

```

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;

```

```

366         l -= mm;
367         require(h < d, 'FullMath::mulDiv: overflow');
368         return fullDiv(l, h, d);
369     }
370 }
371
372 library FixedPoint {
373     struct uq112x112 {
374         uint224 _x;
375     }
376
377     struct uq144x112 {
378         uint256 _x;
379     }
380
381     uint8 private constant RESOLUTION = 112;
382     uint256 private constant Q112 = 0x100000000000000000000000000000000;
383     uint256 private constant Q224 = 0x1000000000000000000000000000000000000000000000000000000000000000;
384     uint256 private constant LOWER_MASK = 0xffffffffffffffffffffffff; // decimal of UQ*x112 (lower 112 bits)
385
386     function decode(uq112x112 memory self) internal pure returns (uint112) {
387         return uint112(self._x >> RESOLUTION);
388     }
389
390     function decode112with18(uq112x112 memory self) internal pure returns (uint) {
391
392         return uint(self._x) / 5192296858534827;
393     }
394
395     function fraction(uint256 numerator, uint256 denominator) internal pure returns (uq112x112 memory) {
396         {
397             require(denominator > 0, 'FixedPoint::fraction: division by zero');
398             if (numerator == 0) return FixedPoint.uq112x112(0);
399
400             if (numerator <= uint144(-1)) {
401                 uint256 result = (numerator << RESOLUTION) / denominator;
402                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
403                 return uq112x112(uint224(result));
404             } else {
405                 uint256 result = FullMath.mulDiv(numerator, Q112, denominator);
406                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
407                 return uq112x112(uint224(result));
408             }
409         }
410     }
411 }

```

```

371         l -= mm;
372         require(h < d, 'FullMath::mulDiv: overflow');
373         return fullDiv(l, h, d);
374     }
375 }
376
377 library FixedPoint {
378     struct uq112x112 {
379         uint224 _x;
380     }
381
382     struct uq144x112 {
383         uint256 _x;
384     }
385
386     uint8 private constant RESOLUTION = 112;
387     uint256 private constant Q112 = 0x100000000000000000000000000000000;
388     uint256 private constant Q224 = 0x1000000000000000000000000000000000000000000000000000000000000000;
389     uint256 private constant LOWER_MASK = 0xffffffffffffffffffffffff; // decimal of UQ*x112 (lower 112 bits)
390
391     function decode(uq112x112 memory self) internal pure returns (uint112) {
392         return uint112(self._x >> RESOLUTION);
393     }
394
395     function decode112with18(uq112x112 memory self) internal pure returns (uint) {
396
397         return uint(self._x) / 5192296858534827;
398     }
399
400     function fraction(uint256 numerator, uint256 denominator) internal pure returns (uq112x112 memory) {
401         {
402             require(denominator > 0, 'FixedPoint::fraction: division by zero');
403             if (numerator == 0) return FixedPoint.uq112x112(0);
404
405             if (numerator <= uint144(-1)) {
406                 uint256 result = (numerator << RESOLUTION) / denominator;
407                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
408                 return uq112x112(uint224(result));
409             } else {
410                 uint256 result = FullMath.mulDiv(numerator, Q112, denominator);
411                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
412                 return uq112x112(uint224(result));
413             }
414         }
415     }
416
417     interface AggregatorV3Interface {
418
419         function decimals() external view returns (uint8);
420         function description() external view returns (string memory);
421         function version() external view returns (uint256);
422     }

