

```

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.encodeWithSelector(token.approve.selector, spender, newAllowance));
311 }
312
313 function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
314     uint256 newAllowance = token.allowance(address(this), spender)
315         .sub(value);
316     _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
317 }
318
319 function _callOptionalReturn(IERC20 token, bytes memory data) private {
320
321     bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
322     if (returndata.length > 0) { // Return data is optional
323         // solhint-disable-next-line max-line-length
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.encodeWithSelector(token.approve.selector, spender, newAllowance));
316 }
317
318 function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
319     uint256 newAllowance = token.allowance(address(this), spender)
320         .sub(value);
321     _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
322 }
323
324 function _callOptionalReturn(IERC20 token, bytes memory data) private {
325
326     bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
327     if (returndata.length > 0) { // Return data is optional
328         // solhint-disable-next-line max-line-length
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
374     struct uq112x112 {
375         uint224 _x;
376     }
377
378     struct uq144x112 {
379         uint256 _x;
380     }
381
382     uint8 private constant RESOLUTION = 112;
383     uint256 private constant Q112 = 0x10000000000000
0000000000000000;
384     uint256 private constant Q224 = 0x10000000000000
000000000000000000000000000000000000000000000000;
385     uint256 private constant LOWER_MASK = 0xffffffff
ffffffffffffffff; // decimal of UQ*x112 (lower
112 bits)
386
387     function decode(uq112x112 memory self) internal
pure returns (uint112) {
388         return uint112(self._x >> RESOLUTION);
389     }
390
391     function decode112with18(uq112x112 memory self)
internal pure returns (uint) {
392
393         return uint(self._x) / 5192296858534827;
394     }
395
396     function fraction(uint256 numerator, uint256 de
nominator) internal pure returns (uq112x112 memory)
{
397         require(denominator > 0, 'FixedPoint::fract
ion: division by zero');
398         if (numerator == 0) return FixedPoint.uq112
x112(0);
399
400         if (numerator <= uint144(-1)) {
401             uint256 result = (numerator << RESOLUTI
ON) / denominator;
402             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
403             return uq112x112(uint224(result));
404         } else {
405             uint256 result = FullMath.mulDiv(numera
tor, Q112, denominator);
406             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
407             return uq112x112(uint224(result));
408         }
409     }
410 }
411
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 {

