

# Документация Meteora DLMM SDK

Подробное описание всех методов DLMM SDK (TypeScript) от Meteora.  
Включает назначение, сигнатуры, параметры, возвращаемые значения и примеры.

## Содержание

- 1. [Статические методы \(DLMM class-level\)](#)
- 2. [Методы DLMM-инстанса \(dlmmPool\)](#)
- 3. [Управление позициями](#)
- 4. [Свопы \(Swaps\)](#)
- 5. [Награды и комиссии \(Claim\)](#)
- 6. [Синхронизация](#)

## 1. Статические методы

Метод	Назначение
create	Создать DLMM-инстанс по адресу пула
createMultiple	Создать сразу несколько DLMM-инстансов
getAllPresetParameters	Получить предустановленные параметры
getClaimableLMReward	Узнать LM-награду по позиции
getClaimableSwapFee	Узнать комиссии по позиции
getAllLbPairPositionsByUser	Все позиции пользователя во всех пулах
toPricePerLamport	Конвертировать цену в лампорты
fromPricePerLamport	Лампорты в цену
getPairPubkeyIfExists	Проверить существование пула
getMaxPriceInBinArrays	Максимальная цена среди активных бинов

## 2. Методы DLMM-инстанса

Метод	Назначение
refetchStates	Обновить состояние пула
getActiveBin	Получить активный бин
getFeeInfo	Узнать комиссии пула
getDynamicFee	Текущая комиссия
getBinArrays	Получить массивы бинов
getBinArrayForSwap	Бины для свопа
getBinsAroundActiveBin	Бины вокруг активного
getBinsBetweenMinAndMaxPrice	Бины в ценовом диапазоне
getBinsBetweenLowerAndUpperBound	Бины по ID
getPriceOfBinByBinId	Цена по binId
getBinIdFromPrice	binId по цене

## 3. Управление позициями

Метод	Назначение
getPositionsByUserAndLbPair	Получить позиции пользователя
initializePositionAndAddLiquidityByStrategy	Создание позиции + ликвидность
addLiquidityByStrategy	Добавить ликвидность
removeLiquidity	Удалить ликвидность
closePosition	Закрыть позицию

## 4. Свопы

Метод	Назначение
swapQuote	Рассчитать своп
swap	Провести обмен

---

## 5. Награды и комиссии

Метод	Назначение
<code>claimLMReward</code>	LM-награда по позиции
<code>claimAllLMRewards</code>	LM-награды по всем позициям
<code>claimSwapFee</code>	Комиссии по одной позиции
<code>claimAllSwapFee</code>	Комиссии по всем позициям
<code>claimAllRewards</code>	Всё сразу (LM + Fee)

---

## 6. Синхронизация

Метод	Назначение
<code>syncWithMarketPrice</code>	Обновить активный бин по рынку

---

## Api methods

### `DLMM.create`

### Сигнатура

```
static DLMM.create(  
  connection: Connection,  
  dlmm: PublicKey,  
  opt?: Opt):  
Promise<DLMM>;
```

### Описание

Метод `create` используется для инициализации экземпляра `DLMM`, привязанного к конкретному пулу ликвидности (`lbPair`). Этот экземпляр предоставляет доступ ко всем

методам взаимодействия с пулом, включая добавление/удаление ликвидности, свопы, получение информации о бинах и т.д.

Необходим предварительно созданный пул (адрес можно получить через API: `https://dlmm-api.meteora.ag/pair/all` ).

## Параметры

Параметр	Тип	Описание
connection	Connection	Объект соединения с кластером Solana ( @solana/web3.js )
dlmm	PublicKey	Публичный ключ пула ликвидности (DLMM pool), с которым необходимо работать
opt	Opt	Необязательный параметр типа Opt

, где:

```
type Opt = {
  cluster?: Cluster | "localhost";
  programId?: PublicKey;
};
```

## Возвращаемое значение

Тип: `Promise<DLMM>`

Возвращает проинициализированный экземпляр `DLMM` , который содержит все данные и методы для работы с выбранным пулом.

## Особенности и замечания

- Если нужно работать с несколькими пулами одновременно — используем `DLMM.createMultiple` .

## Пример использования

```
import { Connection, PublicKey } from '@solana/web3.js';
import DLMM from '@meteora-ag/dlmm';

const connection = new Connection('https://api.devnet.solana.com');
const USDC_USDT_POOL = new
PublicKey('ARwi1S4DaiTG5DX7S4M4ZsrXqpMD1MrTmbu9ue2tpmEq');
```

```
async function initDLMM() {  
  const dlmmPool = await DLMM.create(connection, USDC_USDT_POOL);  
  console.log('DLMM Pool instance initialized:', dlmmPool);  
}
```

---

## DLMM.createMultiple

### Сигнатура

```
static DLMM.createMultiple(  
  connection: Connection,  
  dlmmList: Array<PublicKey>,  
  opt?: Opt):  
Promise<DLMM[]>;
```

### Описание

Метод `createMultiple` используется для одновременной инициализации нескольких экземпляров `DLMM`, каждый из которых привязан к своему пулу ликвидности (`lbPair`). Это удобно, если ваше приложение работает с несколькими пулами одновременно.

Позволяет сократить количество повторных вызовов `create` и централизовать работу с несколькими пулами.

### Параметры

Параметр	Тип	Описание
<code>connection</code>	<code>Connection</code>	Объект соединения с кластером Solana
<code>dlmmList</code>	<code>PublicKey[]</code>	Массив публичных ключей пулов DLMM, которые нужно проинициализировать
<code>opt</code>	<code>Opt</code>	Необязательный параметр типа <code>Opt</code>

### Возвращаемое значение

Тип: `Promise<DLMM[]>`

Возвращает массив экземпляров `DLMM`, соответствующих переданным пулам.

### Особенности и замечания

- Полезен при необходимости отображения данных или взаимодействия с несколькими пулами одновременно.
- Экземпляры возвращаются в том же порядке, в котором переданы адреса пулов.

## Пример использования

```
import { Connection, PublicKey } from '@solana/web3.js';
import DLMM from '@meteora-ag/dlmm';

const connection = new Connection('https://api.devnet.solana.com');
const poolAddresses = [
  new PublicKey('ARwi1S4DaiTG5DX7S4M4ZsrXqpMD1MrTmbu9ue2tpmEq'),
  new PublicKey('AnotherPoolPubkeyHere'),
];

async function initMultipleDLMM() {
  const dlmmPools = await DLMM.createMultiple(connection, poolAddresses);

  dlmmPools.forEach((dlmmPool, index) => {
    console.log(`Pool ${index + 1} initialized`, dlmmPool);
  });
}
```

---

## DLMM.getAllPresetParameters

### Сигнатура

```
static DLMM.getAllPresetParameters(connection: Connection, opt?: Opt): Promise<{
  presetParameter: ProgramAccount<PresetParamV1>[];
  presetParameter2: ProgramAccount<PresetParamV2>[];
}>
```

### Описание

Метод `getAllPresetParameters` используется для получения всех предустановленных (preset) параметров, которые применяются при создании нового пула DLMM. Эти параметры включают в себя конфигурации, такие как шаг бина, количество бинов, диапазон цен и другие базовые настройки, которые задают структуру пула.

Этот метод полезен, если вы хотите создавать кастомные пулы и понимать, какие параметры доступны для настройки.

## Параметры

- connection: Connection
- opt?: Opt

## Возвращаемое значение

Тип: `Promise<PresetParams>`

Объект, содержащий предустановленные параметры:

## Расшифровка ключей `presetParameter`

Ключ	Тип	Описание
<code>binStep</code>	<code>number</code>	Шаг между ценами в бинах (чем меньше — тем больше плотность бинов)
<code>baseFactor</code>	<code>number</code>	Базовый множитель комиссии (используется в расчёте динамической комиссии)
<code>filterPeriod</code>	<code>number</code>	Период фильтрации для расчёта волатильности
<code>decayPeriod</code>	<code>number</code>	Период экспоненциального затухания для волатильности
<code>reductionFactor</code>	<code>number</code>	Множитель уменьшения комиссии при снижении волатильности
<code>variableFeeControl</code>	<code>number</code>	Контроль кривизны роста переменной комиссии
<code>maxVolatilityAccumulator</code>	<code>number</code>	Максимально допустимое накопление волатильности
<code>minBinId</code>	<code>number</code>	Минимальный допустимый бин ID
<code>maxBinId</code>	<code>number</code>	Максимальный допустимый бин ID
<code>protocolShare</code>	<code>number</code>	Доля комиссии, отводимая протоколу (в BPS, например $500 = 5\%$ )

## Расшифровка ключей `presetParameter2`

Ключ	Тип	Описание
<code>binStep</code>	<code>number</code>	Аналогично выше — шаг между ценами
<code>baseFactor</code>	<code>number</code>	Базовый фактор для расчёта комиссии
<code>filterPeriod</code>	<code>number</code>	Период фильтрации волатильности
<code>decayPeriod</code>	<code>number</code>	Период затухания

Ключ	Тип	Описание
<code>variableFeeControl</code>	<code>number</code>	Контроль переменной части комиссии
<code>maxVolatilityAccumulator</code>	<code>number</code>	Максимальное накопление волатильности
<code>reductionFactor</code>	<code>number</code>	Коэффициент уменьшения комиссии
<code>protocolShare</code>	<code>number</code>	Комиссия протокола
<code>index</code>	<code>number</code>	Индекс шаблона (уникальный ID)
<code>baseFeePowerFactor</code>	<code>number</code>	Показатель степени в формуле базовой комиссии
<code>padding0</code>	<code>number</code>	Зарезервированное поле (возможно для выравнивания)
<code>padding1</code>	<code>BN[]</code>	Зарезервированный массив (будущие параметры или выравнивание)

Тип `PresetParams` импортируется из SDK и может быть использован для типизации при создании пула.