```



```

412 interface ITreasury {
413     function deposit( uint _amount, address _token,
uint _profit ) external returns ( bool );
414     function valueOf( address _token, uint _amount
) external view returns ( uint value_ );
415 }
416
417 interface IBondCalculator {
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     using FixedPoint for *;
432     using SafeERC20 for IERC20;
433     using LowGasSafeMath for uint;
434     using LowGasSafeMath for uint32;
435
436
437
438
439
440     /* ===== EVENTS ===== */
441
442     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUSD );
423 // getRoundData and latestRoundData should both r
aise "No data present"
424 // if they do not have data to report, instead of
returning unset values
425 // which could be misinterpreted as actual report
ed values.
426 function getRoundData(uint80 _roundId)
427 external
428 view
429 returns (
430     uint80 roundId,
431     int256 answer,
432     uint256 startedAt,
433     uint256 updatedAt,
434     uint80 answeredInRound
435 );
436 function latestRoundData()
437 external
438 view
439 returns (
440     uint80 roundId,
441     int256 answer,
442     uint256 startedAt,
443     uint256 updatedAt,
444     uint80 answeredInRound
445 );
446 }
447
448 interface ITreasury {
449     function deposit( uint _amount, address _token,
uint _profit ) external returns ( uint );
450     function valueOfToken( address _token, uint _am
ount ) external view returns ( uint value_ );
451     function mintRewards( address _recipient, uint
_amount ) external;
452 }
453
454 interface IBondCalculator {
455     function valuation( address _LP, uint _amount )
external view returns ( uint );
456     function markdown( address _LP ) external view
returns ( uint );
457 }
458
459 interface ISTaking {
460     function stake( uint _amount, address _recipien
t ) external returns ( bool );
461 }
462
463 interface ISTakingHelper {
464     function stake( uint _amount, address _recipien
t ) external;
465 }
466
467 contract TimeBondDepository is Ownable {
468     using FixedPoint for *;
469     using SafeERC20 for IERC20;
470     using LowGasSafeMath for uint;
471     using LowGasSafeMath for uint32;
472
473
474
475
476
477     /* ===== EVENTS ===== */
478
479     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUSD );

```

```

443     event BondRedeemed( address indexed recipient,
      uint payout, uint remaining );
444     event BondPriceChanged( uint indexed priceInUSD,
      uint indexed internalPrice, uint indexed debtRatio );
445     event ControlVariableAdjustment( uint initialBCV,
      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 stakingContract,
      bool isHelper);
450     event LogRecoverLostToken( address indexed tokenToRecover,
      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 used to create bond
458     ITreasury public immutable treasury; // mints Time when receives principle
459     address public immutable DAO; // receives profit share from bond
460
461     bool public immutable isLiquidityBond; // LP and Reserve bonds are treated slightly different
462     IBondCalculator public immutable bondCalculator; // calculates value of LP tokens
463
464     IStaking public staking; // to auto-stake payout
465     IStakingHelper public stakingHelper; // to stake and claim if no staking warmup
466     bool public useHelper;
467
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 allowedZappers;
477
478
479
480
481     /* ===== STRUCTS ===== */
482
483     // Info for creating new bonds
484     struct Terms {
485         uint controlVariable; // scaling variable for price
486         uint minimumPrice; // vs principle value
487         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
488         uint fee; // as % of bond payout, in hundredths. ( 500 = 5% = 0.05 for every 1 paid)
489         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt

```

```

480     event BondRedeemed( address indexed recipient,
      uint payout, uint remaining );
481     event BondPriceChanged( uint indexed priceInUSD,
      uint indexed internalPrice, uint indexed debtRatio );
482     event ControlVariableAdjustment( uint initialBCV,
      uint newBCV, uint adjustment, bool addition );
483     event InitTerms( Terms terms);
484     event LogSetTerms(PARAMETER param, uint value);
485     event LogSetAdjustment( Adjust adjust);
486     event LogSetStaking( address indexed stakingContract,
      bool isHelper);
487     event LogRecoverLostToken( address indexed tokenToRecover,
      uint amount);
488
489
490
491     /* ===== STATE VARIABLES ===== */
492
493     IERC20 public immutable Time; // token given as payment for bond
494     IERC20 public immutable principle; // token used to create bond
495     ITreasury public immutable treasury; // mints Time when receives principle
496     address public immutable DAO; // receives profit share from bond
497
498     AggregatorV3Interface public priceFeed;
499
500     IStaking public staking; // to auto-stake payout
501     IStakingHelper public stakingHelper; // to stake and claim if no staking warmup
502     bool public useHelper;
503
504     Terms public terms; // stores terms for new bonds
505     Adjust public adjustment; // stores adjustment to BCV data
506
507     mapping( address => Bond ) public bondInfo; // stores bond information for depositors
508
509     uint public totalDebt; // total value of outstanding bonds; used for pricing
510     uint32 public lastDecay; // reference time for debt decay
511
512     mapping (address => bool) public allowedZappers;
513
514
515
516
517     /* ===== STRUCTS ===== */
518
519     // Info for creating new bonds
520     struct Terms {
521         uint controlVariable; // scaling variable for price
522         uint minimumPrice; // vs principle value
523         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
524         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt

