EUROe Stablecoin Smart Contract Test Coverage Report

100% Statements `39/39` **100%** Branches `12/12` **100%** Functions `16/16` **100%** Lines `40/40`

File		Statements		Branches		Functions		Lines	
contracts/		100%	39/39	100%	12/12	100%	16/16	100%	40/40

EUROe.sol

100% Statements `39/39` **100%** Branches `12/12` **100%** Functions `16/16` **100%** Lines `40/40`

```
1      // SPDX-License-Identifier: MIT
2
3      /*
4      Copyright (c) 2023 Membrane Finance Oy
5
6      Permission is hereby granted, free of charge, to any person obtaining a copy
7      of this software and associated documentation files (the "Software"), to deal
8      in the Software without restriction, including without limitation the rights
9      to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
10     copies of the Software, and to permit persons to whom the Software is
11     furnished to do so, subject to the following conditions:
12
13     The above copyright notice and this permission notice shall be included in all
14     copies or substantial portions of the Software.
15
16     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
17     IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
18     FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
19     AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
20     LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
21     OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
22     SOFTWARE.
23     */
24
25     pragma solidity 0.8.4;
26
27     import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
28     import "@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
29     import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol";
30     import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
31     import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
32     import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-ERC20PermitUpgradeable.sol";
33     import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
34     import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
35     import "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";
36
37     /**
38     @title A stablecoin ERC20 token contract for EUROe
39     @author Membrane Finance
40     @notice This contract implements the EUROe stablecoin along with its core functionality, such as minting and burning
41     @dev This contract is upgradable. It is implemented as an EIP-1967 transparent upgradable proxy. The PROXYOWNER_ROLE controls upgrades t
42      */
43     contract EUROe is
44         Initializable,
45         ERC20Upgradeable,
46         ERC20BurnableUpgradeable,
47         PausableUpgradeable,
48         AccessControlUpgradeable,
49         ERC20PermitUpgradeable,
50         UUPSUpgradeable
51     {
52         using SafeERC20Upgradeable for IERC20Upgradeable;
53
54         bytes32 public constant PROXYOWNER_ROLE = keccak256("PROXYOWNER_ROLE");
55         bytes32 public constant BLOCKLISTER_ROLE = keccak256("BLOCKLISTER_ROLE");
56         bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
57         bytes32 public constant UNPAUSER_ROLE = keccak256("UNPAUSER_ROLE");
58         bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
59         bytes32 public constant BLOCKED_ROLE = keccak256("BLOCKED_ROLE");
60         bytes32 public constant RESCUER_ROLE = keccak256("RESCUER_ROLE");
61         bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");
62
63         /**
64          * @dev Emitted once a minting set has been completed
65          * @param id External identifier for the minting set
66          */
67         event MintingSetCompleted(uint256 indexed id);
68
69         /// @custom:oz-upgrades-unsafe-allow constructor
70         constructor() {
71 6×          _disableInitializers();
72         }
73
74         /**
75          * @dev Initializes the (upgradeable) contract.
76          * @param proxyOwner Address for whom to give the proxyOwner role
77          * @param admin Address for whom to give the admin role
78          * @param blocklister Address for whom to give the blocklister role
79          * @param pauser Address for whom to give the pauser role
80          * @param unpauser Address for whom to give the unpauser role
81          * @param minter Address for whom to give the minter role
82          */
83         function initialize(
84             address proxyOwner,
85             address admin,
86             address blocklister,
```

```
87                      address pauser,
88                      address unpauser,
89                      address minter,
90                      address rescuer,
91                      address burner
92                  ) external initializer {
93  3×                  __ERC20_init("EUROe Stablecoin", "EUROe");
94  3×                  __ERC20Burnable_init();
95  3×                  __Pausable_init();
96  3×                  __AccessControl_init();
97  3×                  __ERC20Permit_init("EUROe Stablecoin");
98  3×                  __UUPSUpgradeable_init();
99
100 3×                  _grantRole(PROXYOWNER_ROLE, proxyOwner);
101 3×                  _grantRole(DEFAULT_ADMIN_ROLE, admin);
102 3×                  _grantRole(BLOCKLISTER_ROLE, blocklister);
103 3×                  _grantRole(PAUSER_ROLE, pauser);
104 3×                  _grantRole(UNPAUSER_ROLE, unpauser);
105 3×                  _grantRole(MINTER_ROLE, minter);
106 3×                  _grantRole(RESCUER_ROLE, rescuer);
107 3×                  _grantRole(BURNER_ROLE, burner);
108
109                     // Add this contract as blocked so it can't receive its own tokens by accident
110 3×                  _grantRole(BLOCKED_ROLE, address(this));
111
112 3×                  _setRoleAdmin(BLOCKED_ROLE, BLOCKLISTER_ROLE);
113             }
114
115             /// @inheritdoc ERC20Upgradeable
116             function decimals() public pure override returns (uint8) {
117 1×                  return 6;
118             }
119
120             /**
121              * @dev Pauses the contract
122              */
123             function pause() external onlyRole(PAUSER_ROLE) {
124 11×                 _pause();
125             }
126
127             /**
128              * @dev Unpauses the contract
129              */
130             function unpause() external onlyRole(UNPAUSER_ROLE) {
131 1×                  _unpause();
132             }
133
134             /// @inheritdoc ERC20BurnableUpgradeable
135             function burn(uint256 amount) public override onlyRole(BURNER_ROLE) {
136 4×                  super.burn(amount);
137             }
138
139             /// @inheritdoc ERC20BurnableUpgradeable
