

```

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 = 0x1000000000000000000000000000000000000000000000000000;
384     uint256 private constant LOWER_MASK = 0xffffffffffffffff; // 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         return uint(self._x) / 5192296858534827;
392     }
393
394     function fraction(uint256 numerator, uint256 denominator) internal pure returns (uq112x112 memory) {
395         {
396             require(denominator > 0, 'FixedPoint::fraction: division by zero');
397             if (numerator == 0) return FixedPoint.uq112x112(0);
398
399             if (numerator <= uint144(-1)) {
400                 uint256 result = (numerator << RESOLUTION) / denominator;
401                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
402                 return uq112x112(uint224(result));
403             } else {
404                 uint256 result = FullMath.mulDiv(numerator, Q112, denominator);
405                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
406                 return uq112x112(uint224(result));
407             }
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

```

```

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 = 0x1000000000000000000000000000000000000000000000000000;
389     uint256 private constant LOWER_MASK = 0xffffffffffffffff; // 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         return uint(self._x) / 5192296858534827;
397     }
398
399     function fraction(uint256 numerator, uint256 denominator) internal pure returns (uq112x112 memory) {
400         {
401             require(denominator > 0, 'FixedPoint::fraction: division by zero');
402             if (numerator == 0) return FixedPoint.uq112x112(0);
403
404             if (numerator <= uint144(-1)) {
405                 uint256 result = (numerator << RESOLUTION) / denominator;
406                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
407                 return uq112x112(uint224(result));
408             } else {
409                 uint256 result = FullMath.mulDiv(numerator, Q112, denominator);
410                 require(result <= uint224(-1), 'FixedPoint::fraction: overflow');
411                 return uq112x112(uint224(result));
412             }
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 _amount ) external view returns ( uint value_ );
420     function mintRewards( address _recipient, uint _amount ) external;
421 }
422

```



```

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
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 interface IBondCalculator {
424     function valuation( address _LP, uint _amount )
        external view returns ( uint );
425     function markdown( address _LP ) external view
        returns ( uint );
426 }
427
428 interface ISTaking {
429     function stake( uint _amount, address _recipien
        t ) external returns ( bool );
430 }
431
432 interface ISTakingHelper {
433     function stake( uint _amount, address _recipien
        t ) external;
434 }
435
436 contract MaiaBondDepository is Ownable {
437
438     using FixedPoint for *;
439     using SafeERC20 for IERC20;
440     using LowGasSafeMath for uint;
441     using LowGasSafeMath for uint32;
442
443
444
445
446     /* ===== EVENTS ===== */
447
448     event BondCreated( uint deposit, uint indexed p
        ayout, uint indexed expires, uint indexed priceInUS
        D );
449     event BondRedeemed( address indexed recipient,
        uint payout, uint remaining );
450     event BondPriceChanged( uint indexed priceInUS
        D, uint indexed internalPrice, uint indexed debtRat
        io );
451     event ControlVariableAdjustment( uint initialBC
        V, uint newBCV, uint adjustment, bool addition );
452     event InitTerms( Terms terms);
453     event LogSetTerms(PARAMETER param, uint value);
454     event LogSetAdjustment( Adjust adjust);
455     event LogSetStaking( address indexed stakingCon
        tract, bool isHelper);
456     event LogRecoverLostToken( address indexed toke
        nToRecover, uint amount);
457
458
459
460     /* ===== STATE VARIABLES ===== */
461
462     IERC20 public immutable Time; // token given as
        payment for bond
463     IERC20 public immutable principle; // token use
        d to create bond
464     ITreasury public immutable treasury; // mints T
        ime when receives principle
465     address public immutable DAO; // receives profi
        t share from bond
466
467     bool public immutable isLiquidityBond; // LP an
        d Reserve bonds are treated slightly different
468     IBondCalculator public immutable bondCalculato
        r; // calculates value of LP tokens
469
470     ISTaking public staking; // to auto-stake payou
        t
471     ISTakingHelper public stakingHelper; // to stak
        e and claim if no staking warmup
472     bool public useHelper;
473

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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 <= 2000, "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         }

```

```

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

```

```

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         require( _maxPrice >= nativePrice, "Slippag
e limit: more than max price" ); // slippage protec
tion
681
682         uint value = treasury.valueOfToken( address
(principle), _amount );
683         uint payout = payoutFor( value ); // payout
to bonder is computed
684         require( totalDebt.add(value) <= terms.maxD
ebt, "Max capacity reached" );
685         require( payout >= 10000000, "Bond too smal
l" ); // must be > 0.01 Time ( underflow protection
)
686         require( payout <= maxPayout(), "Bond too l
arge"); // size protection because there is no slip
page
687
688
689         /**
        principle is transferred in

