

# Documentation on Contract

## 1. Терминология

### On-chain транзакции

- **Определение:** Операции, отправляемые непосредственно в сеть блокчейна, которые изменяют её состояние.
- **Пример:** Вызовы функций `transfer()`, `mint()`, `burn()` и других, изменения записываются в блокчейн и требуют оплаты газа.

### Off-chain подписи

- **Определение:** Цифровые подписи, создаваемые локально с использованием приватного ключа, которые сами по себе не изменяют состояние блокчейна и не требуют оплаты газа.
- **Пример:** Механизм `ERC20Permit` (EIP-2612), позволяющий вместо отдельной on-chain транзакции для `approve()` использовать подпись.

### ERC20

- **Определение:** Стандарт токенов, определяющий общий интерфейс (функции `balanceOf`, `transfer`, `approve` и т.д.) для управления токенами на Ethereum и совместимых сетях.
- **Значение:** Обеспечивает совместимость токена с кошельками, биржами и dApps.

### ERC20Permit (EIP-2612)

- **Определение:** Расширение стандарта ERC20, позволяющее пользователям одобрять списание токенов посредством off-chain подписи (`permit`).
- **Значение:** Снижает необходимость в отдельной on-chain транзакции для `approve()`, что экономит газ.

### Upgradeable / Proxy Pattern

- **Определение:** Архитектурный шаблон, позволяющий разделять логику контракта и его данные, что дает возможность обновлять функциональность без изменения адреса контракта.
- **Значение:** Позволяет вносить изменения в логику контракта после деплоя, сохраняя при этом все данные и адрес.

## UUPSUpgradeable

- **Определение:** Легковесный механизм обновления прокси, в котором логика обновления реализована в контракте-имплементации.
- **Значение:** Обновление (upgrade) может осуществляться только авторизованными пользователями (например, администраторами).

## AccessControl

- **Определение:** Модуль для управления доступом, позволяющий назначать роли (например, `DEFAULT_ADMIN_ROLE`, `PAUSER_ROLE`, `MINTER_ROLE`, `BURNER_ROLE`) и ограничивать выполнение функций определёнными аккаунтами.
- **Значение:** Обеспечивает контроль над критическими операциями (mint, burn, pause, обновление параметров).

## Pausable

- **Определение:** Модуль, позволяющий приостанавливать выполнение функций контракта в экстренных ситуациях (например, переводы, mint, burn).
- **Значение:** Позволяет быстро остановить работу контракта при обнаружении уязвимостей или иных проблем.

## ReentrancyGuard

- **Определение:** Механизм защиты от повторных (reentrant) вызовов функций, изменяющих состояние, предотвращающий атаки повторного входа.
- **Значение:** Обеспечивает дополнительную безопасность для критически важных функций.

## Oracle

- **Определение:** Внешний источник данных (например, Chainlink Price Feed), предоставляющий актуальную информацию (в нашем случае — текущую цену токена).
- **Значение:** Используется для получения данных, необходимых для динамической корректировки предложения токенов.

## Basis Points

- **Определение:** Единица измерения, равная 1/100 процента (1 bp = 0,01%).
- **Значение:** Используется для задания размеров комиссий и параметра отклонения. Например, комиссия в 1 bp означает 0,01% от суммы перевода.

## Tolerance

- **Определение:** Допустимое отклонение от целевой цены, заданное в базисных пунктах.
- **Значение:** Если `tolerance = 100`, это означает, что допустимое отклонение составляет 100 базисных пунктов, то есть 1%.

## Peg (Stability Mechanism)

- **Определение:** Метод динамической корректировки предложения токенов посредством mint и burn для поддержания их стоимости близкой к заданному целевому уровню (peg).
  - **Значение:** Позволяет стейблкоину сохранять свою стоимость, привязанную к арабской валюте (например, 1 Кувейтский динар = 3,24 USD, где  $3,24 \times 10^8 = 324000000$ ).
- 

## 2. Общая логика

1. При запуске устанавливаются базовые параметры: источник данных для получения цены, адрес для сбора комиссий, начальные значения комиссии, целевой уровень цены и допустимое отклонение.
2. При каждом переводе токенов автоматически удерживается небольшая комиссия, которая направляется на заранее определенный адрес, а остаток суммы передается получателю. Также проверяется, что ни отправитель, ни получатель не находятся в списке заблокированных адресов.
3. Контракт регулярно получает актуальную рыночную цену из внешнего источника (Oracle) и сравнивает её с установленным целевым уровнем с учетом допустимого отклонения. Если цена выходит за пределы допустимого диапазона, контракт корректирует общее предложение токенов: если цена слишком высокая, выпускаются дополнительные токены; если цена слишком низкая, часть токенов сжигается.
4. Все важные операции защищены механизмами контроля доступа, паузы и защиты от повторного входа, что обеспечивает безопасность и возможность оперативного управления.
5. Логика контракта обновляется без изменения его адреса благодаря прокси-паттерну, что позволяет вносить изменения в функциональность без потери данных.



