

```

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 dividend
92     /// @param y The divisor
93     /// @return z The quotient of x and y
94     function div(uint256 x, uint256 y) internal pure
95     returns (uint256 z) {
96         require(y > 0);
97         z = x / y;
98     }
99     /// @notice Returns x % y, reverts if overflows
100     or underflows
101     /// @param x The dividend
102     /// @param y The divisor
103     /// @return z The remainder of x and y
104     function mod(uint256 x, uint256 y) internal pure
105     returns (uint256 z) {
106         require(y > 0);
107         z = x % y;
108     }
109 }
110 library Address {

```

```

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 dividend
92     /// @param y The divisor
93     /// @return z The quotient of x and y
94     function div(uint256 x, uint256 y) internal pure
95     returns (uint256 z) {
96         require(y > 0);
97         z = x / y;
98     }
99     /// @notice Returns x % y, reverts if overflows
100     or underflows
101     /// @param x The dividend
102     /// @param y The divisor
103     /// @return z The remainder of x and y
104     function mod(uint256 x, uint256 y) internal pure
105     returns (uint256 z) {
106         require(y > 0);
107         z = x % y;
108     }
109 }
110 library Address {

```

```

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
399 interface AggregatorV3Interface {
400
401     function decimals() external view returns (uint
8);
402     function description() external view returns (str
ing memory);
403     function version() external view returns (uint25
6);
404
405     // getRoundData and latestRoundData should both r
aise "No data present"
406     // if they do not have data to report, instead of
returning unset values
407     // which could be misinterpreted as actual report
ed values.
408     function getRoundData(uint80 _roundId)
external
409         view
410         returns (
411             uint80 roundId,
412             int256 answer,
413             uint256 startedAt,
414             uint256 updatedAt,
415             uint80 answeredInRound
416         );
417     function latestRoundData()
external
418         view
419         returns (
420             uint80 roundId,
421             int256 answer,
422             uint256 startedAt,
423             uint256 updatedAt,
424             uint80 answeredInRound
425         );
426

```



```

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     using FixedPoint for *;
446     using SafeERC20 for IERC20;
447     using SafeERC20 for IWAVAX9;
448     using LowGasSafeMath for uint;
449     using LowGasSafeMath for uint32;
450
451     /* ===== EVENTS ===== */
452
453     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUS
D );
454
455     event BondRedeemed( address indexed recipient,
uint payout, uint remaining );
456
457     event BondPriceChanged( uint indexed priceInUS
D, uint indexed internalPrice, uint indexed debtRat
io );
458
459     event ControlVariableAdjustment( uint initialBC
V, uint newBCV, uint adjustment, bool addition );
460
461     /* ===== STATE VARIABLES ===== */
462     IERC20 public immutable Time; // token given as
payment for bond
463     IWAVAX9 public immutable principle; // token us
ed to create bond
464
465     ITreasury public immutable treasury; // mints T
ime when receives principle
466     address public immutable DAO; // receives profi
t share from bond
467
468     AggregatorV3Interface public priceFeed;
469
470     ISTaking public staking; // to auto-stake payou
t
471
472     ISTakingHelper public stakingHelper; // to stak
e and claim if no staking warmup
473
474     bool public useHelper;
475
476

