product of x and y
102    function mul(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
103        require(x == 0 || (z = x * y) / x == y);
104    }
105
106    /// @notice Returns x + y, reverts if overflows
    or underflows
107    /// @param x The augend

```

```

51     }
52
53     /// @notice Needs to be called by `pendingOwner`
    ` to claim ownership.
54     function claimOwnership() public {
55         address _pendingOwner = pendingOwner;
56
57         // Checks
58         require(msg.sender == _pendingOwner, "Ownab
le: caller != pending owner");
59
60         // Effects
61         emit OwnershipTransferred(owner, _pendingOw
ner);
62         owner = _pendingOwner;
63         pendingOwner = address(0);
64     }
65
66     /// @notice Only allows the `owner` to execute
    the function.
67     modifier onlyOwner() {
68         require(msg.sender == owner, "Ownable: call
er is not the owner");
69         _;
70     }
71 }
72
73 library LowGasSafeMath {
74     /// @notice Returns x + y, reverts if sum overf
    lows uint256
75     /// @param x The augend
76     /// @param y The addend
77     /// @return z The sum of x and y
78     function add(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
79         require((z = x + y) >= x);
80     }
81
82     function add32(uint32 x, uint32 y) internal pur
    e returns (uint32 z) {
83         require((z = x + y) >= x);
84     }
85
86     /// @notice Returns x - y, reverts if underflow
    s
87     /// @param x The minuend
88     /// @param y The subtrahend
89     /// @return z The difference of x and y
90     function sub(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
91         require((z = x - y) <= x);
92     }
93
94     function sub32(uint32 x, uint32 y) internal pur
    e returns (uint32 z) {
95         require((z = x - y) <= x);
96     }
97
98     /// @notice Returns x * y, reverts if overflows
99     /// @param x The multiplicand
100    /// @param y The multiplier
101    /// @return z The product of x and y
102    function mul(uint256 x, uint256 y) internal pur
    e returns (uint256 z) {
103        require(x == 0 || (z = x * y) / x == y);
104    }
105
106    /// @notice Returns x + y, reverts if overflows
    or underflows
107    /// @param x The augend

```

```

108     /// @param y The addend
109     /// @return z The sum of x and y
110     function add(int256 x, int256 y) internal pure
        returns (int256 z) {
111         require((z = x + y) >= x == (y >= 0));
112     }
113
114     /// @notice Returns x - y, reverts if overflows
    or underflows
115     /// @param x The minuend
116     /// @param y The subtrahend
117     /// @return z The difference of x and y
118     function sub(int256 x, int256 y) internal pure
        returns (int256 z) {
119         require((z = x - y) <= x == (y >= 0));
120     }
121 }
122
123 library Address {
124
125     function isContract(address account) internal v
        iew returns (bool) {
126
127         uint256 size;
128         // solhint-disable-next-line no-inline-asse
        mbly
129         assembly { size := extcodesize(account) }
130         return size > 0;
131     }
132
133     function sendValue(address payable recipient, u
        int256 amount) internal {
134         require(address(this).balance >= amount, "A
        ddress: insufficient balance");
135
136         // solhint-disable-next-line avoid-low-leve
        l-calls, avoid-call-value
137         (bool success, ) = recipient.call{ value: a
        mount }("");
138         require(success, "Address: unable to send v
        alue, recipient may have reverted");
139     }
140
141     function functionCall(address target, bytes mem
        ory data) internal returns (bytes memory) {
142         return functionCall(target, data, "Address: l
        ow-level call failed");
143     }
144
145     function functionCall(
146         address target,
147         bytes memory data,
148         string memory errorMessage
149     ) internal returns (bytes memory) {
150         return _functionCallWithValue(target, data,
        0, errorMessage);
151     }
152
153     function functionCallWithValue(address target,
        bytes memory data, uint256 value) internal returns
        (bytes memory) {
154         return functionCallWithValue(target, data,
        value, "Address: low-level call with value faile
        d");

```

```

108     /// @param y The addend
109     /// @return z The sum of x and y
110     function add(int256 x, int256 y) internal pure
        returns (int256 z) {
111         require((z = x + y) >= x == (y >= 0));
112     }
113
114     /// @notice Returns x - y, reverts if overflows
    or underflows
115     /// @param x The minuend
116     /// @param y The subtrahend
117     /// @return z The difference of x and y
118     function sub(int256 x, int256 y) internal pure
        returns (int256 z) {
119         require((z = x - y) <= x == (y >= 0));
120     }
121
122     function div(uint256 x, uint256 y) internal pur
        e returns(uint256 z){
123         require(y > 0);
124         z=x/y;
125     }
126 }
127
128 library Address {
129
130     function isContract(address account) internal v
        iew returns (bool) {
131
132         uint256 size;
133         // solhint-disable-next-line no-inline-asse
        mbly
134         assembly { size := extcodesize(account) }
135         return size > 0;
136     }
137
138     function sendValue(address payable recipient, u
        int256 amount) internal {
139         require(address(this).balance >= amount, "A
        ddress: insufficient balance");
140
141         // solhint-disable-next-line avoid-low-leve
        l-calls, avoid-call-value
142         (bool success, ) = recipient.call{ value: a
        mount }("");
143         require(success, "Address: unable to send v
        alue, recipient may have reverted");
144     }
145
146     function functionCall(address target, bytes mem
        ory data) internal returns (bytes memory) {
147         return functionCall(target, data, "Address: l
        ow-level call failed");
148     }
149
150     function functionCall(
151         address target,
152         bytes memory data,
153         string memory errorMessage
154     ) internal returns (bytes memory) {
155         return _functionCallWithValue(target, data,
        0, errorMessage);
156     }
157
158     function functionCallWithValue(address target,
        bytes memory data, uint256 value) internal returns
        (bytes memory) {
159         return functionCallWithValue(target, data,
        value, "Address: low-level call with value faile
        d");

```

```

155     }
156
157     function functionCallWithValue(
158         address target,
159         bytes memory data,
160         uint256 value,
161         string memory errorMessage
162     ) internal returns (bytes memory) {
163         require(address(this).balance >= value, "Address: insufficient balance for call");
164         require(isContract(target), "Address: call to non-contract");
165
166         // solhint-disable-next-line avoid-low-level-calls
167         (bool success, bytes memory returndata) = target.call{ value: value }(data);
168         return _verifyCallResult(success, returndata, errorMessage);
169     }
170
171     function _functionCallWithValue(
172         address target,
173         bytes memory data,
174         uint256 weiValue,
175         string memory errorMessage
176     ) private returns (bytes memory) {
177         require(isContract(target), "Address: call to non-contract");
178
179         // solhint-disable-next-line avoid-low-level-calls
180         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
181         if (success) {
182             return returndata;
183         } else {
184             // Look for revert reason and bubble it up if present
185             if (returndata.length > 0) {
186                 // The easiest way to bubble the revert reason is using memory via assembly
187
188                 // solhint-disable-next-line no-inline-assembly
189                 assembly {
190                     let returndata_size := mload(returndata)
191                     revert(add(32, returndata), returndata_size)
192                 }
193             } else {
194                 revert(errorMessage);
195             }
196         }
197     }
198
199     function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
200         return functionStaticCall(target, data, "Address: low-level static call failed");
201     }
202
203     function functionStaticCall(
204         address target,
205         bytes memory data,
206         string memory errorMessage
207     ) internal view returns (bytes memory) {

```