```

```

371     l -= mm;
372     require(h < d, 'FullMath::mulDiv: overflow
w');
373     return fullDiv(l, h, d);
374 }
375 }
376
377 library FixedPoint {
378
379     struct uq112x112 {
380         uint224 _x;
381     }
382
383     struct uq144x112 {
384         uint256 _x;
385     }
386
387     uint8 private constant RESOLUTION = 112;
388     uint256 private constant Q112 = 0x100000000000000
0000000000000000;
389     uint256 private constant Q224 = 0x100000000000000
00000000000000000000000000000000000000000000000;
390     uint256 private constant LOWER_MASK = 0xffffffff
ffffffffffffffff; // decimal of UQ*x112 (lower
112 bits)
391
392     function decode(uq112x112 memory self) internal
pure returns (uint112) {
393         return uint112(self._x >> RESOLUTION);
394     }
395
396     function decode112with18(uq112x112 memory self)
internal pure returns (uint) {
397
398         return uint(self._x) / 5192296858534827;
399     }
400
401     function fraction(uint256 numerator, uint256 de
nominator) internal pure returns (uq112x112 memory)
{
402         require(denominator > 0, 'FixedPoint::fract
ion: division by zero');
403         if (numerator == 0) return FixedPoint.uq112
x112(0);
404
405         if (numerator <= uint144(-1)) {
406             uint256 result = (numerator << RESOLUTI
ON) / denominator;
407             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
408             return uq112x112(uint224(result));
409         } else {
410             uint256 result = FullMath.mulDiv(numera
tor, Q112, denominator);
411             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
412             return uq112x112(uint224(result));
413         }
414     }
415 }
416
417 interface ITreasury {
418     function deposit( uint _amount, address _token,
uint _profit ) external returns ( uint );
419     function valueOfToken( address _token, uint _am
ount ) external view returns ( uint value_ );
420 }
421
422 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
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 TimeBondDepository 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
480     /* ===== STRUCTS ===== */
481
482     // Info for creating new bonds
483     struct Terms {
484         uint controlVariable; // scaling variable for price
485         uint minimumPrice; // vs principle value
486         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
487         uint fee; // as % of bond payout, in hundredths. ( 500 = 5% = 0.05 for every 1 paid)
488         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt
489         uint32 vestingTerm; // in seconds
490     }
491
492     // Info for bond holder
493     struct Bond {
494         uint payout; // Time remaining to be paid
495         uint pricePaid; // In DAI, for front end vesting
496         uint32 lastTime; // Last interaction
497         uint32 vesting; // Seconds left to vest
498     }
499
500     // Info for incremental adjustments to control variable
501     struct Adjust {
502         bool add; // addition or subtraction
503         uint rate; // increment
504         uint target; // BCV when adjustment finished
505         uint32 buffer; // minimum length (in seconds) between adjustments
506         uint32 lastTime; // time when last adjustment made
507     }
508
509
510
511
512     /* ===== INITIALIZATION ===== */
513
514     constructor (
515         address _Time,
516         address _principle,
517         address _treasury,
518         address _DAO,
519         address _bondCalculator
520     ) {
521         require( _Time != address(0) );
522         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     uint256 public startRedeem;
485
486     /* ===== STRUCTS ===== */
487
488     // Info for creating new bonds
489     struct Terms {
490         uint controlVariable; // scaling variable for price
491         uint minimumPrice; // vs principle value
492         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
493         uint fee; // as % of bond payout, in hundredths. ( 500 = 5% = 0.05 for every 1 paid)
494         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt
495         uint32 vestingTerm; // in seconds
496     }
497
498     // Info for bond holder
499     struct Bond {
500         uint payout; // Time remaining to be paid
501         uint pricePaid; // In DAI, for front end vesting
502         uint32 lastTime; // Last interaction
503         uint32 vesting; // Seconds left to vest
504     }
505
506     // Info for incremental adjustments to control variable
507     struct Adjust {
508         bool add; // addition or subtraction
509         uint rate; // increment
510         uint target; // BCV when adjustment finished
511         uint32 buffer; // minimum length (in seconds) between adjustments
512         uint32 lastTime; // time when last adjustment made
513     }
514
515
516
517
518     /* ===== INITIALIZATION ===== */
519
520     constructor (
521         address _Time,
522         address _principle,
523         address _treasury,
524         address _DAO,
525         address _bondCalculator
526     ) {
527         require( _Time != address(0) );
528         Time = IERC20(_Time);

```

```

523     require( _principle != address(0) );
524     principle = IERC20(_principle);
525     require( _treasury != address(0) );
526     treasury = ITreasury(_treasury);
527     require( _DAO != address(0) );
528     DAO = _DAO;
529     // bondCalculator should be address(0) if not LP bond
530     bondCalculator = IBondCalculator(_bondCalculator);
531     isLiquidityBond = ( _bondCalculator != address(0) );
532 }
533
534 /**
535  * @notice initializes bond parameters
536  * @param _controlVariable uint
537  * @param _vestingTerm uint32
538  * @param _minimumPrice uint
539  * @param _maxPayout uint
540  * @param _fee uint
541  * @param _maxDebt uint
542  */
543 function initializeBondTerms(
544     uint _controlVariable,
545     uint _minimumPrice,
546     uint _maxPayout,
547     uint _fee,
548     uint _maxDebt,
549     uint32 _vestingTerm
550 ) external onlyOwner() {
551     require( terms.controlVariable == 0, "Bonds
552 must be initialized from 0" );
553     require( _controlVariable >= 40, "Can lock
554 adjustment" );
555     require( _maxPayout <= 1000, "Payout cannot
556 be above 1 percent" );
557     require( _vestingTerm >= 129600, "Vesting must
558 be longer than 36 hours" );
559     require( _fee <= 10000, "DAO fee cannot exceed
560 payout" );
561     terms = Terms ({
562         controlVariable: _controlVariable,
563         minimumPrice: _minimumPrice,
564         maxPayout: _maxPayout,
565         fee: _fee,
566         maxDebt: _maxDebt,
567         vestingTerm: _vestingTerm
568     });
569     lastDecay = uint32(block.timestamp);
570     emit InitTerms(terms);
571 }
572
573 /**
574  * @notice set parameters for new bonds
575  * @param _parameter PARAMETER
576  * @param _input uint
577  */
578 function setBondTerms ( PARAMETER _parameter, uint
579 _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 <= 1000, "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     startRedeem = block.timestamp + 10 days;
576     lastDecay = uint32(block.timestamp);
577     emit InitTerms(terms);
578 }
579
580 /**
581  * @notice set parameters for new bonds
582  * @param _parameter PARAMETER
583  * @param _input uint
584  */
585 function setBondTerms ( PARAMETER _parameter, uint
586 _input ) external onlyOwner() {