```

```

427     );
428 }
429
430 interface ITreasury {
431     function deposit( uint _amount, address _token,
uint _profit ) external returns ( uint );
432     function valueOfToken( address _token, uint _am
ount ) external view returns ( uint value_ );
433     function mintRewards( address _recipient, uint
_amount ) external;
434 }
435
436 interface ISTaking {
437     function stake( uint _amount, address _recipien
t ) external returns ( bool );
438 }
439
440 interface ISTakingHelper {
441     function stake( uint _amount, address _recipien
t ) external;
442 }
443
444 interface IWMATIC9 is IERC20 {
445     /// @notice Deposit ether to get wrapped ether
446     function deposit() external payable;
447 }
448
449 contract MaiaBondDepository is Ownable {
450     using FixedPoint for *;
451     using SafeERC20 for IERC20;
452     using SafeERC20 for IWMATIC9;
453     using LowGasSafeMath for uint;
454     using LowGasSafeMath for uint32;
455
456     /* ===== EVENTS ===== */
457
458     event BondCreated( uint deposit, uint indexed p
ayout, uint indexed expires, uint indexed priceInUS
D );
459
460     event BondRedeemed( address indexed recipient,
uint payout, uint remaining );
461
462     event BondPriceChanged( uint indexed priceInUS
D, uint indexed internalPrice, uint indexed debtRat
io );
463
464     event ControlVariableAdjustment( uint initialBC
V, uint newBCV, uint adjustment, bool addition );
465
466     /* ===== STATE VARIABLES ===== */
467     IERC20 public immutable Time; // token given as
payment for bond
468     IWMATIC9 public immutable principle; // token u
sed to create bond
469
470     ITreasury public immutable treasury; // mints T
ime when receives principle
471     address public immutable DAO; // receives profi
t share from bond
472
473     AggregatorV3Interface public priceFeed;
474
475     ISTaking public staking; // to auto-stake payou
t
476
477     ISTakingHelper public stakingHelper; // to stak
e and claim if no staking warmup
478
479     bool public useHelper;
480
481

```

```

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
517     constructor (
518         address _Time,
519         address _principle,
520         address _treasury,
521         address _DAO,
522         address _feed
523     ) {
524         require( _Time != address(0) );
525         Time = IERC20(_Time);
526         require( _principle != address(0) );
527         principle = IWAAX9(_principle);

```

```

482     Terms public terms; // stores terms for new bonds
483     Adjust public adjustment; // stores adjustment
    to BCV data
484
485     mapping( address => Bond ) public bondInfo; //
    stores bond information for depositors
486
487     uint public totalDebt; // total value of outstanding bonds; used for pricing
488     uint32 public lastDecay; // reference time for debt decay
489
490
491     mapping (address => bool) public allowedZappers;
492
493     /* ===== STRUCTS ===== */
494     // Info for creating new bonds
495     struct Terms {
496         uint controlVariable; // scaling variable for price
497         uint minimumPrice; // vs principle value. 4 decimals (1500 = 0.15)
498         uint maxPayout; // in thousandths of a %. i.e. 500 = 0.5%
499         uint maxDebt; // 9 decimal debt ratio, max % total supply created as debt
500         uint32 vestingTerm; // in seconds
501     }
502
503     // Info for bond holder
504     struct Bond {
505         uint payout; // Time remaining to be paid
506         uint pricePaid; // In DAI, for front end vesting
507         uint32 vesting; // Seconds left to vest
508         uint32 lastTime; // Last interaction
509     }
510
511     // Info for incremental adjustments to control variable
512     struct Adjust {
513         bool add; // addition or subtraction
514         uint rate; // increment
515         uint target; // BCV when adjustment finished
516         uint32 buffer; // minimum length (in seconds) between adjustments
517         uint32 lastTime; // time when last adjustment made
518     }
519
520     /* ===== INITIALIZATION ===== */
521
522     constructor (
523         address _Time,
524         address _principle,
525         address _treasury,
526         address _DAO,
527         address _feed
528     ) {
529         require( _Time != address(0) );
530         Time = IERC20(_Time);
531         require( _principle != address(0) );
532         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;