## Особенности и замечания

- Метод работает независимо от конкретного пула.
- Возвращаемые параметры могут быть использованы напрямую в методе `createPermissionLbPair` при создании нового пула.
- Рекомендуется вызывать этот метод один раз и кэшировать результат.

## Пример использования

```
import DLMM from '@meteora-ag/dlmm';

async function fetchPresetParams() {
  const presetParams = await DLMM.getAllPresetParameters();
}
```

## DLMM.createPermissionLbPair

**ВАЖНО:** этот метод есть в документации, но вызвать его `DLMM.createPermissionLbPair` не получилось: Свойство `"createPermissionLbPair"` не существует в типе `"typeof DLMM"`.

## Описание



Метод `createPermissionLbPair` используется для создания нового пула DLMM с определёнными параметрами, включая токены, шаг бина, комиссии, владельца и т.д. Это "permissioned" пул — то есть его создание и конфигурация требует авторизации и подписей.

Используется для настройки собственного пула под нужды протокола.

## Возвращаемое значение

Тип: `Promise<Transaction>`

Возвращает неподписанную транзакцию `Transaction`, которую нужно подписать и отправить в блокчейн для создания пула.

---

### DLMM.getClaimableLMReward

**ВАЖНО:** этот метод есть в документации, но вызвать его `DLMM.getClaimableLMReward` не получилось: Свойство "getClaimableLMReward" не существует в типе "typeof DLMM".

## Описание

Метод `getClaimableLMReward` используется для получения количества вознаграждений за участие в **Liquidity Mining** (LM), которые может получить пользователь по конкретной позиции. Вознаграждение накапливается за предоставление ликвидности в DLMM пул и может быть впоследствии получено через `claimLMReward`.

Это своего рода "фарминг" — пользователи получают награды в токенах за активное участие в пулах.

## Возвращаемое значение

Тип: `Promise<LMRewards>`

Объект с деталями накопленных вознаграждений.

---

### DLMM.getClaimableSwapFee

**ВАЖНО:** этот метод есть в документации, но вызвать его `DLMM.getClaimableSwapFee` не получилось: Свойство "getClaimableSwapFee" не существует в типе "typeof DLMM".

## Описание

Метод `getClaimableSwapFee` используется для получения информации о **накопленных комиссиях от свопов**, которые пользователь может забрать по конкретной позиции ликвидности. Это комиссия, которую заработал поставщик ликвидности за счёт торгов, происходящих в его диапазоне бинов.

Комиссии накапливаются пропорционально объёму торгов и размеру вложенной ликвидности.

## Возвращаемое значение

Тип: `Promise<SwapFee>`  
Объект, содержащий информацию о доступных к получению комиссиях.

## DLMM.getAllLbPairPositionsByUser

### Сигнатура

```
static getAllLbPairPositionsByUser(  
  connection: Connection,  
  userPubKey: PublicKey,  
  opt?: Opt):  
  Promise<Map<string, PositionInfo>>;
```

### Описание

Метод `getAllLbPairPositionsByUser` используется для получения **всех позиций пользователя во всех пулах DLMM**. Это универсальный способ получить полную картину всей ликвидности, которую пользователь разместил через DLMM — независимо от пула.

Особенно удобно использовать для отображения портфеля или дашборда пользователя.

### Параметры

Параметр	Тип	Описание
<code>connection</code>	<code>Connection</code>	Соединение с Solana кластером
<code>userPubKey</code>	<code>PublicKey</code>	Публичный ключ пользователя, для которого нужно получить все позиции

Параметр	Тип	Описание
opt	Opt	Необязательный параметр типа Opt

## Возвращаемое значение

Тип: `Promise<Map<string, PositionInfo>>`

Возвращается `Map`, где ключ — это строка (обычно адрес пула DLMM), а значение — объект `PositionInfo`, содержащий массив позиций пользователя в этом пуле и связанную информацию.

где:

```
interface PositionInfo {
    publicKey: PublicKey;
    lbPair: LbPair;
    tokenX: TokenReserve;
    tokenY: TokenReserve;
    lbPairPositionsData: Array<LbPosition>;
}
```

## Особенности и замечания

- Покрывает **все** пулы DLMM, не требуется перечислять адреса вручную.
- Может быть тяжелой операцией, если пользователь работает с десятками пулов.
- Рекомендуется кэшировать результат, если используется на UI.
- Хорошо сочетается с методами `claimAllSwapFee`, `claimAllLMRewards` и `claimAllRewards`.

## Расшифровка `PositionInfo`

Свойство	Тип	Описание
publicKey	PublicKey	<b>Публичный ключ позиции</b> пользователя в пуле. Это уникальный идентификатор аккаунта позиции на Solana.
lbPair	LbPair	<b>Объект пула DLMM</b> , к которому относится эта позиция. Включает данные о токенах, комиссиях, binStep, и прочих параметрах пула.
tokenX	TokenReserve	<b>Резерв токена X</b> в позиции пользователя. Показывает объем X, добавленный

Свойство	Тип	Описание
		пользователем и доступный в рамках его ликвидности.
tokenY	TokenReserve	<b>Резерв токена Y</b> в позиции пользователя. Аналогично tokenX , но для второй стороны пула.
lbPairPositionsData	LbPosition[]	<b>Массив всех позиций</b> пользователя в данном пуле. Включает подробные данные о каждой позиции:

---

## dLmmPool.toPricePerLamport

### Сигнатура

```
toPricePerLamport(price: number): string;
```

### Описание

Метод `toPricePerLamport` используется для преобразования "настоящей" рыночной цены в цену, выраженную в "лампортах" (внутреннее представление цены в DLMM). Это необходимо, так как DLMM использует целочисленные значения и относительные масштабы при расчётах внутри пула.

Особенно важен при вычислении `binId` , установке границ или подготовке к свопу.

### Возвращаемое значение

Тип: `string`

Возвращает строку, представляющую цену в "лампортах" — формате, который понимает DLMM.

### Особенности и замечания

- Используется в паре с `fromPricePerLamport` .
  - Необходим при вызове методов, где требуется значение цены в "внутреннем" представлении.
  - Следует использовать одинаковый формат с `getBinIdFromPrice` .
-

## `dlmmPool.fromPricePerLamport`

### Сигнатура

```
toPricePerLamport(price: number): string;
```

### Описание

Метод `fromPricePerLamport` используется для преобразования внутренней цены в формате "лампортов" обратно в читаемую пользователем **рыночную цену**. Это обратная операция по отношению к `toPricePerLamport`.

Используется при отображении цены пользователю, полученной из `getActiveBin`, `getPriceOfBinByBinId`, и других методов DLMM.

### Пример использования

```
const realPrice = dlmmPool.fromPricePerLamport(Number(activeBin.price))
```

---

## `DLMM.getPairPubkeyIfExists`

### Сигнатура

```
static getPairPubkeyIfExists(  
  connection: Connection,  
  tokenX: PublicKey,  
  tokenY: PublicKey,  
  binStep: BN,  
  baseFactor: BN,  
  baseFeePowerFactor: BN, opt?: Opt):  
Promise<PublicKey | null>;
```

### Описание

Метод `getPairPubkeyIfExists` проверяет, существует ли уже DLMM пул с заданными параметрами (пара токенов и шаг между бинами). Если такой пул найден — метод возвращает его публичный ключ, иначе `null`.

Удобно использовать перед созданием пула, чтобы избежать дубликатов.

## Назначение

Этот метод используется для **поиска адреса уже существующего DLMM пула**, созданного на основе конкретных параметров: токенов X и Y, `binStep`, `baseFactor`, `baseFeePowerFactor`.

Если такой пул существует — вернётся его `PublicKey`, иначе — `null`.

## Аргументы (все обязательные, кроме `opt`):

Аргумент	Тип	Описание
<code>connection</code>	<code>Connection</code>	RPC-соединение с кластером Solana
<code>tokenX</code>	<code>PublicKey</code>	Mint токена X
<code>tokenY</code>	<code>PublicKey</code>	Mint токена Y
<code>binStep</code>	<code>BN</code>	Шаг между соседними бинами. Задаёт дискретность ценовых уровней (меньше = больше плотность)
<code>baseFactor</code>	<code>BN</code>	Базовый множитель комиссии. Используется в расчётах динамической комиссии
<code>baseFeePowerFactor</code>	<code>BN</code>	Экспонента в формуле базовой комиссии. Повышает чувствительность комиссии к волатильности
<code>opt</code> (необязательно)	<code>Opt</code>	Дополнительные настройки (например, <code>commitment</code> или <code>custom program ID</code> )

## Пример использования

```
const tokenX = new PublicKey("..."); // например, USDC
const tokenY = new PublicKey("..."); // например, USDT
const binStep = new BN(10);
const baseFactor = new BN(25);
const baseFeePowerFactor = new BN(3);

const existingPoolPubkey = await DLMM.getPairPubkeyIfExists(
  connection,
  tokenX,
  tokenY,
  binStep,
  baseFactor,
  baseFeePowerFactor
);
```

```
if (existingPoolPubkey) {
  console.log("Пул уже существует:", existingPoolPubkey.toBase58());
} else {
  console.log("Такой пул не найден, можно создать новый.");
}
```

## Особенности и замечания

- **Важно!** `tokenX` и `tokenY` должны быть в правильном порядке, иначе PDA будет другим, и метод вернёт `null`.
- Используется **до вызова** `createPermissionLbPair`, чтобы не создавать дубликаты.