---

### 3. Подробная Документация Функций

#### State Variables

## Roles and Access Control

- **PAUSER\_ROLE, MINTER\_ROLE, BURNER\_ROLE:**

Константы типа `bytes32`, определяющие уникальные идентификаторы ролей.

- **PAUSER\_ROLE:** Разрешает вызов функций `pause()` и `unpause()`.
- **MINTER\_ROLE:** Разрешает вызов функции `mint()`.
- **BURNER\_ROLE:** Разрешает вызов функции `burn()`.

## Commission Parameters

- **transferFeeBasisPoints (uint256):**

Размер комиссии за перевод, выраженный в базисных пунктах.

*Default:* 1 (0.01%).

- **feeCollector (address):**

Адрес, на который направляются собранные комиссии.

Может быть обновлен через функцию `updateFeeCollector()`.

## Механизм Заморозки (Blacklist)

- **frozen (mapping(address => bool)):**

Состояние заморозки адресов; если значение `true`, адрес заблокирован для проведения переводов. Управляется через функции `freeze()` и `unfreeze()`.

## Oracle и Параметры Стабильности

- **priceFeed (AggregatorV3Interface):**

Интерфейс для взаимодействия с Chainlink Price Feed, откуда получают актуальную цену.

- **REFERENCE\_SUPPLY (uint256 constant):**

Базовое значение ( $1e24$ ) для справочных расчётов целевого предложения (используется в качестве ориентира).

- **peggedPrice (uint256):**

Целевая цена, к которой должен стремиться токен. По умолчанию равна 324000000 (что соответствует 3,24 USD при 8 десятичных знаках). Может обновляться через `updatePeggedPrice()`.

- **tolerance (uint256):**

Допустимое отклонение от целевой цены, заданное в базисных пунктах (по умолчанию 100 bp = 1%). Может обновляться через `updateTolerance()`.

## События

- **FeeUpdated:**

Генерируется при обновлении `transferFeeBasisPoints`.

- **FeeCollectorUpdated:**  
Генерируется при изменении `feeCollector`.
- **AccountFrozen / AccountUnfrozen:**  
Генерируются при заморозке или разморозке адреса.
- **SupplyAdjusted:**  
Генерируется после корректировки общего предложения токенов функцией `adjustSupply()`.
- **PeggedPriceUpdated:**  
Генерируется при обновлении значения `peggedPrice`.
- **ToleranceUpdated:**  
Генерируется при обновлении `tolerance`.

## Custom Errors

- **Unauthorized():**  
Выбрасывается, если вызывающий не обладает требуемыми правами или если один из адресов перевода заморожен.
- **ZeroAddress():**  
Выбрасывается, если передан нулевой адрес там, где это не допускается.
- **PriceInvalid():**  
Выбрасывается, если полученная из Oracle цена неверна ( $\leq 0$ ).

## Functions

### `initialize(address _priceFeed, address _feeCollector)`

- **Purpose:**  
Инициализирует контракт при деплое через прокси, устанавливая базовые параметры, роли и инициализируя модули.
- **Parameters:**
  - `_priceFeed`: Адрес Chainlink Price Feed. Должен быть ненулевым.
  - `_feeCollector`: Адрес для сбора комиссий. Должен быть ненулевым.
- **Internal Actions:**
  - Вызываются инициализирующие функции для ERC20, ERC20Permit, Pausable, AccessControl, UUPS и ReentrancyGuard.
  - Устанавливаются роли для `msg.sender` (DEFAULT\_ADMIN\_ROLE, PAUSER\_ROLE, MINTER\_ROLE, BURNER\_ROLE).
  - Инициализируются переменные `priceFeed`, `feeCollector`, `transferFeeBasisPoints` (1 bp), `peggedPrice` (324000000) и `tolerance` (100 bp).
- **Output:**  
Изменяет состояние контракта.