```

```

538     require( _treasury != address(0) );
539     treasury = ITreasury(_treasury);
540     require( _DAO != address(0) );
541     DAO = _DAO;
542     require( _feed != address(0) );
543     priceFeed = AggregatorV3Interface( _feed );
544 }
545
546 /**
547  * @notice initializes bond parameters
548  * @param _controlVariable uint
549  * @param _vestingTerm uint
550  * @param _minimumPrice uint
551  * @param _maxPayout uint
552  * @param _maxDebt uint
553  */
554 function initializeBondTerms(
555     uint _controlVariable,
556     uint _minimumPrice,
557     uint _maxPayout,
558     uint _maxDebt,
559     uint32 _vestingTerm
560 ) external onlyPolicy() {
561     require( currentDebt() == 0, "Debt must be
0 for initialization" );
562     require( _controlVariable >= 40, "Can lock
adjustment" );
563     require( _maxPayout <= 1000, "Payout cannot
be above 1 percent" );
564     require( _vestingTerm >= 129600, "Vesting m
ust be longer than 36 hours" );
565     terms = Terms ( {
566         controlVariable: _controlVariable,
567         vestingTerm: _vestingTerm,
568         minimumPrice: _minimumPrice,
569         maxPayout: _maxPayout,
570         maxDebt: _maxDebt
571     });
572     lastDecay = uint32(block.timestamp);
573 }
574
575
576
577
578 /* ===== POLICY FUNCTIONS ===== */
579
580 enum PARAMETER { VESTING, PAYOUT, DEBT, MINPRIC
E }
581 /**
582  * @notice set parameters for new bonds
583  * @param _parameter PARAMETER
584  * @param _input uint
585  */
586 function setBondTerms ( PARAMETER _parameter, u
int _input ) external onlyPolicy() {
587     if ( _parameter == PARAMETER.VESTING ) { //
0
588         require( _input >= 129600, "Vesting mus
t be longer than 36 hours" );
589         terms.vestingTerm = uint32(_input);
590     } else if ( _parameter == PARAMETER.PAYOUT
) { // 1
591         require( _input <= 1000, "Payout cannot
be above 1 percent" );
592         terms.maxPayout = _input;
593     } else if ( _parameter == PARAMETER.DEBT )
{ // 2
594         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 /**

```

```

595     } else if ( _parameter == PARAMETER.MINPRIC
E ) { // 3
596         terms.minimumPrice = _input;
597     }
598 }
599
600 /**
601  * @notice set control variable adjustment
602  * @param _addition bool
603  * @param _increment uint
604  * @param _target uint
605  * @param _buffer uint
606  */
607 function setAdjustment (
608     bool _addition,
609     uint _increment,
610     uint _target,
611     uint32 _buffer
612 ) external onlyPolicy() {
613     require( _increment <= terms.controlVariabl
e.mul( 25 )/ 1000, "Increment too large" );
614     require(_target >= 40, "Next Adjustment cou
ld be locked");
615     adjustment = Adjust({
616         add: _addition,
617         rate: _increment,
618         target: _target,
619         buffer: _buffer,
620         lastTime: uint32(block.timestamp)
621     });
622 }
623
624 /**
625  * @notice set contract for auto stake
626  * @param _staking address
627  * @param _helper bool
628  */
629 function setStaking( address _staking, bool _he
lper ) external onlyPolicy() {
630     require( _staking != address(0) , "IA");
631     if ( _helper ) {
632         useHelper = true;
633         stakingHelper = IStakingHelper(_stakin
g);
634     } else {
635         useHelper = false;
636         staking = IStaking(_staking);
637     }
638 }
639
640 function allowZapper(address zapper) external o
nlyPolicy {
641     require(zapper != address(0), "ZNA");
642
643     allowedZappers[zapper] = true;
644 }
645
646 function removeZapper(address zapper) external
onlyPolicy {
647
648     allowedZappers[zapper] = false;
649 }
650
651
652
653
654 /* ===== USER FUNCTIONS ===== */
655
656 /**

```

```

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,

```

```

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 payable returns ( uint ) {
668     require( _depositor != address(0), "Invalid
address" );
669     require(msg.sender == _depositor || allowed
Zappers[msg.sender], "LFNA");
670     decayDebt();
671     require( totalDebt <= terms.maxDebt, "Max c
apacity reached" );
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.valueOfToken( address
(principle), _amount );
679     uint payout = payoutFor( value ); // payout
to bonder is computed
680
681     require( payout >= 100000000, "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     /**
685         asset carries risk and is not minted ag
ainst
686         asset transfered to treasury and reward
s minted as payout
687     */
688     if (address(this).balance >= _amount) {
689         // pay with WETH9
690         require(msg.value == _amount, "UA");
691         principle.deposit{value: _amount}(); //
wrap only what is needed to pay
692         principle.transfer(address(treasury), _
amount);
693     } else {
694         principle.safeTransferFrom( msg.sender,
address(treasury), _amount );
695     }
696
697     treasury.mintRewards( address(this), payout
);
698
699     // total debt is increased
700     totalDebt = totalDebt.add( value );
701
702     // depositor info is stored
703     bondInfo[ _depositor ] = Bond({
704         payout: bondInfo[ _depositor ].payout.a
dd( payout ),
705         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