```

```

490     uint32 vestingTerm; // in seconds
491 }
492
493 // Info for bond holder
494 struct Bond {
495     uint payout; // Time remaining to be paid
496     uint pricePaid; // In DAI, for front end vi
ewing
497     uint32 lastTime; // Last interaction
498     uint32 vesting; // Seconds left to vest
499 }
500
501 // Info for incremental adjustments to control
variable
502 struct Adjust {
503     bool add; // addition or subtraction
504     uint rate; // increment
505     uint target; // BCV when adjustment finishe
d
506     uint32 buffer; // minimum length (in second
s) between adjustments
507     uint32 lastTime; // time when last adjustme
nt made
508 }
509
510
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 n
ot LP bond
531     bondCalculator = IBondCalculator(_bondCalcu
lator);
532     isLiquidityBond = ( _bondCalculator != addr
ess(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

```

```

525     uint32 vestingTerm; // in seconds
526 }
527
528 // Info for bond holder
529 struct Bond {
530     uint payout; // Time remaining to be paid
531     uint pricePaid; // In DAI, for front end vi
ewing
532     uint32 lastTime; // Last interaction
533     uint32 vesting; // Seconds left to vest
534 }
535
536 // Info for incremental adjustments to control
variable
537 struct Adjust {
538     bool add; // addition or subtraction
539     uint rate; // increment
540     uint target; // BCV when adjustment finishe
d
541     uint32 buffer; // minimum length (in second
s) between adjustments
542     uint32 lastTime; // time when last adjustme
nt made
543 }
544
545
546
547
548 /* ===== INITIALIZATION ===== */
549
550 constructor (
551     address _Time,
552     address _principle,
553     address _treasury,
554     address _DAO,
555     address _feed
556 ) {
557     require( _Time != address(0) );
558     Time = IERC20(_Time);
559     require( _principle != address(0) );
560     principle = IERC20(_principle);
561     require( _treasury != address(0) );
562     treasury = ITreasury(_treasury);
563     require( _DAO != address(0) );
564     DAO = _DAO;
565     require( _feed != address(0) );
566     priceFeed = AggregatorV3Interface( _feed );
567 }
568
569 /**
570  * @notice initializes bond parameters
571  * @param _controlVariable uint
572  * @param _vestingTerm uint32
573  * @param _minimumPrice uint
574  * @param _maxPayout uint
575  * @param _maxDebt uint
576  */
577 function initializeBondTerms(
578     uint _controlVariable,
579     uint _minimumPrice,
580     uint _maxPayout,
581     uint _maxDebt,
582     uint32 _vestingTerm

```

```

551     ) external onlyOwner() {
552         require( terms.controlVariable == 0, "Bonds
must be initialized from 0" );
553         require( _controlVariable >= 40, "Can lock
adjustment" );
554         require( _maxPayout <= 1000, "Payout cannot
be above 1 percent" );
555         require( _vestingTerm >= 129600, "Vesting m
ust be longer than 36 hours" );
556         require( _fee <= 10000, "DAO fee cannot exc
eed payout" );
557         terms = Terms ( {
558             controlVariable: _controlVariable,
559             minimumPrice: _minimumPrice,
560             maxPayout: _maxPayout,
561             fee: _fee,
562             maxDebt: _maxDebt,
563             vestingTerm: _vestingTerm
564         });
565         lastDecay = uint32(block.timestamp);
566         emit InitTerms(terms);
567     }
568
569
570
571
572     /* ===== POLICY FUNCTIONS ===== */
573
574     enum PARAMETER { VESTING, PAYOUT, FEE, DEBT, MI
NPRICE }
575     /**
576      * @notice set parameters for new bonds
577      * @param _parameter PARAMETER
578      * @param _input uint
579      */
580     function setBondTerms ( PARAMETER _parameter, u
int _input ) external onlyOwner() {
581         if ( _parameter == PARAMETER.VESTING ) { //
0
582             require( _input >= 129600, "Vesting mus
t 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 cann
ot exceed payout" );
589             terms.fee = _input;
590         } else if ( _parameter == PARAMETER.DEBT )
{ // 3
591             terms.maxDebt = _input;
592         } else if ( _parameter == PARAMETER.MINPRIC
E ) { // 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

