

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 interface IOwnable {
5     function policy() external view returns (address);
6
7     function renounceManagement() external;
8
9     function pushManagement( address newOwner_ ) external;
10
11     function pullManagement() external;
12 }
13
14 contract Ownable is IOwnable {
15     address internal _owner;
16     address internal _newOwner;
17
18     event OwnershipPushed(address indexed previousOwner, address indexed newOwner);
19     event OwnershipPulled(address indexed previousOwner, address indexed newOwner);
20
21     constructor () {
22         _owner = msg.sender;
23         emit OwnershipPushed( address(0), _owner );
24     }
25
26     function policy() public view override returns (address) {
27         return _owner;
28     }
29
30     modifier onlyPolicy() {
31         require( _owner == msg.sender, "Ownable: caller is not the owner" );
32         _;
33     }
34
35     function renounceManagement() public virtual override onlyPolicy() {
36         emit OwnershipPushed( _owner, address(0) );
37         _owner = address(0);
38     }
39
40     function pushManagement( address newOwner_ ) public virtual override onlyPolicy() {
41         require( newOwner_ != address(0), "Ownable: new owner is the zero address" );
42         emit OwnershipPushed( _owner, newOwner_ );
43         _newOwner = newOwner_;
44     }
45
46     function pullManagement() public virtual override {
47         require( msg.sender == _newOwner, "Ownable: must be new owner to pull" );
48         emit OwnershipPulled( _owner, _newOwner );
49         _owner = _newOwner;
50     }
51 }
52
53

```

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 interface IOwnable {
5     function policy() external view returns (address);
6
7     function renounceManagement() external;
8
9     function pushManagement( address newOwner_ ) external;
10
11     function pullManagement() external;
12 }
13
14 contract Ownable is IOwnable {
15     address internal _owner;
16     address internal _newOwner;
17
18     event OwnershipPushed(address indexed previousOwner, address indexed newOwner);
19     event OwnershipPulled(address indexed previousOwner, address indexed newOwner);
20
21     constructor () {
22         _owner = msg.sender;
23         emit OwnershipPushed( address(0), _owner );
24     }
25
26     function policy() public view override returns (address) {
27         return _owner;
28     }
29
30     modifier onlyPolicy() {
31         require( _owner == msg.sender, "Ownable: caller is not the owner" );
32         _;
33     }
34
35     function renounceManagement() public virtual override onlyPolicy() {
36         emit OwnershipPushed( _owner, address(0) );
37         _owner = address(0);
38     }
39
40     function pushManagement( address newOwner_ ) public virtual override onlyPolicy() {
41         require( newOwner_ != address(0), "Ownable: new owner is the zero address" );
42         emit OwnershipPushed( _owner, newOwner_ );
43         _newOwner = newOwner_;
44     }
45
46     function pullManagement() public virtual override {
47         require( msg.sender == _newOwner, "Ownable: must be new owner to pull" );
48         emit OwnershipPulled( _owner, _newOwner );
49         _owner = _newOwner;
50     }
51 }
52
53

```

```

54 library LowGasSafeMath {
55     /// @notice Returns x + y, reverts if sum overflows
56     uint256
57     /// @param x The augend
58     /// @param y The addend
59     /// @return z The sum of x and y
60     function add(uint256 x, uint256 y) internal pure
61     returns (uint256 z) {
62         require((z = x + y) >= x);
63     }
64     function add32(uint32 x, uint32 y) internal pure
65     returns (uint32 z) {
66         require((z = x + y) >= x);
67     }
68     /// @notice Returns x - y, reverts if underflows
69     s
70     /// @param x The minuend
71     /// @param y The subtrahend
72     /// @return z The difference of x and y
73     function sub(uint256 x, uint256 y) internal pure
74     returns (uint256 z) {
75         require((z = x - y) <= x);
76     }
77     function sub32(uint32 x, uint32 y) internal pure
78     returns (uint32 z) {
79         require((z = x - y) <= x);
80     }
81     /// @notice Returns x * y, reverts if overflows
82     /// @param x The multiplicand
83     /// @param y The multiplier
84     /// @return z The product of x and y
85     function mul(uint256 x, uint256 y) internal pure
86     returns (uint256 z) {
87         require(x == 0 || (z = x * y) / x == y);
88     }
89     /// @notice Returns x + y, reverts if overflows
90     or underflows
91     /// @param x The augend
92     /// @param y The addend
93     /// @return z The sum of x and y
94     function add(int256 x, int256 y) internal pure
95     returns (int256 z) {
96         require((z = x + y) >= x == (y >= 0));
97     }
98     /// @notice Returns x - y, reverts if overflows
99     or underflows
100    /// @param x The minuend
101    /// @param y The subtrahend
102    /// @return z The difference of x and y
103    function sub(int256 x, int256 y) internal pure
104    returns (int256 z) {
105        require((z = x - y) <= x == (y >= 0));
106    }
107 }
108
109 library Address {
110

```

```

54 library LowGasSafeMath {
55     /// @notice Returns x + y, reverts if sum overflows
56     uint256
57     /// @param x The augend
58     /// @param y The addend
59     /// @return z The sum of x and y
60     function add(uint256 x, uint256 y) internal pure
61     returns (uint256 z) {
62         require((z = x + y) >= x);
63     }
64     function add32(uint32 x, uint32 y) internal pure
65     returns (uint32 z) {
66         require((z = x + y) >= x);
67     }
68     /// @notice Returns x - y, reverts if underflows
69     s
70     /// @param x The minuend
71     /// @param y The subtrahend
72     /// @return z The difference of x and y
73     function sub(uint256 x, uint256 y) internal pure
74     returns (uint256 z) {
75         require((z = x - y) <= x);
76     }
77     function sub32(uint32 x, uint32 y) internal pure
78     returns (uint32 z) {
79         require((z = x - y) <= x);
80     }
81     /// @notice Returns x * y, reverts if overflows
82     /// @param x The multiplicand
83     /// @param y The multiplier
84     /// @return z The product of x and y
85     function mul(uint256 x, uint256 y) internal pure
86     returns (uint256 z) {
87         require(x == 0 || (z = x * y) / x == y);
88     }
89     /// @notice Returns x + y, reverts if overflows
90     or underflows
91     /// @param x The augend
92     /// @param y The addend
93     /// @return z The sum of x and y
94     function add(int256 x, int256 y) internal pure
95     returns (int256 z) {
96         require((z = x + y) >= x == (y >= 0));
97     }
98     /// @notice Returns x - y, reverts if overflows
99     or underflows
100    /// @param x The minuend
101    /// @param y The subtrahend
102    /// @return z The difference of x and y
103    function sub(int256 x, int256 y) internal pure
104    returns (int256 z) {
105        require((z = x - y) <= x == (y >= 0));
106    }
107 }
108
109 library Address {
110