```

```

706         lastTime: uint32(block.timestamp),
707         pricePaid: priceInUSD
708     });
709
710     // indexed events are emitted
711     emit BondCreated( _amount, payout, block.timestamp.add( terms.vestingTerm ), priceInUSD );
712     emit BondPriceChanged( bondPriceInUSD(), _bondPrice(), debtRatio() );
713
714     adjust(); // control variable is adjusted
715     return payout;
716 }
717
718 /**
719  * @notice redeem bond for user
720  * @param _recipient address
721  * @param _stake bool
722  * @return uint
723  */
724 function redeem( address _recipient, bool _stake ) external returns ( uint ) {
725     require(msg.sender == _recipient, "NA");
726     Bond memory info = bondInfo[ _recipient ];
727     uint percentVested = percentVestedFor( _recipient ); // (seconds since last interaction / vesting term remaining)
728
729     if ( percentVested >= 10000 ) { // if fully vested
730         delete bondInfo[ _recipient ]; // delete user info
731         emit BondRedeemed( _recipient, info.payout, 0 ); // emit bond data
732         return stakeOrSend( _recipient, _stake, info.payout ); // pay user everything due
733     } else { // if unfinished
734         // calculate payout vested
735         uint payout = info.payout.mul( percentVested ) / 10000;
736
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         emit BondRedeemed( _recipient, payout, bondInfo[ _recipient ].payout );
745         return stakeOrSend( _recipient, _stake, payout );
746     }
747 }
748
749 }
750
751
752
753
754 /* ===== INTERNAL HELPER FUNCTIONS =====
755 */
756
757 /**
758  * @notice allow user to stake payout automatically
759  * @param _stake bool
760  * @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

```

```

760     * @return uint
761     */
762     function stakeOrSend( address _recipient, bool
       _stake, uint _amount ) internal returns ( uint ) {
763         if ( !_stake ) { // if user does not want t
o stake
764             Time.transfer( _recipient, _amount );
            // send payout
765         } else { // if user wants to stake
766             if ( useHelper ) { // use if staking wa
rmup is 0
767                 Time.approve( address(stakingHelve
r), _amount );
768                 stakingHelper.stake( _amount, _reci
pient );
769             } else {
770                 Time.approve( address(staking), _am
ount );
771                 staking.stake( _amount, _recipient
);
772             }
773         }
774         return _amount;
775     }
776     /**
777     * @notice makes incremental adjustment to con
trol variable
778     */
779     function adjust() internal {
780         uint timeCanAdjust = adjustment.lastTime.a
dd32( adjustment.buffer );
781         if( adjustment.rate != 0 && block.timestam
p >= timeCanAdjust ) {
782             uint initial = terms.controlVariable;
783             if ( adjustment.add ) {
784                 terms.controlVariable = terms.contr
olVariable.add( adjustment.rate );
785             if ( terms.controlVariable >= adjus
tment.target ) {
786                 adjustment.rate = 0;
787             }
788         } else {
789             terms.controlVariable = terms.contr
olVariable.sub( adjustment.rate );
790             if ( terms.controlVariable <= adjus
tment.target ) {
791                 adjustment.rate = 0;
792             }
793         }
794         adjustment.lastTime = uint32(block.time
stamp);
795         emit ControlVariableAdjustment( initia
l, terms.controlVariable, adjustment.rate, adjustme
nt.add );
796     }
797 }
798 }
799
800 /**
801 * @notice reduce total debt
802 */
803 function decayDebt() internal {
804     totalDebt = totalDebt.sub( debtDecay() );
805     lastDecay = uint32(block.timestamp);
806 }
807
808
809
810