```

```

583     ) external onlyOwner() {
584         require( terms.controlVariable == 0, "Bonds
must be initialized from 0" );
585         require( _controlVariable >= 40, "Can lock
adjustment" );
586         require( _maxPayout <= 1000, "Payout cannot
be above 1 percent" );
587         require( _vestingTerm >= 129600, "Vesting m
ust be longer than 36 hours" );
588         terms = Terms ( {
589             controlVariable: _controlVariable,
590             minimumPrice: _minimumPrice,
591             maxPayout: _maxPayout,
592             maxDebt: _maxDebt,
593             vestingTerm: _vestingTerm
594         });
595         lastDecay = uint32(block.timestamp);
596         emit InitTerms(terms);
597     }
598
599
600
601
602     /* ===== POLICY FUNCTIONS ===== */
603
604     enum PARAMETER { VESTING, PAYOUT, FEE, DEBT, MI
NPRICE }
605     /**
606      * @notice set parameters for new bonds
607      * @param _parameter PARAMETER
608      * @param _input uint
609      */
610     function setBondTerms ( PARAMETER _parameter, u
int _input ) external onlyOwner() {
611         if ( _parameter == PARAMETER.VESTING ) { //
0
612             require( _input >= 129600, "Vesting mus
t be longer than 36 hours" );
613             terms.vestingTerm = uint32(_input);
614         } else if ( _parameter == PARAMETER.PAYOUT
) { // 1
615             require( _input <= 1000, "Payout cannot
be above 1 percent" );
616             terms.maxPayout = _input;
617         } else if ( _parameter == PARAMETER.DEBT )
{ // 2
618             terms.maxDebt = _input;
619         } else if ( _parameter == PARAMETER.MINPRIC
E ) { // 3
620             terms.minimumPrice = _input;
621         }
622         emit LogSetTerms(_parameter, _input);
623     }
624
625     /**
626      * @notice set control variable adjustment
627      * @param _addition bool
628      * @param _increment uint
629      * @param _target uint
630      * @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 cou
ld 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 _he
lper ) external onlyOwner() {
629         require( _staking != address(0), "IA" );
630         if ( _helper ) {
631             useHelper = true;
632             stakingHelper = IStakingHelper(_stakin
g);
633         } else {
634             useHelper = false;
635             staking = IStaking(_staking);
636         }
637         emit LogSetStaking(_staking, _helper);
638     }
639
640     function allowZapper(address zapper) external o
nlyOwner {
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

```

```

631     */
632     function setAdjustment (
633         bool _addition,
634         uint _increment,
635         uint _target,
636         uint32 _buffer
637     ) external onlyOwner() {
638         require( _increment <= terms.controlVariabl
e.mul( 25 ) / 1000 , "Increment too large" );
639         require(_target >= 40, "Next Adjustment cou
ld be locked");
640         adjustment = Adjust({
641             add: _addition,
642             rate: _increment,
643             target: _target,
644             buffer: _buffer,
645             lastTime: uint32(block.timestamp)
646         });
647         emit LogSetAdjustment(adjustment);
648     }
649
650     /**
651      * @notice set contract for auto stake
652      * @param _staking address
653      * @param _helper bool
654      */
655     function setStaking( address _staking, bool _he
lper ) external onlyOwner() {
656         require( _staking != address(0), "IA" );
657         if ( _helper ) {
658             useHelper = true;
659             stakingHelper = IStakingHelper(_stakin
g);
660         } else {
661             useHelper = false;
662             staking = IStaking(_staking);
663         }
664         emit LogSetStaking(_staking, _helper);
665     }
666
667     function allowZapper(address zapper) external o
nlyOwner {
668         require(zapper != address(0), "ZNA");
669
670         allowedZappers[zapper] = true;
671     }
672
673     function removeZapper(address zapper) external
onlyOwner {
674
675         allowedZappers[zapper] = false;
676     }
677
678
679
680
681     /* ===== USER FUNCTIONS ===== */
682
683     /**
684      * @notice deposit bond
685      * @param _amount uint
686      * @param _maxPrice uint
687      * @param _depositor address
688      * @return uint
689      */
690     function deposit(
691         uint _amount,
692         uint _maxPrice,
693         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
686         uint fee = payout.mul( terms.fee )/ 10000 ;
687         uint profit = value.sub( payout ).sub( fee
);
688
689         uint balanceBefore = Time.balanceOf(address
(this));
690         /**
691         principle is transferred in
692         approved and
693         deposited into the treasury, returning
        (_amount - profit) Time
694         */
695         principle.safeTransferFrom( msg.sender, add
ress(this), _amount );
696         principle.approve( address( treasury ), _am
ount );
697         treasury.deposit( _amount, address(principl
e), profit );
698
699         if ( fee != 0 ) { // fee is transferred to
dao
700             Time.safeTransfer( DAO, fee );
701         }
702         require(balanceBefore.add(profit) == Time.b
alanceOf(address(this)), "Not enough Time to cover
profit");
703         // total debt is increased
704         totalDebt = totalDebt.add( value );
705
706         // depositor info is stored
707         bondInfo[ _depositor ] = Bond({
708             payout: bondInfo[ _depositor ].payout.a
dd( payout ),
709             vesting: terms.vestingTerm,
710             lastTime: uint32(block.timestamp),
711             pricePaid: priceInUSD
});