### **freeze(address account) / unfreeze(address account)**

- **Purpose:**  
Замораживают или размораживают указанный адрес, запрещая или разрешая переводы.
- **Parameters:**
  - **account:** Адрес, который необходимо заморозить или разморозить.
- **Access:**  
Только для DEFAULT\_ADMIN\_ROLE.
- **Output:**  
Эмитируют события **AccountFrozen** или **AccountUnfrozen**.

### **updateFee(uint256 newFeeBasisPoints) / updateFeeCollector(address newCollector)**

- **Purpose:**  
Позволяют обновлять параметры комиссии.
- **Parameters:**
  - **newFeeBasisPoints:** Новое значение комиссии (например, 1 bp = 0,01%).
  - **newCollector:** Новый адрес для сбора комиссий (не может быть нулевым).
- **Access:**  
Только для DEFAULT\_ADMIN\_ROLE.
- **Output:**  
Эмитируют события **FeeUpdated** и **FeeCollectorUpdated**.

### **updatePeggedPrice(uint256 newPeggedPrice) / updateTolerance(uint256 newTolerance)**

- **Purpose:**  
Позволяют изменять параметры стабильности.
- **Parameters:**
  - **newPeggedPrice:** Новая целевая цена (не может быть 0).
  - **newTolerance:** Новое значение допуска (в базисных пунктах).
- **Access:**  
Только для DEFAULT\_ADMIN\_ROLE.
- **Output:**  
Эмитируют события **PeggedPriceUpdated** и **ToleranceUpdated**.

### **\_transfer(address sender, address recipient, uint256 amount)**

- **Purpose:**  
Расширяет стандартную функцию ERC20 **\_transfer**, добавляя:
  - Проверку, что ни **sender**, ни **recipient** не находятся в списке заблокированных (frozen).
  - Вычисление комиссии, которая переводится на **feeCollector**.
  - Перевод оставшейся суммы получателю.

- **Parameters:**
  - **sender:** Адрес отправителя.
  - **recipient:** Адрес получателя.
  - **amount:** Сумма токенов для перевода.
- **Logic:**  
Если один из адресов заморожен, выбрасывается ошибка **Unauthorized()**.  
Комиссия вычисляется по формуле:  $fee = amount \times transferFeeBasisPoints / 10000$  и списывается с **sender** на **feeCollector**. Остаток переводится **recipient**.
- **Output:**  
Обновляет балансы токенов.

#### **\_beforeTokenTransfer(address from, address to, uint256 amount)**

- **Purpose:**  
Гарантирует, что все операции перевода (включая mint и burn) выполняются только когда контракт не приостановлен.
- **Parameters:**
  - **from:** Адрес отправителя (или нулевой адрес при mint).
  - **to:** Адрес получателя (или нулевой адрес при burn).
  - **amount:** Количество токенов.
- **Logic:**  
Используется модификатор **whenNotPaused**.
- **Output:**  
Не возвращает значение.

#### **mint(address to, uint256 amount) / burn(uint256 amount)**

- **Purpose:**
  - **mint:** Выпускает новые токены и зачисляет их на указанный адрес.
  - **burn:** Сжигает токены с баланса вызывающего.
- **Parameters:**
  - Для mint:
    - **to:** Адрес для зачисления токенов.
    - **amount:** Количество токенов для выпуска.
  - Для burn:
    - **amount:** Количество токенов для сжигания с баланса вызывающего.
- **Access:**
  - mint доступна только для MINTER\_ROLE.
  - burn доступна только для BURNER\_ROLE.
- **Modifiers:**  
**whenNotPaused** и **nonReentrant**.
- **Output:**  
Обновляют общий баланс токенов.



### **getLatestPrice() public view returns (uint256)**

- **Purpose:**  
Получает актуальную цену из Oracle (например, Chainlink Price Feed).
- **Parameters:**  
Нет.
- **Logic:**  
Вызывает метод `latestRoundData()` у `priceFeed`. Если возвращаемая цена  $\leq 0$ , выбрасывается ошибка `PriceInvalid()`.
- **Output:**  
Возвращает текущую цену в формате `uint256`.