```

```

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

```

```

587         if ( _parameter == PARAMETER.VESTING ) { //
0
588             require( _input >= 129600, "Vesting must
be longer than 36 hours" );
589             terms.vestingTerm = uint32(_input);
590         } else if ( _parameter == PARAMETER.PAYOUT
) { // 1
591             require( _input <= 1000, "Payout cannot
be above 1 percent" );
592             terms.maxPayout = _input;
593         } else if ( _parameter == PARAMETER.FEE ) {
// 2
594             require( _input <= 10000, "DAO fee cannot
exceed payout" );
595             terms.fee = _input;
596         } else if ( _parameter == PARAMETER.DEBT )
{ // 3
597             terms.maxDebt = _input;
598         } else if ( _parameter == PARAMETER.MINPRICE
) { // 4
599             terms.minimumPrice = _input;
600         }
601         emit LogSetTerms(_parameter, _input);
602     }
603
604     /**
605     * @notice set control variable adjustment
606     * @param _addition bool
607     * @param _increment uint
608     * @param _target uint
609     * @param _buffer uint
610     */
611     function setAdjustment (
612         bool _addition,
613         uint _increment,
614         uint _target,
615         uint32 _buffer
616     ) external onlyOwner() {
617         require( _increment <= terms.controlVariable
e.mul( 25 ) / 1000 , "Increment too large" );
618         require(_target >= 40, "Next Adjustment could
be locked");
619         adjustment = Adjust({
620             add: _addition,
621             rate: _increment,
622             target: _target,
623             buffer: _buffer,
624             lastTime: uint32(block.timestamp)
625         });
626         emit LogSetAdjustment(adjustment);
627     }
628
629     /**
630     * @notice set contract for auto stake
631     * @param _staking address
632     * @param _helper bool
633     */
634     function setStaking( address _staking, bool _hel
per ) external onlyOwner() {
635         require( _staking != address(0), "IA" );
636         if ( _helper ) {
637             useHelper = true;
638             stakingHelper = IStakingHelper(_staking
g);
639         } else {
640             useHelper = false;
641             staking = IStaking(_staking);
642         }

```

```

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

```

```

643         emit LogSetStaking(_staking, _helper);
644     }
645
646     function allowZapper(address zapper) external o
nlyOwner {
647         require(zapper != address(0), "ZNA");
648
649         allowedZappers[zapper] = true;
650     }
651
652     function removeZapper(address zapper) external
onlyOwner {
653
654         allowedZappers[zapper] = false;
655     }
656
657
658
659
660     /* ===== USER FUNCTIONS ===== */
661
662     /**
663     * @notice deposit bond
664     * @param _amount uint
665     * @param _maxPrice uint
666     * @param _depositor address
667     * @return uint
668     */
669     function deposit(
670         uint _amount,
671         uint _maxPrice,
672         address _depositor
673     ) external returns ( uint ) {
674         require( _depositor != address(0), "Invalid
address" );
675         require(msg.sender == _depositor || allowed
Zappers[msg.sender], "LFNA");
676         decayDebt();
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         /**