```

160     }
161
162     function functionCallWithValue(
163         address target,
164         bytes memory data,
165         uint256 value,
166         string memory errorMessage
167     ) internal returns (bytes memory) {
168         require(address(this).balance >= value, "Address: insufficient balance for call");
169         require(isContract(target), "Address: call to non-contract");
170
171         // solhint-disable-next-line avoid-low-level-calls
172         (bool success, bytes memory returndata) = target.call{ value: value }(data);
173         return _verifyCallResult(success, returndata, errorMessage);
174     }
175
176     function _functionCallWithValue(
177         address target,
178         bytes memory data,
179         uint256 weiValue,
180         string memory errorMessage
181     ) private returns (bytes memory) {
182         require(isContract(target), "Address: call to non-contract");
183
184         // solhint-disable-next-line avoid-low-level-calls
185         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
186         if (success) {
187             return returndata;
188         } else {
189             // Look for revert reason and bubble it up if present
190             if (returndata.length > 0) {
191                 // The easiest way to bubble the revert reason is using memory via assembly
192
193                 // solhint-disable-next-line no-inline-assembly
194                 assembly {
195                     let returndata_size := mload(returndata)
196                     revert(add(32, returndata), returndata_size)
197                 }
198             } else {
199                 revert(errorMessage);
200             }
201         }
202     }
203
204     function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
205         return functionStaticCall(target, data, "Address: low-level static call failed");
206     }
207
208     function functionStaticCall(
209         address target,
210         bytes memory data,
211         string memory errorMessage
212     ) internal view returns (bytes memory) {

```

```

208         require(isContract(target), "Address: stati
c call to non-contract");
209
210         // solhint-disable-next-line avoid-low-leve
l-calls
211         (bool success, bytes memory returndata) = t
arget.staticcall(data);
212         return _verifyCallResult(success, returndat
a, errorMessage);
213     }
214
215     function functionDelegateCall(address target, b
ytes memory data) internal returns (bytes memory) {
216         return functionDelegateCall(target, data,
"Address: low-level delegate call failed");
217     }
218
219     function functionDelegateCall(
220         address target,
221         bytes memory data,
222         string memory errorMessage
223     ) internal returns (bytes memory) {
224         require(isContract(target), "Address: deleg
ate call to non-contract");
225
226         // solhint-disable-next-line avoid-low-leve
l-calls
227         (bool success, bytes memory returndata) = t
arget.delegatecall(data);
228         return _verifyCallResult(success, returndat
a, errorMessage);
229     }
230
231     function _verifyCallResult(
232         bool success,
233         bytes memory returndata,
234         string memory errorMessage
235     ) private pure returns (bytes memory) {
236         if (success) {
237             return returndata;
238         } else {
239             if (returndata.length > 0) {
240
241                 assembly {
242                     let returndata_size := mload(re
turndata)
243                     revert(add(32, returndata), ret
urndata_size)
244                 }
245             } else {
246                 revert(errorMessage);
247             }
248         }
249     }
250
251     function addressToString(address _address) inte
rnal pure returns (string memory) {
252         bytes32 _bytes = bytes32(uint256(_adres
s));
253         bytes memory HEX = "0123456789abcdef";
254         bytes memory _addr = new bytes(42);
255
256         _addr[0] = '0';
257         _addr[1] = 'x';
258
259         for(uint256 i = 0; i < 20; i++) {
260             _addr[2+i*2] = HEX[uint8(_bytes[i + 12]
>> 4)];

```

```

213         require(isContract(target), "Address: stati
c call to non-contract");
214
215         // solhint-disable-next-line avoid-low-leve
l-calls
216         (bool success, bytes memory returndata) = t
arget.staticcall(data);
217         return _verifyCallResult(success, returndat
a, errorMessage);
218     }
219
220     function functionDelegateCall(address target, b
ytes memory data) internal returns (bytes memory) {
221         return functionDelegateCall(target, data,
"Address: low-level delegate call failed");
222     }
223
224     function functionDelegateCall(
225         address target,
226         bytes memory data,
227         string memory errorMessage
228     ) internal returns (bytes memory) {
229         require(isContract(target), "Address: deleg
ate call to non-contract");
230
231         // solhint-disable-next-line avoid-low-leve
l-calls
232         (bool success, bytes memory returndata) = t
arget.delegatecall(data);
233         return _verifyCallResult(success, returndat
a, errorMessage);
234     }
235
236     function _verifyCallResult(
237         bool success,
238         bytes memory returndata,
239         string memory errorMessage
240     ) private pure returns (bytes memory) {
241         if (success) {
242             return returndata;
243         } else {
244             if (returndata.length > 0) {
245
246                 assembly {
247                     let returndata_size := mload(re
turndata)
248                     revert(add(32, returndata), ret
urndata_size)
249                 }
250             } else {
251                 revert(errorMessage);
252             }
253         }
254     }
255
256     function addressToString(address _address) inte
rnal pure returns (string memory) {
257         bytes32 _bytes = bytes32(uint256(_adres
s));
258         bytes memory HEX = "0123456789abcdef";
259         bytes memory _addr = new bytes(42);
260
261         _addr[0] = '0';
262         _addr[1] = 'x';
263
264         for(uint256 i = 0; i < 20; i++) {
265             _addr[2+i*2] = HEX[uint8(_bytes[i + 12]
>> 4)];

```

```

261         _addr[3+i*2] = HEX[uint8(_bytes[i + 12]
262         & 0x0f)];
263     }
264     return string(_addr);
265 }
266 }
267 }
268 interface IERC20 {
269     function decimals() external view returns (uint
270     8);
271     function totalSupply() external view returns (u
272     int256);
273     function balanceOf(address account) external vi
274     ew returns (uint256);
275     function transfer(address recipient, uint256 am
276     ount) external returns (bool);
277     function allowance(address owner, address spend
278     er) external view returns (uint256);
279     function approve(address spender, uint256 amoun
280     t) external returns (bool);
281     function transferFrom(address sender, address r
282     ecipient, uint256 amount) external returns (bool);
283     event Transfer(address indexed from, address in
284     dexed to, uint256 value);
285     event Approval(address indexed owner, address i
286     ndexed spender, uint256 value);
287 }
288 library SafeERC20 {
289     using LowGasSafeMath for uint256;
290     using Address for address;
291 }
292     function safeTransfer(IERC20 token, address to,
293     uint256 value) internal {
294         _callOptionalReturn(token, abi.encodeWithSe
295         lector(token.transfer.selector, to, value));
296     }
297     function safeTransferFrom(IERC20 token, address
298     from, address to, uint256 value) internal {
299         _callOptionalReturn(token, abi.encodeWithSe
300         lector(token.transferFrom.selector, from, to, valu
301         e));
302     }
303     function safeApprove(IERC20 token, address spen
304     der, uint256 value) internal {
305         require((value == 0) || (token.allowance(ad
306         dress(this), spender) == 0),
307         "SafeERC20: approve from non-zero to no
308         n-zero allowance"
309         );
310         _callOptionalReturn(token, abi.encodeWithSe
311         lector(token.approve.selector, spender, value));
312     }
313     function safeIncreaseAllowance(IERC20 token, ad
314     dress spender, uint256 value) internal {
315         uint256 newAllowance = token.allowance(addr
316         ess(this), spender).add(value);

```

