Free Token

WHITE PAPER

Free Token is the people's cryptocurrency made free.

1st arrived, 1st served. On the Ethereum Blockchain.

What is $FREET?

- $FREET is a simple ERC20 token on the Ethereum blockchain.
- $FREET is not mintable nor pausable (we could not create more $FREET).
- Total Supply & final supply is: 1,000,000,000 $FREET.

$FREET Distribution

- 1st arrived, 1st served.
- 100% of the total supply is available through the distribution contract.
- Distribution contract set a distribution cap at 1,000 $FREET per address.

How do I get $FREET?

Send 0 Ether (ETH) to this automatic distribution address using your favorite wallet and get 1,000 $FREET instantly for free:
0x1B8e9B85679f845057426dd4Ba1eB1ba38319677

Some more questions?

Why create a token for free?
Simple, Free Token is the future of cryptocurrencies.

But hey! Could someone get all the stock if it is for free?
No he cannot. The distribution is capped at 1,000 $FREET per address. To get all the stock someone would need to make 1,000,000 transactions with 1,000,000 different addresses. This is impossible because of the gaz (even through multi-sender applications).

But if you want more $FREET, feel free to get more by sending 0 Ether (ETH) to the distribution contract with another Ethereum address.

Will $FREE ever have a value some day?
When $FREET will be HODL by 1 million users it will surely have the value of the people.
And when someone decides to create a liquidity pool on Uniswap then $FREET will start to be tradable against any other cryptocurrencies.

Any catch?
Free Token source code is available on Github as well as on Etherscan.io (all contracts are verified and consultable).

Can I tip $FREET creators?

Yes indeed, you can tip $FREET creators by sending Ether (ETH) to the creators' address

Ethereum addresses

- Token address: 0x8c1b9072F78FfB869C817453dE44b2101C4DE645
- Distribution contract: 0x1B8e9B85679f845057426dd4Ba1eB1ba38319677
- Creator's address: 0x43621b2dc33FAf55c50f9713CD20f0119d8a7646