```

```

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

```

```

694         principle is transferred in
695         approved and
696         deposited into the treasury, returning
        (_amount - profit) Time
697     */
698     principle.safeTransferFrom( msg.sender, add
ress(this), _amount );
699     principle.approve( address( treasury ), _am
ount );
700     treasury.deposit( _amount, address(principl
e), profit );
701
702     if ( fee != 0 ) { // fee is transferred to
dao
703         Time.safeTransfer( DAO, fee );
704     }
705     require(balanceBefore.add(payout) == Time.b
alanceOf(address(this)), "Not enough Time to cover
profit");
706     // total debt is increased
707     totalDebt = totalDebt.add( value );
708
709     // depositor info is stored
710     bondInfo[ _depositor ] = Bond({
711         payout: bondInfo[ _depositor ].payout.a
dd( payout ),
712         vesting: terms.vestingTerm,
713         lastTime: uint32(block.timestamp),
714         pricePaid: priceInUSD
715     });
716
717     // indexed events are emitted
718     emit BondCreated( _amount, payout, block.ti
mestamp.add( terms.vestingTerm ), priceInUSD );
719     emit BondPriceChanged( bondPriceInUSD(), _b
ondPrice(), debtRatio() );
720
721     adjust(); // control variable is adjusted
722     return payout;
723 }
724
725 /**
726  * @notice redeem bond for user
727  * @param _recipient address
728  * @param _stake bool
729  * @return uint
730  */
731 function redeem( address _recipient, bool _stak
e ) external returns ( uint ) {
732     require(block.timestamp >= startRedeem, "Re
deem has not started yet.");
733     require(msg.sender == _recipient, "NA");
734     Bond memory info = bondInfo[ _recipient ];
735     // (seconds since last interaction / vestin
g term remaining)
736     uint percentVested = percentVestedFor( _rec
ipient );
737
738     if ( percentVested >= 10000 ) { // if fully
vested
739         delete bondInfo[ _recipient ]; // delet
e user info
740         emit BondRedeemed( _recipient, info.pay
out, 0 ); // emit bond data
741         return stakeOrSend( _recipient, _stake,
info.payout ); // pay user everything due
742
743     } else { // if unfinished

```

```

738         // calculate payout vested
739         uint payout = info.payout.mul( percentV
ested ) / 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
748         emit BondRedeemed( _recipient, payout,
bondInfo[ _recipient ].payout );
749         return stakeOrSend( _recipient, _stake,
payout );
750     }
751 }
752
753
754
755
756 /* ===== INTERNAL HELPER FUNCTIONS =====
*/
757
758 /**
759  * @notice allow user to stake payout automati
cally
760  * @param _stake bool
761  * @param _amount uint
762  * @return uint
763  */
764 function stakeOrSend( address _recipient, bool
_stake, uint _amount ) internal returns ( uint ) {
765     if ( !_stake ) { // if user does not want t
o stake
766         Time.transfer( _recipient, _amount );
// send payout
767     } else { // if user wants to stake
768         if ( useHelper ) { // use if staking wa
rmup is 0
769             Time.approve( address(stakingHelpe
r), _amount );
770             stakingHelper.stake( _amount, _reci
pient );
771         } else {
772             Time.approve( address(staking), _am
ount );
773             staking.stake( _amount, _recipient
);
774         }
775     }
776     return _amount;
777 }
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 ) {