```

266         _addr[3+i*2] = HEX[uint8(_bytes[i + 12]
267         & 0x0f)];
268     }
269     return string(_addr);
270 }
271 }
272 }
273 interface IERC20 {
274     function decimals() external view returns (uint
275     8);
276     function totalSupply() external view returns (u
277     int256);
278     function balanceOf(address account) external vi
279     ew returns (uint256);
280     function transfer(address recipient, uint256 am
281     ount) external returns (bool);
282     function allowance(address owner, address spend
283     er) external view returns (uint256);
284     function approve(address spender, uint256 amoun
285     t) external returns (bool);
286     function transferFrom(address sender, address r
287     ecipient, uint256 amount) external returns (bool);
288     event Transfer(address indexed from, address in
289     dexed to, uint256 value);
290     event Approval(address indexed owner, address i
291     ndexed spender, uint256 value);
292 }
293 library SafeERC20 {
294     using LowGasSafeMath for uint256;
295     using Address for address;
296 }
297     function safeTransfer(IERC20 token, address to,
298     uint256 value) internal {
299         _callOptionalReturn(token, abi.encodeWithSe
300         lector(token.transfer.selector, to, value));
301     }
302     function safeTransferFrom(IERC20 token, address
303     from, address to, uint256 value) internal {
304         _callOptionalReturn(token, abi.encodeWithSe
305         lector(token.transferFrom.selector, from, to, valu
306         e));
307     }
308     function safeApprove(IERC20 token, address spen
309     der, uint256 value) internal {
310         require((value == 0) || (token.allowance(ad
311         dress(this), spender) == 0),
312         "SafeERC20: approve from non-zero to no
313         n-zero allowance"
314         );
315         _callOptionalReturn(token, abi.encodeWithSe
316         lector(token.approve.selector, spender, value));
317     }
318     function safeIncreaseAllowance(IERC20 token, ad
319     dress spender, uint256 value) internal {
320         uint256 newAllowance = token.allowance(addr
321         ess(this), spender).add(value);

```

```

310     _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
311 }
312
313 function safeDecreaseAllowance(IERC20 token, ad
dress spender, uint256 value) internal {
314     uint256 newAllowance = token.allowance(addr
ess(this), spender)
315         .sub(value);
316     _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
317 }
318
319 function _callOptionalReturn(IERC20 token, byte
s memory data) private {
320
321     bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
322     if (returndata.length > 0) { // Return data
is optional
323         // solhint-disable-next-line max-line-l
ength
324         require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
325     }
326 }
327 }
328
329 library FullMath {
330     function fullMul(uint256 x, uint256 y) private
pure returns (uint256 l, uint256 h) {
331         uint256 mm = mulmod(x, y, uint256(-1));
332         l = x * y;
333         h = mm - l;
334         if (mm < l) h -= 1;
335     }
336
337     function fullDiv(
338         uint256 l,
339         uint256 h,
340         uint256 d
341     ) private pure returns (uint256) {
342         uint256 pow2 = d & -d;
343         d /= pow2;
344         l /= pow2;
345         l += h * ((-pow2) / pow2 + 1);
346         uint256 r = 1;
347         r *= 2 - d * r;
348         r *= 2 - d * r;
349         r *= 2 - d * r;
350         r *= 2 - d * r;
351         r *= 2 - d * r;
352         r *= 2 - d * r;
353         r *= 2 - d * r;
354         r *= 2 - d * r;
355         return l * r;
356     }
357
358     function mulDiv(
359         uint256 x,
360         uint256 y,
361         uint256 d
362     ) internal pure returns (uint256) {
363         (uint256 l, uint256 h) = fullMul(x, y);
364         uint256 mm = mulmod(x, y, d);
365         if (mm > l) h -= 1;

```

```

315     _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
316 }
317
318 function safeDecreaseAllowance(IERC20 token, ad
dress spender, uint256 value) internal {
319     uint256 newAllowance = token.allowance(addr
ess(this), spender)
320         .sub(value);
321     _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
322 }
323
324 function _callOptionalReturn(IERC20 token, byte
s memory data) private {
325
326     bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
327     if (returndata.length > 0) { // Return data
is optional
328         // solhint-disable-next-line max-line-l
ength
329         require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
330     }
331 }
332 }
333
334 library FullMath {
335     function fullMul(uint256 x, uint256 y) private
pure returns (uint256 l, uint256 h) {
336         uint256 mm = mulmod(x, y, uint256(-1));
337         l = x * y;
338         h = mm - l;
339         if (mm < l) h -= 1;
340     }
341
342     function fullDiv(
343         uint256 l,
344         uint256 h,
345         uint256 d
346     ) private pure returns (uint256) {
347         uint256 pow2 = d & -d;
348         d /= pow2;
349         l /= pow2;
350         l += h * ((-pow2) / pow2 + 1);
351         uint256 r = 1;
352         r *= 2 - d * r;
353         r *= 2 - d * r;
354         r *= 2 - d * r;
355         r *= 2 - d * r;
356         r *= 2 - d * r;
357         r *= 2 - d * r;
358         r *= 2 - d * r;
359         r *= 2 - d * r;
360         return l * r;
361     }
362
363     function mulDiv(
364         uint256 x,
365         uint256 y,
366         uint256 d
367     ) internal pure returns (uint256) {
368         (uint256 l, uint256 h) = fullMul(x, y);
369         uint256 mm = mulmod(x, y, d);
370         if (mm > l) h -= 1;

```



```

418     function valuation( address _LP, uint _amount )
external view returns ( uint );
419     function markdown( address _LP ) external view
returns ( uint );
420 }
421
422 interface ISTaking {
423     function stake( uint _amount, address _recipien
t ) external returns ( bool );
424 }
425
426 interface ISTakingHelper {
427     function stake( uint _amount, address _recipien
t ) external;
428 }
429
430 contract TimeBondDepository is Ownable {
431
432     using FixedPoint for *;
433     using SafeERC20 for IERC20;
434     using LowGasSafeMath for uint;
435     using LowGasSafeMath for uint32;
436
437
438
439
440     /* ===== EVENTS ===== */
441
442     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUS
D );
443     event BondRedeemed( address indexed recipient,
uint payout, uint remaining );
444     event BondPriceChanged( uint indexed priceInUS
D, uint indexed internalPrice, uint indexed debtRat
io );
445     event ControlVariableAdjustment( uint initialBC
V, uint newBCV, uint adjustment, bool addition );
446     event InitTerms( Terms terms);
447     event LogSetTerms(PARAMETER param, uint value);
448     event LogSetAdjustment( Adjust adjust);
449     event LogSetStaking( address indexed stakingCon
tract, bool isHelper);
450     event LogRecoverLostToken( address indexed toke
nToRecover, uint amount);
451
452
453
454     /* ===== STATE VARIABLES ===== */
455
456     IERC20 public immutable Time; // token given as
payment for bond
457     IERC20 public immutable principle; // token use
d to create bond
458     ITreasury public immutable treasury; // mints T
ime when receives principle
459     address public immutable DAO; // receives profi
t share from bond
460
461     bool public immutable isLiquidityBond; // LP an
d Reserve bonds are treated slightly different
462     IBondCalculator public immutable bondCalculato
r; // calculates value of LP tokens
463
464     ISTaking public staking; // to auto-stake payou
t
465     ISTakingHelper public stakingHelper; // to stak
e and claim if no staking warmup
466     bool public useHelper;
467

```