Source code $FREET token ERC20 :

```
// SPDX-
License-
Identifier:
MIT

pragma solidity ^0.8.0;

/*
 * @dev Provides information about the current execution context,
including the
 * sender of the transaction and its data. While these are generally
available
 * via msg.sender and msg.data, they should not be accessed in such a
direct
 * manner, since when dealing with meta-transactions the account sending
and
 * paying for execution may not be the actual sender (as far as an
application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like
contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        this; // silence state mutability warning without generating
bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);
```

```
    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to
`recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns
(bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This
is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view
returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's
tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method
brings the risk
     * that someone may use both the old and the new allowance by
unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set
the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
```

```
    function approve(address spender, uint256 amount) external returns
(bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256
amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account
(`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256
value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is
set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256
value);
}

/**
 * @dev Interface for the optional metadata functions from the ERC20
standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);
```

```
    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This
means
 * that a supply mechanism has to be added in a derived contract using
{_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert
instead
 * of returning `false` on failure. This behavior is nonetheless
conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to
{transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts
just
 * by listening to said events. Other implementations of the EIP may not
emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around
setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping (address => uint256) private _balances;
```

```
    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The defaut value of {decimals} is 18. To select a different value
for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once
during
     * construction.
     */
    constructor (string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of
the
     * name.
     */
    function symbol() public view virtual override returns (string memory)
{
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user
representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens
should
```

```
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship
between
     * Ether and Wei. This is the value {ERC20} uses, unless this function
is
     * overridden;
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual override returns (uint8) {
        return 18;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual override returns (uint256)
{
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view virtual override
returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual
override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
```

```
    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual
override returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual
override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is
not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at
least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256
amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);

        uint256 currentAllowance = _allowances[sender][_msgSender()];
        require(currentAllowance >= amount, "ERC20: transfer amount
exceeds allowance");
        _approve(sender, _msgSender(), currentAllowance - amount);

        return true;
```

```
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the
caller.
     *
     * This is an alternative to {approve} that can be used as a
mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public
virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender]
+ addedValue);
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the
caller.
     *
     * This is an alternative to {approve} that can be used as a
mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue)
public virtual returns (bool) {
        uint256 currentAllowance = _allowances[_msgSender()][spender];
        require(currentAllowance >= subtractedValue, "ERC20: decreased
allowance below zero");
        _approve(_msgSender(), spender, currentAllowance -
subtractedValue);
```

```
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be
used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount)
internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero
address");
        require(recipient != address(0), "ERC20: transfer to the zero
address");

        _beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = _balances[sender];
        require(senderBalance >= amount, "ERC20: transfer amount exceeds
balance");
        _balances[sender] = senderBalance - amount;
        _balances[recipient] += amount;

        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`,
increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
```

```
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds
balance");
        _balances[account] = accountBalance - amount;
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s
tokens.
     *
     * This internal function is equivalent to `approve`, and can be used
to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
```

```
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount)
internal virtual {
        require(owner != address(0), "ERC20: approve from the zero
address");
        require(spender != address(0), "ERC20: approve to the zero
address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This
includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s
tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-
contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256
amount) internal virtual { }
}

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both
their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20 {
    /**
```

```
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the
caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at
least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) public virtual {
        uint256 currentAllowance = allowance(account, _msgSender());
        require(currentAllowance >= amount, "ERC20: burn amount exceeds
allowance");
        _approve(account, _msgSender(), currentAllowance - amount);
        _burn(account, amount);
    }
}


/**
 * @dev {ERC20} token, including:
 *
 *  - Preminted initial supply
 *  - Ability for holders to burn (destroy) their tokens
 *  - No access control mechanism (for minting/pausing) and hence no
governance
 *
 * This contract uses {ERC20Burnable} to include burn capabilities - head
to
 * its documentation for details.
 *
 * _Available since v3.4._
 */
contract FreeTokenFREET is ERC20Burnable {
```

```
    /**
     * @dev Mints `initialSupply` amount of token and transfers them to
`owner`.
     *
     * See {ERC20-constructor}.
     */
    constructor(
        string memory name,
        string memory symbol,
        uint256 initialSupply,
        address owner
    ) ERC20(name, symbol) {
        _mint(owner, initialSupply);
    }
}
```