```

```

694     ) external returns ( uint ) {
695         require( _depositor != address(0), "Invalid
address" );
696         require(msg.sender == _depositor || allowed
Zappers[msg.sender], "LFNA");
697         decayDebt();
698
699         uint priceInUSD = bondPriceInUSD(); // Stor
ed in bond info
700         uint nativePrice = _bondPrice();
701
702         require( _maxPrice >= nativePrice, "Slippag
e limit: more than max price" ); // slippage protec
tion
703
704         uint value = treasury.valueOfToken( address
(principle), _amount );
705         uint payout = payoutFor( value ); // payout
to bonder is computed
706         require( totalDebt.add(value) <= terms.maxD
ebt, "Max capacity reached" );
707         require( payout >= 10000000, "Bond too smal
l" ); // must be > 0.01 Time ( underflow protection
)
708         require( payout <= maxPayout(), "Bond too l
arge"); // size protection because there is no slip
page
709
710         principle.safeTransferFrom( msg.sender, add
ress(treasury), _amount );
711
712         treasury.mintRewards( address(this), payout
);
713
714         // total debt is increased
715         totalDebt = totalDebt.add( value );
716
717         // depositor info is stored
718         bondInfo[ _depositor ] = Bond({
719             payout: bondInfo[ _depositor ].payout.a
dd( payout ),
720             vesting: terms.vestingTerm,
721             lastTime: uint32(block.timestamp),
722             pricePaid: priceInUSD
});

```

```

712
713     // indexed events are emitted
714     emit BondCreated( _amount, payout, block.timestamp.add( terms.vestingTerm ), priceInUSD );
715     emit BondPriceChanged( bondPriceInUSD(), _bondPrice(), 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 _stake ) external returns ( uint ) {
728     require(msg.sender == _recipient, "NA");
729     Bond memory info = bondInfo[ _recipient ];
730     // (seconds since last interaction / vesting term remaining)
731     uint percentVested = percentVestedFor( _recipient );
732
733     if ( percentVested >= 10000 ) { // if fully vested
734         delete bondInfo[ _recipient ]; // delete user info
735         emit BondRedeemed( _recipient, info.payout, 0 ); // emit bond data
736         return stakeOrSend( _recipient, _stake, info.payout ); // pay user everything due
737     } else { // if unfinished
738         // calculate payout vested
739         uint payout = info.payout.mul( percentVested ) / 10000 ;
740         // store updated deposit info
741         bondInfo[ _recipient ] = Bond({
742             payout: info.payout.sub( payout ),
743             vesting: info.vesting.sub32( uint32( block.timestamp ).sub32( info.lastTime ) ),
744             lastTime: uint32(block.timestamp),
745             pricePaid: info.pricePaid
746         });
747         emit BondRedeemed( _recipient, payout, bondInfo[ _recipient ].payout );
748         return stakeOrSend( _recipient, _stake, payout );
749     }
750 }
751
752 }
753
754
755
756
757 /* ===== INTERNAL HELPER FUNCTIONS =====
758 */
759 /**
760  * @notice allow user to stake payout automatically
761  * @param _stake bool
762  * @param _amount uint
763  * @return uint
764  */
765 function stakeOrSend( address _recipient, bool _stake, uint _amount ) internal returns ( uint ) {