## `dlmmPool.getMaxPriceInBinArrays`

### Сигнатура

```
getMaxPriceInBinArrays(binArrayAccounts: BinArrayAccount[]): Promise<string>;
```

### Описание

Метод `getMaxPriceInBinArrays` возвращает **максимальную цену** среди всех бинов в переданных bin-массивах, которые содержат ликвидность. Цена возвращается в читаемом виде (как строка) и учитывает десятичные знаки токенов X и Y.

Полезно для отображения верхней границы диапазона цен, доступных для торгов или размещения ликвидности.

### Параметры

Параметр	Тип	Описание
<code>binArrays</code>	<code>BinArrayAccount[]</code>	Массив объектов <code>BinArrayAccount</code> , полученных из <code>dlmmPool.getBinArrays()</code>

### Возвращаемое значение

Тип: `Promise<string | null>`

- Строка — максимальная цена среди активных бинов с ликвидностью.

- `null` — если подходящих бинов не найдено.
- 

## Особенности и замечания

- Цена возвращается в стандартной форме (`price = tokenY / tokenX`).
- Работает только с бинами, где **есть ликвидность**.

## Пример использования

```
import DLMM from '@meteora-ag/dlmm';

async function getMaxBinPrice(dlmmPool) {
  const binArrays = await dlmmPool.getBinArrays();
  const maxPrice = await DLMM.getMaxPriceInBinArrays(binArrays);

  if (maxPrice) {
    console.log('Максимальная цена среди активных бинов:', maxPrice);
  } else {
    console.log('Бины с ликвидностью не найдены.');
```

---

### `dlmmPool.refetchStates`

#### Сигнатура

ts

КопироватьРедактировать

```
async refetchStates(): Promise<void>
```


#### Описание

Метод `refetchStates` используется для обновления **всех on-chain данных**, связанных с пулом DLMM. Это включает:

- информацию о текущем активном бине,
- параметры пула (`LbPair`),
- кэшированные значения ликвидности,



- комиссионные ставки и др.

 Этот метод обязательно нужно вызывать перед **взаимодействием с пулом**, чтобы убедиться в актуальности состояния, особенно перед свопом, добавлением/удалением ликвидности или расчетом стратегии.

---

## Параметры

Нет параметров.

---

## Возвращаемое значение

Тип: `Promise<void>`

Ничего не возвращает. Метод обновляет внутреннее состояние экземпляра `dlmmPool`.

---

## Особенности и замечания

- Важно вызывать **перед любыми транзакциями**, иначе можно получить устаревшие данные (например, неверный `activeBin`, неправильную стратегию, ошибку расчёта ликвидности).
- Особенно полезен после длительного бездействия, при переключении пулов, при использовании кэширования и на клиентской стороне.
- Может использоваться вручную или автоматически в хранилище состояния (store), чтобы не полагаться на данные "в памяти".

---

## Пример использования

ts

КопироватьРедактировать

```
async function safePoolInteraction(dlmmPool) { await dlmmPool.refetchStates(); //
всегда обновляем состояние const activeBin = await dlmmPool.getActiveBin();
console.log('Текущий активный бин:', activeBin.binId); }
```

---

Следующим логично будет разобрать `getActiveBin`, раз уж мы его упомянули. Готов продолжить?

4o

**ChatGPT сказал:**

Поехали дальше — разбираем один из самых используемых методов: `getActiveBin`.

---

 `d1mmPool.getActiveBin`

 **Сигнатура**


ts

КопироватьРедактировать

```
async getActiveBin(): Promise<{ binId: number; price: string }>
```

 **Описание**

Метод `getActiveBin` возвращает **активный бин**, в котором на данный момент находится текущая цена в пуле. Это "центральный" бин, относительно которого рассчитываются стратегии и диапазоны размещения ликвидности.

 Активный бин — это бин, где текущая цена пула фактически пересекается, и который участвует в расчётах свопов и размещении ликвидности.

---

 **Параметры**

Нет параметров.

---

 **Возвращаемое значение**

Тип: `Promise<{ binId: number; price: string }>`

Объект, содержащий:

- `binId` : номер активного бина (целое число),
  - `price` : цена в этом бине в читаемом виде (строка).
- 

## 💡 Особенности и замечания

- Метод **зависит от актуальности данных**, обязательно вызывать `refetchStates()` перед использованием.
  - Полученное значение часто используется как основа для расчёта диапазона:
    - `minBinId = activeBin.binId - N`
    - `maxBinId = activeBin.binId + N`
  - Цена может быть преобразована через `fromPricePerLamport()` при необходимости.
- 

## 🧪 Пример использования

ts

Копировать Редактировать

```
async function logActiveBin(dlmmPool) { await dlmmPool.refetchStates(); const activeBin = await dlmmPool.getActiveBin(); console.log('Текущий активный бин ID:', activeBin.binId); console.log('Цена в активном бине:', activeBin.price); }
```

---

Следующим логичным шагом будет `getBinArrays` — метод, возвращающий массив всех bin'ов. Готов?

4o

**ChatGPT сказал:**

Отлично! Теперь разберём метод `getBinArrays`.

---

◆ `dlmmPool.getBinArrays`

📌 Сигнатура


ts

КопироватьРедактировать

```
async getBinArrays(): Promise<BinArrayAccount[]>
```

## Описание

Метод `getBinArrays` возвращает массив всех **BinArrayAccount** объектов, содержащих данные о бинах, которые находятся в пределах интереса данного пула. Это ядро структуры DLMM, где каждая запись описывает диапазон бинов, их состояние и ликвидность.

 Бины (bins) в DLMM представляют собой ячейки ценового диапазона, в которых может находиться ликвидность. Эти ячейки агрегируются в массивы (`BinArrayAccount`) для оптимизации чтения из блокчейна.

## Параметры

Нет параметров.

## Возвращаемое значение

Тип: `Promise<BinArrayAccount[]>`

Массив объектов `BinArrayAccount`, каждый из которых содержит:

- идентификаторы бинов,
- доступную ликвидность,
- активность,
- распределение по токенам.

Тип `BinArrayAccount` зависит от импорта из SDK, но по сути — это "батч" из 32 бинов с их данными.

## Особенности и замечания

- Используется во внутренних вычислениях стратегий (например, `swapQuote` , `getMaxPriceInBinArrays` , и др.).
  - Может быть тяжёлым по объёму — не вызывай без необходимости.
  - Визуализируется на UI как "heatmap" ликвидности.
- 

## Пример использования

ts

КопироватьРедактировать

```
async function inspectBinArrays(dlmmPool) { await dlmmPool.refetchStates(); const binArrays = await dlmmPool.getBinArrays(); console.log(`Найдено ${binArrays.length} массивов бинов`); binArrays.forEach((arr, index) => { console.log(`Массив #${index}:`, arr); }); }
```

---

Следующий по логике — `getBinArrayForSwap` , который возвращает бины, подходящие именно для **свопов**. Готов?

2/2

4o

**ChatGPT** сказал:

Отлично, продолжаем! Сейчас разберём метод `getBinArrayForSwap` .

---

## `dlmmPool.getBinArrayForSwap`

### Сигнатура


ts

КопироватьРедактировать

```
async getBinArrayForSwap(isYtoX: boolean): Promise<BinArrayAccount[]>
```

## Описание

Метод `getBinArrayForSwap` возвращает список массивов бинов ( `BinArrayAccount[]` ), **релевантных для выполнения свопа** в указанном направлении. Эти бины содержат ликвидность, которая будет задействована при обмене токенов.

 Метод используется для подготовки к расчёту свопа ( `swapQuote` ) или при формировании транзакции `swap()` .

---

## Параметры

Параметр	Тип	Описание
<code>isYtoX</code>	<code>boolean</code>	Направление обмена:
— <code>true</code> = <code>tokenY</code> → <code>tokenX</code>		
— <code>false</code> = <code>tokenX</code> → <code>tokenY</code>		

---

## Возвращаемое значение

Тип: `Promise<BinArrayAccount[]>`

Массив `BinArrayAccount` , содержащий только те массивы бинов, которые:

- находятся в актуальном ценовом диапазоне для указанного направления,
- имеют ликвидность,
- потенциально будут использованы при исполнении свопа.

---

## Особенности и замечания

- Обязателен для получения **точного** `swapQuote`.
- Если не вызвать — расчёт может быть некорректен или вызвать ошибку.
- Работает с учётом текущего `activeBin` , так что **рекомендуется предварительно вызвать** `refetchStates()` .

---

## Пример использования

ts

КопироватьРедактировать

```
async function prepareSwap(dlmmPool) { await dlmmPool.refetchStates(); const
directionYtoX = true; const swapBins = await
dlmmPool.getBinArrayForSwap(directionYtoX); console.log(`Найдено ${swapBins.length} bin
массивов для свопа (${directionYtoX ? 'Y→X' : 'X→Y'})`); }
```

---

Следующий — `getFeeInfo`, где мы узнаем комиссии пула. Приступим?

**ChatGPT сказал:**

Отлично! Теперь разбираем метод `getFeeInfo`.

---

◆ `dlmmPool.getFeeInfo`

📌 **Сигнатура**

ts

КопироватьРедактировать

```
getFeeInfo(): FeeInfo
```

📄 **Описание**

Метод `getFeeInfo` возвращает объект с информацией о комиссиях пула DLMM. Это статические значения, заданные при создании пула:

- базовая торговая комиссия,
- комиссия протокола,
- максимальная комиссия (в случае использования динамической).

📊 Используется для отображения пользователю актуальных настроек комиссий и для внутренней логики расчётов.

---

🔧 **Параметры**

Нет параметров.