```

423     function valuation( address _LP, uint _amount )
external view returns ( uint );
424     function markdown( address _LP ) external view
returns ( uint );
425 }
426
427 interface ISTaking {
428     function stake( uint _amount, address _recipien
t ) external returns ( bool );
429 }
430
431 interface ISTakingHelper {
432     function stake( uint _amount, address _recipien
t ) external;
433 }
434
435 contract MaiaBondDepository is Ownable {
436
437     using FixedPoint for *;
438     using SafeERC20 for IERC20;
439     using LowGasSafeMath for uint;
440     using LowGasSafeMath for uint32;
441
442
443
444
445     /* ===== EVENTS ===== */
446
447     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUS
D );
448     event BondRedeemed( address indexed recipient,
uint payout, uint remaining );
449     event BondPriceChanged( uint indexed priceInUS
D, uint indexed internalPrice, uint indexed debtRat
io );
450     event ControlVariableAdjustment( uint initialBC
V, uint newBCV, uint adjustment, bool addition );
451     event InitTerms( Terms terms);
452     event LogSetTerms(PARAMETER param, uint value);
453     event LogSetAdjustment( Adjust adjust);
454     event LogSetStaking( address indexed stakingCon
tract, bool isHelper);
455     event LogRecoverLostToken( address indexed toke
nToRecover, uint amount);
456
457
458
459     /* ===== STATE VARIABLES ===== */
460
461     IERC20 public immutable Time; // token given as
payment for bond
462     IERC20 public immutable principle; // token use
d to create bond
463     ITreasury public immutable treasury; // mints T
ime when receives principle
464     address public immutable DAO; // receives profi
t share from bond
465
466     bool public immutable isLiquidityBond; // LP an
d Reserve bonds are treated slightly different
467     IBondCalculator public immutable bondCalculato
r; // calculates value of LP tokens
468
469     ISTaking public staking; // to auto-stake payou
t
470     ISTakingHelper public stakingHelper; // to stak
e and claim if no staking warmup
471     bool public useHelper;
472

```

```

468     Terms public terms; // stores terms for new bonds
469     Adjust public adjustment; // stores adjustment to BCV data
470
471     mapping( address => Bond ) public bondInfo; // stores bond information for depositors
472
473     uint public totalDebt; // total value of outstanding bonds; used for pricing
474     uint32 public lastDecay; // reference time for debt decay
475
476     mapping (address => bool) public allowedZapper
477     s;
478
479     /* ===== STRUCTS ===== */
480
481     // Info for creating new bonds
482     struct Terms {
483         uint controlVariable; // scaling variable for price
484         uint minimumPrice; // vs principle value
485         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
486         uint fee; // as % of bond payout, in hundredths. ( 500 = 5% = 0.05 for every 1 paid)
487         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt
488         uint32 vestingTerm; // in seconds
489     }
490
491     // Info for bond holder
492     struct Bond {
493         uint payout; // Time remaining to be paid
494         uint pricePaid; // In DAI, for front end viewing
495         uint32 lastTime; // Last interaction
496         uint32 vesting; // Seconds left to vest
497     }
498
499     // Info for incremental adjustments to control variable
500     struct Adjust {
501         bool add; // addition or subtraction
502         uint rate; // increment
503         uint target; // BCV when adjustment finished
504         uint32 buffer; // minimum length (in seconds) between adjustments
505         uint32 lastTime; // time when last adjustment made
506     }
507
508     /* ===== INITIALIZATION ===== */
509
510     constructor (
511         address _Time,
512         address _principle,
513         address _treasury,
514         address _DAO,
515         address _bondCalculator
516     ) {
517         require( _Time != address(0) );
518         Time = IERC20(_Time);

```

```

473     Terms public terms; // stores terms for new bonds
474     Adjust public adjustment; // stores adjustment to BCV data
475
476     mapping( address => Bond ) public bondInfo; // stores bond information for depositors
477
478     uint public totalDebt; // total value of outstanding bonds; used for pricing
479     uint32 public lastDecay; // reference time for debt decay
480
481     mapping (address => bool) public allowedZapper
482     s;
483
484     /* ===== STRUCTS ===== */
485
486     // Info for creating new bonds
487     struct Terms {
488         uint controlVariable; // scaling variable for price
489         uint minimumPrice; // vs principle value
490         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
491         uint fee; // as % of bond payout, in hundredths. ( 500 = 5% = 0.05 for every 1 paid)
492         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt
493         uint32 vestingTerm; // in seconds
494     }
495
496     // Info for bond holder
497     struct Bond {
498         uint payout; // Time remaining to be paid
499         uint pricePaid; // In DAI, for front end viewing
500         uint32 lastTime; // Last interaction
501         uint32 vesting; // Seconds left to vest
502     }
503
504     // Info for incremental adjustments to control variable
505     struct Adjust {
506         bool add; // addition or subtraction
507         uint rate; // increment
508         uint target; // BCV when adjustment finished
509         uint32 buffer; // minimum length (in seconds) between adjustments
510         uint32 lastTime; // time when last adjustment made
511     }
512
513     /* ===== INITIALIZATION ===== */
514
515     constructor (
516         address _Time,
517         address _principle,
518         address _treasury,
519         address _DAO,
520         address _bondCalculator
521     ) {
522         require( _Time != address(0) );
523         Time = IERC20(_Time);

```

```

524     require( _principle != address(0) );
525     principle = IERC20(_principle);
526     require( _treasury != address(0) );
527     treasury = ITreasury(_treasury);
528     require( _DAO != address(0) );
529     DAO = _DAO;
530     // bondCalculator should be address(0) if not LP bond
531     bondCalculator = IBondCalculator(_bondCalculator);
532     isLiquidityBond = ( _bondCalculator != address(0) );
533 }
534
535 /**
536  * @notice initializes bond parameters
537  * @param _controlVariable uint
538  * @param _vestingTerm uint32
539  * @param _minimumPrice uint
540  * @param _maxPayout uint
541  * @param _fee uint
542  * @param _maxDebt uint
543  */
544 function initializeBondTerms(
545     uint _controlVariable,
546     uint _minimumPrice,
547     uint _maxPayout,
548     uint _fee,
549     uint _maxDebt,
550     uint32 _vestingTerm
551 ) external onlyOwner() {
552     require( terms.controlVariable == 0, "Bonds
553 must be initialized from 0" );
554     require( _controlVariable >= 40, "Can lock
555 adjustment" );
556     require( _maxPayout <= 1000, "Payout cannot
557 be above 1 percent" );
558     require( _vestingTerm >= 129600, "Vesting must
559 be longer than 36 hours" );
560     require( _fee <= 10000, "DAO fee cannot exceed
561 payout" );
562     terms = Terms ({
563         controlVariable: _controlVariable,
564         minimumPrice: _minimumPrice,
565         maxPayout: _maxPayout,
566         fee: _fee,
567         maxDebt: _maxDebt,
568         vestingTerm: _vestingTerm
569     });
570     lastDecay = uint32(block.timestamp);
571     emit InitTerms(terms);
572 }
573
574 /* ===== POLICY FUNCTIONS ===== */
575
576 enum PARAMETER { VESTING, PAYOUT, FEE, DEBT, MINPRICE }
577
578 /**
579  * @notice set parameters for new bonds
580  * @param _parameter PARAMETER
581  * @param _input uint
582  */
583 function setBondTerms ( PARAMETER _parameter, uint
584 _input ) external onlyOwner() {

```