```

```

806  /* ===== VIEW FUNCTIONS ===== */
807
808  /**
809   * @notice determine maximum bond size
810   * @return uint
811   */
812  function maxPayout() public view returns ( uint
) {
813      return Time.totalSupply().mul( terms.maxPay
out )/ 100000;
814  }
815
816  /**
817   * @notice calculate interest due for new bond
818   * @param _value uint
819   * @return uint
820   */
821  function payoutFor( uint _value ) public view r
eturns ( uint ) {
822      return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18()/ 1e14;
823  }
824
825
826  /**
827   * @notice calculate current bond premium
828   * @return price_ uint
829   */
830  function bondPrice() public view returns ( uint
price_ ) {
831      price_ = terms.controlVariable.mul( debtRat
io() )/ 1e5;
832      if ( price_ < terms.minimumPrice ) {
833          price_ = terms.minimumPrice;
834      }
835  }
836
837  /**
838   * @notice calculate current bond price and re
move floor if above
839   * @return price_ uint
840   */
841  function _bondPrice() internal returns ( uint p
rice_ ) {
842      price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
843      if ( price_ < terms.minimumPrice ) {
844          price_ = terms.minimumPrice;
845      } else if ( terms.minimumPrice != 0 ) {
846          terms.minimumPrice = 0;
847      }
848  }
849
850  /**
851   * @notice get asset price from chainlink
852   */
853  function assetPrice() public view returns (int)
{
854      ( , int price, , , ) = priceFeed.latestRoun
dData();
855      return price;
856  }
857
858  /**
859   * @notice converts bond price to DAI value
860   * @return price_ uint
861   */
862  function bondPriceInUSD() public view returns (
uint price_ ) {

```

```

811  /* ===== VIEW FUNCTIONS ===== */
812
813  /**
814   * @notice determine maximum bond size
815   * @return uint
816   */
817  function maxPayout() public view returns ( uint
) {
818      return Time.totalSupply().mul( terms.maxPay
out )/ 100000;
819  }
820
821  /**
822   * @notice calculate interest due for new bond
823   * @param _value uint
824   * @return uint
825   */
826  function payoutFor( uint _value ) public view r
eturns ( uint ) {
827      return FixedPoint.fraction( _value, bondPri
ce() ).decode112with18()/ 1e14;
828  }
829
830
831  /**
832   * @notice calculate current bond premium
833   * @return price_ uint
834   */
835  function bondPrice() public view returns ( uint
price_ ) {
836      price_ = terms.controlVariable.mul( debtRat
io() )/ 1e5;
837      if ( price_ < terms.minimumPrice ) {
838          price_ = terms.minimumPrice;
839      }
840  }
841
842  /**
843   * @notice calculate current bond price and re
move floor if above
844   * @return price_ uint
845   */
846  function _bondPrice() internal returns ( uint p
rice_ ) {
847      price_ = terms.controlVariable.mul( debtRat
io() ).add( 10000000000 ) / 1e7;
848      if ( price_ < terms.minimumPrice ) {
849          price_ = terms.minimumPrice;
850      } else if ( terms.minimumPrice != 0 ) {
851          terms.minimumPrice = 0;
852      }
853  }
854
855  /**
856   * @notice get asset price from chainlink
857   */
858  function assetPrice() public view returns (int)
{
859      ( , int price, , , ) = priceFeed.latestRoun
dData();
860      return price;
861  }
862
863  /**
864   * @notice converts bond price to DAI value
865   * @return price_ uint
866   */
867  function bondPriceInUSD() public view returns (
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     * @notice calculate how far into vesting a de
      positor is
909     * @param _depositor address
910     * @return percentVested_ uint
911     */
912     function percentVestedFor( address _depositor )
      public view returns ( uint percentVested_ ) {
913         Bond memory bond = bondInfo[ _depositor ];
914         uint secondsSinceLast = uint32(block.timest
      amp).sub32( bond.lastTime );
915         uint vesting = bond.vesting;
916
917