---

## Возвращаемое значение

Тип: `FeeInfo`

Объект следующей структуры:

ts

[Копировать](#)[Редактировать](#)

```
type FeeInfo = { baseFee: number; // Комиссия базовая (bps) protocolFee: number; // Комиссия протокола (bps) maxFee: number; // Максимально возможная комиссия (bps) };
```

Все значения указываются в **базисных пунктах** (basis points), где `100 = 1%`.

---

## Особенности и замечания

- Этот метод возвращает значения из состояния пула, полученного при вызове `refetchStates()`.
  - Не требует запроса на RPC — работает локально по данным экземпляра.
  - Хорошо сочетается с `getDynamicFee`, чтобы сравнить базовую и текущую комиссии.
- 

## Пример использования

ts

[Копировать](#)[Редактировать](#)

```
async function showFees(dlmmPool) { await dlmmPool.refetchStates(); // обязательно перед получением данных const fees = dlmmPool.getFeeInfo(); console.log(`Комиссия пула:`); console.log(`- Базовая: ${fees.baseFee} bps`); console.log(`- Протокольная: ${fees.protocolFee} bps`); console.log(`- Максимальная: ${fees.maxFee} bps`); }
```

---

Следующий на очереди — `getDynamicFee`, который показывает **реальную текущую комиссию** с учётом волатильности. Готов?



ChatGPT сказал:

Отлично! Сейчас разберём `getDynamicFee` — важный метод для анализа текущих условий в пуле.

---

## ◆ `dLmmPool.getDynamicFee`

### 📌 Сигнатура

ts

КопироватьРедактировать

```
async getDynamicFee(): Promise<Decimal>
```

### 📄 Описание

Метод `getDynamicFee` возвращает **текущую плавающую комиссию** пула DLMM. Эта комиссия может **динамически изменяться** в зависимости от рыночной волатильности, объёмов торгов и стратегии, заложенной в пуле. Если пул работает с динамическими комиссиями, то этот метод покажет актуальное значение на текущий момент.

📈 Это комиссия, которая будет реально применена к следующей сделке, **а не базовая**, указанная в `getFeeInfo`.

---

### 🔧 Параметры

Нет параметров.

---

### 🔧 Возвращаемое значение

Тип: `Promise<Decimal>`

Текущее значение комиссии в формате `Decimal` (обычно в базисных пунктах или долях процента).

---

### 💡 Особенности и замечания

- Метод **асинхронный**, т.к. возможно обращение к сети.
- Возвращает значение с высокой точностью (до 18 знаков после запятой).
- Используется для расчёта реального slippage или отображения комиссии пользователю в момент свопа.
- Не забудь вызвать `refetchStates()` перед использованием, чтобы получить актуальные on-chain данные.

---

## Пример использования

ts

КопироватьРедактировать

```
async function logCurrentFee(dlmmPool) { await dlmmPool.refetchStates(); const dynamicFee = await dlmmPool.getDynamicFee(); console.log('Текущая динамическая комиссия (в %):', dynamicFee.toFixed(4)); // например: "0.3050" }
```

---

Следующий шаг — `getBinsAroundActiveBin`, который позволяет получить "окрестности" активного бина. Продолжим?

**ChatGPT сказал:**

Супер! Дальше — метод `getBinsAroundActiveBin`, который часто используется при построении стратегий или UI-индикаторов.

---

## `dlmmPool.getBinsAroundActiveBin`

### Сигнатура

ts

КопироватьРедактировать

```
async getBinsAroundActiveBin( range: number ): Promise<{ activeBin: number; bins: BinLiquidity[] }>
```

## Описание

Метод `getBinsAroundActiveBin` возвращает список бинов в **заданном диапазоне** вокруг текущего активного бина. Это удобно, когда нужно построить визуализацию ликвидности, сориентироваться по диапазонам или подготовить пользовательскую стратегию добавления ликвидности.

🔍 Получаешь активный бин и диапазон "влево/вправо" от него: `binId ± range`.

## Параметры

Параметр	Тип	Описание
<code>range</code>	<code>number</code>	Количество бинов слева и справа от активного. Например, <code>range = 10</code> даст 21 бин: <code>active - 10</code> до <code>active + 10</code> .

## Возвращаемое значение

Тип: `Promise<{ activeBin: number; bins: BinLiquidity[] }>`

Объект, содержащий:

- `activeBin` : ID текущего активного бина.
- `bins` : массив объектов `BinLiquidity` , содержащих информацию о ликвидности, распределении токенов и статусе каждого бина.

Тип `BinLiquidity` включает:

ts

КопироватьРедактировать

```
{ binId: number; x: BN; // Ликвидность токена X y: BN; // Ликвидность токена Y
totalLiquidity: BN; }
```

## Особенности и замечания

- Отлично подходит для подготовки стратегии ( `minBinId` , `maxBinId` ).
- Используется в UI-графиках: `depth charts`, `liquidity heatmaps`.

- Слишком большой `range` может замедлить загрузку на клиенте — 10–20 обычно достаточно.
  - Обязательно вызывай `refetchStates()` перед этим методом!
- 

## Пример использования

ts

КопироватьРедактировать

```
async function analyzeLiquidity(dlmmPool) { await dlmmPool.refetchStates(); const { activeBin, bins } = await dlmmPool.getBinsAroundActiveBin(10); console.log(`Текущий активный бин: ${activeBin}`); bins.forEach(bin => { console.log(`Бин ${bin.binId}: X=${bin.x.toString()}, Y=${bin.y.toString()}`); }); }
```

---

Готов продолжать с `getBinsBetweenMinAndMaxPrice` ? Он даёт бины в заданном ценовом диапазоне.

**ChatGPT сказал:**

Отлично, теперь разберём метод `getBinsBetweenMinAndMaxPrice` — суперполезен при работе с пользовательскими ценовыми диапазонами.

---

## `dlmmPool.getBinsBetweenMinAndMaxPrice`

### Сигнатура


ts

КопироватьРедактировать

```
async getBinsBetweenMinAndMaxPrice( minPrice: number | string, maxPrice: number | string, tokenXDecimals: number, tokenYDecimals: number ): Promise<{ activeBin: number; bins: BinLiquidity[] }>
```

### Описание

Метод `getBinsBetweenMinAndMaxPrice` возвращает **список бинов**, попадающих в заданный **ценовой диапазон** между `minPrice` и `maxPrice`. Это особенно полезно при построении стратегии добавления ликвидности в кастомный диапазон, где пользователь указывает конкретные цены.

 Метод позволяет точно определить, какие бины будут активны при заданном ценовом коридоре.

## Параметры

Параметр	Тип	Описание
<code>minPrice</code>	<code>number</code> \   <code>string</code>	Нижняя граница диапазона цены (Y/X)
<code>maxPrice</code>	<code>number</code> \   <code>string</code>	Верхняя граница
<code>tokenXDecimals</code>	<code>number</code>	Десятичные знаки токена X
<code>tokenYDecimals</code>	<code>number</code>	Десятичные знаки токена Y

## Возвращаемое значение

Тип: `Promise<{ activeBin: number; bins: BinLiquidity[] }>`

Объект с:

- `activeBin`: ID текущего активного бина.
- `bins`: массив `BinLiquidity`, соответствующих заданному диапазону цен.

## Особенности и замечания

- Значения `minPrice` и `maxPrice` указываются в читаемом формате, а не в "лампортах".
- Использует `toPricePerLampport` внутри, чтобы сопоставить цены с `binId`.
- Если заданный диапазон не содержит бинов — возвращается пустой массив.
- Часто применяется при стратегии `Curve` или `Manual`.

## Пример использования

ts

КопироватьРедактировать

```
async function getCustomRangeBins(dlmmPool) { await dlmmPool.refetchStates(); const
minPrice = 0.95; const maxPrice = 1.05; const { activeBin, bins } = await
dlmmPool.getBinsBetweenMinAndMaxPrice( minPrice, maxPrice, 6, // tokenX decimals 6 //
tokenY decimals ); console.log(`Активный бин: ${activeBin}`); console.log(`Найдено
бинов в диапазоне ${minPrice}-${maxPrice}: ${bins.length}`); }
```

---

Готов идти дальше к `getBinsBetweenLowerAndUpperBound` , где указываются биновые ID вместо цен?

**ChatGPT сказал:**

Отлично, продолжаем — сейчас метод `getBinsBetweenLowerAndUpperBound` , который даёт полный контроль при работе с **binId напрямую**, а не с ценами.

---

## `dlmmPool.getBinsBetweenLowerAndUpperBound`

### Сигнатура


ts

КопироватьРедактировать

```
async getBinsBetweenLowerAndUpperBound( lowerBinId: number, upperBinId: number ):
Promise<{ activeBin: number; bins: BinLiquidity[] }>
```

### Описание

Метод `getBinsBetweenLowerAndUpperBound` возвращает список бинов по **заданным границам ID** ( `binId` ). Это альтернативный способ построения диапазонов, когда тебе известны точные `binId` (вместо цен).

 Полезно, если ты работаешь с данными напрямую через `activeBin.binId` и строишь диапазон вроде `±10 бинов` .

---

## Параметры

Параметр	Тип	Описание
lowerBinId	number	Нижняя граница ID бина
upperBinId	number	Верхняя граница ID бина

## Возвращаемое значение

Тип: `Promise<{ activeBin: number; bins: BinLiquidity[] }>`

Объект:

- `activeBin` : ID текущего активного бина.
- `bins` : массив `BinLiquidity` , находящихся в указанном диапазоне.