```

```

744         // calculate payout vested
745         uint payout = info.payout.mul( percentV
ested ) / 10000 ;
746         // store updated deposit info
747         bondInfo[ _recipient ] = Bond({
748             payout: info.payout.sub( payout ),
749             vesting: info.vesting.sub32( uint32
( block.timestamp ).sub32( info.lastTime ) ),
750             lastTime: uint32(block.timestamp),
751             pricePaid: info.pricePaid
752         });
753
754         emit BondRedeemed( _recipient, payout,
bondInfo[ _recipient ].payout );
755         return stakeOrSend( _recipient, _stake,
payout );
756     }
757 }
758
759
760
761
762 /* ===== INTERNAL HELPER FUNCTIONS =====
*/
763
764 /**
765  * @notice allow user to stake payout automati
cally
766  * @param _stake bool
767  * @param _amount uint
768  * @return uint
769  */
770 function stakeOrSend( address _recipient, bool
_stake, uint _amount ) internal returns ( uint ) {
771     if ( !_stake ) { // if user does not want t
o stake
772         Time.transfer( _recipient, _amount );
// send payout
773     } else { // if user wants to stake
774         if ( useHelper ) { // use if staking wa
rmup is 0
775             Time.approve( address(stakingHelpe
r), _amount );
776             stakingHelper.stake( _amount, _reci
pient );
777         } else {
778             Time.approve( address(staking), _am
ount );
779             staking.stake( _amount, _recipient
);
780         }
781     }
782     return _amount;
783 }
784
785 /**
786  * @notice makes incremental adjustment to con
trol variable
787  */
788 function adjust() internal {
789     uint timeCanAdjust = adjustment.lastTime.ad
d32( adjustment.buffer );
790     if( adjustment.rate != 0 && block.timestamp
>= timeCanAdjust ) {
791         uint initial = terms.controlVariable;
792         uint bcv = initial;
793         if ( adjustment.add ) {
794             bcv = bcv.add(adjustment.rate);
795             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  * @notice reduce total debt
807  */
808
809 function decayDebt() internal {
810     totalDebt = totalDebt.sub( debtDecay() );
811     lastDecay = uint32(block.timestamp);
812 }
813
814
815
816
817 /* ===== VIEW FUNCTIONS ===== */
818
819 /**
820  * @notice determine maximum bond size
821  * @return uint
822  */
823
824 function maxPayout() public view returns ( uint
) {
825     return Time.totalSupply().mul( terms.maxPay
out ) / 100000 ;
826 }
827 /**
828  * @notice calculate interest due for new bond
829  * @param _value uint
830  * @return uint
831  */
832
833 function payoutFor( uint _value ) public view r
eturns ( uint ) {
834     return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18() / 1e16 ;
835 }
836
837 /**
838  * @notice calculate current bond premium
839  * @return price_ uint
840  */
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  * @notice calculate current bond price and re
move floor if above
850  * @return price_ uint

```

```

796         adjustment.rate = 0;
797         bcv = adjustment.target;
798     }
799 } else {
800     bcv = bcv.sub(adjustment.rate);
801     if ( bcv <= adjustment.target ) {
802         adjustment.rate = 0;
803         bcv = adjustment.target;
804     }
805 }
806 terms.controlVariable = bcv;
807 adjustment.lastTime = uint32(block.time
stamp);
808     emit ControlVariableAdjustment( initia
l, bcv, adjustment.rate, adjustment.add );
809 }
810 }
811 /**
812  * @notice reduce total debt
813  */
814
815 function decayDebt() internal {
816     totalDebt = totalDebt.sub( debtDecay() );
817     lastDecay = uint32(block.timestamp);
818 }
819
820
821
822
823 /* ===== VIEW FUNCTIONS ===== */
824
825 /**
826  * @notice determine maximum bond size
827  * @return uint
828  */
829
830 function maxPayout() public view returns ( uint
) {
831     return (Time.totalSupply().add(50000000000
00)).mul( terms.maxPayout ) / 100000 ;
832 }
833 /**
834  * @notice calculate interest due for new bond
835  * @param _value uint
836  * @return uint
837  */
838
839 function payoutFor( uint _value ) public view r
eturns ( uint ) {
840     return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18() / 1e16 ;
841 }
842
843 /**
844  * @notice calculate current bond premium
845  * @return price_ uint
846  */
847
848 function bondPrice() public view returns ( uint
price_ ) {
849     price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
850     if ( price_ < terms.minimumPrice ) {
851         price_ = terms.minimumPrice;
852     }
853 }
854 /**
855  * @notice calculate current bond price and re
move floor if above
856  * @return price_ uint