```

529     require( _principle != address(0) );
530     principle = IERC20(_principle);
531     require( _treasury != address(0) );
532     treasury = ITreasury(_treasury);
533     require( _DAO != address(0) );
534     DAO = _DAO;
535     // bondCalculator should be address(0) if not LP bond
536     bondCalculator = IBondCalculator(_bondCalculator);
537     isLiquidityBond = ( _bondCalculator != address(0) );
538 }
539
540 /**
541  * @notice initializes bond parameters
542  * @param _controlVariable uint
543  * @param _vestingTerm uint32
544  * @param _minimumPrice uint
545  * @param _maxPayout uint
546  * @param _fee uint
547  * @param _maxDebt uint
548  */
549 function initializeBondTerms(
550     uint _controlVariable,
551     uint _minimumPrice,
552     uint _maxPayout,
553     uint _fee,
554     uint _maxDebt,
555     uint32 _vestingTerm
556 ) external onlyOwner() {
557     require( terms.controlVariable == 0, "Bonds
558 must be initialized from 0" );
559     require( _controlVariable >= 40, "Can lock
560 adjustment" );
561     require( _maxPayout <= 2000, "Payout cannot
562 be above 1 percent" );
563     require( _vestingTerm >= 129600, "Vesting must
564 be longer than 36 hours" );
565     require( _fee <= 10000, "DAO fee cannot exceed
566 payout" );
567     terms = Terms ({
568         controlVariable: _controlVariable,
569         minimumPrice: _minimumPrice,
570         maxPayout: _maxPayout,
571         fee: _fee,
572         maxDebt: _maxDebt,
573         vestingTerm: _vestingTerm
574     });
575     lastDecay = uint32(block.timestamp);
576     emit InitTerms(terms);
577 }
578
579 /* ===== POLICY FUNCTIONS ===== */
580
581 enum PARAMETER { VESTING, PAYOUT, FEE, DEBT, MINPRICE }
582
583 /**
584  * @notice set parameters for new bonds
585  * @param _parameter PARAMETER
586  * @param _input uint
587  */
588 function setBondTerms ( PARAMETER _parameter, uint
589 _input ) external onlyOwner() {

```

```

581         if ( _parameter == PARAMETER.VESTING ) { //
0
582             require( _input >= 129600, "Vesting must
be longer than 36 hours" );
583             terms.vestingTerm = uint32(_input);
584         } else if ( _parameter == PARAMETER.PAYOUT
) { // 1
585             require( _input <= 1000, "Payout cannot
be above 1 percent" );
586             terms.maxPayout = _input;
587         } else if ( _parameter == PARAMETER.FEE ) {
// 2
588             require( _input <= 10000, "DAO fee cannot
exceed payout" );
589             terms.fee = _input;
590         } else if ( _parameter == PARAMETER.DEBT )
{ // 3
591             terms.maxDebt = _input;
592         } else if ( _parameter == PARAMETER.MINPRICE
) { // 4
593             terms.minimumPrice = _input;
594         }
595         emit LogSetTerms(_parameter, _input);
596     }
597
598     /**
599     * @notice set control variable adjustment
600     * @param _addition bool
601     * @param _increment uint
602     * @param _target uint
603     * @param _buffer uint
604     */
605     function setAdjustment (
606         bool _addition,
607         uint _increment,
608         uint _target,
609         uint32 _buffer
610     ) external onlyOwner() {
611         require( _increment <= terms.controlVariable
e.mul( 25 ) / 1000 , "Increment too large" );
612         require(_target >= 40, "Next Adjustment could
be locked");
613         adjustment = Adjust({
614             add: _addition,
615             rate: _increment,
616             target: _target,
617             buffer: _buffer,
618             lastTime: uint32(block.timestamp)
619         });
620         emit LogSetAdjustment(adjustment);
621     }
622
623     /**
624     * @notice set contract for auto stake
625     * @param _staking address
626     * @param _helper bool
627     */
628     function setStaking( address _staking, bool _hel
per ) external onlyOwner() {
629         require( _staking != address(0), "IA" );
630         if ( _helper ) {
631             useHelper = true;
632             stakingHelper = IStakingHelper(_staking
g);
633         } else {
634             useHelper = false;
635             staking = IStaking(_staking);
636         }

```

```

586         if ( _parameter == PARAMETER.VESTING ) { //
0
587             require( _input >= 129600, "Vesting must
be longer than 36 hours" );
588             terms.vestingTerm = uint32(_input);
589         } else if ( _parameter == PARAMETER.PAYOUT
) { // 1
590             require( _input <= 2000, "Payout cannot
be above 1 percent" );
591             terms.maxPayout = _input;
592         } else if ( _parameter == PARAMETER.FEE ) {
// 2
593             require( _input <= 10000, "DAO fee cannot
exceed payout" );
594             terms.fee = _input;
595         } else if ( _parameter == PARAMETER.DEBT )
{ // 3
596             terms.maxDebt = _input;
597         } else if ( _parameter == PARAMETER.MINPRICE
) { // 4
598             terms.minimumPrice = _input;
599         }
600         emit LogSetTerms(_parameter, _input);
601     }
602
603     /**
604     * @notice set control variable adjustment
605     * @param _addition bool
606     * @param _increment uint
607     * @param _target uint
608     * @param _buffer uint
609     */
610     function setAdjustment (
611         bool _addition,
612         uint _increment,
613         uint _target,
614         uint32 _buffer
615     ) external onlyOwner() {
616         require( _increment <= terms.controlVariable
e.mul( 25 ) / 1000 , "Increment too large" );
617         require(_target >= 40, "Next Adjustment could
be locked");
618         adjustment = Adjust({
619             add: _addition,
620             rate: _increment,
621             target: _target,
622             buffer: _buffer,
623             lastTime: uint32(block.timestamp)
624         });
625         emit LogSetAdjustment(adjustment);
626     }
627
628     /**
629     * @notice set contract for auto stake
630     * @param _staking address
631     * @param _helper bool
632     */
633     function setStaking( address _staking, bool _hel
per ) external onlyOwner() {
634         require( _staking != address(0), "IA" );
635         if ( _helper ) {
636             useHelper = true;
637             stakingHelper = IStakingHelper(_staking
g);
638         } else {
639             useHelper = false;
640             staking = IStaking(_staking);
641         }

```

```

637         emit LogSetStaking(_staking, _helper);
638     }
639
640     function allowZapper(address zapper) external onlyOwner {
641         require(zapper != address(0), "ZNA");
642
643         allowedZappers[zapper] = true;
644     }
645
646     function removeZapper(address zapper) external onlyOwner {
647
648         allowedZappers[zapper] = false;
649     }
650
651
652
653
654     /* ===== USER FUNCTIONS ===== */
655
656     /**
657      * @notice deposit bond
658      * @param _amount uint
659      * @param _maxPrice uint
660      * @param _depositor address
661      * @return uint
662      */
663     function deposit(
664         uint _amount,
665         uint _maxPrice,
666         address _depositor
667     ) external returns ( uint ) {
668         require( _depositor != address(0), "Invalid
address" );
669         require(msg.sender == _depositor || allowed
Zappers[msg.sender], "LFNA");
670         decayDebt();
671
672
673         uint priceInUSD = bondPriceInUSD(); // Stor
ed in bond info
674         uint nativePrice = _bondPrice();
675
676         require( _maxPrice >= nativePrice, "Slippag
e limit: more than max price" ); // slippage protec
tion
677
678         uint value = treasury.valueOf( address(prin
ciple), _amount );
679         uint payout = payoutFor( value ); // payout
to bonder is computed
680         require( totalDebt.add(value) <= terms.maxD
ebt, "Max capacity reached" );
681         require( payout >= 10000000, "Bond too smal
l" ); // must be > 0.01 Time ( underflow protection
)
682         require( payout <= maxPayout(), "Bond too l
arge"); // size protection because there is no slip
page
683
684         // profits are calculated
685         uint fee = payout.mul( terms.fee )/ 10000 ;
686
687         uint profit = value.sub( payout ).sub( fee
);
688
689         uint balanceBefore = Time.balanceOf(address
(this));
690
691         /**

```