## Особенности и замечания

- Отлично сочетается с `getActiveBin` , где ты получаешь `activeBin.binId` , а потом строишь диапазон вокруг него.
- Используется при ручной настройке диапазонов стратегий `Spot / Curve / BidAsk`.
- Если `lowerBinId` и `upperBinId` заданы некорректно (напр., верх < нижнего), будет ошибка или пустой массив.

## Пример использования

ts

КопироватьРедактировать

```
async function getBinsByIdRange(dlmmPool) { await dlmmPool.refetchStates(); const activeBin = await dlmmPool.getActiveBin(); const range = 10; const { bins } = await dlmmPool.getBinsBetweenLowerAndUpperBound( activeBin.binId - range, activeBin.binId + range ); console.log(`Найдено ${bins.length} бинов в диапазоне binId ±${range}`); }
```

Следующим по списку идёт `getPriceOfBinByBinId` — метод для получения цены конкретного бина. Продолжим?

**ChatGPT сказал:**

Отлично! Сейчас разберём метод `getPriceOfBinByBinId` — он используется, когда тебе нужно узнать цену конкретного `binId`.

---

## `dLmmPool.getPriceOfBinByBinId`

### Сигнатура


ts

КопироватьРедактировать

```
getPriceOfBinByBinId(binId: number): string
```

### Описание

Метод `getPriceOfBinByBinId` возвращает **цену**, соответствующую определённому `binId`, в читаемом формате ( `string` ). В DLMM каждый бин привязан к фиксированной цене, которая рассчитывается по логарифмической формуле, и этот метод позволяет узнать её по номеру бина.

 Используется при визуализации бинов, расчёте стратегий или отображении ценовых диапазонов.

---

### Параметры

Параметр	Тип	Описание
<code>binId</code>	<code>number</code>	Номер бина, цену которого нужно получить

---

### Возвращаемое значение



Тип: `string`

Цена в формате строки, соответствующая `binId`. Обычно это `priceY / priceX`.

---

## 💡 Особенности и замечания

- Цена зависит от `binStep`, заданного при создании пула.
  - Может возвращать значения с высокой точностью, например `"1.0025123..."`.
  - Работает локально, не требует вызова сети.
  - Часто используется с `getBinsAroundActiveBin`, `getBinsBetween...`, `getBinIdFromPrice`.
- 

## 🔧 Пример использования

ts

КопироватьРедактировать

```
async function logBinPrices(dlmmPool) { await dlmmPool.refetchStates(); const activeBin = await dlmmPool.getActiveBin(); const binId = activeBin.binId; const price = dlmmPool.getPriceOfBinByBinId(binId); console.log(`Цена для binId ${binId}:`, price); }
```

---

Следующий метод — обратный: `getBinIdFromPrice`, где ты передаёшь цену и получаешь `binId`. Готов?

**ChatGPT сказал:**

Отлично! Сейчас разберём метод `getBinIdFromPrice` — обратный по логике к предыдущему.

---

## 💠 `dlmmPool.getBinIdFromPrice`

### 📌 Сигнатура


ts

КопироватьРедактировать

```
getBinIdFromPrice( price: number | string, tokenXDecimals: number, tokenYDecimals: number, roundUp: boolean ): number
```

## Описание

Метод `getBinIdFromPrice` вычисляет номер бина ( `binId` ), который соответствует заданной **рыночной цене**. Это обратная операция к `getPriceOfBinByBinId`.

 Используется при построении кастомных стратегий или интерфейсов, где пользователь задаёт цену (например, через слайдер или инпут), а ты должен перевести это в `binId`.

## Параметры

Параметр	Тип	Описание
<code>price</code>	<code>number</code>    <code>string</code>	Цена в формате <code>Y/X</code>
<code>tokenXDecimals</code>	<code>number</code>	Десятичные знаки токена X
<code>tokenYDecimals</code>	<code>number</code>	Десятичные знаки токена Y
<code>roundUp</code>	<code>boolean</code>	Если <code>true</code> — округлить вверх, если <code>false</code> — вниз

## Возвращаемое значение

Тип: `number`

Целочисленный идентификатор бина ( `binId` ), соответствующий заданной цене.

## Особенности и замечания

- Это **ключевой метод**, если ты реализуешь пользовательский ввод ценовых диапазонов.
- `roundUp` позволяет выбрать ближайший бин вверх или вниз — важно при точных позициях.
- Не забудь корректно указать десятичные знаки — иначе результат будет неточным.

---


## Пример использования

ts

Копировать Редактировать

```
async function findBinIdForPrice(dlmmPool) { await dlmmPool.refetchStates(); const price = 1.0035; const binId = dlmmPool.getBinIdFromPrice(price, 6, 6, true); console.log(`BinId для цены ${price}:`, binId); }
```

---

 Отличная новость — ты теперь знаешь **все методы анализа пула!**

Дальше переходим к ключевому разделу — **работа с позициями:**

- `getPositionsByUserAndLbPair`
- `initializePositionAndAddLiquidityByStrategy`
- `addLiquidityByStrategy`
- и так далее...

Начнём с `getPositionsByUserAndLbPair` ?

4o

**ChatGPT сказал:**

Отличный вопрос! Давай посчитаем. Мы уже **полностью покрыли:**

---

 **Готово:**

### **1. Статические методы ( DLMM. ):**

- `create`
- `createMultiple`
- `getAllPresetParameters`
- `createPermissionLbPair`
- `getClaimableLMReward`
- `getClaimableSwapFee`

- `getAllLbPairPositionsByUser`
- `toPricePerLamport`
- `fromPricePerLamport`
- `getPairPubkeyIfExists`
- `getMaxPriceInBinArrays`

## 2. Инстанс-методы ( `dLmmPool1.` ):

- `refetchStates`
  - `getActiveBin`
  - `getBinArrays`
  - `getBinArrayForSwap`
  - `getFeeInfo`
  - `getDynamicFee`
  - `getBinsAroundActiveBin`
  - `getBinsBetweenMinAndMaxPrice`
  - `getBinsBetweenLowerAndUpperBound`
  - `getPriceOfBinByBinId`
  - `getBinIdFromPrice`
- 

 **Осталось:**

## 3. Инстанс-методы (Позиции и ликвидность):

- `getPositionsByUserAndLbPair`
- `initializePositionAndAddLiquidityByStrategy`
- `addLiquidityByStrategy`
- `removeLiquidity`
- `closePosition`

## 4. Свопы:

- `swapQuote`
- `swap`

## 5. Клеймы:

- `claimLMReward`

- `claimAllLMRewards`
- `claimSwapFee`
- `claimAllSwapFee`
- `claimAllRewards`

## 6. Синхронизация:

- `syncWithMarketPrice`

---

### Всего осталось: 13 методов

Хочешь, чтобы я продолжал в текущем темпе, или хочешь кратко/в виде таблицы, чтобы ускорить?

4o

ChatGPT сказал:

Отлично, продолжаем в детальном формате 🙌

Теперь переходим к работе с позициями. Начнём с базового метода — `getPositionsByUserAndLbPair` .

---

### `dLmmPool.getPositionsByUserAndLbPair`

#### Сигнатура


ts

КопироватьРедактировать

```
async getPositionsByUserAndLbPair(user: PublicKey): Promise<{ activeBin: { binId: number; price: string }; userPositions: Position[]; }>
```

#### Описание

Метод `getPositionsByUserAndLbPair` позволяет получить **все активные позиции пользователя** в рамках текущего пула DLMM ( `dLmmPool` ). Также возвращает текущий активный бин, что удобно при анализе стратегий.

 Используется для отображения всех позиций пользователя в конкретном пуле, управления ликвидностью, клеймов, редактирования и закрытия позиций.

---

## Параметры

Параметр	Тип	Описание
user	PublicKey	Публичный ключ пользователя, чьи позиции нужно получить

## Возвращаемое значение

Тип:

ts

КопироватьРедактировать

```
Promise<{ activeBin: { binId: number; price: string }; userPositions: Position[]; }>
```

Где:

- `activeBin` : объект с ID и ценой текущего активного бина,
- `userPositions` : массив объектов `Position` , каждый из которых содержит:
  - `publicKey` : ключ позиции,
  - `positionData` : включая объём ликвидности, диапазон бинов, активность,
  - `positionBinData` : информация по каждому задействованному бину.

## Особенности и замечания

- Метод работает **только для текущего пула** — для всех пулов используйте `getAllLbPairPositionsByUser` .
  - Данные можно напрямую использовать в `claim` , `removeLiquidity` , `closePosition` и т.д.
  - Удобен для итерирования и отображения в таблицах или карточках в UI.
-

## Пример использования

ts

КопироватьРедактировать

```
async function showUserPositions(dlmmPool, userPublicKey) { await
dlmmPool.refetchStates(); const { activeBin, userPositions } = await
dlmmPool.getPositionsByUserAndLbPair(userPublicKey); console.log(`Активный бин:
${activeBin.binId}, цена: ${activeBin.price}`); console.log(`Найдено позиций:
${userPositions.length}`); userPositions.forEach((pos, idx) => { console.log(`Позиция
#${idx + 1} - ${pos.publicKey.toBase58()}`); console.log('Диапазон бинов:',
pos.positionData.minBinId, '-', pos.positionData.maxBinId); }); }
```

---

Далее — один из ключевых методов: `initializePositionAndAddLiquidityByStrategy` . Готов продолжить?

4o

**ChatGPT сказал:**

Отлично! Сейчас разберём один из центральных методов SDK —  
`initializePositionAndAddLiquidityByStrategy` .