Source code FreeToken Distribution contract :

```
pragma
solidity
^0.5.0;


/*
 * @dev Provides information about the current execution context, including
the
 * sender of the transaction and its data. While these are generally
available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending
and
 * paying for execution may not be the actual sender (as far as an
application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly
deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode
- see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}



/**
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
```

```
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no
nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked
as
 * `nonReentrant` may not call one another. This can be worked around by
making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After
Istanbul].
 *
 * _Since v2.5.0:_ this module is now much more gas efficient, given net gas
 * metering changes introduced in the Istanbul hardfork.
 */
contract ReentrancyGuard {
    bool private _notEntered;

    constructor () internal {
        // Storing an initial non-zero value makes deployment a bit more
        // expensive, but in exchange the refund on every call to
nonReentrant
        // will be lower in amount. Since refunds are capped to a percetange
of
        // the total transaction's gas, it is best to keep them low in cases
        // like this one, to increase the likelihood of the full refund
coming
        // into effect.
        _notEntered = true;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from
happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
```

```
        require(_notEntered, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _notEntered = false;

        _;

        // By storing the original value once again, a refund is triggered
(see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _notEntered = true;
    }
}


/**
 * @title Crowdsale
 * @dev Crowdsale is a base contract for managing a token crowdsale,
 * allowing investors to purchase tokens with ether. This contract implements
 * such functionality in its most fundamental form and can be extended to
provide additional
 * functionality and/or custom behavior.
 * The external interface represents the basic interface for purchasing
tokens, and conforms
 * the base architecture for crowdsales. It is *not* intended to be modified
/ overridden.
 * The internal interface conforms the extensible and modifiable surface of
crowdsales. Override
 * the methods to add functionality. Consider using 'super' where appropriate
to concatenate
 * behavior.
 */
contract Crowdsale is Context, ReentrancyGuard {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // The token being sold
    IERC20 private _token;

    // Address where funds are collected
    address payable private _wallet;

    // How many token units a buyer gets per wei.
    // The rate is the conversion between wei and the smallest and
indivisible token unit.
```

```
    // So, if you are using a rate of 1 with a ERC20Detailed token with 3
decimals called TOK
    // 1 wei will give you 1 unit, or 0.001 TOK.
    uint256 private _rate;

    // Amount of wei raised
    uint256 private _weiRaised;

    /**
     * Event for token purchase logging
     * @param purchaser who paid for the tokens
     * @param beneficiary who got the tokens
     * @param value weis paid for purchase
     * @param amount amount of tokens purchased
     */
    event TokensPurchased(address indexed purchaser, address indexed
beneficiary, uint256 value, uint256 amount);

    /**
     * @param rate Number of token units a buyer gets per wei
     * @dev The rate is the conversion between wei and the smallest and
indivisible
     * token unit. So, if you are using a rate of 1 with a ERC20Detailed
token
     * with 3 decimals called TOK, 1 wei will give you 1 unit, or 0.001 TOK.
     * @param wallet Address where collected funds will be forwarded to
     * @param token Address of the token being sold
     */
    constructor (uint256 rate, address payable wallet, IERC20 token) public {
        require(rate >= 0, "Crowdsale: rate must be above 0");
        require(wallet != address(0), "Crowdsale: wallet is the zero
address");
        require(address(token) != address(0), "Crowdsale: token is the zero
address");

        _rate = rate;
        _wallet = wallet;
        _token = token;
    }

    /**
     * @dev fallback function ***DO NOT OVERRIDE***
     * Note that other contracts will transfer funds with a base gas stipend
     * of 2300, which is not enough to call buyTokens. Consider calling
     * buyTokens directly when purchasing tokens from a contract.
     */
```

```
function () external payable {
    buyTokens(_msgSender());
}

/**
 * @return the token being sold.
 */
function token() public view returns (IERC20) {
    return _token;
}

/**
 * @return the address where funds are collected.
 */
function wallet() public view returns (address payable) {
    return _wallet;
}

/**
 * @return the number of token units a buyer gets per wei.
 */
function rate() public view returns (uint256) {
    return _rate;
}

/**
 * @return the amount of wei raised.
 */
function weiRaised() public view returns (uint256) {
    return _weiRaised;
}

/**
 * @dev low level token purchase ***DO NOT OVERRIDE***
 * This function has a non-reentrancy guard, so it shouldn't be called by
 * another `nonReentrant` function.