```



```

851     */
852     function _bondPrice() internal returns ( uint p
rice_ ) {
853         price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
854         if ( price_ < terms.minimumPrice ) {
855             price_ = terms.minimumPrice;
856         } else if ( terms.minimumPrice != 0 ) {
857             terms.minimumPrice = 0;
858         }
859     }
860
861     /**
862     * @notice converts bond price to DAI value
863     * @return price_ uint
864     */
865     function bondPriceInUSD() public view returns (
uint price_ ) {
866         if( isLiquidityBond ) {
867             price_ = bondPrice().mul( bondCalculato
r.markdown( address(principle) ) ) / 100 ;
868         } else {
869             price_ = bondPrice().mul( 10 ** princip
le.decimals() ) / 100;
870         }
871     }
872
873
874     /**
875     * @notice calculate current ratio of debt to
Time supply
876     * @return debtRatio_ uint
877     */
878     function debtRatio() public view returns ( uint
debtRatio_ ) {
879         uint supply = Time.totalSupply();
880         debtRatio_ = FixedPoint.fraction(
881             currentDebt().mul( 1e9 ),
882             supply
883         ).decode112with18() / 1e18;
884     }
885
886     /**
887     * @notice debt ratio in same terms for reserv
e or liquidity bonds
888     * @return uint
889     */
890     function standardizedDebtRatio() external view
returns ( uint ) {
891         if ( isLiquidityBond ) {
892             return debtRatio().mul( bondCalculator.
markdown( address(principle) ) ) / 1e9;
893         } else {
894             return debtRatio();
895         }
896     }
897
898     /**
899     * @notice calculate debt factoring in decay
900     * @return uint
901     */
902     function currentDebt() public view returns ( ui
nt ) {
903         return totalDebt.sub( debtDecay() );
904     }
905
906     /**
907     * @notice amount to decay total debt by

```

```

857     */
858     function _bondPrice() internal returns ( uint p
rice_ ) {
859         price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
860         if ( price_ < terms.minimumPrice ) {
861             price_ = terms.minimumPrice;
862         } else if ( terms.minimumPrice != 0 ) {
863             terms.minimumPrice = 0;
864         }
865     }
866
867     /**
868     * @notice converts bond price to DAI value
869     * @return price_ uint
870     */
871     function bondPriceInUSD() public view returns (
uint price_ ) {
872         if( isLiquidityBond ) {
873             price_ = bondPrice().mul( bondCalculato
r.markdown( address(principle) ) ) / 100 ;
874         } else {
875             price_ = bondPrice().mul( 10 ** princip
le.decimals() ) / 100;
876         }
877     }
878
879
880     /**
881     * @notice calculate current ratio of debt to
Time supply
882     * @return debtRatio_ uint
883     */
884     function debtRatio() public view returns ( uint
debtRatio_ ) {
885         uint supply = Time.totalSupply().add(500000
000000000);
886         debtRatio_ = FixedPoint.fraction(
887             currentDebt().mul( 1e9 ),
888             supply
889         ).decode112with18() / 1e18;
890     }
891
892     /**
893     * @notice debt ratio in same terms for reserv
e or liquidity bonds
894     * @return uint
895     */
896     function standardizedDebtRatio() external view
returns ( uint ) {
897         if ( isLiquidityBond ) {
898             return debtRatio().mul( bondCalculator.
markdown( address(principle) ) ) / 1e9;
899         } else {
900             return debtRatio();
901         }
902     }
903
904     /**
905     * @notice calculate debt factoring in decay
906     * @return uint
907     */
908     function currentDebt() public view returns ( ui
nt ) {
909         return totalDebt.sub( debtDecay() );
910     }
911
912     /**
913     * @notice amount to decay total debt by