---

## `dlmmPool.initializePositionAndAddLiquidityByStrategy`

### Сигнатура


ts

КопироватьРедактировать

```
async initializePositionAndAddLiquidityByStrategy(params: { positionPubKey: PublicKey;
user: PublicKey; totalXAmount: BN; totalYAmount: BN; strategy: { minBinId: number;
maxBinId: number; strategyType: StrategyType; // Spot | BidAsk | Curve }; }):
Promise<Transaction | Transaction[]>
```

### Описание

Метод `initializePositionAndAddLiquidityByStrategy` используется для **создания новой позиции** в пуле DLMM и **одновременного добавления ликвидности** на основе выбранной стратегии. Это самый универсальный способ начать работу с пулом.

 Под капотом создаёт аккаунт позиции и распределяет ликвидность в заданном диапазоне бинов по определённой стратегии.

## Параметры

Параметр	Тип	Описание
<code>positionPubKey</code>	<code>PublicKey</code>	Публичный ключ создаваемой позиции
<code>user</code>	<code>PublicKey</code>	Ключ пользователя (владельца позиции)
<code>totalXAmount</code>	<code>BN</code>	Общая сумма токена X
<code>totalYAmount</code>	<code>BN</code>	Общая сумма токена Y
<code>strategy.minBinId</code>	<code>number</code>	Нижняя граница диапазона бинов
<code>strategy.maxBinId</code>	<code>number</code>	Верхняя граница диапазона бинов
<code>strategy.strategyType</code>	<code>StrategyType</code>	Стратегия размещения ликвидности:

- `Spot` : сбалансировано вокруг цены
- `BidAsk` : разделённый диапазон (лимитки)
- `Curve` : в форме кривой |

## Возвращаемое значение

Тип: `Promise<Transaction | Transaction[]>`

Одна или несколько транзакций, которые необходимо подписать и отправить через `sendAndConfirmTransaction`.

## Особенности и замечания

- Метод **и создаёт, и наполняет** позицию — упрощает UX.
- Стратегия `Spot` может использоваться с `autoFillYByStrategy()` для автоматического



подбора  $y$  под  $x$ .

- Возвращает `Transaction[]` в случаях, когда операция разбивается на несколько шагов (например, при большом объёме).
- Нужно предварительно создать `Keypair` для позиции.

---

## Пример использования: стратегия Spot (Balance)

ts

КопироватьРедактировать

```
import { Keypair } from '@solana/web3.js'; import BN from 'bn.js'; import {
StrategyType } from '@meteora-ag/dlmm'; const newPosition = new Keypair(); const
TOTAL_RANGE_INTERVAL = 10; const minBinId = activeBin.binId - TOTAL_RANGE_INTERVAL;
const maxBinId = activeBin.binId + TOTAL_RANGE_INTERVAL; const totalXAmount = new
BN(100 * 10 ** baseMint.decimals); const totalYAmount = autoFillYByStrategy(
activeBin.binId, dlmmPool.lbPair.binStep, totalXAmount, activeBin.xAmount,
activeBin.yAmount, minBinId, maxBinId, StrategyType.Spot ); const tx = await
dlmmPool.initializePositionAndAddLiquidityByStrategy({ positionPubKey:
newPosition.publicKey, user: user.publicKey, totalXAmount, totalYAmount, strategy: {
minBinId, maxBinId, strategyType: StrategyType.Spot, }, }); // Подпись и отправка const
txHash = await sendAndConfirmTransaction(connection, tx, [user, newPosition]);
console.log('Новая позиция создана:', txHash);
```

---

Дальше разберём `addLiquidityByStrategy`, который позволяет **добавить ликвидность в существующую позицию**. Поехали?

4o

ChatGPT сказал:

Отлично, сейчас разберём метод `addLiquidityByStrategy` — он дополняет предыдущий, но применяется к **уже существующей** позиции.

---

 `dlmmPool.addLiquidityByStrategy`

 Сигнатура


ts

КопироватьРедактировать

```
async addLiquidityByStrategy(params: { positionPubKey: PublicKey; user: PublicKey; totalXAmount: BN; totalYAmount: BN; strategy: { minBinId: number; maxBinId: number; strategyType: StrategyType; }; }): Promise<Transaction | Transaction[]>
```

## Описание

Метод `addLiquidityByStrategy` позволяет **добавить ликвидность в существующую позицию** в DLMM пуле, используя одну из встроенных стратегий. Очень удобен для наращивания позиции, усреднения, DCA и повторного входа в рынок.

 Можно использовать стратегию `Spot`, `Curve`, или `BidAsk` — точно так же, как при создании новой позиции.

## Параметры

Параметр	Тип	Описание
<code>positionPubKey</code>	<code>PublicKey</code>	Ключ уже существующей позиции
<code>user</code>	<code>PublicKey</code>	Ключ владельца позиции
<code>totalXAmount</code>	<code>BN</code>	Кол-во токена X
<code>totalYAmount</code>	<code>BN</code>	Кол-во токена Y
<code>strategy.minBinId</code>	<code>number</code>	Нижняя граница бинов
<code>strategy.maxBinId</code>	<code>number</code>	Верхняя граница бинов
<code>strategy.strategyType</code>	<code>StrategyType</code>	Тип стратегии (например, <code>Spot</code> )

## Возвращаемое значение

Тип: `Promise<Transaction | Transaction[]>`

Одна или несколько транзакций, которые нужно подписать и отправить.

## 💡 Особенности и замечания

- Используется **только с уже инициализированной позицией**.
  - Подходит для стратегий DCA (добавления в однонаправленную позицию).
  - Стратегия и диапазон должны быть совместимы с уже существующей логикой позиции (иначе может быть ошибка или неэффективное распределение).
  - Убедись, что пользователь имеет достаточный баланс токенов.
- 

## 🔧 Пример использования

ts

КопироватьРедактировать

```
const TOTAL_RANGE_INTERVAL = 10; const minBinId = activeBin.binId -
TOTAL_RANGE_INTERVAL; const maxBinId = activeBin.binId + TOTAL_RANGE_INTERVAL; const
totalXAmount = new BN(100 * 10 ** baseMint.decimals); const totalYAmount =
autoFillYByStrategy( activeBin.binId, dlmmPool.lbPair.binStep, totalXAmount,
activeBin.xAmount, activeBin.yAmount, minBinId, maxBinId, StrategyType.Spot ); const tx
= await dlmmPool.addLiquidityByStrategy({ positionPubKey: existingPosition.publicKey,
user: user.publicKey, totalXAmount, totalYAmount, strategy: { minBinId, maxBinId,
strategyType: StrategyType.Spot, }, }); await sendAndConfirmTransaction(connection, tx,
[user]); console.log('Ликвидность добавлена.');
```

---

Дальше — метод `removeLiquidity`, с помощью которого можно **частично или полностью извлечь ликвидность** из позиции. Готов продолжать?

40

ChatGPT сказал:

Супер! Сейчас разберём метод `removeLiquidity` — один из важнейших для управления позицией.

---

💠 `dlmmPool.removeLiquidity`

📌 Сигнатура


ts

КопироватьРедактировать

```
async removeLiquidity(params: { position: PublicKey; user: PublicKey; fromBinId: number; toBinId: number; liquiditiesBpsToRemove: BN[]; // в базисных пунктах: 10000 = 100% shouldClaimAndClose?: boolean; }): Promise<Transaction | Transaction[]>
```

## Описание

Метод `removeLiquidity` используется для **частичного или полного изъятия ликвидности** из существующей позиции. Можно указать диапазон бинов, процент ликвидности для удаления и опционально запросить **автоклейм** и **закрытие** позиции в одной транзакции.

 Позволяет гибко управлять позициями — делать rebalancing, вывод, вывод с клеймом или полный выход с закрытием.

## Параметры

Параметр	Тип	Описание
<code>position</code>	<code>PublicKey</code>	Ключ позиции, из которой выводим ликвидность
<code>user</code>	<code>PublicKey</code>	Владелец позиции
<code>fromBinId</code>	<code>number</code>	ID первого бина, из которого нужно удалить ликвидность
<code>toBinId</code>	<code>number</code>	ID последнего бина
<code>liquiditiesBpsToRemove</code>	<code>BN[]</code>	Массив процентов ликвидности для удаления в каждом бине (в BPS: 10000 = 100% )
<code>shouldClaimAndClose</code>	<code>boolean</code> (необязательно)	Если <code>true</code> , автоматически клеймит комиссии и закрывает позицию после удаления

## Возвращаемое значение

Тип: `Promise<Transaction | Transaction[]>`

Одна или несколько транзакций, готовых к подписи.

---

## 💡 Особенности и замечания

- `liquiditiesBpsToRemove.length` должен быть равен количеству бинов в диапазоне.
  - Если хочешь удалить всё — используй `new BN(100 * 100)` для каждого бина (10000 = 100%).
  - Если `shouldClaimAndClose = true`, позиция будет закрыта сразу после вывода.
  - Лучше предварительно получить `binIds` из `positionBinData`.
- 

## 🔧 Пример использования

ts

[Копировать](#)[Редактировать](#)

```
const userPosition = userPositions.find(p =>
p.publicKey.equals(existingPosition.publicKey) ); const binIds =
userPosition.positionData.positionBinData.map(bin => bin.binId); const
removeLiquidityTx = await dlmmPool.removeLiquidity({ position: userPosition.publicKey,
user: user.publicKey, fromBinId: binIds[0], toBinId: binIds[binIds.length - 1],
liquiditiesBpsToRemove: new Array(binIds.length).fill(new BN(10000)), // 100%
shouldClaimAndClose: false, // или true }); for (const tx of
Array.isArray(removeLiquidityTx) ? removeLiquidityTx : [removeLiquidityTx]) { await
sendAndConfirmTransaction(connection, tx, [user]); } console.log('Ликвидность успешно
удалена.');
```

---

Следующий метод — `closePosition`, который закрывает позицию (после удаления ликвидности). Поехали?