```

```

723
724     // indexed events are emitted
725     emit BondCreated( _amount, payout, block.timestamp.add( terms.vestingTerm ), priceInUSD );
726     emit BondPriceChanged( bondPriceInUSD(), _bondPrice(), debtRatio() );
727
728     adjust(); // control variable is adjusted
729     return payout;
730 }
731
732 /**
733  * @notice redeem bond for user
734  * @param _recipient address
735  * @param _stake bool
736  * @return uint
737  */
738 function redeem( address _recipient, bool _stake ) external returns ( uint ) {
739     require(msg.sender == _recipient, "NA");
740     Bond memory info = bondInfo[ _recipient ];
741     // (seconds since last interaction / vesting term remaining)
742     uint percentVested = percentVestedFor( _recipient );
743
744     if ( percentVested >= 10000 ) { // if fully vested
745         delete bondInfo[ _recipient ]; // delete user info
746         emit BondRedeemed( _recipient, info.payout, 0 ); // emit bond data
747         return stakeOrSend( _recipient, _stake, info.payout ); // pay user everything due
748     } else { // if unfinished
749         // calculate payout vested
750         uint payout = info.payout.mul( percentVested ) / 10000 ;
751         // store updated deposit info
752         bondInfo[ _recipient ] = Bond({
753             payout: info.payout.sub( payout ),
754             vesting: info.vesting.sub32( uint32( block.timestamp ).sub32( info.lastTime ) ),
755             lastTime: uint32(block.timestamp),
756             pricePaid: info.pricePaid
757         });
758         emit BondRedeemed( _recipient, payout, bondInfo[ _recipient ].payout );
759         return stakeOrSend( _recipient, _stake, payout );
760     }
761 }
762
763 }
764
765
766
767
768 /* ===== INTERNAL HELPER FUNCTIONS =====
769 */
770 /**
771  * @notice allow user to stake payout automatically
772  * @param _stake bool
773  * @param _amount uint
774  * @return uint
775  */
776 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(stakingHelve
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     * @notice makes incremental adjustment to con
trol variable
781     */
782     function adjust() internal {
783         uint timeCanAdjust = adjustment.lastTime.ad
d32( adjustment.buffer );
784         if( adjustment.rate != 0 && block.timestamp
>= timeCanAdjust ) {
785             uint initial = terms.controlVariable;
786             uint bcv = initial;
787             if ( adjustment.add ) {
788                 bcv = bcv.add(adjustment.rate);
789                 if ( bcv >= adjustment.target ) {
790                     adjustment.rate = 0;
791                     bcv = adjustment.target;
792                 }
793             } else {
794                 bcv = bcv.sub(adjustment.rate);
795                 if ( bcv <= adjustment.target ) {
796                     adjustment.rate = 0;
797                     bcv = adjustment.target;
798                 }
799             }
800             terms.controlVariable = bcv;
801             adjustment.lastTime = uint32(block.time
stamp);
802             emit ControlVariableAdjustment( initia
l, bcv, adjustment.rate, adjustment.add );
803         }
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 /* ===== VIEW FUNCTIONS ===== */
816
817 /**
818 * @notice determine maximum bond size
819 * @return uint
820 */