```

```

908     * @return decay_ uint
909     */
910     function debtDecay() public view returns ( uint
decay_ ) {
911         uint32 timeSinceLast = uint32(block.timesta
mp).sub32( lastDecay );
912         decay_ = totalDebt.mul( timeSinceLast ) / t
erms.vestingTerm;
913         if ( decay_ > totalDebt ) {
914             decay_ = totalDebt;
915         }
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
929         if ( vesting > 0 ) {
930             percentVested_ = secondsSinceLast.mul(
10000 ) / vesting;
931         } else {
932             percentVested_ = 0;
933         }
934     }
935
936     /**
937     * @notice calculate amount of Time available
for claim by depositor
938     * @param _depositor address
939     * @return pendingPayout_ uint
940     */
941     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
942         uint percentVested = percentVestedFor( _dep
ositor );
943         uint payout = bondInfo[ _depositor ].payou
t;
944
945         if ( percentVested >= 10000 ) {
946             pendingPayout_ = payout;
947         } else {
948             pendingPayout_ = payout.mul( percentVes
ted ) / 10000;
949         }
950     }
951
952
953
954
955     /* ===== AUXILLIARY ===== */
956
957     /**
958     * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
959     * @return bool
960     */
961     function recoverLostToken(IERC20 _token ) exter
nal returns ( bool ) {
962         require( _token != Time, "NAT" );
963         require( _token != principle, "NAP" );

```

```

914     * @return decay_ uint
915     */
916     function debtDecay() public view returns ( uint
decay_ ) {
917         uint32 timeSinceLast = uint32(block.timesta
mp).sub32( lastDecay );
918         decay_ = (totalDebt.mul( timeSinceLast ))div(terms.vestingTerm);
919         if ( decay_ > totalDebt ) {
920             decay_ = totalDebt;
921         }
922     }
923
924
925     /**
926     * @notice calculate how far into vesting a de
positor is
927     * @param _depositor address
928     * @return percentVested_ uint
929     */
930     function percentVestedFor( address _depositor )
public view returns ( uint percentVested_ ) {
931         Bond memory bond = bondInfo[ _depositor ];
932         uint secondsSinceLast = uint32(block.timest
amp).sub32( bond.lastTime );
933         uint vesting = bond.vesting;
934
935         if ( vesting > 0 ) {
936             percentVested_ = secondsSinceLast.mul(
10000 ) / vesting;
937         } else {
938             percentVested_ = 0;
939         }
940     }
941
942     /**
943     * @notice calculate amount of Time available
for claim by depositor
944     * @param _depositor address
945     * @return pendingPayout_ uint
946     */
947     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
948         uint percentVested = percentVestedFor( _dep
ositor );
949         uint payout = bondInfo[ _depositor ].payou
t;
950
951         if ( percentVested >= 10000 ) {
952             pendingPayout_ = payout;
953         } else {
954             pendingPayout_ = payout.mul( percentVes
ted ) / 10000;
955         }
956     }
957
958
959
960
961     /* ===== AUXILLIARY ===== */
962
963     /**
964     * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
965     * @return bool
966     */
967     function recoverLostToken(IERC20 _token ) exter
nal returns ( bool ) {
968         require( _token != Time, "NAT" );
969         require( _token != principle, "NAP" );

```

```

964         uint balance = _token.balanceOf( address(th
is));
965         _token.safeTransfer( DAO,  balance );
966         emit LogRecoverLostToken(address(_token), b
alance);
967         return true;

```

```

968     }
969 }

```

```

970         uint balance = _token.balanceOf( address(th
is));
971         _token.safeTransfer( DAO,  balance );
972         emit LogRecoverLostToken(address(_token), b
alance);
973         return true;
974     }
975
976     function recoverLostETH() internal {
977         if (address(this).balance > 0) safeTransfer
ETH(DAO, address(this).balance);
978     }
979
980     /// @notice Transfers ETH to the recipient addr
ess
981     /// @dev Fails with `STE`
982     /// @param to The destination of the transfer
983     /// @param value The value to be transferred
984     function safeTransferETH(address to, uint256 va
lue) internal {
985         (bool success, ) = to.call{value: value}(ne
w bytes(0));
986         require(success, 'STE');
987     }
988 }

```