```

642         emit LogSetStaking(_staking, _helper);
643     }
644
645     function allowZapper(address zapper) external o
nlyOwner {
646         require(zapper != address(0), "ZNA");
647
648         allowedZappers[zapper] = true;
649     }
650
651     function removeZapper(address zapper) external
onlyOwner {
652
653         allowedZappers[zapper] = false;
654     }
655
656
657
658
659     /* ===== USER FUNCTIONS ===== */
660
661     /**
662      * @notice deposit bond
663      * @param _amount uint
664      * @param _maxPrice uint
665      * @param _depositor address
666      * @return uint
667      */
668     function deposit(
669         uint _amount,
670         uint _maxPrice,
671         address _depositor
672     ) external returns ( uint ) {
673         require( _depositor != address(0), "Invalid
address" );
674         require(msg.sender == _depositor || allowed
Zappers[msg.sender], "LFNA");
675         decayDebt();
676
677
678         uint priceInUSD = bondPriceInUSD(); // Stor
ed in bond info
679         uint nativePrice = _bondPrice();
680
681         require( _maxPrice >= nativePrice, "Slippag
e limit: more than max price" ); // slippage protec
tion
682
683         uint value = treasury.valueOfToken( address
(principle), _amount );
684         uint payout = payoutFor( value ); // payout
to bonder is computed
685         require( totalDebt.add(value) <= terms.maxD
ebt, "Max capacity reached" );
686         require( payout >= 10000000, "Bond too smal
l" ); // must be > 0.01 Time ( underflow protection
)
687         require( payout <= maxPayout(), "Bond too l
arge"); // size protection because there is no slip
page
688
689         // profits are calculated
690         uint fee = (payout.mul( terms.fee )).div(10
000);
691
692         uint profit = value.sub( payout ).sub( fee
);
693
694         uint balanceBefore = Time.balanceOf(address
(this));
695
696         /**

```

```

690         principle is transferred in
691         approved and
692         deposited into the treasury, returning
        (_amount - profit) Time
693     */
694     principle.safeTransferFrom( msg.sender, add
ress(this), _amount );
695     principle.approve( address( treasury ), _am
ount );
696     treasury.deposit( _amount, address(principl
e), profit );
697
698     if ( fee != 0 ) { // fee is transferred to
dao
699         Time.safeTransfer( DAO, fee );
700     }
701     require(balanceBefore.add(profit) == Time.b
alanceOf(address(this)), "Not enough Time to cover
profit");
702     // total debt is increased
703     totalDebt = totalDebt.add( value );
704
705     // depositor info is stored
706     bondInfo[ _depositor ] = Bond({
707         payout: bondInfo[ _depositor ].payout.a
dd( payout ),
708         vesting: terms.vestingTerm,
709         lastTime: uint32(block.timestamp),
710         pricePaid: priceInUSD
711     });
712
713     // indexed events are emitted
714     emit BondCreated( _amount, payout, block.ti
mestamp.add( terms.vestingTerm ), priceInUSD );
715     emit BondPriceChanged( bondPriceInUSD(), _b
ondPrice(), debtRatio() );
716
717     adjust(); // control variable is adjusted
718     return payout;
719 }
720
721 /**
722  * @notice redeem bond for user
723  * @param _recipient address
724  * @param _stake bool
725  * @return uint
726  */
727 function redeem( address _recipient, bool _stak
e ) external returns ( uint ) {
728     require(msg.sender == _recipient, "NA");
729     Bond memory info = bondInfo[ _recipient ];
730     // (seconds since last interaction / vestin
g term remaining)
731     uint percentVested = percentVestedFor( _rec
ipient );
732
733     if ( percentVested >= 10000 ) { // if fully
vested
734         delete bondInfo[ _recipient ]; // delet
e user info
735         emit BondRedeemed( _recipient, info.pay
out, 0 ); // emit bond data
736         return stakeOrSend( _recipient, _stake,
info.payout ); // pay user everything due
737     } else { // if unfinished
738         // calculate payout vested

```

```

693         principle is transferred in
694         approved and
695         deposited into the treasury, returning
        (_amount - profit) Time
696     */
697     principle.safeTransferFrom( msg.sender, add
ress(this), _amount );
698     principle.approve( address( treasury ), _am
ount );
699     treasury.deposit( _amount, address(principl
e), profit );
700
701     if ( fee != 0 ) { // fee is transferred to
dao
702         Time.safeTransfer( DAO, fee );
703     }
704     require(balanceBefore.add(payout) == Time.b
alanceOf(address(this)), "Not enough Time to cover
profit");
705     // total debt is increased
706     totalDebt = totalDebt.add( value );
707
708     // depositor info is stored
709     bondInfo[ _depositor ] = Bond({
710         payout: bondInfo[ _depositor ].payout.a
dd( payout ),
711         vesting: terms.vestingTerm,
712         lastTime: uint32(block.timestamp),
713         pricePaid: priceInUSD
714     });
715
716     // indexed events are emitted
717     emit BondCreated( _amount, payout, block.ti
mestamp.add( terms.vestingTerm ), priceInUSD );
718     emit BondPriceChanged( bondPriceInUSD(), _b
ondPrice(), debtRatio() );
719
720     adjust(); // control variable is adjusted
721     return payout;
722 }
723
724 /**
725  * @notice redeem bond for user
726  * @param _recipient address
727  * @param _stake bool
728  * @return uint
729  */
730 function redeem( address _recipient, bool _stak
e ) external returns ( uint ) {
731     require(msg.sender == _recipient, "NA");
732     Bond memory info = bondInfo[ _recipient ];
733     // (seconds since last interaction / vestin
g term remaining)
734     uint percentVested = percentVestedFor( _rec
ipient );
735
736     if ( percentVested >= 10000 ) { // if fully
vested
737         delete bondInfo[ _recipient ]; // delet
e user info
738         emit BondRedeemed( _recipient, info.pay
out, 0 ); // emit bond data
739         return stakeOrSend( _recipient, _stake,
info.payout ); // pay user everything due
740     } else { // if unfinished
741         // calculate payout vested

```

