

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 interface IERC20 {
5     /**
6      * @dev Returns the amount of tokens in existenc
7      e.
8      */
9     function totalSupply() external view returns (uin
10 t256);
11
12     /**
13      * @dev Returns the amount of tokens owned by `ac
14 count`.
15      */
16     function balanceOf(address account) external view
17 returns (uint256);
18
19     /**
20      * @dev Moves `amount` tokens from the caller's a
21 ccount to `recipient`.
22      *
23      * Returns a boolean value indicating whether the
24 operation succeeded.
25      *
26      * Emits a {Transfer} event.
27      */
28     function transfer(address recipient, uint256 amou
29 nt) external returns (bool);
30
31     /**
32      * @dev Returns the remaining number of tokens th
33 at `spender` will be
34      * allowed to spend on behalf of `owner` through
35 {transferFrom}. This is
36      * zero by default.
37      *
38      * This value changes when {approve} or {transfer
39 From} are called.
40      */
41     function allowance(address owner, address spende
42 r) external view returns (uint256);
43
44     /**
45      * @dev Sets `amount` as the allowance of `spende
46 r` over the caller's tokens.
47      *
48      * Returns a boolean value indicating whether the
49 operation succeeded.
50      *
51      * IMPORTANT: Beware that changing an allowance w
52 ith this method brings the risk
53      * that someone may use both the old and the new
54 allowance by unfortunate
55      * transaction ordering. One possible solution to
56 mitigate this race
57      * condition is to first reduce the spender's all
58 owance to 0 and set the
59      * desired value afterwards:
60      * https://github.com/ethereum/EIPs/issues/20#iss
61 uecomment-263524729
62      *
63      * Emits an {Approval} event.

```

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 interface IERC20 {
5     /**
6      * @dev Returns the amount of tokens in existenc
7      e.
8      */
9     function totalSupply() external view returns (uin
10 t256);
11
12     /**
13      * @dev Returns the amount of tokens owned by `ac
14 count`.
15      */
16     function balanceOf(address account) external view
17 returns (uint256);
18
19     /**
20      * @dev Moves `amount` tokens from the caller's a
21 ccount to `recipient`.
22      *
23      * Returns a boolean value indicating whether the
24 operation succeeded.
25      *
26      * Emits a {Transfer} event.
27      */
28     function transfer(address recipient, uint256 amou
29 nt) external returns (bool);
30
31     /**
32      * @dev Returns the remaining number of tokens th
33 at `spender` will be
34      * allowed to spend on behalf of `owner` through
35 {transferFrom}. This is
36      * zero by default.
37      *
38      * This value changes when {approve} or {transfer
39 From} are called.
40      */
41     function allowance(address owner, address spende
42 r) external view returns (uint256);
43
44     /**
45      * @dev Sets `amount` as the allowance of `spende
46 r` over the caller's tokens.
47      *
48      * Returns a boolean value indicating whether the
49 operation succeeded.
50      *
51      * IMPORTANT: Beware that changing an allowance w
52 ith this method brings the risk
53      * that someone may use both the old and the new
54 allowance by unfortunate
55      * transaction ordering. One possible solution to
56 mitigate this race
57      * condition is to first reduce the spender's all
58 owance to 0 and set the
59      * desired value afterwards:
60      * https://github.com/ethereum/EIPs/issues/20#iss
61 uecomment-263524729
62      *
63      * Emits an {Approval} event.