```

```

106     function isContract(address account) internal v
107     iew returns (bool) {
108         uint256 size;
109         // solhint-disable-next-line no-inline-asse
110         mbly
111         assembly { size := extcodesize(account) }
112         return size > 0;
113     }
114     function sendValue(address payable recipient, u
115     int256 amount) internal {
116         require(address(this).balance >= amount, "A
117         ddress: insufficient balance");
118         // solhint-disable-next-line avoid-low-leve
119         l-calls, avoid-call-value
120         (bool success, ) = recipient.call{ value: a
121         mount }("");
122         require(success, "Address: unable to send v
123         alue, recipient may have reverted");
124     }
125     function functionCall(address target, bytes mem
126     ory data) internal returns (bytes memory) {
127         return functionCall(target, data, "Address: l
128         ow-level call failed");
129     }
130     function functionCall(
131     address target,
132     bytes memory data,
133     string memory errorMessage
134     ) internal returns (bytes memory) {
135         return _functionCallWithValue(target, data,
136         0, errorMessage);
137     }
138     function functionCallWithValue(address target,
139     bytes memory data, uint256 value) internal returns
140     (bytes memory) {
141         return functionCallWithValue(target, data,
142         value, "Address: low-level call with value faile
143         d");
144     }
145     function functionCallWithValue(
146     address target,
147     bytes memory data,
148     uint256 value,
149     string memory errorMessage
150     ) internal returns (bytes memory) {
151         require(address(this).balance >= value, "Ad
152         dress: insufficient balance for call");
153         require(isContract(target), "Address: call
154         to non-contract");
155         // solhint-disable-next-line avoid-low-leve
156         l-calls
157         (bool success, bytes memory returndata) = t
158         arget.call{ value: value }(data);
159         return _verifyCallResult(success, returndat
160         a, errorMessage);
161     }
162     function _functionCallWithValue(
163     address target,
164     bytes memory data,
165     uint256 weiValue,

```

```

111     function isContract(address account) internal v
112     iew returns (bool) {
113         uint256 size;
114         // solhint-disable-next-line no-inline-asse
115         mbly
116         assembly { size := extcodesize(account) }
117         return size > 0;
118     }
119     function sendValue(address payable recipient, u
120     int256 amount) internal {
121         require(address(this).balance >= amount, "A
122         ddress: insufficient balance");
123         // solhint-disable-next-line avoid-low-leve
124         l-calls, avoid-call-value
125         (bool success, ) = recipient.call{ value: a
126         mount }("");
127         require(success, "Address: unable to send v
128         alue, recipient may have reverted");
129     }
130     function functionCall(address target, bytes mem
131     ory data) internal returns (bytes memory) {
132         return functionCall(target, data, "Address: l
133         ow-level call failed");
134     }
135     function functionCall(
136     address target,
137     bytes memory data,
138     string memory errorMessage
139     ) internal returns (bytes memory) {
140         return _functionCallWithValue(target, data,
141         0, errorMessage);
142     }
143     function functionCallWithValue(address target,
144     bytes memory data, uint256 value) internal returns
145     (bytes memory) {
146         return functionCallWithValue(target, data,
147         value, "Address: low-level call with value faile
148         d");
149     }
150     function functionCallWithValue(
151     address target,
152     bytes memory data,
153     uint256 value,
154     string memory errorMessage
155     ) internal returns (bytes memory) {
156         require(address(this).balance >= value, "Ad
157         dress: insufficient balance for call");
158         require(isContract(target), "Address: call
159         to non-contract");
160         // solhint-disable-next-line avoid-low-leve
161         l-calls
162         (bool success, bytes memory returndata) = t
163         arget.call{ value: value }(data);
164         return _verifyCallResult(success, returndat
165         a, errorMessage);
166     }
167     function _functionCallWithValue(
168     address target,
169     bytes memory data,
170     uint256 weiValue,

```

```

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

```

```

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

```

```

207 // solhint-disable-next-line avoid-low-level-calls
208 (bool success, bytes memory returndata) = target.delegatecall(data);
209 return _verifyCallResult(success, returndata, errorMessage);
210 }
211
212 function _verifyCallResult(
213     bool success,
214     bytes memory returndata,
215     string memory errorMessage
216 ) private pure returns(bytes memory) {
217     if (success) {
218         return returndata;
219     } else {
220         if (returndata.length > 0) {
221             assembly {
222                 let returndata_size := mload(returndata)
223                 revert(add(32, returndata), returndata_size)
224             }
225         } else {
226             revert(errorMessage);
227         }
228     }
229 }
230
231 function addressToString(address _address) internal pure returns(string memory) {
232     bytes32 _bytes = bytes32(uint256(_address));
233     bytes memory HEX = "0123456789abcdef";
234     bytes memory _addr = new bytes(42);
235     _addr[0] = '0';
236     _addr[1] = 'x';
237     for(uint256 i = 0; i < 20; i++) {
238         _addr[2+i*2] = HEX[uint8(_bytes[i + 12] >> 4)];
239         _addr[3+i*2] = HEX[uint8(_bytes[i + 12] & 0x0f)];
240     }
241     return string(_addr);
242 }
243
244 interface IERC20 {
245     function decimals() external view returns (uint8);
246     function totalSupply() external view returns (uint256);
247     function balanceOf(address account) external view returns (uint256);
248     function transfer(address recipient, uint256 amount) external returns (bool);
249     function allowance(address owner, address spender) external view returns (uint256);
250     function approve(address spender, uint256 amount) external returns (bool);

```

```

212 // solhint-disable-next-line avoid-low-level-calls
213 (bool success, bytes memory returndata) = target.delegatecall(data);
214 return _verifyCallResult(success, returndata, errorMessage);
215 }
216
217 function _verifyCallResult(
218     bool success,
219     bytes memory returndata,
220     string memory errorMessage
221 ) private pure returns(bytes memory) {
222     if (success) {
223         return returndata;
224     } else {
225         if (returndata.length > 0) {
226             assembly {
227                 let returndata_size := mload(returndata)
228                 revert(add(32, returndata), returndata_size)
229             }
230         } else {
231             revert(errorMessage);
232         }
233     }
234 }
235
236 function addressToString(address _address) internal pure returns(string memory) {
237     bytes32 _bytes = bytes32(uint256(_address));
238     bytes memory HEX = "0123456789abcdef";
239     bytes memory _addr = new bytes(42);
240     _addr[0] = '0';
241     _addr[1] = 'x';
242     for(uint256 i = 0; i < 20; i++) {
243         _addr[2+i*2] = HEX[uint8(_bytes[i + 12] >> 4)];
244         _addr[3+i*2] = HEX[uint8(_bytes[i + 12] & 0x0f)];
245     }
246     return string(_addr);
247 }
248
249 interface IERC20 {
250     function decimals() external view returns (uint8);
251     function totalSupply() external view returns (uint256);
252     function balanceOf(address account) external view returns (uint256);
253     function transfer(address recipient, uint256 amount) external returns (bool);
254     function allowance(address owner, address spender) external view returns (uint256);
255     function approve(address spender, uint256 amount) external returns (bool);