### **adjustSupply() external**

- **Purpose:**  
Корректирует общее предложение токенов для поддержания цены близкой к целевой (`peggedPrice`).  
Функция может вызываться вручную или автоматически через Chainlink Keepers.
- **Logic:**
  1. Получается текущая цена с помощью `getLatestPrice()`.
  2. Вычисляются пороги:
    - **Верхний порог:** `upperThreshold=peggedPrice*(10000+tolerance)/10000`
    - **Нижний порог:** `lowerThreshold=peggedPrice*(10000-tolerance)/10000`
  3. Если текущая цена выше верхнего порога:
    - Вычисляется целевое предложение: `targetSupply=currentSupply*currentPrice/peggedPrice`
    - Если `targetSupply` больше текущего предложения, происходит mint разницы на адрес `feeCollector`.
  4. Если текущая цена ниже нижнего порога:
    - Аналогичным образом вычисляется `targetSupply`, и если текущее предложение превышает `targetSupply`, происходит burn разницы с адреса `feeCollector`.
- **Output:**  
Генерируется событие `SupplyAdjusted` с новым общим предложением токенов.

### **Chainlink Keepers Functions**

- **checkUpkeep(bytes calldata) external view returns (bool upkeepNeeded, bytes memory)**
  - **Purpose:**  
Проверяет, требуется ли корректировка предложения.
  - **Logic:**  
Вычисляются верхний и нижний пороги (как в `adjustSupply()`), затем

сравнивается текущая цена. Если текущая цена выше верхнего порога или ниже нижнего, возвращается `upkeepNeeded = true`.

- **Output:**  
Возвращает кортеж `(upkeepNeeded, "")`.
- **performUpkeep(bytes calldata) external**
  - **Purpose:**  
Вызывается Кеепер-нодами, если `checkUpkeep` возвращает true, и запускает функцию `adjustSupply()`.
  - **Output:**  
Запускает процесс корректировки предложения.

#### **\_authorizeUpgrade(address newImplementation) internal**

- **Purpose:**  
Защищает механизм обновления логики контракта (паттерн UUPS), разрешая обновление только администраторам.
- **Parameters:**
  - `newImplementation`: Адрес новой реализации контракта.
- **Access:**  
Только для DEFAULT\_ADMIN\_ROLE.
- **Output:**  
Не возвращает значения.

#### **pause() external и unpause() external**

- **Purpose:**  
Приостанавливают и возобновляют работу контракта, блокируя или разрешая операции (transfer, mint, burn, adjustSupply).
- **Access:**  
Только для PAUSER\_ROLE.
- **Output:**  
Изменяют состояние контракта (paused/unpaused).

---

## **4. Механизм Стабильности: Привязка к Арабской Валюте**

- **Целевой Курс (peggedPrice):**  
Значение `peggedPrice` установлено равным 324000000, что соответствует курсу «1 Кувейтский динар = 3,24 USD» при использовании 8 десятичных знаков. Это значение служит опорой для корректировки предложения.

- **Допустимое Отклонение (tolerance):**

Параметр `tolerance` задаёт максимально допустимое отклонение от целевой цены в базисных пунктах (например, 100 bp = 1%). На его основе вычисляются верхний и нижний пороги:

- **Верхний порог:**  $upperThreshold = peggedPrice \times (10000 + tolerance) / 10000$  Если текущая цена превышает этот порог, предложение должно увеличиваться.
- **Нижний порог:**  $lowerThreshold = peggedPrice \times (10000 - tolerance) / 10000$  Если текущая цена ниже этого порога, предложение должно сокращаться.

- **Корректировка Предложения (adjustSupply):**

Функция `adjustSupply()` автоматически корректирует предложение токенов:

- Если текущая цена выше верхнего порога, вычисляется целевое предложение  $targetSupply = currentSupply \times currentPrice / peggedPrice$  и если `targetSupply` больше текущего, контракт выпускает (mint) недостающие токены на адрес `feeCollector`.
- Если текущая цена ниже нижнего порога, аналогично вычисляется `targetSupply`, и если текущее предложение больше `targetSupply`, контракт сжигает (burn) разницу с адреса `feeCollector`.

- **Интеграция с Oracle:**

Функция `getLatestPrice()` обращается к Chainlink Price Feed для получения актуальной цены в реальном времени. Эта цена используется для вычисления порогов и определения необходимости корректировки предложения.

- **Автоматизация через Chainlink Keepers:**

Функции `checkUpkeep()` и `performUpkeep()` позволяют автоматически запускать корректировку предложения, если текущая цена отклоняется от целевой (с учетом `tolerance`) более, чем допустимо.