```

```

777         if ( !_stake ) { // if user does not want t
o stake
778             Time.transfer( _recipient, _amount );
// send payout
779         } else { // if user wants to stake
780             if ( useHelper ) { // use if staking wa
rmup is 0
781                 Time.approve( address(stakingHelve
r), _amount );
782                 stakingHelper.stake( _amount, _reci
pient );
783             } else {
784                 Time.approve( address(staking), _am
ount );
785                 staking.stake( _amount, _recipient
);
786             }
787         }
788         return _amount;
789     }
790     /**
791     * @notice makes incremental adjustment to con
trol variable
792     */
793     function adjust() internal {
794         uint timeCanAdjust = adjustment.lastTime.ad
d32( adjustment.buffer );
795         if( adjustment.rate != 0 && block.timestamp
>= timeCanAdjust ) {
796             uint initial = terms.controlVariable;
797             uint bcv = initial;
798             if ( adjustment.add ) {
799                 bcv = bcv.add(adjustment.rate);
800                 if ( bcv >= adjustment.target ) {
801                     adjustment.rate = 0;
802                     bcv = adjustment.target;
803                 }
804             } else {
805                 bcv = bcv.sub(adjustment.rate);
806                 if ( bcv <= adjustment.target ) {
807                     adjustment.rate = 0;
808                     bcv = adjustment.target;
809                 }
810             }
811             terms.controlVariable = bcv;
812             adjustment.lastTime = uint32(block.time
stamp);
813             emit ControlVariableAdjustment( initia
l, bcv, adjustment.rate, adjustment.add );
814         }
815     }
816 }
817 /**
818 * @notice reduce total debt
819 */
820 function decayDebt() internal {
821     totalDebt = totalDebt.sub( debtDecay() );
822     lastDecay = uint32(block.timestamp);
823 }
824 }
825
826 /* ===== VIEW FUNCTIONS ===== */
827
828 /**
829 * @notice determine maximum bond size
830 * @return uint
831 */

```



```

823     */
824     function maxPayout() public view returns ( uint
) {
825         return Time.totalSupply().mul( terms.maxPay
out ) / 100000 ;
826     }
827
828     /**
829     * @notice calculate interest due for new bond
830     * @param _value uint
831     * @return uint
832     */
833     function payoutFor( uint _value ) public view r
eturns ( uint ) {
834         return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18() / 1e16 ;
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     */
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 {

```

```

834     */
835     function maxPayout() public view returns ( uint
) {
836         return Time.totalSupply().mul( terms.maxPay
out ) / 100000 ;
837     }
838
839     /**
840     * @notice calculate interest due for new bond
841     * @param _value uint
842     * @return uint
843     */
844     function payoutFor( uint _value ) public view r
eturns ( uint ) {
845         return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18() / 1e16 ;
846     }
847
848
849     /**
850     * @notice calculate current bond premium
851     * @return price_ uint
852     */
853     function bondPrice() public view returns ( uint
price_ ) {
854         price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
855         if ( price_ < terms.minimumPrice ) {
856             price_ = terms.minimumPrice;
857         }
858     }
859
860     /**
861     * @notice calculate current bond price and re
move floor if above
862     * @return price_ uint
863     */
864     function _bondPrice() internal returns ( uint p
rice_ ) {
865         price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
866         if ( price_ < terms.minimumPrice ) {
867             price_ = terms.minimumPrice;
868         } else if ( terms.minimumPrice != 0 ) {
869             terms.minimumPrice = 0;
870         }
871     }
872
873     /**
874     * @notice get asset price from chainlink
875     */
876     function assetPrice() public view returns (int)
{
877         ( , int price, , , ) = priceFeed.latestRoun
dData();
878         return price;
879     }
880
881     /**
882     * @notice converts bond price to DAI value
883     * @return price_ uint
884     */
885     function bondPriceInUSD() public view returns (
uint price_ ) {
886         price_ = bondPrice().mul( uint( assetPrice
() ) ).mul( 1e6 );

```

```

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  */
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  */

```

```

887 }
888
889
890 /**
891  * @notice calculate current ratio of debt to
Time supply
892  * @return debtRatio_ uint
893  */
894 function debtRatio() public view returns ( uint
debtRatio_ ) {
895     uint supply = Time.totalSupply();
896     debtRatio_ = FixedPoint.fraction(
897         currentDebt().mul( 1e9 ),
898         supply
899     ).decode112with18() / 1e18;
900 }
901
902 /**
903  * @notice debt ratio in same terms as reserve
bonds
904  * @return uint
905  */
906 function standardizedDebtRatio() external view
returns ( uint ) {
907     return debtRatio().mul( uint( assetPrice()
) ) / 10**priceFeed.decimals(); // ETH feed is 8 de
cimals
908 }
909
910 /**
911  * @notice calculate debt factoring in decay
912  * @return uint
913  */
914 function currentDebt() public view returns ( ui
nt ) {
915     return totalDebt.sub( debtDecay() );
916 }
917
918 /**
919  * @notice amount to decay total debt by
920  * @return decay_ uint
921  */
922 function debtDecay() public view returns ( uint
decay_ ) {
923     uint32 timeSinceLast = uint32(block.timesta
mp).sub32( lastDecay );
924     decay_ = (totalDebt.mul( timeSinceLast )).d
iv(terms.vestingTerm);
925     if ( decay_ > totalDebt ) {
926         decay_ = totalDebt;
927     }
928 }
929
930 /**
931  * @notice calculate how far into vesting a de
positor is
932  * @param _depositor address
933  * @return percentVested_ uint
934  */