```

```

46  */
47  function approve(address spender, uint256 amount)
    external returns (bool);
48
49  /**
50   * @dev Moves `amount` tokens from `sender` to `recipient` using the
    allowance mechanism. `amount` is then deducted
    from the caller's
51   * allowance.
52   *
53   * Returns a boolean value indicating whether the
    operation succeeded.
54   *
55   * Emits a {Transfer} event.
56   */
57  function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
58
59  /**
60   * @dev Emitted when `value` tokens are moved from one account (`from`) to
    another (`to`).
61   *
62   * Note that `value` may be zero.
63   */
64  event Transfer(address indexed from, address indexed to, uint256 value);
65
66  /**
67   * @dev Emitted when the allowance of a `spender` for an `owner` is set by
    a call to {approve}. `value` is the new allowance.
68   */
69  event Approval(address indexed owner, address indexed spender, uint256 value);
70
71  }
72
73  library LowGasSafeMath {
74      /// @notice Returns x + y, reverts if sum overflows uint256
75      /// @param x The augend
76      /// @param y The addend
77      /// @return z The sum of x and y
78      function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
79          require((z = x + y) >= x);
80      }
81
82      /// @notice Returns x - y, reverts if underflows
83      /// @param x The minuend
84      /// @param y The subtrahend
85      /// @return z The difference of x and y
86      function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
87          require((z = x - y) <= x);
88      }
89
90      /// @notice Returns x * y, reverts if overflows
91      /// @param x The multiplicand
92      /// @param y The multiplier
93      /// @return z The product of x and y
94      function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
95          require(x == 0 || (z = x * y) / x == y);
96      }
97
98  }
99

```

```

46  */
47  function approve(address spender, uint256 amount)
    external returns (bool);
48
49  /**
50   * @dev Moves `amount` tokens from `sender` to `recipient` using the
    allowance mechanism. `amount` is then deducted
    from the caller's
51   * allowance.
52   *
53   * Returns a boolean value indicating whether the
    operation succeeded.
54   *
55   * Emits a {Transfer} event.
56   */
57  function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
58
59  /**
60   * @dev Emitted when `value` tokens are moved from one account (`from`) to
    another (`to`).
61   *
62   * Note that `value` may be zero.
63   */
64  event Transfer(address indexed from, address indexed to, uint256 value);
65
66  /**
67   * @dev Emitted when the allowance of a `spender` for an `owner` is set by
    a call to {approve}. `value` is the new allowance.
68   */
69  event Approval(address indexed owner, address indexed spender, uint256 value);
70
71  }
72
73  library LowGasSafeMath {
74      /// @notice Returns x + y, reverts if sum overflows uint256
75      /// @param x The augend
76      /// @param y The addend
77      /// @return z The sum of x and y
78      function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
79          require((z = x + y) >= x);
80      }
81
82      /// @notice Returns x - y, reverts if underflows
83      /// @param x The minuend
84      /// @param y The subtrahend
85      /// @return z The difference of x and y
86      function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
87          require((z = x - y) <= x);
88      }
89
90      /// @notice Returns x * y, reverts if overflows
91      /// @param x The multiplicand
92      /// @param y The multiplier
93      /// @return z The product of x and y
94      function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
95          require(x == 0 || (z = x * y) / x == y);
96      }
97
98  }
99

```

```

100     /// @notice Returns x + y, reverts if overflows
    or underflows
101     /// @param x The augend
102     /// @param y The addend
103     /// @return z The sum of x and y
104     function add(int256 x, int256 y) internal pure
    returns (int256 z) {
105         require((z = x + y) >= x == (y >= 0));
106     }
107
108     /// @notice Returns x - y, reverts if overflows
    or underflows
109     /// @param x The minuend
110     /// @param y The subtrahend
111     /// @return z The difference of x and y
112     function sub(int256 x, int256 y) internal pure
    returns (int256 z) {
113         require((z = x - y) <= x == (y >= 0));
114     }
115 }
116
117 abstract contract ERC20 is IERC20 {
118
119     using LowGasSafeMath for uint256;
120
121     // Present in ERC777
122     mapping (address => uint256) internal _balances;
123
124     // Present in ERC777
125     mapping (address => mapping (address => uint256))
    internal _allowances;
126
127     // Present in ERC777
128     uint256 internal _totalSupply;
129
130     // Present in ERC777
131     string internal _name;
132
133     // Present in ERC777
134     string internal _symbol;
135
136     // Present in ERC777
137     uint8 internal _decimals;
138
139     constructor (string memory name_, string memory s
    ymbol_, uint8 decimals_) {
140         _name = name_;
141         _symbol = symbol_;
142         _decimals = decimals_;
143     }
144
145     function name() public view returns (string memor
    y) {
146         return _name;
147     }
148
149     function symbol() public view returns (string mem
    ory) {
150         return _symbol;
151     }
152
153     function decimals() public view returns (uint8) {
154         return _decimals;
155     }
156
157     function totalSupply() public view override retur
    ns (uint256) {
158         return _totalSupply;
159     }
160