4o

**ChatGPT сказал:**

Отлично! Сейчас разберём финальный метод для управления позициями — `closePosition`.

---

## `dLmmPool.closePosition`

### Сигнатура


ts

КопироватьРедактировать

```
async closePosition(params: { owner: PublicKey; position: PublicKey; }):  
Promise<Transaction | Transaction[]>
```

### Описание

Метод `closePosition` используется для **закрытия позиции ликвидности** после того, как из неё полностью удалена ликвидность. Закрытие освобождает занятые аккаунты, возвращает аренду и "удаляет" позицию с цепочки в логическом смысле.

 Обязательный финальный шаг после `removeLiquidity`, если `shouldClaimAndClose` не был использован.

---

### Параметры

Параметр	Тип	Описание
<code>owner</code>	<code>PublicKey</code>	Владелец позиции
<code>position</code>	<code>PublicKey</code>	Публичный ключ позиции, которую нужно закрыть

### Возвращаемое значение

Тип: `Promise<Transaction | Transaction[]>`

Транзакция для закрытия позиции.

---

### Особенности и замечания

- Перед закрытием **вся ликвидность должна быть удалена**.

- Если вызвать метод слишком рано (до удаления ликвидности), будет ошибка.
  - Можно использовать после `removeLiquidity(...)` с `shouldClaimAndClose: false`.
  - Рекомендуется вызывать `getPositionsByUserAndLbPair` повторно после закрытия, чтобы обновить состояние.
- 

## Пример использования

ts

КопироватьРедактировать

```
const tx = await dlmmPool.closePosition({ owner: user.publicKey, position:
positionPublicKey, }); await sendAndConfirmTransaction(connection, tx, [user]);
console.log('Позиция успешно закрыта.');
```

---

Теперь мы готовы перейти к **операциям свопа**:

- `swapQuote`
- `swap`

Начинаем с `swapQuote` ?

**ChatGPT** сказал:

Отлично! Переходим к **операциям свопа**, начиная с подготовительного метода — `swapQuote`.

---

## `dlmmPool.swapQuote`

## Сигнатура


ts

КопироватьРедактировать

```
async swapQuote( inAmount: BN, isYtoX: boolean, slippageBps: BN, binArrays:
BinArrayAccount[] ): Promise<SwapQuote>
```

## Описание

Метод `swapQuote` рассчитывает **приблизительный результат свопа**: сколько пользователь получит (или потеряет), сколько будет минимальный вывод с учётом `slippage`, какие бины будут задействованы, и какую комиссию он заплатит.

 Это симуляция до фактического обмена. Используется для UI, проверки приемлемости цены, расчёта `minOutAmount`.

## Параметры

Параметр	Тип	Описание
<code>inAmount</code>	BN	Сумма входного токена
<code>isYtoX</code>	boolean	Направление свопа: <code>true</code> = $Y \rightarrow X$ , <code>false</code> = $X \rightarrow Y$
<code>slippageBps</code>	BN	Допустимое проскальзывание (в BPS, например, <code>100</code> = 1%)
<code>binArrays</code>	<code>BinArrayAccount[]</code>	Массив бинов, полученных из <code>getBinArrayForSwap()</code>

## Возвращаемое значение

Тип: `Promise<SwapQuote>`

Объект `SwapQuote` с примерной структурой:

ts

[Копировать](#)[Редактировать](#)

```
type SwapQuote = { minOutAmount: BN; // с учётом slippage expectedOutAmount: BN; // фактическое, без проскальзывания totalFee: BN; // комиссия binArraysPubkey: PublicKey; }
```

## Особенности и замечания

- Обязательно вызывать `getBinArrayForSwap()` перед этим методом.



- Slippage указывается в **BPS** (1% = 100).
- Используется перед `swap()` для передачи `minOutAmount` и проверки выгодности курса.
- Значения в `BN` — важно правильно форматировать на фронте (например, делить на `10 ** decimals`).

---

## Пример использования

ts

КопироватьРедактировать

```
const swapAmount = new BN(0.1 * 10 ** 9); // 0.1 токена (например, SOL) const isYtoX = true; const binArrays = await dlmmPool.getBinArrayForSwap(isYtoX); const slippageBps = new BN(100); // 1% const quote = await dlmmPool.swapQuote(swapAmount, isYtoX, slippageBps, binArrays); console.log(`Ожидаемый выход: ${quote.expectedOutAmount.toString()}`); console.log(`Минимум с учётом slippage: ${quote.minOutAmount.toString()}`); console.log(`Комиссия: ${quote.totalFee.toString()}`);
```

---

Далее — боевой метод `swap`, который создаёт транзакцию для фактического обмена. Готов?

4o

**ChatGPT сказал:**

Отлично! Переходим к основному методу для выполнения обмена — `swap`.

---

## `dlmmPool.swap`

### Сигнатура

ts

КопироватьРедактировать

```
async swap(params: { inToken: PublicKey; binArraysPubkey: PublicKey; inAmount: BN; lbPair: PublicKey; user: PublicKey; minOutAmount: BN; outToken: PublicKey; }):
```

Promise<Transaction>

## Описание

Метод `swap` создаёт транзакцию для выполнения **реального свопа** между двумя токенами в рамках текущего DLMM пула. Он использует рассчитанный ранее `swapQuote`, чтобы защититься от проскальзывания, и позволяет пользователю совершить обмен прямо через Solana.

 Это финальный шаг в цепочке: `getBinArrayForSwap` → `swapQuote` → `swap`.

## Параметры

Параметр	Тип	Описание
<code>inToken</code>	<code>PublicKey</code>	Токен, который отдаёт пользователь
<code>binArraysPubkey</code>	<code>PublicKey</code>	Адрес массива бинов из <code>swapQuote.binArraysPubkey</code>
<code>inAmount</code>	<code>BN</code>	Количество входного токена
<code>lbPair</code>	<code>PublicKey</code>	Адрес пула DLMM
<code>user</code>	<code>PublicKey</code>	Пользователь, совершающий обмен
<code>minOutAmount</code>	<code>BN</code>	Минимальный допустимый результат после проскальзывания
<code>outToken</code>	<code>PublicKey</code>	Токен, который пользователь хочет получить

## Возвращаемое значение

Тип: `Promise<Transaction>`

Транзакция, готовая к подписи и отправке.

## Особенности и замечания

- Перед вызовом **обязательно** рассчитать `swapQuote`, чтобы получить `binArraysPubkey` и `minOutAmount`.
- `minOutAmount` защищает от плохих условий (slippage protection).

- Токены должны быть предварительно одобрены (владелец должен иметь АТА аккаунты).
  - `swap()` не отправляет транзакцию — тебе нужно отдельно вызвать `sendAndConfirmTransaction`.
- 

## Пример использования

ts

КопироватьРедактировать

```
const swapAmount = new BN(0.1 * 10 ** 9); // 0.1 SOL const isYtoX = true; const
binArrays = await dlmmPool.getBinArrayForSwap(isYtoX); const quote = await
dlmmPool.swapQuote(swapAmount, isYtoX, new BN(100), binArrays); // 1% slippage const
swapTx = await dlmmPool.swap({ inToken: dlmmPool.tokenY.publicKey, outToken:
dlmmPool.tokenX.publicKey, inAmount: swapAmount, minOutAmount: quote.minOutAmount,
lbPair: dlmmPool.pubkey, user: user.publicKey, binArraysPubkey: quote.binArraysPubkey,
}); await sendAndConfirmTransaction(connection, swapTx, [user]); console.log('Своп
успешно выполнен!');
```

---

Готов перейти к методам **claim** (вознаграждения и комиссии)? Начнём с `claimLMReward` ?

4o

**ChatGPT** сказал:

Отлично! Переходим к блоку **claim-методов**, начиная с `claimLMReward`.

---

## `dlmmPool.claimLMReward`

## Сигнатура

ts

КопироватьРедактировать

```
async claimLMReward(params: { owner: PublicKey; position: PublicKey; }):
Promise<Transaction>
```

## Описание

Метод `claimLMReward` позволяет пользователю **получить вознаграждение за Liquidity Mining (LM)** для конкретной позиции. Это награды, начисленные за предоставление ликвидности в пул, и их можно забирать отдельно от своп-комиссий.

💎 Если ты добавляешь ликвидность и пул участвует в программе стимулирования — этот метод даст тебе LM-токены.

## Параметры

Параметр	Тип	Описание
<code>owner</code>	<code>PublicKey</code>	Ключ пользователя, который клеймит награду
<code>position</code>	<code>PublicKey</code>	Ключ позиции, по которой начислена награда

## Возвращаемое значение

Тип: `Promise<Transaction>`

Транзакция для получения LM-наград. Её нужно подписать и отправить через `sendAndConfirmTransaction`.

## Особенности и замечания

- Используй `getClaimableLMReward(position)` перед вызовом, чтобы узнать, есть ли что клеймить.
- Награда может быть в другом токене (например, `MTR`), не в `X` или `Y`.
- Метод **не закрывает позицию** — просто выдаёт токены.
- Может вызываться периодически для разных позиций.

## Пример использования

ts

КопироватьРедактировать

```
const tx = await dlmmPool.claimLMReward({ owner: user.publicKey, position: positionPubkey, }); await sendAndConfirmTransaction(connection, tx, [user]); console.log('LM-награда успешно получена!');
```

---

Следующий метод — `claimAllLMRewards`, который делает то же самое, но **сразу по всем позициям пользователя в этом пуле**. Продолжим?

4o

**ChatGPT сказал:**

Отлично, продолжаем — сейчас метод `claimAllLMRewards`.