```

```

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

```

```

690         approved and
691         deposited into the treasury
692     */
693
694     principle.approve( address( treasury ), _am
ount );
695     principle.safeTransferFrom( msg.sender, add
ress(treasury), _amount );
696     treasury.mintRewards( address(this), payout
);
697
698     // total debt is increased
699     totalDebt = totalDebt.add( value );
700
701     // depositor info is stored
702     bondInfo[ _depositor ] = Bond({
703         payout: bondInfo[ _depositor ].payout.a
dd( payout ),
704         vesting: terms.vestingTerm,
705         lastTime: uint32(block.timestamp),
706         pricePaid: priceInUSD
707     });
708     // indexed events are emitted
709     emit BondCreated( _amount, payout, block.ti
mestamp.add( terms.vestingTerm ), priceInUSD );
710     emit BondPriceChanged( bondPriceInUSD(), _b
ondPrice(), debtRatio() );
711
712     adjust(); // control variable is adjusted
713     return payout;
714 }
715
716 /**
717  * @notice redeem bond for user
718  * @param _recipient address
719  * @param _stake bool
720  * @return uint
721  */
722
723 function redeem( address _recipient, bool _stak
e ) external returns ( uint ) {
724     require(msg.sender == _recipient, "NA");
725     Bond memory info = bondInfo[ _recipient ];
726     // (seconds since last interaction / vestin
g term remaining)
727     uint percentVested = percentVestedFor( _rec
ipient );
728
729     if ( percentVested >= 10000 ) { // if fully
vested
730         delete bondInfo[ _recipient ]; // delet
e user info
731         emit BondRedeemed( _recipient, info.pay
out, 0 ); // emit bond data
732         return stakeOrSend( _recipient, _stake,
info.payout ); // pay user everything due
733
734     } else { // if unfinished
735         // calculate payout vested

```

```

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

```

```

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

```

```

792         bcv = adjustment.target;
793     }
794     } else {
795         bcv = bcv.sub(adjustment.rate);
796         if ( bcv <= adjustment.target ) {
797             adjustment.rate = 0;
798             bcv = adjustment.target;
799         }
800     }
801     terms.controlVariable = bcv;
802     adjustment.lastTime = uint32(block.time
stamp);
803     emit ControlVariableAdjustment( initia
l, bcv, adjustment.rate, adjustment.add );
804 }
805 }
806
807 /**
808  * @notice reduce total debt
809  */
810 function decayDebt() internal {
811     totalDebt = totalDebt.sub( debtDecay() );
812     lastDecay = uint32(block.timestamp);
813 }
814
815
816
817
818 /* ===== VIEW FUNCTIONS ===== */
819
820 /**
821  * @notice determine maximum bond size
822  * @return uint
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  */

```

```

788         bcv = adjustment.target;
789     }
790     } else {
791         bcv = bcv.sub(adjustment.rate);
792         if ( bcv <= adjustment.target ) {
793             adjustment.rate = 0;
794             bcv = adjustment.target;
795         }
796     }
797     terms.controlVariable = bcv;
798     adjustment.lastTime = uint32(block.time
stamp);
799     emit ControlVariableAdjustment( initia
l, bcv, adjustment.rate, adjustment.add );
800 }
801 }
802
803 /**
804  * @notice reduce total debt
805  */
806 function decayDebt() internal {
807     totalDebt = totalDebt.sub( debtDecay() );
808     lastDecay = uint32(block.timestamp);
809 }
810
811
812
813
814 /* ===== VIEW FUNCTIONS ===== */
815
816 /**
817  * @notice determine maximum bond size
818  * @return uint
819  */
820 function maxPayout() public view returns ( uint
) {
821     return Time.totalSupply().mul( terms.maxPay
out ) / 100000 ;
822 }
823
824 /**
825  * @notice calculate interest due for new bond
826  * @param _value uint
827  * @return uint
828  */
829 function payoutFor( uint _value ) public view r
eturns ( uint ) {
830     return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18() / 1e14 ;
831 }
832
833
834 /**
835  * @notice calculate current bond premium
836  * @return price_ uint
837  */
838 function bondPrice() public view returns ( uint
price_ ) {
839     price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
840     if ( price_ < terms.minimumPrice ) {
841         price_ = terms.minimumPrice;
842     }
843 }
844
845 /**
846  * @notice calculate current bond price and re
move floor if above
847  * @return price_ uint
848  */