140             function burnFrom(address account, uint256 amount)
141                 public
142                 override
143                 onlyRole(BURNER_ROLE)
144             {
145 9×                  super.burnFrom(account, amount);
146             }
147
148             /**
149              * @dev Consumes a received permit and burns tokens based on the permit
150              * @param owner Source of the permit and allowance
151              * @param spender Target of the permit and allowance
152              * @param value How many tokens were permitted to be burned
153              * @param deadline Until what timestamp the permit is valid
154              * @param v The v portion of the permit signature
155              * @param r The r portion of the permit signature
156              * @param s The s portion of the permit signature
157              */
158             function burnFromWithPermit(
159                 address owner,
160                 address spender,
161                 uint256 value,
162                 uint256 deadline,
163                 uint8 v,
164                 bytes32 r,
165                 bytes32 s
166             ) public onlyRole(BURNER_ROLE) {
167 6×                  require(msg.sender == spender, "Invalid spender");
168 5×                  super.permit(owner, spender, value, deadline, v, r, s);
169 5×                  super.burnFrom(owner, value);
170             }
171
172             /**
173              * @dev Mints tokens to the given account
174              * @param account The account to mint tokens to
175              * @param amount How many tokens to mint
176              */
177             function mint(address account, uint256 amount)
178                 external
179                 onlyRole(MINTER_ROLE)
```

```
180                 {
181 8×                   _mint(account, amount);
182                 }
183
184                 /**
185                  * @dev Performs a batch of mints
186                  * @param targets Array of addresses for which to mint
187                  * @param amounts Array of amounts to mint for the corresponding addresses
188                  * @param id An external identifier given for the minting set
189                  * @param checksum A checksum to make sure none of the input data has changed
190                  */
191                 function mintSet(
192                     address[] calldata targets,
193                     uint256[] calldata amounts,
194                     uint256 id,
195                     bytes32 checksum
196                 ) external onlyRole(MINTER_ROLE) {
197 23×                 require(targets.length == amounts.length, "Unmatching mint lengths");
198 22×                 require(targets.length > 0, "Nothing to mint");
199
200 21×                 bytes32 calculated = keccak256(abi.encode(targets, amounts, id));
201 21×                 require(calculated == checksum, "Checksum mismatch");
202
203 11×                 for (uint256 i = 0; i < targets.length; i++) {
204 16×                     require(amounts[i] > 0, "Mint amount not greater than 0");
205 15×                     _mint(targets[i], amounts[i]);
206                     }
207 7×                 emit MintingSetCompleted(id);
208                 }
209
210                 /**
211                  * @dev Modifier that checks that an account is not blocked. Reverts
212                  * if the account is blocked
213                  */
214                 modifier whenNotBlocked(address account) {
215 157×                require(!hasRole(BLOCKED_ROLE, account), "Blocked user");
216 145×                _;
217                 }
218
219                 /**
220                  * @dev Checks that the contract is not paused and that neither sender nor receiver are blocked before transferring tokens. See {ERC
221                  * @param from source of the transfer
222                  * @param to target of the transfer
223                  * @param amount amount of tokens to be transferred
224                  */
225                 function _beforeTokenTransfer(
226                     address from,
227                     address to,
228                     uint256 amount
229                 ) internal override whenNotPaused whenNotBlocked(from) whenNotBlocked(to) {
230 69×                 super._beforeTokenTransfer(from, to, amount);
231                 }
232
233                 /**
234                  * @dev Restricts who can upgrade the contract. Executed when anyone tries to upgrade the contract
235                  * @param newImplementation Address of the new implementation
236                  */
237                 function _authorizeUpgrade(address newImplementation)
238                     internal
239                     override
240                     onlyRole(PROXYOWNER_ROLE)
241                 {}
242
243                 /**
244                  * @dev Returns the address of the implementation behind the proxy
245                  */
246                 function getImplementation() external view returns (address) {
247 2×                  return _getImplementation();
248                 }
249
250                 /**
251                  * @notice Used for rescuing tokens sent to the contract. Contact EUROe if you have accidentally sent tokens to the contract.
252                  * @dev Allows the rescue of an arbitrary token sent accidentally to the contract
253                  * @param token Which token we want to rescue
254                  * @param to Where should the rescued tokens be sent to
255                  * @param amount How many should be rescued
256                  */
257                 function rescueERC20(
258                     IERC20Upgradeable token,
259                     address to,
260                     uint256 amount
261                 ) external onlyRole(RESCUER_ROLE) {
262 5×                  token.safeTransfer(to, amount);
263                 }
264
265                 /**
266                  * @dev Prevent anyone from removing their own role (override OZ function)
267                  */
268                 function renounceRole(bytes32, address) public override {
269 8×                  revert("Not supported");
270                 }
271
272                 /**
```

```
273         * @dev This empty reserved space is put in place to allow future versions to add new
274         * variables without shifting down storage in the inheritance chain.
275         * See https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
276         */
277        uint256[50] private __gap;
278    }
279
```