```

```

100     /// @notice Returns x + y, reverts if overflows
    or underflows
101     /// @param x The augend
102     /// @param y The addend
103     /// @return z The sum of x and y
104     function add(int256 x, int256 y) internal pure
    returns (int256 z) {
105         require((z = x + y) >= x == (y >= 0));
106     }
107
108     /// @notice Returns x - y, reverts if overflows
    or underflows
109     /// @param x The minuend
110     /// @param y The subtrahend
111     /// @return z The difference of x and y
112     function sub(int256 x, int256 y) internal pure
    returns (int256 z) {
113         require((z = x - y) <= x == (y >= 0));
114     }
115 }
116
117 abstract contract ERC20 is IERC20 {
118
119     using LowGasSafeMath for uint256;
120
121     // Present in ERC777
122     mapping (address => uint256) internal _balances;
123
124     // Present in ERC777
125     mapping (address => mapping (address => uint256))
    internal _allowances;
126
127     // Present in ERC777
128     uint256 internal _totalSupply;
129
130     // Present in ERC777
131     string internal _name;
132
133     // Present in ERC777
134     string internal _symbol;
135
136     // Present in ERC777
137     uint8 internal _decimals;
138
139     constructor (string memory name_, string memory s
    ymbol_, uint8 decimals_) {
140         _name = name_;
141         _symbol = symbol_;
142         _decimals = decimals_;
143     }
144
145     function name() public view returns (string memor
    y) {
146         return _name;
147     }
148
149     function symbol() public view returns (string mem
    ory) {
150         return _symbol;
151     }
152
153     function decimals() public view returns (uint8) {
154         return _decimals;
155     }
156
157     function totalSupply() public view override retur
    ns (uint256) {
158         return _totalSupply;
159     }
160

```

```

161 function balanceOf(address account) public view v
    irtual override returns (uint256) {
162     return _balances[account];
163 }
164
165 function transfer(address recipient, uint256 amou
    nt) public virtual override returns (bool) {
166     _transfer(msg.sender, recipient, amount);
167     return true;
168 }
169
170 function allowance(address owner, address spend
    er) public view virtual override returns (uint256)
    {
171     return _allowances[owner][spender];
172 }
173
174 function approve(address spender, uint256 amoun
    t) public virtual override returns (bool) {
175     _approve(msg.sender, spender, amount);
176     return true;
177 }
178
179 function transferFrom(address sender, address r
    ecipient, uint256 amount) public virtual override r
    eturns (bool) {
180     _transfer(sender, recipient, amount);
181     _approve(sender, msg.sender, _allowances[se
    nder][msg.sender]
182         .sub(amount));
183     return true;
184 }
185
186 function increaseAllowance(address spender, uin
    t256 addedValue) public virtual returns (bool) {
187     _approve(msg.sender, spender, _allowances[m
    sg.sender][spender].add(addedValue));
188     return true;
189 }
190
191 function decreaseAllowance(address spender, uin
    t256 subtractedValue) public virtual returns (bool)
    {
192     _approve(msg.sender, spender, _allowances[m
    sg.sender][spender]
193         .sub(subtractedValue));
194     return true;
195 }
196
197 function _transfer(address sender, address reci
    pient, uint256 amount) internal virtual {
198     require(sender != address(0), "ERC20: transfe
    r from the zero address");
199     require(recipient != address(0), "ERC20: tran
    sfer to the zero address");
200
201     _beforeTokenTransfer(sender, recipient, amoun
    t);
202
203     _balances[sender] = _balances[sender].sub(amou
    nt);
204     _balances[recipient] = _balances[recipient].a
    dd(amount);
205     emit Transfer(sender, recipient, amount);
206 }
207
208 function _mint(address account_, uint256 amount
    _) internal virtual {
209     require(account_ != address(0), "ERC20: min
    t to the zero address");