```

```

263     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
264
265     event Transfer(address indexed from, address indexed to, uint256 value);
266
267     event Approval(address indexed owner, address indexed spender, uint256 value);
268 }
269
270 library SafeERC20 {
271     using LowGasSafeMath for uint256;
272     using Address for address;
273
274     function safeTransfer(IERC20 token, address to, uint256 value) internal {
275         _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
276     }
277
278     function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
279         _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
280     }
281
282     function safeApprove(IERC20 token, address spender, uint256 value) internal {
283         require((value == 0) || (token.allowance(address(this), spender) == 0),
284             "SafeERC20: approve from non-zero to non-zero allowance");
285         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
286     }
287
288     function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
289         uint256 newAllowance = token.allowance(address(this), spender).add(value);
290         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
291     }
292
293     function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
294         uint256 newAllowance = token.allowance(address(this), spender).sub(value);
295         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
296     }
297
298     function _callOptionalReturn(IERC20 token, bytes memory data) private {
299         bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
300         if (returndata.length > 0) { // Return data is optional
301             // solhint-disable-next-line max-line-length
302             require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
303         }
304     }

```

```

268     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
269
270     event Transfer(address indexed from, address indexed to, uint256 value);
271
272     event Approval(address indexed owner, address indexed spender, uint256 value);
273 }
274
275 library SafeERC20 {
276     using LowGasSafeMath for uint256;
277     using Address for address;
278
279     function safeTransfer(IERC20 token, address to, uint256 value) internal {
280         _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
281     }
282
283     function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
284         _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
285     }
286
287     function safeApprove(IERC20 token, address spender, uint256 value) internal {
288         require((value == 0) || (token.allowance(address(this), spender) == 0),
289             "SafeERC20: approve from non-zero to non-zero allowance");
290         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
291     }
292
293     function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
294         uint256 newAllowance = token.allowance(address(this), spender).add(value);
295         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
296     }
297
298     function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
299         uint256 newAllowance = token.allowance(address(this), spender).sub(value);
300         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
301     }
302
303     function _callOptionalReturn(IERC20 token, bytes memory data) private {
304         bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
305         if (returndata.length > 0) { // Return data is optional
306             // solhint-disable-next-line max-line-length
307             require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
308         }
309     }

```



```

369     function decode(uq112x112 memory self) internal
pure returns (uint112) {
370         return uint112(self._x >> RESOLUTION);
371     }
372
373     function decode112with18(uq112x112 memory self)
internal pure returns (uint) {
374
375         return uint(self._x) / 5192296858534827;
376     }
377
378     function fraction(uint256 numerator, uint256 de
nominator) internal pure returns (uq112x112 memory)
{
379         require(denominator > 0, 'FixedPoint::fract
ion: division by zero');
380         if (numerator == 0) return FixedPoint.uq112
x112(0);
381
382         if (numerator <= uint144(-1)) {
383             uint256 result = (numerator << RESOLUTI
ON) / denominator;
384             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
385             return uq112x112(uint224(result));
386         } else {
387             uint256 result = FullMath.mulDiv(numera
tor, Q112, denominator);
388             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
389             return uq112x112(uint224(result));
390         }
391     }
392 }

```

```

393
394 interface AggregatorV3Interface {
395
396     function decimals() external view returns (uint
8);
397     function description() external view returns (str
ing memory);
398     function version() external view returns (uint25
6);
399
400     // getRoundData and latestRoundData should both r
aise "No data present"
401     // if they do not have data to report, instead of
returning unset values
402     // which could be misinterpreted as actual report
ed values.
403     function getRoundData(uint80 _roundId)
external
404     view
405     returns (
406         uint80 roundId,
407         int256 answer,
408         uint256 startedAt,
409         uint256 updatedAt,
410         uint80 answeredInRound
411     );
412     function latestRoundData()
external
413     view
414     returns (
415         uint80 roundId,
416         int256 answer,
417         uint256 startedAt,
418         uint256 updatedAt,
419         uint80 answeredInRound
420     );
421

```

```

374     function decode(uq112x112 memory self) internal
pure returns (uint112) {
375         return uint112(self._x >> RESOLUTION);
376     }
377
378     function decode112with18(uq112x112 memory self)
internal pure returns (uint) {
379
380         return uint(self._x) / 5192296858534827;
381     }
382
383     function fraction(uint256 numerator, uint256 de
nominator) internal pure returns (uq112x112 memory)
{
384         require(denominator > 0, 'FixedPoint::fract
ion: division by zero');
385         if (numerator == 0) return FixedPoint.uq112
x112(0);
386
387         if (numerator <= uint144(-1)) {
388             uint256 result = (numerator << RESOLUTI
ON) / denominator;
389             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
390             return uq112x112(uint224(result));
391         } else {
392             uint256 result = FullMath.mulDiv(numera
tor, Q112, denominator);
393             require(result <= uint224(-1), 'FixedPo
int::fraction: overflow');
394             return uq112x112(uint224(result));
395         }
396     }
397 }
398

```



```

422     );
423 }
424
425 interface ITreasury {
426     function deposit( uint _amount, address _token,
uint _profit ) external returns ( bool );
427     function valueOf( address _token, uint _amount
) external view returns ( uint value_ );
428     function mintRewards( address _recipient, uint
_amount ) external;
429 }
430
431 interface ISTaking {
432     function stake( uint _amount, address _recipien
t ) external returns ( bool );
433 }
434
435 interface ISTakingHelper {
436     function stake( uint _amount, address _recipien
t ) external;
437 }
438
439 interface IWAVAX9 is IERC20 {
440     /// @notice Deposit ether to get wrapped ether
441     function deposit() external payable;
442 }
443
444 contract TimeBondDepository is Ownable {
445
446     using FixedPoint for *;
447     using SafeERC20 for IERC20;
448     using SafeERC20 for IWAVAX9;
449     using LowGasSafeMath for uint;
450     using LowGasSafeMath for uint32;
451
452
453
454
455     /* ===== EVENTS ===== */
456
457     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUS
D );
458     event BondRedeemed( address indexed recipient,
uint payout, uint remaining );
459     event BondPriceChanged( uint indexed priceInUS
D, uint indexed internalPrice, uint indexed debtRat
io );
460     event ControlVariableAdjustment( uint initialBC
V, uint newBCV, uint adjustment, bool addition );
461
462
463
464
465     /* ===== STATE VARIABLES ===== */
466     IERC20 public immutable Time; // token given as
payment for bond
467     IWAVAX9 public immutable principle; // token us
ed to create bond
468     ITreasury public immutable treasury; // mints T
ime when receives principle
469     address public immutable DAO; // receives profi
t share from bond
470
471     AggregatorV3Interface public priceFeed;
472
473     ISTaking public staking; // to auto-stake payou
t
474     ISTakingHelper public stakingHelper; // to stak
e and claim if no staking warmup
475     bool public useHelper;
476