```

```

868         price_ = bondPrice().mul( uint( assetPrice
      ( ) ) ).mul( 1e6 );
869     }
870
871
872     /**
873     * @notice calculate current ratio of debt to
      Time supply
874     * @return debtRatio_ uint
875     */
876     function debtRatio() public view returns ( uint
      debtRatio_ ) {
877         uint supply = Time.totalSupply();
878         debtRatio_ = FixedPoint.fraction(
879             currentDebt().mul( 1e9 ),
880             supply
881         ).decode112with18()/ 1e18;
882     }
883
884     /**
885     * @notice debt ratio in same terms as reserve
      bonds
886     * @return uint
887     */
888     function standardizedDebtRatio() external view
      returns ( uint ) {
889         return debtRatio().mul( uint( assetPrice()
      ) ) / 10**priceFeed.decimals(); // ETH feed is 8 de
      cimals
890     }
891
892     /**
893     * @notice calculate debt factoring in decay
894     * @return uint
895     */
896     function currentDebt() public view returns ( ui
      nt ) {
897         return totalDebt.sub( debtDecay() );
898     }
899
900     /**
901     * @notice amount to decay total debt by
902     * @return decay_ uint
903     */
904     function debtDecay() public view returns ( uint
      decay_ ) {
905         uint32 timeSinceLast = uint32(block.timesta
      mp).sub32( lastDecay );
906         decay_ = (totalDebt.mul( timeSinceLast )).d
      iv(terms.vestingTerm);
907         if ( decay_ > totalDebt ) {
908             decay_ = totalDebt;
909         }
910     }
911
912     /**
913     * @notice calculate how far into vesting a de
      positor is
914     * @param _depositor address
915     * @return percentVested_ uint
916     */
917     function percentVestedFor( address _depositor )
      public view returns ( uint percentVested_ ) {
918         Bond memory bond = bondInfo[ _depositor ];
919         uint secondsSinceLast = uint32(block.timest
      amp).sub32( bond.lastTime );
920         uint vesting = bond.vesting;
921
922

```

```

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 }

```

```

923         if ( vesting > 0 ) {
924             percentVested_ = (secondsSinceLast.mul(
10000 )).div(vesting);
925         } else {
926             percentVested_ = 0;
927         }
928     }
929
930     /**
931      * @notice calculate amount of Time available
      for claim by depositor
932      * @param _depositor address
933      * @return pendingPayout_ uint
934      */
935     function pendingPayoutFor( address _depositor )
external view returns ( uint pendingPayout_ ) {
936         uint percentVested = percentVestedFor( _dep
ositor );
937         uint payout = bondInfo[ _depositor ].payout;
938
939         if ( percentVested >= 10000 ) {
940             pendingPayout_ = payout;
941         } else {
942             pendingPayout_ = payout.mul( percentVes
ted )/ 10000;
943         }
944     }
945
946
947
948
949     /* ===== AUXILLIARY ===== */
950
951     /**
952      * @notice allow anyone to send lost tokens (e
xcluding principle or Time) to the DAO
953      * @return bool
954      */
955     function recoverLostToken( IERC20 _token ) exte
rnal returns ( bool ) {
956         require( _token != Time, "NAT" );
957         require( _token != principle, "NAP" );
958         _token.safeTransfer( DAO, _token.balanceOf(
address(this) ) );
959         return true;
960     }
961
962     function recoverLostETH() internal {
963         if ( address(this).balance > 0 ) safeTransfer
ETH(DAO, address(this).balance);
964     }
965
966     /// @notice Transfers ETH to the recipient addr
ess
967     /// @dev Fails with `STE`
968     /// @param to The destination of the transfer
969     /// @param value The value to be transferred
970     function safeTransferETH(address to, uint256 va
lue) internal {
971         (bool success, ) = to.call{value: value}(ne
w bytes(0));
972         require(success, 'STE');
973     }
974 }

```