```

```

161 function balanceOf(address account) public view v
    irtual override returns (uint256) {
162     return _balances[account];
163 }
164
165 function transfer(address recipient, uint256 amou
    nt) public virtual override returns (bool) {
166     _transfer(msg.sender, recipient, amount);
167     return true;
168 }
169
170 function allowance(address owner, address spend
    er) public view virtual override returns (uint256)
    {
171     return _allowances[owner][spender];
172 }
173
174 function approve(address spender, uint256 amoun
    t) public virtual override returns (bool) {
175     _approve(msg.sender, spender, amount);
176     return true;
177 }
178
179 function transferFrom(address sender, address r
    ecipient, uint256 amount) public virtual override r
    eturns (bool) {
180     _transfer(sender, recipient, amount);
181     _approve(sender, msg.sender, _allowances[se
    nder][msg.sender]
182         .sub(amount));
183     return true;
184 }
185
186 function increaseAllowance(address spender, uin
    t256 addedValue) public virtual returns (bool) {
187     _approve(msg.sender, spender, _allowances[m
    sg.sender][spender].add(addedValue));
188     return true;
189 }
190
191 function decreaseAllowance(address spender, uin
    t256 subtractedValue) public virtual returns (bool)
    {
192     _approve(msg.sender, spender, _allowances[m
    sg.sender][spender]
193         .sub(subtractedValue));
194     return true;
195 }
196
197 function _transfer(address sender, address reci
    pient, uint256 amount) internal virtual {
198     require(sender != address(0), "ERC20: transfe
    r from the zero address");
199     require(recipient != address(0), "ERC20: tran
    sfer to the zero address");
200
201     _beforeTokenTransfer(sender, recipient, amoun
    t);
202
203     _balances[sender] = _balances[sender].sub(amou
    nt);
204     _balances[recipient] = _balances[recipient].a
    dd(amount);
205     emit Transfer(sender, recipient, amount);
206 }
207
208 function _mint(address account_, uint256 amount
    _) internal virtual {
209     require(account_ != address(0), "ERC20: min
    t to the zero address");

```

```

210     _beforeTokenTransfer(address( this ), accou
nt_, amount_);
211     _totalSupply = _totalSupply.add(amount_);
212     _balances[account_] = _balances[account_].a
dd(amount_);
213     emit Transfer(address(0), account_, amount
_);
214 }
215
216 function _burn(address account, uint256 amount)
internal virtual {
217     require(account != address(0), "ERC20: burn
from the zero address");
218
219     _beforeTokenTransfer(account, address(0), a
mount);
220
221     _balances[account] = _balances[account].sub
(amount);
222     _totalSupply = _totalSupply.sub(amount);
223     emit Transfer(account, address(0), amount);
224 }
225
226 function _approve(address owner, address spende
r, uint256 amount) internal virtual {
227     require(owner != address(0), "ERC20: approv
e from the zero address");
228     require(spender != address(0), "ERC20: appr
ove to the zero address");
229
230     _allowances[owner][spender] = amount;
231     emit Approval(owner, spender, amount);
232 }
233
234 function _beforeTokenTransfer( address from_, add
ress to_, uint256 amount_ ) internal virtual { }
235 }
236
237 library Counters {
238     using LowGasSafeMath for uint256;
239
240     struct Counter {
241         uint256 _value; // default: 0
242     }
243
244     function current(Counter storage counter) inter
nal view returns (uint256) {
245         return counter._value;
246     }
247
248     function increment(Counter storage counter) int
ernal {
249         counter._value += 1;
250     }
251
252     function decrement(Counter storage counter) int
ernal {
253         counter._value = counter._value.sub(1);
254     }
255 }
256
257 interface IERC2612Permit {
258
259     function permit(
260         address owner,
261         address spender,
262         uint256 amount,
263         uint256 deadline,
264         uint8 v,
265         bytes32 r,