```

```

399
400 interface ITreasury {
401     function deposit( uint _amount, address _token,
uint _profit ) external returns ( uint );
402     function valueOfToken( address _token, uint _am
ount ) external view returns ( uint value_ );
403     function mintRewards( address _recipient, uint
_amount ) external;
404 }
405
406 interface ISTaking {
407     function stake( uint _amount, address _recipien
t ) external returns ( bool );
408 }
409
410 interface ISTakingHelper {
411     function stake( uint _amount, address _recipien
t ) external;
412 }
413
414 interface IWMATIC9 is IERC20 {
415     /// @notice Deposit ether to get wrapped ether
416     function deposit() external payable;
417 }
418
419 contract MaiaBondDepository is Ownable {
420
421     using FixedPoint for *;
422     using SafeERC20 for IERC20;
423     using SafeERC20 for IWMATIC9;
424     using LowGasSafeMath for uint;
425     using LowGasSafeMath for uint32;
426
427
428
429
430     /* ===== EVENTS ===== */
431
432     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUS
D );
433     event BondRedeemed( address indexed recipient,
uint payout, uint remaining );
434     event BondPriceChanged( uint indexed priceInUS
D, uint indexed internalPrice, uint indexed debtRat
io );
435     event ControlVariableAdjustment( uint initialBC
V, uint newBCV, uint adjustment, bool addition );
436
437
438
439
440     /* ===== STATE VARIABLES ===== */
441     IERC20 public immutable Time; // token given as
payment for bond
442     IWMATIC9 public immutable principle; // token u
sed to create bond
443     ITreasury public immutable treasury; // mints T
ime when receives principle
444     address public immutable DAO; // receives profi
t share from bond
445
446     ISTaking public staking; // to auto-stake payou
t
447     ISTakingHelper public stakingHelper; // to stak
e and claim if no staking warmup
448     bool public useHelper;
449

```

```

477     Terms public terms; // stores terms for new bonds
478     Adjust public adjustment; // stores adjustment to BCV data
479
480     mapping( address => Bond ) public bondInfo; // stores bond information for depositors
481
482     uint public totalDebt; // total value of outstanding bonds; used for pricing
483     uint32 public lastDecay; // reference time for debt decay
484
485
486     mapping (address => bool) public allowedZappers;
487
488     /* ===== STRUCTS ===== */
489     // Info for creating new bonds
490     struct Terms {
491         uint controlVariable; // scaling variable for price
492         uint minimumPrice; // vs principle value. 4 decimals (1500 = 0.15)
493         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
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 vesting; // Seconds left to vest
503         uint32 lastTime; // Last interaction
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     /* ===== INITIALIZATION ===== */
516     constructor (
517         address _Time,
518         address _principle,
519         address _treasury,
520         address _DAO,
521         address _feed
522     ) {
523         require( _Time != address(0) );
524         Time = IERC20(_Time);
525         require( _principle != address(0) );
526         principle = IWAAX9(_principle);

```

```

450     Terms public terms; // stores terms for new bonds
451     Adjust public adjustment; // stores adjustment to BCV data
452
453     mapping( address => Bond ) public bondInfo; // stores bond information for depositors
454
455     uint public totalDebt; // total value of outstanding bonds; used for pricing
456     uint32 public lastDecay; // reference time for debt decay
457
458
459     mapping (address => bool) public allowedZappers;
460
461     /* ===== STRUCTS ===== */
462     // Info for creating new bonds
463     struct Terms {
464         uint controlVariable; // scaling variable for price
465         uint minimumPrice; // vs principle value. 4 decimals (1500 = 0.15)
466         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
467         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt
468         uint32 vestingTerm; // in seconds
469     }
470
471     // Info for bond holder
472     struct Bond {
473         uint payout; // Time remaining to be paid
474         uint pricePaid; // In DAI, for front end vesting
475         uint32 vesting; // Seconds left to vest
476         uint32 lastTime; // Last interaction
477     }
478
479     // Info for incremental adjustments to control variable
480     struct Adjust {
481         bool add; // addition or subtraction
482         uint rate; // increment
483         uint target; // BCV when adjustment finished
484         uint32 buffer; // minimum length (in seconds) between adjustments
485         uint32 lastTime; // time when last adjustment made
486     }
487
488     /* ===== INITIALIZATION ===== */
489     constructor (
490         address _Time,
491         address _principle,
492         address _treasury,
493         address _DAO
494     ) {
495         require( _Time != address(0) );
496         Time = IERC20(_Time);
497         require( _principle != address(0) );
498         principle = IWMATIC9(_principle);

```

```

533     require( _treasury != address(0) );
534     treasury = ITreasury(_treasury);
535     require( _DAO != address(0) );
536     DAO = _DAO;
537     require( _feed != address(0) );
538     priceFeed = AggregatorV3Interface( _feed );
539 }
540
541 /**
542  * @notice initializes bond parameters
543  * @param _controlVariable uint
544  * @param _vestingTerm uint
545  * @param _minimumPrice uint
546  * @param _maxPayout uint
547  * @param _maxDebt uint
548  */
549 function initializeBondTerms(
550     uint _controlVariable,
551     uint _minimumPrice,
552     uint _maxPayout,
553     uint _maxDebt,
554     uint32 _vestingTerm
555 ) external onlyPolicy() {
556     require( currentDebt() == 0, "Debt must be
0 for initialization" );
557     require( _controlVariable >= 40, "Can lock
adjustment" );
558     require( _maxPayout <= 1000, "Payout cannot
be above 1 percent" );
559     require( _vestingTerm >= 129600, "Vesting m
ust be longer than 36 hours" );
560     terms = Terms ( {
561         controlVariable: _controlVariable,
562         vestingTerm: _vestingTerm,
563         minimumPrice: _minimumPrice,
564         maxPayout: _maxPayout,
565         maxDebt: _maxDebt
566     });
567     lastDecay = uint32(block.timestamp);
568 }
569
570
571
572
573 /* ===== POLICY FUNCTIONS ===== */
574
575 enum PARAMETER { VESTING, PAYOUT, DEBT, MINPRIC
E }
576 /**
577  * @notice set parameters for new bonds
578  * @param _parameter PARAMETER
579  * @param _input uint
580  */
581 function setBondTerms ( PARAMETER _parameter, u
int _input ) external onlyPolicy() {
582     if ( _parameter == PARAMETER.VESTING ) { //
0
583         require( _input >= 129600, "Vesting mus
t be longer than 36 hours" );
584         terms.vestingTerm = uint32(_input);
585     } else if ( _parameter == PARAMETER.PAYOUT
) { // 1
586         require( _input <= 1000, "Payout cannot
be above 1 percent" );
587         terms.maxPayout = _input;
588     } else if ( _parameter == PARAMETER.DEBT )
{ // 2
589         terms.maxDebt = _input;