 * @param beneficiary Recipient of the token purchase
 */
function buyTokens(address beneficiary) public nonReentrant payable {
     uint256 weiAmount = msg.value;


      // calculate token amount to be created
    uint256 tokens = _getTokenAmount(weiAmount) + 100000000000000000000;
    _preValidatePurchase(beneficiary, weiAmount, tokens);
```

```
        // update state
        _weiRaised = _weiRaised.add(weiAmount);

        _processPurchase(beneficiary, tokens);
        emit TokensPurchased(_msgSender(), beneficiary, weiAmount, tokens);

        _updatePurchasingState(beneficiary, weiAmount, tokens);

        _forwardFunds();
        _postValidatePurchase(beneficiary, weiAmount);
    }

    /**
     * @dev Validation of an incoming purchase. Use require statements to
revert state when conditions are not met.
     * Use `super` in contracts that inherit from Crowdsale to extend their
validations.
     * Example from CappedCrowdsale.sol's _preValidatePurchase method:
     *     super._preValidatePurchase(beneficiary, weiAmount);
     *     require(weiRaised().add(weiAmount) <= cap);
     * @param beneficiary Address performing the token purchase
     * @param weiAmount Value in wei involved in the purchase
     */
    function _preValidatePurchase(address beneficiary, uint256 weiAmount,
uint256 tokens) internal view {
        require(beneficiary != address(0), "Crowdsale: beneficiary is the
zero address");
        require(weiAmount >= 0, "Crowdsale: weiAmount must 0");
        require(tokens != 0, "test warning");
        this; // silence state mutability warning without generating bytecode
- see https://github.com/ethereum/solidity/issues/2691
    }

    /**
     * @dev Validation of an executed purchase. Observe state and use revert
statements to undo rollback when valid
     * conditions are not met.
     * @param beneficiary Address performing the token purchase
     * @param weiAmount Value in wei involved in the purchase
     */
    function _postValidatePurchase(address beneficiary, uint256 weiAmount)
internal view {
        // solhint-disable-previous-line no-empty-blocks
    }

    /**
```

```
     * @dev Source of tokens. Override this method to modify the way in which
the crowdsale ultimately gets and sends
     * its tokens.
     * @param beneficiary Address performing the token purchase
     * @param tokenAmount Number of tokens to be emitted
     */
    function _deliverTokens(address beneficiary, uint256 tokenAmount)
internal {
        _token.safeTransfer(beneficiary, tokenAmount);
    }

    /**
     * @dev Executed when a purchase has been validated and is ready to be
executed. Doesn't necessarily emit/send
     * tokens.
     * @param beneficiary Address receiving the tokens
     * @param tokenAmount Number of tokens to be purchased
     */
    function _processPurchase(address beneficiary, uint256 tokenAmount)
internal {
        _deliverTokens(beneficiary, tokenAmount);
    }

    /**
     * @dev Override for extensions that require an internal state to check
for validity (current user contributions,
     * etc.)
     * @param beneficiary Address receiving the tokens
     * @param weiAmount Value in wei involved in the purchase
     */
    function _updatePurchasingState(address beneficiary, uint256 weiAmount,
uint256 tokens) internal {
        // solhint-disable-previous-line no-empty-blocks
    }

    /**
     * @dev Override to extend the way in which ether is converted to tokens.
     * @param weiAmount Value in wei to be converted into tokens
     * @return Number of tokens that can be purchased with the specified
_weiAmount
     */
    function _getTokenAmount(uint256 weiAmount) internal view returns
(uint256) {
        return weiAmount.mul(_rate);
    }
```

```
    /**
     * @dev Determines how ETH is stored/forwarded on purchases.
     */
    function _forwardFunds() internal {
        _wallet.transfer(msg.value);
    }
}
```

```
/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not
include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns
(bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns
(uint256);
```

```
    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's
tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings
the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns
(bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`)
to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set
by
     * a call to {approve}. `value` is the new allowance.
     */
```

```
    event Approval(address indexed owner, address indexed spender, uint256
value);
}




/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an
entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
```

```
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with
custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal
pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being
zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-
contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
```

```
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts
with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal
pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this
doesn't hold

        return c;
    }

    /**
```

```
     * @dev Returns the remainder of dividing two unsigned integers.
(unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers.
(unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal
pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}



/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
```

```
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet
created accounts
        // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address
payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas
cost
```

```
     * of certain opcodes, possibly making contracts go over the 2300 gas
limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-
transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-
considerations.html#use-the-checks-effects-interactions-pattern[checks-
effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have
reverted");
    }
}


/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the
token
 * contract returns false). Tokens that return no value (and instead revert
or
 * throw on failure) are also supported, non-reverting calls are assumed to
be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement
to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`,
etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;
```

```
    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token,
abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256
value) internal {
        callOptionalReturn(token,
abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value)
internal {
        // safeApprove should only be called when setting an initial
allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) ==
0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token,
abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256
value) internal {
        uint256 newAllowance = token.allowance(address(this),
spender).add(value);
        callOptionalReturn(token,
abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256
value) internal {
        uint256 newAllowance = token.allowance(address(this),
spender).sub(value, "SafeERC20: decreased allowance below zero");
        callOptionalReturn(token,
abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call
to a contract), relaxing the requirement
```

```
     * on the return value: the return value is optional (but if data is
returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its
variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's
return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //  1. The target address is checked to verify it contains contract
code
        //  2. The call itself is made, and success asserted
        //  3. The return value is decoded, which in turn checks the size of
the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-
contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20
operation did not succeed");
        }
    }
}
```

```
/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @dev Returns the largest of two numbers.
     */
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
```

```
    }

    /**
     * @dev Returns the smallest of two numbers.
     */
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}


/**
 * @title AllowanceCrowdsale
 * @dev Extension of Crowdsale where tokens are held by a wallet, which
approves an allowance to the crowdsale.
 */
contract AllowanceCrowdsale is Crowdsale {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    address private _tokenWallet;

    /**
     * @dev Constructor, takes token wallet address.
     * @param tokenWallet Address holding the tokens, which has approved
allowance to the crowdsale.
     */
    constructor (address tokenWallet) public {
        require(tokenWallet != address(0), "AllowanceCrowdsale: token wallet
is the zero address");
        _tokenWallet = tokenWallet;
    }

    /**
     * @return the address of the wallet that will hold the tokens.
     */
    function tokenWallet() public view returns (address) {
```

```
        return _tokenWallet;
    }

    /**
     * @dev Checks the amount of tokens left in the allowance.
     * @return Amount of tokens left in the allowance
     */
    function remainingTokens() public view returns (uint256) {
        return Math.min(token().balanceOf(_tokenWallet),
token().allowance(_tokenWallet, address(this)));
    }

    /**
     * @dev Overrides parent behavior by transferring tokens from wallet.
     * @param beneficiary Token purchaser
     * @param tokenAmount Amount of tokens purchased
     */
    function _deliverTokens(address beneficiary, uint256 tokenAmount)
internal {
        token().safeTransferFrom(_tokenWallet, beneficiary, tokenAmount);
    }
}

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
```

```
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns
(bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}


contract PauserRole is Context {
    using Roles for Roles.Role;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(_msgSender());
    }

    modifier onlyPauser() {
        require(isPauser(_msgSender()), "PauserRole: caller does not have the
Pauser role");
        _;
    }

    function isPauser(address account) public view returns (bool) {
        return _pausers.has(account);
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function renouncePauser() public {
        _removePauser(_msgSender());
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
```

```
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}


/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
contract Pausable is Context, PauserRole {
    /**
     * @dev Emitted when the pause is triggered by a pauser (`account`).