```

```

210     _beforeTokenTransfer(address( this ), accou
nt_, amount_);
211     _totalSupply = _totalSupply.add(amount_);
212     _balances[account_] = _balances[account_].a
dd(amount_);
213     emit Transfer(address(0), account_, amount
_);
214 }
215
216 function _burn(address account, uint256 amount)
internal virtual {
217     require(account != address(0), "ERC20: burn
from the zero address");
218
219     _beforeTokenTransfer(account, address(0), a
mount);
220
221     _balances[account] = _balances[account].sub
(amount);
222     _totalSupply = _totalSupply.sub(amount);
223     emit Transfer(account, address(0), amount);
224 }
225
226 function _approve(address owner, address spende
r, uint256 amount) internal virtual {
227     require(owner != address(0), "ERC20: approv
e from the zero address");
228     require(spender != address(0), "ERC20: appr
ove to the zero address");
229
230     _allowances[owner][spender] = amount;
231     emit Approval(owner, spender, amount);
232 }
233
234 function _beforeTokenTransfer( address from_, add
ress to_, uint256 amount_ ) internal virtual { }
235 }
236
237 library Counters {
238     using LowGasSafeMath for uint256;
239
240     struct Counter {
241         uint256 _value; // default: 0
242     }
243
244     function current(Counter storage counter) inter
nal view returns (uint256) {
245         return counter._value;
246     }
247
248     function increment(Counter storage counter) int
ernal {
249         counter._value += 1;
250     }
251
252     function decrement(Counter storage counter) int
ernal {
253         counter._value = counter._value.sub(1);
254     }
255 }
256
257 interface IERC2612Permit {
258
259     function permit(
260         address owner,
261         address spender,
262         uint256 amount,
263         uint256 deadline,
264         uint8 v,
265         bytes32 r,

```

```

266         bytes32 s
267     ) external;
268
269     function nonces(address owner) external view re
turns (uint256);
270 }
271
272 abstract contract ERC20Permit is ERC20, IERC2612Per
mit {
273     using Counters for Counters.Counter;
274
275     mapping(address => Counters.Counter) private _n
onces;
276
277     // keccak256("Permit(address owner,address spen
der,uint256 value,uint256 nonce,uint256 deadlin
e)");
278     bytes32 public constant PERMIT_TYPEHASH = 0x6e7
1edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64
845d6126c9;
279
280     bytes32 public DOMAIN_SEPARATOR;
281
282     constructor() {
283         uint256 chainID;
284         assembly {
285             chainID := chainid()
286         }
287
288         DOMAIN_SEPARATOR = keccak256(
289             abi.encode(
290                 keccak256("EIP712Domain(string nam
e,string version,uint256 chainId,address verifyingC
ontract)"),
291                 keccak256(bytes(name())),
292                 keccak256(bytes("1")), // Version
293                 chainID,
294                 address(this)
295             )
296         );
297     }
298
299     function permit(
300         address owner,
301         address spender,
302         uint256 amount,
303         uint256 deadline,
304         uint8 v,
305         bytes32 r,
306         bytes32 s
307     ) public virtual override {
308         require(block.timestamp <= deadline, "Permi
t: expired deadline");
309
310         bytes32 hashStruct =
311             keccak256(abi.encode(PERMIT_TYPEHASH, o
wner, spender, amount, _nonces[owner].current(), de
adline));
312
313         bytes32 _hash = keccak256(abi.encodePacked
(uint16(0x1901), DOMAIN_SEPARATOR, hashStruct));
314
315         address signer = ecrecover(_hash, v, r, s);
316         require(signer != address(0) && signer == o
wner, "ERC20Permit: Invalid signature");
317
318         _nonces[owner].increment();
319         _approve(owner, spender, amount);
320     }
321