```

```

505     require( _treasury != address(0) );
506     treasury = ITreasury(_treasury);
507     require( _DAO != address(0) );
508     DAO = _DAO;
509 }
510
511 /**
512  * @notice initializes bond parameters
513  * @param _controlVariable uint
514  * @param _vestingTerm uint
515  * @param _minimumPrice uint
516  * @param _maxPayout uint
517  * @param _maxDebt uint
518  */
519 function initializeBondTerms(
520     uint _controlVariable,
521     uint _minimumPrice,
522     uint _maxPayout,
523     uint _maxDebt,
524     uint32 _vestingTerm
525 ) external onlyPolicy() {
526     require( currentDebt() == 0, "Debt must be
0 for initialization" );
527     require( _controlVariable >= 40, "Can lock
adjustment" );
528     require( _maxPayout <= 1000, "Payout cannot
be above 1 percent" );
529     require( _vestingTerm >= 129600, "Vesting m
ust be longer than 36 hours" );
530     terms = Terms ( {
531         controlVariable: _controlVariable,
532         vestingTerm: _vestingTerm,
533         minimumPrice: _minimumPrice,
534         maxPayout: _maxPayout,
535         maxDebt: _maxDebt
536     });
537     lastDecay = uint32(block.timestamp);
538 }
539
540
541
542
543 /* ===== POLICY FUNCTIONS ===== */
544
545 enum PARAMETER { VESTING, PAYOUT, DEBT, MINPRIC
E }
546 /**
547  * @notice set parameters for new bonds
548  * @param _parameter PARAMETER
549  * @param _input uint
550  */
551 function setBondTerms ( PARAMETER _parameter, u
int _input ) external onlyPolicy() {
552     if ( _parameter == PARAMETER.VESTING ) { //
0
553         require( _input >= 129600, "Vesting mus
t be longer than 36 hours" );
554         terms.vestingTerm = uint32(_input);
555     } else if ( _parameter == PARAMETER.PAYOUT
) { // 1
556         require( _input <= 1000, "Payout cannot
be above 1 percent" );
557         terms.maxPayout = _input;
558     } else if ( _parameter == PARAMETER.DEBT )
{ // 2
559         terms.maxDebt = _input;

```

```

590     } else if ( _parameter == PARAMETER.MINPRIC
E ) { // 3
591         terms.minimumPrice = _input;
592     }
593 }
594
595 /**
596  * @notice set control variable adjustment
597  * @param _addition bool
598  * @param _increment uint
599  * @param _target uint
600  * @param _buffer uint
601  */
602 function setAdjustment (
603     bool _addition,
604     uint _increment,
605     uint _target,
606     uint32 _buffer
607 ) external onlyPolicy() {
608     require( _increment <= terms.controlVariabl
e.mul( 25 )/ 1000, "Increment too large" );
609     require(_target >= 40, "Next Adjustment cou
ld be locked");
610     adjustment = Adjust({
611         add: _addition,
612         rate: _increment,
613         target: _target,
614         buffer: _buffer,
615         lastTime: uint32(block.timestamp)
616     });
617 }
618
619 /**
620  * @notice set contract for auto stake
621  * @param _staking address
622  * @param _helper bool
623  */
624 function setStaking( address _staking, bool _he
lper ) external onlyPolicy() {
625     require( _staking != address(0) , "IA");
626     if ( _helper ) {
627         useHelper = true;
628         stakingHelper = IStakingHelper(_stakin
g);
629     } else {
630         useHelper = false;
631         staking = IStaking(_staking);
632     }
633 }
634
635 function allowZapper(address zapper) external o
nlyPolicy {
636     require(zapper != address(0), "ZNA");
637
638     allowedZappers[zapper] = true;
639 }
640
641 function removeZapper(address zapper) external
onlyPolicy {
642
643     allowedZappers[zapper] = false;
644 }
645
646
647
648
649 /* ===== USER FUNCTIONS ===== */
650
651 /**

```

```

560     } else if ( _parameter == PARAMETER.MINPRIC
E ) { // 3
561         terms.minimumPrice = _input;
562     }
563 }
564
565 /**
566  * @notice set control variable adjustment
567  * @param _addition bool
568  * @param _increment uint
569  * @param _target uint
570  * @param _buffer uint
571  */
572 function setAdjustment (
573     bool _addition,
574     uint _increment,
575     uint _target,
576     uint32 _buffer
577 ) external onlyPolicy() {
578     require( _increment <= terms.controlVariabl
e.mul( 25 )/ 1000, "Increment too large" );
579     require(_target >= 40, "Next Adjustment cou
ld be locked");
580     adjustment = Adjust({
581         add: _addition,
582         rate: _increment,
583         target: _target,
584         buffer: _buffer,
585         lastTime: uint32(block.timestamp)
586     });
587 }
588
589 /**
590  * @notice set contract for auto stake
591  * @param _staking address
592  * @param _helper bool
593  */
594 function setStaking( address _staking, bool _he
lper ) external onlyPolicy() {
595     require( _staking != address(0) , "IA");
596     if ( _helper ) {
597         useHelper = true;
598         stakingHelper = IStakingHelper(_stakin
g);
599     } else {
600         useHelper = false;
601         staking = IStaking(_staking);
602     }
603 }
604
605 function allowZapper(address zapper) external o
nlyPolicy {
606     require(zapper != address(0), "ZNA");
607
608     allowedZappers[zapper] = true;
609 }
610
611 function removeZapper(address zapper) external
onlyPolicy {
612
613     allowedZappers[zapper] = false;
614 }
615
616
617
618
619 /* ===== USER FUNCTIONS ===== */
620
621 /**

```

```

652 * @notice deposit bond
653 * @param _amount uint
654 * @param _maxPrice uint
655 * @param _depositor address
656 * @return uint
657 */
658 function deposit(
659     uint _amount,
660     uint _maxPrice,
661     address _depositor
662 ) external payable returns ( uint ) {
663     require( _depositor != address(0), "Invalid
address" );
664     require(msg.sender == _depositor || allowed
Zappers[msg.sender], "LFNA");
665     decayDebt();
666     require( totalDebt <= terms.maxDebt, "Max c
apacity reached" );
667
668     uint priceInUSD = bondPriceInUSD(); // Stor
ed in bond info
669     uint nativePrice = _bondPrice();
670
671     require( _maxPrice >= nativePrice, "Slippag
e limit: more than max price" ); // slippage protec
tion
672
673     uint value = treasury.valueOf( address(prin
ciple), _amount );
674     uint payout = payoutFor( value ); // payout
to bonder is computed
675
676     require( payout >= 100000000, "Bond too smal
l" ); // must be > 0.01 Time ( underflow protection
)
677     require( payout <= maxPayout(), "Bond too l
arge"); // size protection because there is no slip
page
678
679     /**
680         asset carries risk and is not minted ag
ainst
681         asset transfered to treasury and reward
s minted as payout
682     */
683     if (address(this).balance >= _amount) {
684         // pay with WETH9
685         require(msg.value == _amount, "UA");
686         principle.deposit{value: _amount}(); //
wrap only what is needed to pay
687         principle.transfer(address(treasury), _
amount);
688     } else {
689         principle.safeTransferFrom( msg.sender,
address(treasury), _amount );
690     }
691
692     treasury.mintRewards( address(this), payout
);
693
694     // total debt is increased
695     totalDebt = totalDebt.add( value );
696
697     // depositor info is stored
698     bondInfo[ _depositor ] = Bond({
699         payout: bondInfo[ _depositor ].payout.a
dd( payout ),
700         vesting: terms.vestingTerm,