---

◆ `dlmmPool.claimAllLMRewards`

📌 **Сигнатура**

ts

КопироватьРедактировать

```
async claimAllLMRewards(params: { owner: PublicKey; positions: Position[]; }): Promise<Transaction[]>
```

📄 **Описание**

Метод `claimAllLMRewards` позволяет **забрать LM-награды сразу по нескольким позициям** пользователя в текущем пуле DLMM. Это удобный способ сэкономить на транзакциях и ускорить получение всех стимулирующих вознаграждений.

🔄 Отлично подходит для регулярного массового сбора наград, особенно если пользователь работает с несколькими диапазонами.

---

🔧 **Параметры**

Параметр	Тип	Описание
owner	PublicKey	Ключ пользователя
positions	Position[]	Массив позиций, по которым нужно клеймить LM-награды (из <code>getPositionsByUserAndLbPair</code> )

---

## Возвращаемое значение

Тип: `Promise<Transaction[]>`

Массив транзакций, каждая из которых предназначена для клейма LM-награды по одной или нескольким позициям. Все транзакции нужно подписать и отправить.

---

## Особенности и замечания

- Получи список позиций с помощью `getPositionsByUserAndLbPair(...)` .
  - Перед вызовом желательно проверить наличие наград через `getClaimableLMReward(...)` для каждой позиции.
  - Этот метод **не объединяет** клеймы в одну транзакцию, но возвращает их все разом — ты сам решаешь, какие отправить.
  - Может возвращать пустой массив, если нечего клеймить.
- 

## Пример использования

ts

[Копировать](#)[Редактировать](#)

```
const { userPositions } = await dlmmPool.getPositionsByUserAndLbPair(user.publicKey);
const txs = await dlmmPool.claimAllLMRewards({ owner: user.publicKey, positions:
userPositions, }); for (const tx of txs) { await sendAndConfirmTransaction(connection,
tx, [user]); } console.log('Все LM-награды получены!');
```

---

Следующий по логике — `claimSwapFee` , который позволяет получить **заработанные комиссии** за свопы по одной позиции. Поехали?

4o

ChatGPT сказал:

Отлично, переходим к следующему claim-методу — `claimSwapFee` .

 `d1mmPool.claimSwapFee`

 **Сигнатура**


ts

КопироватьРедактировать

```
async claimSwapFee(params: { owner: PublicKey; position: PublicKey; }):  
Promise<Transaction>
```

 **Описание**

Метод `claimSwapFee` позволяет **получить комиссии**, накопленные в результате свопов через диапазон ликвидности текущей позиции. Это доход, который зарабатывает поставщик ликвидности за активность в пуле.

 Комиссии начисляются в токенах X и Y в зависимости от торгов в соответствующих бинах.

 **Параметры**

Параметр	Тип	Описание
owner	PublicKey	Адрес пользователя
position	PublicKey	Адрес позиции, по которой нужно получить комиссии

 **Возвращаемое значение**

Тип: `Promise<Transaction>`

Транзакция, которую нужно подписать и отправить для получения комиссии.

---

## 💡 Особенности и замечания

- Работает **только для одной позиции**.
  - Перед вызовом желательно проверить наличие комиссий через `getClaimableSwapFee(...)`.
  - Комиссии могут быть получены даже без LM-программы.
  - Токены отправляются напрямую на `associated token accounts` пользователя.
- 

## 💡 Пример использования

ts

КопироватьРедактировать

```
const tx = await dlmmPool.claimSwapFee({ owner: user.publicKey, position: positionPubkey, }); await sendAndConfirmTransaction(connection, tx, [user]); console.log('Комиссии за свопы успешно получены!');
```

---

Следующим идёт `claimAllSwapFee` — аналогичный, но сразу по всем позициям в пуле. Готов?

4o

**ChatGPT сказал:**

Отлично! Сейчас разберём `claimAllSwapFee` — чтобы забирать комиссии **по всем позициям** в пуле.

---

◆ `dlmmPool.claimAllSwapFee`

📌 **Сигнатура**




ts

КопироватьРедактировать

```
async claimAllSwapFee(params: { owner: PublicKey; positions: Position[]; }):  
Promise<Transaction[]>
```

## Описание

Метод `claimAllSwapFee` позволяет пользователю **получить комиссии за свопы по всем своим позициям** в текущем пуле DLMM. Это упрощённый способ массового получения всех доступных комиссионных доходов.

 Идеально подходит для пользователей с несколькими позициями в одном пуле — сокращает количество ручных операций.

## Параметры

Параметр	Тип	Описание
<code>owner</code>	<code>PublicKey</code>	Ключ пользователя
<code>positions</code>	<code>Position[]</code>	Массив позиций (например, из <code>getPositionsByUserAndLbPair</code> )

## Возвращаемое значение

Тип: `Promise<Transaction[]>`

Массив транзакций для получения комиссий — по одной или нескольким позициям. Транзакции необходимо подписать и отправить вручную.

## Особенности и замечания

- Используется совместно с `getPositionsByUserAndLbPair` .
- Перед вызовом можно дополнительно фильтровать позиции, у которых есть комиссии через `getClaimableSwapFee` .
- Не объединяет клеймы в одну транзакцию, но позволяет отправить все транзакции последовательно.

- Может вернуть пустой массив, если нет комиссий к получению.

---

## Пример использования

ts

КопироватьРедактировать

```
const { userPositions } = await dlmmPool.getPositionsByUserAndLbPair(user.publicKey);
const txs = await dlmmPool.claimAllSwapFee({ owner: user.publicKey, positions:
userPositions, }); for (const tx of txs) { await sendAndConfirmTransaction(connection,
tx, [user]); } console.log('Все комиссии за свопы получены!');
```

---

Следующий метод — `claimAllRewards`, который сочетает оба типа наград: **LM + swap fee**.  
Готов?

40

**ChatGPT сказал:**

Супер! Сейчас разберём универсальный метод `claimAllRewards` — он позволяет получить **все виды наград** сразу.

---

## `dlmmPool.claimAllRewards`

### Сигнатура

ts

КопироватьРедактировать

```
async claimAllRewards(params: { owner: PublicKey; positions: Position[]; }):
Promise<Transaction[]>
```

## Описание

Метод `claimAllRewards` объединяет в себе оба типа наград:

-  **Liquidity Mining (LM)** — за участие в программе стейкинга ликвидности,

-  **Swap Fees** — комиссии от трейдеров, прошедших через диапазон позиции.

Это самый удобный способ **одним вызовом получить все доступные награды** в текущем пуле DLMM.

---

## Параметры

Параметр	Тип	Описание
<code>owner</code>	<code>PublicKey</code>	Публичный ключ пользователя
<code>positions</code>	<code>Position[]</code>	Массив позиций в текущем пуле (например, из <code>getPositionsByUserAndLbPair</code> )

---

## Возвращаемое значение

Тип: `Promise<Transaction[]>`

Массив транзакций, каждая из которых включает `claimLMReward` и/или `claimSwapFee` по определённой позиции.

---

## Особенности и замечания

- Проверяет каждую позицию: если в ней есть что-то клеймить — создаёт транзакцию.
- Эквивалентно вызову `claimAllLMRewards` + `claimAllSwapFee` , но объединено в одну логику.
- Отлично подходит для UI-кнопки "Claim All Rewards".
- Может вернуть пустой массив, если нет доступных наград.

---

## Пример использования

ts

КопироватьРедактировать

```
const { userPositions } = await dlmmPool.getPositionsByUserAndLbPair(user.publicKey);
const txs = await dlmmPool.claimAllRewards({ owner: user.publicKey, positions:
userPositions, }); for (const tx of txs) { await sendAndConfirmTransaction(connection,
tx, [user]); } console.log('Все награды успешно получены!');
```

---

📌 Остался последний метод в списке — `syncWithMarketPrice`, который позволяет **синхронизировать активный бин с рыночной ценой**. Перейдём к нему?

4o

**ChatGPT сказал:**

Отлично, завершаем обзор SDK последним методом — `syncWithMarketPrice`.

---

## ◆ `dlmmPool.syncWithMarketPrice`

### 📌 Сигнатура

ts

КопироватьРедактировать

```
async syncWithMarketPrice(): Promise<Transaction>
```

### 📄 Описание

Метод `syncWithMarketPrice` используется для **принудительного обновления активного бина** пула DLMM, чтобы он соответствовал **актуальной рыночной цене**. Это важно, если произошёл внешний ценовой сдвиг, и активный бин устарел.

📈 Обычно это делает оракул или keeper, но ты можешь синхронизировать вручную, если необходимо.

---

### 🔧 Параметры

Нет параметров.

---

## Возвращаемое значение

Тип: `Promise<Transaction>`

Транзакция, которую нужно подписать и отправить, чтобы обновить активный бин пула.

---

## Особенности и замечания

- Требуется достаточных прав — транзакция должна быть подписана авторизованным аккаунтом (например, от `keeper`'а).
  - Используется только в пулах с динамической логикой, где бин не обновляется автоматически.
  - Может быть полезен после длительной неактивности или внешнего ценового скачка.
  - Безопасен — не изменяет ликвидность или награды, только перемещает "ценовой указатель".
- 


## Пример использования







ts

[Копировать](#)[Редактировать](#)

```
const tx = await dlmmPool.syncWithMarketPrice(); await  
sendAndConfirmTransaction(connection, tx, [keeper]); console.log('Пул синхронизирован с  
рыночной ценой.');
```

---

 **Поздравляю!** Ты только что полностью разобрал весь DLMM TypeScript SDK от Meteora:

-  Статические методы
-  Методы пула
-  Работа с позициями
-  Свопы
-  Клеймы и награды
-  Административные функции