```

```

266         bytes32 s
267     ) external;
268
269     function nonces(address owner) external view re
turns (uint256);
270 }
271
272 abstract contract ERC20Permit is ERC20, IERC2612Per
mit {
273     using Counters for Counters.Counter;
274
275     mapping(address => Counters.Counter) private _n
onces;
276
277     // keccak256("Permit(address owner,address spen
der,uint256 value,uint256 nonce,uint256 deadlin
e)");
278     bytes32 public constant PERMIT_TYPEHASH = 0x6e7
1edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64
845d6126c9;
279
280     bytes32 public DOMAIN_SEPARATOR;
281
282     constructor() {
283         uint256 chainID;
284         assembly {
285             chainID := chainid()
286         }
287
288         DOMAIN_SEPARATOR = keccak256(
289             abi.encode(
290                 keccak256("EIP712Domain(string nam
e,string version,uint256 chainId,address verifyingC
ontract)"),
291                 keccak256(bytes(name())),
292                 keccak256(bytes("1")), // Version
293                 chainID,
294                 address(this)
295             )
296         );
297     }
298
299     function permit(
300         address owner,
301         address spender,
302         uint256 amount,
303         uint256 deadline,
304         uint8 v,
305         bytes32 r,
306         bytes32 s
307     ) public virtual override {
308         require(block.timestamp <= deadline, "Permi
t: expired deadline");
309
310         bytes32 hashStruct =
311             keccak256(abi.encode(PERMIT_TYPEHASH, o
wner, spender, amount, _nonces[owner].current(), de
adline));
312
313         bytes32 _hash = keccak256(abi.encodePacked
(uint16(0x1901), DOMAIN_SEPARATOR, hashStruct));
314
315         address signer = ecrecover(_hash, v, r, s);
316         require(signer != address(0) && signer == o
wner, "ERC20Permit: Invalid signature");
317
318         _nonces[owner].increment();
319         _approve(owner, spender, amount);
320     }
321

```

```

322     function nonces(address owner) public view over
ride returns (uint256) {
323         return _nonces[owner].current();
324     }
325 }
326
327 interface IOwnable {
328     function owner() external view returns (address);
329
330     function renounceOwnership() external;
331
332     function transferOwnership( address newOwner_ ) e
xternal;
333 }
334
335 contract Ownable is IOwnable {
336
337     address internal _owner;
338
339     event OwnershipTransferred(address indexed previo
usOwner, address indexed newOwner);
340
341     constructor () {
342         _owner = msg.sender;
343         emit OwnershipTransferred( address(0), _owner
);
344     }
345
346     function owner() public view override returns (ad
dress) {
347         return _owner;
348     }
349
350     modifier onlyOwner() {
351         require( _owner == msg.sender, "Ownable: caller
is not the owner" );
352         _;
353     }
354
355     function renounceOwnership() public virtual overr
ide onlyOwner() {
356         emit OwnershipTransferred( _owner, address(0)
);
357         _owner = address(0);
358     }
359
360     function transferOwnership( address newOwner_ ) p
ublic virtual override onlyOwner() {
361         require( newOwner_ != address(0), "Ownable: new
owner is the zero address");
362         emit OwnershipTransferred( _owner, newOwner_ );
363         _owner = newOwner_;
364     }
365 }
366
367 contract VaultOwned is Ownable {
368
369     address internal _vault;
370
371     event VaultTransferred(address indexed newVault);
372
373     function setVault( address vault_ ) external only
Owner() {
374         require(vault_ != address(0), "IA0");
375         _vault = vault_;
376         emit VaultTransferred( _vault );
377     }
378
379     function vault() public view returns (address) {
380         return _vault;