```

```

622 * @notice deposit bond
623 * @param _amount uint
624 * @param _maxPrice uint
625 * @param _depositor address
626 * @return uint
627 */
628 function deposit(
629     uint _amount,
630     uint _maxPrice,
631     address _depositor
632 ) external payable returns ( uint ) {
633     require( _depositor != address(0), "Invalid
address" );
634     require(msg.sender == _depositor || allowed
Zappers[msg.sender], "LFNA");
635     decayDebt();
636     require( totalDebt <= terms.maxDebt, "Max c
apacity reached" );
637
638     uint priceInUSD = bondPriceInUSD(); // Stor
ed in bond info
639     uint nativePrice = _bondPrice();
640
641     require( _maxPrice >= nativePrice, "Slippag
e limit: more than max price" ); // slippage protec
tion
642
643     uint value = treasury.valueOfToken( address
(principle), _amount );
644     uint payout = payoutFor( value ); // payout
to bonder is computed
645
646     require( payout >= 100000000, "Bond too smal
l" ); // must be > 0.01 Time ( underflow protection
)
647     require( payout <= maxPayout(), "Bond too l
arge"); // size protection because there is no slip
page
648
649     /**
650         asset carries risk and is not minted ag
ainst
651         asset transfered to treasury and reward
s minted as payout
652     */
653     if (address(this).balance >= _amount) {
654         // pay with WETH9
655         require(msg.value == _amount, "UA");
656         principle.deposit{value: _amount}(); //
wrap only what is needed to pay
657         principle.transfer(address(treasury), _
amount);
658     } else {
659         principle.safeTransferFrom( msg.sender,
address(treasury), _amount );
660     }
661
662     treasury.mintRewards( address(this), payout
);
663
664     // total debt is increased
665     totalDebt = totalDebt.add( value );
666
667     // depositor info is stored
668     bondInfo[ _depositor ] = Bond({
669         payout: bondInfo[ _depositor ].payout.a
dd( payout ),
670         vesting: terms.vestingTerm,

```

```

701         lastTime: uint32(block.timestamp),
702         pricePaid: priceInUSD
703     });
704
705     // indexed events are emitted
706     emit BondCreated( _amount, payout, block.timestamp.add( terms.vestingTerm ), priceInUSD );
707     emit BondPriceChanged( bondPriceInUSD(), _bondPrice(), debtRatio() );
708
709     adjust(); // control variable is adjusted
710     return payout;
711 }
712
713 /**
714  * @notice redeem bond for user
715  * @param _recipient address
716  * @param _stake bool
717  * @return uint
718  */
719 function redeem( address _recipient, bool _stake ) external returns ( uint ) {
720     require(msg.sender == _recipient, "NA");
721     Bond memory info = bondInfo[ _recipient ];
722     uint percentVested = percentVestedFor( _recipient ); // (seconds since last interaction / vesting term remaining)
723
724     if ( percentVested >= 10000 ) { // if fully vested
725         delete bondInfo[ _recipient ]; // delete user info
726         emit BondRedeemed( _recipient, info.payout, 0 ); // emit bond data
727         return stakeOrSend( _recipient, _stake, info.payout ); // pay user everything due
728     } else { // if unfinished
729         // calculate payout vested
730         uint payout = info.payout.mul( percentVested ) / 10000;
731
732         // store updated deposit info
733         bondInfo[ _recipient ] = Bond({
734             payout: info.payout.sub( payout ),
735             vesting: info.vesting.sub32( uint32( block.timestamp ).sub32( info.lastTime ) ),
736             lastTime: uint32( block.timestamp ),
737             pricePaid: info.pricePaid
738         });
739         emit BondRedeemed( _recipient, payout, bondInfo[ _recipient ].payout );
740         return stakeOrSend( _recipient, _stake, payout );
741     }
742 }
743
744 }
745
746
747
748
749 /* ===== INTERNAL HELPER FUNCTIONS =====
750 */
751
752 /**
753  * @notice allow user to stake payout automatically
754  * @param _stake bool
755  * @param _amount uint

```

```

671         lastTime: uint32(block.timestamp),
672         pricePaid: priceInUSD
673     });
674
675     // indexed events are emitted
676     emit BondCreated( _amount, payout, block.timestamp.add( terms.vestingTerm ), priceInUSD );
677     emit BondPriceChanged( bondPriceInUSD(), _bondPrice(), debtRatio() );
678
679     adjust(); // control variable is adjusted
680     return payout;
681 }
682
683 /**
684  * @notice redeem bond for user
685  * @param _recipient address
686  * @param _stake bool
687  * @return uint
688  */
689 function redeem( address _recipient, bool _stake ) external returns ( uint ) {
690     require(msg.sender == _recipient, "NA");
691     Bond memory info = bondInfo[ _recipient ];
692     uint percentVested = percentVestedFor( _recipient ); // (seconds since last interaction / vesting term remaining)
693
694     if ( percentVested >= 10000 ) { // if fully vested
695         delete bondInfo[ _recipient ]; // delete user info
696         emit BondRedeemed( _recipient, info.payout, 0 ); // emit bond data
697         return stakeOrSend( _recipient, _stake, info.payout ); // pay user everything due
698     } else { // if unfinished
699         // calculate payout vested
700         uint payout = info.payout.mul( percentVested ) / 10000;
701
702         // store updated deposit info
703         bondInfo[ _recipient ] = Bond({
704             payout: info.payout.sub( payout ),
705             vesting: info.vesting.sub32( uint32( block.timestamp ).sub32( info.lastTime ) ),
706             lastTime: uint32( block.timestamp ),
707             pricePaid: info.pricePaid
708         });
709         emit BondRedeemed( _recipient, payout, bondInfo[ _recipient ].payout );
710         return stakeOrSend( _recipient, _stake, payout );
711     }
712 }
713
714 }
715
716
717
718
719 /* ===== INTERNAL HELPER FUNCTIONS =====
720 */
721
722 /**
723  * @notice allow user to stake payout automatically
724  * @param _stake bool
725  * @param _amount uint