```

```

925     function percentVestedFor( address _depositor )
public view returns ( uint percentVested_ ) {
926         Bond memory bond = bondInfo[ _depositor ];
927         uint secondsSinceLast = uint32(block.timestamp).sub32( bond.lastTime );
928         uint vesting = bond.vesting;
929
930         if ( vesting > 0 ) {
931             percentVested_ = secondsSinceLast.mul(
10000 ) / vesting;
932         } else {
933             percentVested_ = 0;
934         }
935     }
936
937     /**
938     * @notice calculate amount of Time available
for claim by depositor
939     * @param _depositor address
940     * @return pendingPayout_ uint
941     */
942     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
943         uint percentVested = percentVestedFor( _depositor );
944         uint payout = bondInfo[ _depositor ].payout;
945
946         if ( percentVested >= 10000 ) {
947             pendingPayout_ = payout;
948         } else {
949             pendingPayout_ = payout.mul( percentVested ) / 10000;
950         }
951     }
952
953
954
955
956     /* ===== AUXILIARY ===== */
957
958     /**
959     * @notice allow anyone to send lost tokens (excluding principle or Time) to the DAO
960     * @return bool
961     */
962     function recoverLostToken(IERC20 _token ) external returns ( bool ) {
963         require( _token != Time, "NAT" );
964         require( _token != principle, "NAP" );
965         uint balance = _token.balanceOf( address(this));
966         _token.safeTransfer( DAO, balance );
967         emit LogRecoverLostToken(address(_token), balance);
968         return true;

```

```

936     function percentVestedFor( address _depositor )
public view returns ( uint percentVested_ ) {
937         Bond memory bond = bondInfo[ _depositor ];
938         uint secondsSinceLast = uint32(block.timestamp).sub32( bond.lastTime );
939         uint vesting = bond.vesting;
940
941         if ( vesting > 0 ) {
942             percentVested_ = secondsSinceLast.mul(
10000 ) / vesting;
943         } else {
944             percentVested_ = 0;
945         }
946     }
947
948     /**
949     * @notice calculate amount of Time available
for claim by depositor
950     * @param _depositor address
951     * @return pendingPayout_ uint
952     */
953     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
954         uint percentVested = percentVestedFor( _depositor );
955         uint payout = bondInfo[ _depositor ].payout;
956
957         if ( percentVested >= 10000 ) {
958             pendingPayout_ = payout;
959         } else {
960             pendingPayout_ = payout.mul( percentVested ) / 10000;
961         }
962     }
963
964
965
966
967     /* ===== AUXILIARY ===== */
968
969     /**
970     * @notice allow anyone to send lost tokens (excluding principle or Time) to the DAO
971     * @return bool
972     */
973     function recoverLostToken(IERC20 _token ) external returns ( bool ) {
974         require( _token != Time, "NAT" );
975         require( _token != principle, "NAP" );
976         uint balance = _token.balanceOf( address(this));
977         _token.safeTransfer( DAO, balance );
978         emit LogRecoverLostToken(address(_token), balance);
979         return true;
980     }
981
982     function recoverLostETH() internal {
983         if ( address(this).balance > 0 ) safeTransferETH(DAO, address(this).balance);
984     }
985
986     /// @notice Transfers ETH to the recipient address
987     /// @dev Fails with `STE`
988     /// @param to The destination of the transfer
989     /// @param value The value to be transferred
990     function safeTransferETH(address to, uint256 value) internal {

```

```
969     }
970 }

991     (bool success, ) = to.call{value: value}(new bytes(0));
992     require(success, 'STE');
993 }
994 }
```