```

```

322     function nonces(address owner) public view over
ride returns (uint256) {
323         return _nonces[owner].current();
324     }
325 }
326
327 interface IOwnable {
328     function owner() external view returns (address);
329
330     function renounceOwnership() external;
331
332     function transferOwnership( address newOwner_ ) e
xternal;
333 }
334
335 contract Ownable is IOwnable {
336
337     address internal _owner;
338
339     event OwnershipTransferred(address indexed previo
usOwner, address indexed newOwner);
340
341     constructor () {
342         _owner = msg.sender;
343         emit OwnershipTransferred( address(0), _owner
);
344     }
345
346     function owner() public view override returns (ad
dress) {
347         return _owner;
348     }
349
350     modifier onlyOwner() {
351         require( _owner == msg.sender, "Ownable: caller
is not the owner" );
352         _;
353     }
354
355     function renounceOwnership() public virtual overr
ide onlyOwner() {
356         emit OwnershipTransferred( _owner, address(0)
);
357         _owner = address(0);
358     }
359
360     function transferOwnership( address newOwner_ ) p
ublic virtual override onlyOwner() {
361         require( newOwner_ != address(0), "Ownable: new
owner is the zero address");
362         emit OwnershipTransferred( _owner, newOwner_ );
363         _owner = newOwner_;
364     }
365 }
366
367 contract VaultOwned is Ownable {
368
369     address internal _vault;
370
371     event VaultTransferred(address indexed newVault);
372
373     function setVault( address vault_ ) external only
Owner() {
374         require(vault_ != address(0), "IA0");
375         _vault = vault_;
376         emit VaultTransferred( _vault );
377     }
378
379     function vault() public view returns (address) {
380         return _vault;

```

```

381 }
382
383 modifier onlyVault() {
384     require( _vault == msg.sender, "VaultOwned: caller is not the Vault" );
385     _;
386 }
387
388 }
389
390 contract TimeERC20Token is ERC20Permit, VaultOwned
391 {
392     using LowGasSafeMath for uint256;
393
394     constructor() ERC20("Time", "TIME", 9) {
395     }
396
397     function mint(address account_, uint256 amount
398     _) external onlyVault() {
399         _mint(account_, amount_);
400     }
401
402     function burn(uint256 amount) external virtual
403     {
404         _burn(msg.sender, amount);
405     }
406
407     function burnFrom(address account_, uint256 amount_) external virtual {
408         _burnFrom(account_, amount_);
409     }
410
411     function _burnFrom(address account_, uint256 amount_) internal virtual {
412         uint256 decreasedAllowance_ =
413             allowance(account_, msg.sender).sub(amount_);
414         _approve(account_, msg.sender, decreasedAllowance_);
415         _burn(account_, amount_);
416     }

```

```

381 }
382
383 modifier onlyVault() {
384     require( _vault == msg.sender, "VaultOwned: caller is not the Vault" );
385     _;
386 }
387
388 }
389
390 contract MaiaERC20Token is ERC20Permit, VaultOwned
391 {
392     using LowGasSafeMath for uint256;
393
394     constructor() ERC20("Maia", "MAIA", 9) {
395     }
396
397     function mint(address account_, uint256 amount
398     _) external onlyVault() {
399         _mint(account_, amount_);
400     }
401
402     function burn(uint256 amount) external virtual
403     {
404         _burn(msg.sender, amount);
405     }
406
407     function burnFrom(address account_, uint256 amount_) external virtual {
408         _burnFrom(account_, amount_);
409     }
410
411     function _burnFrom(address account_, uint256 amount_) internal virtual {
412         uint256 decreasedAllowance_ =
413             allowance(account_, msg.sender).sub(amount_);
414         _approve(account_, msg.sender, decreasedAllowance_);
415         _burn(account_, amount_);
416     }

```