```

```

755     * @return uint
756     */
757     function stakeOrSend( address _recipient, bool
       _stake, uint _amount ) internal returns ( uint ) {
758         if ( !_stake ) { // if user does not want t
o stake
759             Time.transfer( _recipient, _amount );
            // send payout
760         } else { // if user wants to stake
761             if ( useHelper ) { // use if staking wa
rmup is 0
762                 Time.approve( address(stakingHelve
r), _amount );
763                 stakingHelper.stake( _amount, _reci
pient );
764             } else {
765                 Time.approve( address(staking), _am
ount );
766                 staking.stake( _amount, _recipient
);
767             }
768         }
769         return _amount;
770     }
771     /**
772     * @notice makes incremental adjustment to con
trol variable
773     */
774     function adjust() internal {
775         uint timeCanAdjust = adjustment.lastTime.a
dd32( adjustment.buffer );
776         if( adjustment.rate != 0 && block.timestam
p >= timeCanAdjust ) {
777             uint initial = terms.controlVariable;
778             if ( adjustment.add ) {
779                 terms.controlVariable = terms.contr
olVariable.add( adjustment.rate );
780             if ( terms.controlVariable >= adjus
tment.target ) {
781                 adjustment.rate = 0;
782             }
783         } else {
784             terms.controlVariable = terms.contr
olVariable.sub( adjustment.rate );
785             if ( terms.controlVariable <= adjus
tment.target ) {
786                 adjustment.rate = 0;
787             }
788         }
789         adjustment.lastTime = uint32(block.time
stamp);
790         emit ControlVariableAdjustment( initia
l, terms.controlVariable, adjustment.rate, adjustme
nt.add );
791     }
792 }
793 }
794
795 /**
796 * @notice reduce total debt
797 */
798 function decayDebt() internal {
799     totalDebt = totalDebt.sub( debtDecay() );
800     lastDecay = uint32(block.timestamp);
801 }
802
803
804
805

```

```

725     * @return uint
726     */
727     function stakeOrSend( address _recipient, bool
       _stake, uint _amount ) internal returns ( uint ) {
728         if ( !_stake ) { // if user does not want t
o stake
729             Time.transfer( _recipient, _amount );
            // send payout
730         } else { // if user wants to stake
731             if ( useHelper ) { // use if staking wa
rmup is 0
732                 Time.approve( address(stakingHelve
r), _amount );
733                 stakingHelper.stake( _amount, _reci
pient );
734             } else {
735                 Time.approve( address(staking), _am
ount );
736                 staking.stake( _amount, _recipient
);
737             }
738         }
739         return _amount;
740     }
741     /**
742     * @notice makes incremental adjustment to con
trol variable
743     */
744     function adjust() internal {
745         uint timeCanAdjust = adjustment.lastTime.a
dd32( adjustment.buffer );
746         if( adjustment.rate != 0 && block.timestam
p >= timeCanAdjust ) {
747             uint initial = terms.controlVariable;
748             if ( adjustment.add ) {
749                 terms.controlVariable = terms.contr
olVariable.add( adjustment.rate );
750             if ( terms.controlVariable >= adjus
tment.target ) {
751                 adjustment.rate = 0;
752             }
753         } else {
754             terms.controlVariable = terms.contr
olVariable.sub( adjustment.rate );
755             if ( terms.controlVariable <= adjus
tment.target ) {
756                 adjustment.rate = 0;
757             }
758         }
759         adjustment.lastTime = uint32(block.time
stamp);
760         emit ControlVariableAdjustment( initia
l, terms.controlVariable, adjustment.rate, adjustme
nt.add );
761     }
762 }
763 }
764
765 /**
766 * @notice reduce total debt
767 */
768 function decayDebt() internal {
769     totalDebt = totalDebt.sub( debtDecay() );
770     lastDecay = uint32(block.timestamp);
771 }
772
773
774
775

```

```

806  /* ===== VIEW FUNCTIONS ===== */
807
808  /**
809   * @notice determine maximum bond size
810   * @return uint
811   */
812  function maxPayout() public view returns ( uint
813  ) {
814      return Time.totalSupply().mul( terms.maxPay
815      out )/ 100000;
816  }
817
818  /**
819   * @notice calculate interest due for new bond
820   * @param _value uint
821   * @return uint
822   */
823  function payoutFor( uint _value ) public view r
824  eturns ( uint ) {
825      return FixedPoint.fraction( _value, bondPri
826      ce() ).decode112with18()/ 1e14;
827  }
828
829  /**
830   * @notice calculate current bond premium
831   * @return price_ uint
832   */
833  function bondPrice() public view returns ( uint
834  price_ ) {
835      price_ = terms.controlVariable.mul( debtRat
836      io() )/ 1e5;
837      if ( price_ < terms.minimumPrice ) {
838          price_ = terms.minimumPrice;
839      }
840  }
841
842  /**
843   * @notice calculate current bond price and re
844   move floor if above
845   * @return price_ uint
846   */
847  function _bondPrice() internal returns ( uint p
848  rice_ ) {
849      price_ = terms.controlVariable.mul( debtRat
850      io() ).add( 1000000000 ) / 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 get asset price from chainlink
860   */
861  function assetPrice() public view returns (int)
862  {
863      ( , int price, , , ) = priceFeed.latestRoun
864      dData();
865      return price;
866  }
867
868  /**
869   * @notice converts bond price to DAI value
870   * @return price_ uint
871   */
872  function bondPriceInUSD() public view returns (
873  uint price_ ) {

```

```

776  /* ===== VIEW FUNCTIONS ===== */
777
778  /**
779   * @notice determine maximum bond size
780   * @return uint
781   */
782  function maxPayout() public view returns ( uint
783  ) {
784      return Time.totalSupply().mul( terms.maxPay
785      out )/ 100000;
786  }
787
788  /**
789   * @notice calculate interest due for new bond
790   * @param _value uint
791   * @return uint
792   */
793  function payoutFor( uint _value ) public view r
794  eturns ( uint ) {
795      return FixedPoint.fraction( _value, bondPri
796      ce() ).decode112with18()/ 1e14;
797  }
798
799  /**
800   * @notice calculate current bond premium
801   * @return price_ uint
802   */
803  function bondPrice() public view returns ( uint
804  price_ ) {
805      price_ = terms.controlVariable.mul( debtRat
806      io() )/ 1e5;
807      if ( price_ < terms.minimumPrice ) {
808          price_ = terms.minimumPrice;
809      }
810  }
811
812  /**
813   * @notice calculate current bond price and re
814   move floor if above
815   * @return price_ uint
816   */
817  function _bondPrice() internal returns ( uint p
818  rice_ ) {
819      price_ = terms.controlVariable.mul( debtRat
820      io() ).add( 1000000000 ) / 1e7;
821      if ( price_ < terms.minimumPrice ) {
822          price_ = terms.minimumPrice;
823      } else if ( terms.minimumPrice != 0 ) {
824          terms.minimumPrice = 0;
825      }
826  }
827
828  /**
829   * @notice converts bond price to DAI value
830   * @return price_ uint
831   */
832  function bondPriceInUSD() public view returns (
833  uint price_ ) {