```

740         uint payout = info.payout.mul( percentV
ested ) / 10000 ;
741         // store updated deposit info
742         bondInfo[ _recipient ] = Bond({
743             payout: info.payout.sub( payout ),
744             vesting: info.vesting.sub32( uint32
( block.timestamp ).sub32( info.lastTime ) ),
745             lastTime: uint32(block.timestamp),
746             pricePaid: info.pricePaid
747         });
748
749         emit BondRedeemed( _recipient, payout,
bondInfo[ _recipient ].payout );
750         return stakeOrSend( _recipient, _stake,
payout );
751     }
752 }
753
754
755
756
757 /* ===== INTERNAL HELPER FUNCTIONS =====
*/
758
759 /**
760  * @notice allow user to stake payout automati
cally
761  * @param _stake bool
762  * @param _amount uint
763  * @return uint
764  */
765 function stakeOrSend( address _recipient, bool
_stake, uint _amount ) internal returns ( uint ) {
766     if ( !_stake ) { // if user does not want t
o stake
767         Time.transfer( _recipient, _amount );
// send payout
768     } else { // if user wants to stake
769         if ( useHelper ) { // use if staking wa
rmup is 0
770             Time.approve( address(stakingHelpe
r), _amount );
771             stakingHelper.stake( _amount, _reci
pient );
772         } else {
773             Time.approve( address(staking), _am
ount );
774             staking.stake( _amount, _recipient
);
775         }
776     }
777     return _amount;
778 }
779
780 /**
781  * @notice makes incremental adjustment to con
trol variable
782  */
783 function adjust() internal {
784     uint timeCanAdjust = adjustment.lastTime.ad
d32( adjustment.buffer );
785     if( adjustment.rate != 0 && block.timestamp
>= timeCanAdjust ) {
786         uint initial = terms.controlVariable;
787         uint bcv = initial;
788         if ( adjustment.add ) {
789             bcv = bcv.add(adjustment.rate);
790             if ( bcv >= adjustment.target ) {
791                 adjustment.rate = 0;

```

```

743         uint payout = info.payout.mul( percentV
ested ) / 10000 ;
744         // store updated deposit info
745         bondInfo[ _recipient ] = Bond({
746             payout: info.payout.sub( payout ),
747             vesting: info.vesting.sub32( uint32
( block.timestamp ).sub32( info.lastTime ) ),
748             lastTime: uint32(block.timestamp),
749             pricePaid: info.pricePaid
750         });
751
752         emit BondRedeemed( _recipient, payout,
bondInfo[ _recipient ].payout );
753         return stakeOrSend( _recipient, _stake,
payout );
754     }
755 }
756
757
758
759
760 /* ===== INTERNAL HELPER FUNCTIONS =====
*/
761
762 /**
763  * @notice allow user to stake payout automati
cally
764  * @param _stake bool
765  * @param _amount uint
766  * @return uint
767  */
768 function stakeOrSend( address _recipient, bool
_stake, uint _amount ) internal returns ( uint ) {
769     if ( !_stake ) { // if user does not want t
o stake
770         Time.transfer( _recipient, _amount );
// send payout
771     } else { // if user wants to stake
772         if ( useHelper ) { // use if staking wa
rmup is 0
773             Time.approve( address(stakingHelpe
r), _amount );
774             stakingHelper.stake( _amount, _reci
pient );
775         } else {
776             Time.approve( address(staking), _am
ount );
777             staking.stake( _amount, _recipient
);
778         }
779     }
780     return _amount;
781 }
782
783 /**
784  * @notice makes incremental adjustment to con
trol variable
785  */
786 function adjust() internal {
787     uint timeCanAdjust = adjustment.lastTime.ad
d32( adjustment.buffer );
788     if( adjustment.rate != 0 && block.timestamp
>= timeCanAdjust ) {
789         uint initial = terms.controlVariable;
790         uint bcv = initial;
791         if ( adjustment.add ) {
792             bcv = bcv.add(adjustment.rate);
793             if ( bcv >= adjustment.target ) {
794                 adjustment.rate = 0;

```

```

792         bcv = adjustment.target;
793     }
794     } else {
795         bcv = bcv.sub(adjustment.rate);
796         if ( bcv <= adjustment.target ) {
797             adjustment.rate = 0;
798             bcv = adjustment.target;
799         }
800     }
801     terms.controlVariable = bcv;
802     adjustment.lastTime = uint32(block.time
stamp);
803     emit ControlVariableAdjustment( initia
l, bcv, adjustment.rate, adjustment.add );
804 }
805 }
806 /**
807  * @notice reduce total debt
808  */
809 function decayDebt() internal {
810     totalDebt = totalDebt.sub( debtDecay() );
811     lastDecay = uint32(block.timestamp);
812 }
813
814
815
816
817
818 /* ===== VIEW FUNCTIONS ===== */
819 /**
820  * @notice determine maximum bond size
821  * @return uint
822  */
823 function maxPayout() public view returns ( uint
) {
824     return Time.totalSupply().mul( terms.maxPay
out ) / 100000 ;
825 }
826
827 /**
828  * @notice calculate interest due for new bond
829  * @param _value uint
830  * @return uint
831  */
832 function payoutFor( uint _value ) public view r
eturns ( uint ) {
833     return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18() / 1e16 ;
834 }
835
836
837
838 /**
839  * @notice calculate current bond premium
840  * @return price_ uint
841  */
842 function bondPrice() public view returns ( uint
price_ ) {
843     price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
844     if ( price_ < terms.minimumPrice ) {
845         price_ = terms.minimumPrice;
846     }
847 }
848
849 /**
850  * @notice calculate current bond price and re
move floor if above
851  * @return price_ uint
852  */

```

```

795         bcv = adjustment.target;
796     }
797     } else {
798         bcv = bcv.sub(adjustment.rate);
799         if ( bcv <= adjustment.target ) {
800             adjustment.rate = 0;
801             bcv = adjustment.target;
802         }
803     }
804     terms.controlVariable = bcv;
805     adjustment.lastTime = uint32(block.time
stamp);
806     emit ControlVariableAdjustment( initia
l, bcv, adjustment.rate, adjustment.add );
807 }
808 }
809 /**
810  * @notice reduce total debt
811  */
812 function decayDebt() internal {
813     totalDebt = totalDebt.sub( debtDecay() );
814     lastDecay = uint32(block.timestamp);
815 }
816
817
818
819
820
821 /* ===== VIEW FUNCTIONS ===== */
822 /**
823  * @notice determine maximum bond size
824  * @return uint
825  */
826 function maxPayout() public view returns ( uint
) {
827     return Time.totalSupply().mul( terms.maxPay
out ) / 100000 ;
828 }
829
830 /**
831  * @notice calculate interest due for new bond
832  * @param _value uint
833  * @return uint
834  */
835 function payoutFor( uint _value ) public view r
eturns ( uint ) {
836     return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18() / 1e16 ;
837 }
838
839
840
841 /**
842  * @notice calculate current bond premium
843  * @return price_ uint
844  */
845 function bondPrice() public view returns ( uint
price_ ) {
846     price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
847     if ( price_ < terms.minimumPrice ) {
848         price_ = terms.minimumPrice;
849     }
850 }
851
852 /**
853  * @notice calculate current bond price and re
move floor if above
854  * @return price_ uint
855  */

```



```

853     function _bondPrice() internal returns ( uint p
      rice_ ) {
854         price_ = terms.controlVariable.mul( debtRat
      io() ).add( 10000000000 ) / 1e7;
855         if ( price_ < terms.minimumPrice ) {
856             price_ = terms.minimumPrice;
857         } else if ( terms.minimumPrice != 0 ) {
858             terms.minimumPrice = 0;
859         }
860     }
861
862     /**
863      * @notice converts bond price to DAI value
864      * @return price_ uint
865      */
866     function bondPriceInUSD() public view returns (
      uint price_ ) {
867         if( isLiquidityBond ) {
868             price_ = bondPrice().mul( bondCalculato
      r.markdown( address(principle) ) ) / 100 ;
869         } else {
870             price_ = bondPrice().mul( 10 ** princip
      le.decimals() ) / 100;
871         }
872     }
873
874
875     /**
876      * @notice calculate current ratio of debt to
      Time supply
877      * @return debtRatio_ uint
878      */
879     function debtRatio() public view returns ( uint
      debtRatio_ ) {
880         uint supply = Time.totalSupply();
881         debtRatio_ = FixedPoint.fraction(
882             currentDebt().mul( 1e9 ),
883             supply
884         ).decode112with18() / 1e18;
885     }
886
887     /**
888      * @notice debt ratio in same terms for reserv
      e or liquidity bonds
889      * @return uint
890      */
891     function standardizedDebtRatio() external view
      returns ( uint ) {
892         if ( isLiquidityBond ) {
893             return debtRatio().mul( bondCalculator.
      markdown( address(principle) ) ) / 1e9;
894         } else {
895             return debtRatio();
896         }
897     }
898
899     /**
900      * @notice calculate debt factoring in decay
901      * @return uint
902      */
903     function currentDebt() public view returns ( ui
      nt ) {
904         return totalDebt.sub( debtDecay() );
905     }
906
907     /**
908      * @notice amount to decay total debt by
909      * @return decay_ uint
910      */

```