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by a pauser (`account`).
     */
    event Unpaused(address account);

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state. Assigns the Pauser
role
     * to the deployer.
     */
    constructor () internal {
        _paused = false;
    }

    /**
     * @dev Returns true if the contract is paused, and false otherwise.
     */
    function paused() public view returns (bool) {
        return _paused;
```

```
    }

    /**
     * @dev Modifier to make a function callable only when the contract is
not paused.
     */
    modifier whenNotPaused() {
        require(!_paused, "Pausable: paused");
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is
paused.
     */
    modifier whenPaused() {
        require(_paused, "Pausable: not paused");
        _;
    }

    /**
     * @dev Called by a pauser to pause, triggers stopped state.
     */
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }

    /**
     * @dev Called by a pauser to unpause, returns to normal state.
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}


/**
 * @title PausableCrowdsale
 * @dev Extension of Crowdsale contract where purchases can be paused and
unpaused by the pauser role.
 */
contract PausableCrowdsale is Crowdsale, Pausable {
    /**
```

```
     * @dev Validation of an incoming purchase. Use require statements to
revert state when conditions are not met.
     * Use super to concatenate validations.
     * Adds the validation that the crowdsale must not be paused.
     * @param _beneficiary Address performing the token purchase
     * @param _weiAmount Value in wei involved in the purchase
     */
    function _preValidatePurchase(address _beneficiary, uint256 _weiAmount,
uint256 tokens) internal view whenNotPaused {
        return super._preValidatePurchase(_beneficiary, _weiAmount, tokens);
    }
}

contract CapperRole is Context {
    using Roles for Roles.Role;

    event CapperAdded(address indexed account);
    event CapperRemoved(address indexed account);

    Roles.Role private _cappers;

    constructor () internal {
        _addCapper(_msgSender());
    }

    modifier onlyCapper() {
        require(isCapper(_msgSender()), "CapperRole: caller does not have the
Capper role");
        _;
    }

    function isCapper(address account) public view returns (bool) {
        return _cappers.has(account);
    }

    function addCapper(address account) public onlyCapper {
        _addCapper(account);
    }

    function renounceCapper() public {
        _removeCapper(_msgSender());
    }

    function _addCapper(address account) internal {
        _cappers.add(account);
        emit CapperAdded(account);
```

```
    }

    function _removeCapper(address account) internal {
        _cappers.remove(account);
        emit CapperRemoved(account);
    }
}

/**
 * @title IndividuallyCappedCrowdsale
 * @dev Crowdsale with per-beneficiary caps.
 */
contract IndividuallyCappedCrowdsale is Crowdsale, CapperRole {
    using SafeMath for uint256;

    mapping(address => uint256) private _contributions;
    mapping(address => uint256) private _caps;

    /**
     * @dev Sets a specific beneficiary's maximum contribution.
     * @param beneficiary Address to be capped
     * @param cap Wei limit for individual contribution
     */
    function setCap(address beneficiary, uint256 cap) external onlyCapper {
        _caps[beneficiary] = cap;
    }

    /**
     * @dev Returns the cap of a specific beneficiary.
     * @param beneficiary Address whose cap is to be checked
     * @return Current cap for individual beneficiary
     */
    function getCap(address beneficiary) public view returns (uint256) {
        return _caps[beneficiary];
    }

    /**
     * @dev Returns the amount contributed so far by a specific beneficiary.
     * @param beneficiary Address of contributor
     * @return Beneficiary contribution so far
     */
    function getContribution(address beneficiary) public view returns
(uint256) {
        return _contributions[beneficiary];
    }
```

```
    /**
     * @dev Extend parent behavior requiring purchase to respect the
beneficiary's funding cap.
     * @param beneficiary Token purchaser
     * @param weiAmount Amount of wei contributed
     */
    function _preValidatePurchase(address beneficiary, uint256 weiAmount,
uint256 tokens) internal view {
        super._preValidatePurchase(beneficiary, weiAmount, tokens);
        // solhint-disable-next-line max-line-length
        require(_contributions[beneficiary] <= _caps[beneficiary],
"IndividuallyCappedCrowdsale: beneficiary's cap exceeded");
    }

    /**
     * @dev Extend parent behavior to update beneficiary contributions.
     * @param beneficiary Token purchaser
     * @param weiAmount Amount of wei contributed
     */
    function _updatePurchasingState(address beneficiary, uint256 weiAmount,
uint256 tokens) internal {
        super._updatePurchasingState(beneficiary, weiAmount, tokens);
        _contributions[beneficiary] =
_contributions[beneficiary].add(tokens);
    }
}




contract DistributionFREET is Crowdsale, AllowanceCrowdsale,
PausableCrowdsale, IndividuallyCappedCrowdsale {
    constructor(
        uint256 rate,
        uint256 cap,
        address payable wallet,
        IERC20 token,
        address tokenWallet  // <- new argument
    )
        AllowanceCrowdsale(tokenWallet)  // <- used here
        Crowdsale(rate, wallet, token)
        public
    {

    }
}
```