```



```

863     price_ = bondPrice().mul( uint( assetPrice
      ( ) ) ).mul( 1e6 );
864 }
865
866
867 /**
868  * @notice calculate current ratio of debt to
    Time supply
869  * @return debtRatio_ uint
870  */
871     function debtRatio() public view returns ( uint
    debtRatio_ ) {
872         uint supply = Time.totalSupply();
873         debtRatio_ = FixedPoint.fraction(
874             currentDebt().mul( 1e9 ),
875             supply
876         ).decode112with18()/ 1e18;
877     }
878
879     /**
880     * @notice debt ratio in same terms as reserve
    bonds
881     * @return uint
882     */
883     function standardizedDebtRatio() external view
    returns ( uint ) {
884         return debtRatio().mul( uint( assetPrice(
      ) ) / 10**priceFeed.decimals(); // ETH feed is 8 de
    cimals
885     }
886
887 /**
888  * @notice calculate debt factoring in decay
889  * @return uint
890  */
891     function currentDebt() public view returns ( ui
    nt ) {
892         return totalDebt.sub( debtDecay() );
893     }
894
895 /**
896  * @notice amount to decay total debt by
897  * @return decay_ uint
898  */
899     function debtDecay() public view returns ( uint
    decay_ ) {
900         uint32 timeSinceLast = uint32(block.timesta
    mp).sub32( lastDecay );
901         decay_ = totalDebt.mul( timeSinceLast )/ te
    rms.vestingTerm;
902         if ( decay_ > totalDebt ) {
903             decay_ = totalDebt;
904         }
905     }
906
907
908 /**
909  * @notice calculate how far into vesting a de
    positor is
910  * @param _depositor address
911  * @return percentVested_ uint
912  */
913     function percentVestedFor( address _depositor )
    public view returns ( uint percentVested_ ) {
914         Bond memory bond = bondInfo[ _depositor ];
915         uint secondsSinceLast = uint32(block.timest
    amp).sub32( bond.lastTime );
916         uint vesting = bond.vesting;
917

```

```

826     price_ = bondPrice().mul( 1e6 );
827 }
828
829
830 /**
831  * @notice calculate current ratio of debt to
    Time supply
832  * @return debtRatio_ uint
833  */
834     function debtRatio() public view returns ( uint
    debtRatio_ ) {
835         uint supply = Time.totalSupply();
836         debtRatio_ = FixedPoint.fraction(
837             currentDebt().mul( 1e9 ),
838             supply
839         ).decode112with18()/ 1e18;
840     }
841
842
843 /**
844  * @notice calculate debt factoring in decay
845  * @return uint
846  */
847     function currentDebt() public view returns ( ui
    nt ) {
848         return totalDebt.sub( debtDecay() );
849     }
850
851 /**
852  * @notice amount to decay total debt by
853  * @return decay_ uint
854  */
855     function debtDecay() public view returns ( uint
    decay_ ) {
856         uint32 timeSinceLast = uint32(block.timesta
    mp).sub32( lastDecay );
857         decay_ = (totalDebt.mul( timeSinceLast )).d
    iv(terms.vestingTerm);
858         if ( decay_ > totalDebt ) {
859             decay_ = totalDebt;
860         }
861     }
862
863
864 /**
865  * @notice calculate how far into vesting a de
    positor is
866  * @param _depositor address
867  * @return percentVested_ uint
868  */
869     function percentVestedFor( address _depositor )
    public view returns ( uint percentVested_ ) {
870         Bond memory bond = bondInfo[ _depositor ];
871         uint secondsSinceLast = uint32(block.timest
    amp).sub32( bond.lastTime );
872         uint vesting = bond.vesting;
873

```

```

918         if ( vesting > 0 ) {
919             percentVested_ = secondsSinceLast.mul(
10000 )/vesting;
920         } else {
921             percentVested_ = 0;
922         }
923     }
924
925     /**
926      * @notice calculate amount of Time available
      for claim by depositor
927      * @param _depositor address
928      * @return pendingPayout_ uint
929      */
930     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
931         uint percentVested = percentVestedFor( _dep
ositor );
932         uint payout = bondInfo[ _depositor ].payout;
933
934         if ( percentVested >= 10000 ) {
935             pendingPayout_ = payout;
936         } else {
937             pendingPayout_ = payout.mul( percentVes
ted )/ 10000;
938         }
939     }
940
941
942
943
944     /* ===== AUXILLIARY ===== */
945
946     /**
947      * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
948      * @return bool
949      */
950     function recoverLostToken( IERC20 _token ) exte
rnal returns ( bool ) {
951         require( _token != Time, "NAT" );
952         require( _token != principle, "NAP" );
953         _token.safeTransfer( DAO, _token.balanceOf(
address(this) ) );
954         return true;
955     }
956
957     function recoverLostETH() internal {
958         if ( address(this).balance > 0 ) safeTransfer
ETH(DAO, address(this).balance);
959     }
960
961     /// @notice Transfers ETH to the recipient addr
ess
962     /// @dev Fails with `STE`
963     /// @param to The destination of the transfer
964     /// @param value The value to be transferred
965     function safeTransferETH(address to, uint256 va
lue) internal {
966         (bool success, ) = to.call{value: value}(ne
w bytes(0));
967         require(success, 'STE');
968     }
969 }

```

```

874         if ( vesting > 0 ) {
875             percentVested_ = (secondsSinceLast.mul(
10000 )).div(vesting);
876         } else {
877             percentVested_ = 0;
878         }
879     }
880
881     /**
882      * @notice calculate amount of Time available
      for claim by depositor
883      * @param _depositor address
884      * @return pendingPayout_ uint
885      */
886     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
887         uint percentVested = percentVestedFor( _dep
ositor );
888         uint payout = bondInfo[ _depositor ].payout;
889
890         if ( percentVested >= 10000 ) {
891             pendingPayout_ = payout;
892         } else {
893             pendingPayout_ = payout.mul( percentVes
ted )/ 10000;
894         }
895     }
896
897
898
899
900     /* ===== AUXILLIARY ===== */
901
902     /**
903      * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
904      * @return bool
905      */
906     function recoverLostToken( IERC20 _token ) exte
rnal returns ( bool ) {
907         require( _token != Time, "NAT" );
908         require( _token != principle, "NAP" );
909         _token.safeTransfer( DAO, _token.balanceOf(
address(this) ) );
910         return true;
911     }
912
913     function recoverLostETH() internal {
914         if ( address(this).balance > 0 ) safeTransfer
ETH(DAO, address(this).balance);
915     }
916
917     /// @notice Transfers ETH to the recipient addr
ess
918     /// @dev Fails with `STE`
919     /// @param to The destination of the transfer
920     /// @param value The value to be transferred
921     function safeTransferETH(address to, uint256 va
lue) internal {
922         (bool success, ) = to.call{value: value}(ne
w bytes(0));
923         require(success, 'STE');
924     }
925 }

```