```



```

853     function _bondPrice() internal returns ( uint p
rice_ ) {
854         price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
855         if ( price_ < terms.minimumPrice ) {
856             price_ = terms.minimumPrice;
857         } else if ( terms.minimumPrice != 0 ) {
858             terms.minimumPrice = 0;
859         }
860     }
861
862     /**
863      * @notice converts bond price to DAI value
864      * @return price_ uint
865      */
866     function bondPriceInUSD() public view returns (
uint price_ ) {
867         if( isLiquidityBond ) {
868             price_ = bondPrice().mul( bondCalculato
r.markdown( address(principle) ) ) / 100 ;
869         } else {
870             price_ = bondPrice().mul( 10 ** princip
le.decimals() ) / 100;
871         }
872     }
873
874
875     /**
876      * @notice calculate current ratio of debt to
Time supply
877      * @return debtRatio_ uint
878      */
879     function debtRatio() public view returns ( uint
debtRatio_ ) {
880         uint supply = Time.totalSupply();
881         debtRatio_ = FixedPoint.fraction(
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
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      */

```

```

849     function _bondPrice() internal returns ( uint p
rice_ ) {
850         price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
851         if ( price_ < terms.minimumPrice ) {
852             price_ = terms.minimumPrice;
853         } else if ( terms.minimumPrice != 0 ) {
854             terms.minimumPrice = 0;
855         }
856     }
857
858     /**
859      * @notice converts bond price to DAI value
860      * @return price_ uint
861      */
862     function bondPriceInUSD() public view returns (
uint price_ ) {
863         if( isLiquidityBond ) {
864             price_ = bondPrice().mul( bondCalculato
r.markdown( address(principle) ) ) / 100 ;
865         } else {
866             price_ = bondPrice().mul( 10 ** princip
le.decimals() ) / 10000;
867         }
868     }
869
870
871     /**
872      * @notice calculate current ratio of debt to
Time supply
873      * @return debtRatio_ uint
874      */
875     function debtRatio() public view returns ( uint
debtRatio_ ) {
876         uint supply = Time.totalSupply();
877         debtRatio_ = FixedPoint.fraction(
currentDebt().mul( 1e9 ),
878         supply
879         ).decode112with18() / 1e18;
880     }
881
882     /**
883      * @notice debt ratio in same terms for reserv
e or liquidity bonds
884      * @return uint
885      */
886
887     function standardizedDebtRatio() external view
returns ( uint ) {
888         if ( isLiquidityBond ) {
889             return debtRatio().mul( bondCalculator.
markdown( address(principle) ) ) / 1e9;
890         } else {
891             return debtRatio();
892         }
893     }
894
895     /**
896      * @notice calculate debt factoring in decay
897      * @return uint
898      */
899     function currentDebt() public view returns ( ui
nt ) {
900         return totalDebt.sub( debtDecay() );
901     }
902
903     /**
904      * @notice amount to decay total debt by
905      * @return decay_ uint
906      */

```

```

911     function debtDecay() public view returns ( uint
decay_ ) {
912         uint32 timeSinceLast = uint32(block.timesta
mp).sub32( lastDecay );
913         decay_ = totalDebt.mul( timeSinceLast ) / t
erms.vestingTerm;
914         if ( decay_ > totalDebt ) {
915             decay_ = totalDebt;
916         }
917     }
918
919     /**
920     * @notice calculate how far into vesting a de
positor is
921     * @param _depositor address
922     * @return percentVested_ uint
923     */
924
925     function percentVestedFor( address _depositor )
public view returns ( uint percentVested_ ) {
926         Bond memory bond = bondInfo[ _depositor ];
927         uint secondsSinceLast = uint32(block.timest
amp).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
943     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
944         uint percentVested = percentVestedFor( _dep
ositor );
945         uint payout = bondInfo[ _depositor ].payou
t;
946
947         if ( percentVested >= 10000 ) {
948             pendingPayout_ = payout;
949         } else {
950             pendingPayout_ = payout.mul( percentVes
ted ) / 10000;
951         }
952     }
953
954     /** ===== AUXILLIARY ===== */
955
956     /**
957     * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
958     * @return bool
959     */
960
961     function recoverLostToken(IERC20 _token ) exter
nal returns ( bool ) {
962         require( _token != Time, "NAT" );
963         require( _token != principle, "NAP" );
964         uint balance = _token.balanceOf( address(th
is));

```

```

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

```

```

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

```

```

969 }
970 }

```

```

962     _token.safeTransfer( DAO, balance );
963     emit LogRecoverLostToken(address(_token), balance);
964     return true;
965 }
966
967     function recoverLostETH() internal {
968         if (address(this).balance > 0) safeTransferETH(DAO, address(this).balance);
969     }
970
971     /// @notice Transfers ETH to the recipient address
972     /// @dev Fails with `STE`
973     /// @param to The destination of the transfer
974     /// @param value The value to be transferred
975     function safeTransferETH(address to, uint256 value) internal {
976         (bool success, ) = to.call{value: value}(new bytes(0));
977         require(success, 'STE');
978     }
979 }

```