```

856     function _bondPrice() internal returns ( uint p
      rice_ ) {
857         price_ = terms.controlVariable.mul( debtRat
      io() ).add( 10000000000 ) / 1e7;
858         if ( price_ < terms.minimumPrice ) {
859             price_ = terms.minimumPrice;
860         } else if ( terms.minimumPrice != 0 ) {
861             terms.minimumPrice = 0;
862         }
863     }
864
865     /**
866      * @notice converts bond price to DAI value
867      * @return price_ uint
868      */
869     function bondPriceInUSD() public view returns (
      uint price_ ) {
870         if( isLiquidityBond ) {
871             price_ = bondPrice().mul( bondCalculato
      r.markdown( address(principle) ) ) / 100 ;
872         } else {
873             price_ = bondPrice().mul( 10 ** princip
      le.decimals() ) / 100;
874         }
875     }
876
877
878     /**
879      * @notice calculate current ratio of debt to
      Time supply
880      * @return debtRatio_ uint
881      */
882     function debtRatio() public view returns ( uint
      debtRatio_ ) {
883         uint supply = Time.totalSupply();
884         debtRatio_ = FixedPoint.fraction(
885             currentDebt().mul( 1e9 ),
886             supply
887         ).decode112with18() / 1e18;
888     }
889
890     /**
891      * @notice debt ratio in same terms for reserv
      e or liquidity bonds
892      * @return uint
893      */
894     function standardizedDebtRatio() external view
      returns ( uint ) {
895         if ( isLiquidityBond ) {
896             return debtRatio().mul( bondCalculator.
      markdown( address(principle) ) ) / 1e9;
897         } else {
898             return debtRatio();
899         }
900     }
901
902     /**
903      * @notice calculate debt factoring in decay
904      * @return uint
905      */
906     function currentDebt() public view returns ( ui
      nt ) {
907         return totalDebt.sub( debtDecay() );
908     }
909
910     /**
911      * @notice amount to decay total debt by
912      * @return decay_ uint
913      */

```

```

911     function debtDecay() public view returns ( uint
decay_ ) {
912         uint32 timeSinceLast = uint32(block.timesta
mp).sub32( lastDecay );
913         decay_ = totalDebt.mul( timeSinceLast ) / t
erms.vestingTerm;
914         if ( decay_ > totalDebt ) {
915             decay_ = totalDebt;
916         }
917     }
918
919     /**
920     * @notice calculate how far into vesting a de
positor is
921     * @param _depositor address
922     * @return percentVested_ uint
923     */
924     function percentVestedFor( address _depositor )
public view returns ( uint percentVested_ ) {
925         Bond memory bond = bondInfo[ _depositor ];
926         uint secondsSinceLast = uint32(block.timest
amp).sub32( bond.lastTime );
927         uint vesting = bond.vesting;
928         if ( vesting > 0 ) {
929             percentVested_ = secondsSinceLast.mul(
10000 ) / vesting;
930         } else {
931             percentVested_ = 0;
932         }
933     }
934
935     /**
936     * @notice calculate amount of Time available
for claim by depositor
937     * @param _depositor address
938     * @return pendingPayout_ uint
939     */
940     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
941         uint percentVested = percentVestedFor( _dep
ositor );
942         uint payout = bondInfo[ _depositor ].payou
t;
943         if ( percentVested >= 10000 ) {
944             pendingPayout_ = payout;
945         } else {
946             pendingPayout_ = payout.mul( percentVes
ted ) / 10000;
947         }
948     }
949
950     /** ===== AUXILLIARY ===== */
951     /**
952     * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
953     * @return bool
954     */
955     function recoverLostToken(IERC20 _token ) exter
nal returns ( bool ) {
956         require( _token != Time, "NAT" );
957         require( _token != principle, "NAP" );
958         uint balance = _token.balanceOf( address(th
is));

```

```

914     function debtDecay() public view returns ( uint
decay_ ) {
915         uint32 timeSinceLast = uint32(block.timesta
mp).sub32( lastDecay );
916         decay_ = (totalDebt.mul( timeSinceLast )).d
iv(terms.vestingTerm);
917         if ( decay_ > totalDebt ) {
918             decay_ = totalDebt;
919         }
920     }
921
922     /**
923     * @notice calculate how far into vesting a de
positor is
924     * @param _depositor address
925     * @return percentVested_ uint
926     */
927     function percentVestedFor( address _depositor )
public view returns ( uint percentVested_ ) {
928         Bond memory bond = bondInfo[ _depositor ];
929         uint secondsSinceLast = uint32(block.timest
amp).sub32( bond.lastTime );
930         uint vesting = bond.vesting;
931         if ( vesting > 0 ) {
932             percentVested_ = secondsSinceLast.mul(
10000 ) / vesting;
933         } else {
934             percentVested_ = 0;
935         }
936     }
937
938     /**
939     * @notice calculate amount of Time available
for claim by depositor
940     * @param _depositor address
941     * @return pendingPayout_ uint
942     */
943     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
944         uint percentVested = percentVestedFor( _dep
ositor );
945         uint payout = bondInfo[ _depositor ].payou
t;
946         if ( percentVested >= 10000 ) {
947             pendingPayout_ = payout;
948         } else {
949             pendingPayout_ = payout.mul( percentVes
ted ) / 10000;
950         }
951     }
952
953     /** ===== AUXILLIARY ===== */
954     /**
955     * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
956     * @return bool
957     */
958     function recoverLostToken(IERC20 _token ) exter
nal returns ( bool ) {
959         require( _token != Time, "NAT" );
960         require( _token != principle, "NAP" );
961         uint balance = _token.balanceOf( address(th
is));

```

```

966     _token.safeTransfer( DAO,  balance );
967     emit LogRecoverLostToken(address(_token), b
alance);
968     return true;

```

```

969 }
970 }

```

```

969     _token.safeTransfer( DAO,  balance );
970     emit LogRecoverLostToken(address(_token), b
alance);
971     return true;
972 }
973
974     function recoverLostETH() internal {
975         if (address(this).balance > 0) safeTransfer
ETH(DAO, address(this).balance);
976     }
977
978     /// @notice Transfers ETH to the recipient addr
ess
979     /// @dev Fails with `STE`
980     /// @param to The destination of the transfer
981     /// @param value The value to be transferred
982     function safeTransferETH(address to, uint256 va
lue) internal {
983         (bool success, ) = to.call{value: value}(ne
w bytes(0));
984         require(success, 'STE');
985     }
986 }

```

