four-flap-meme-sdk English Docs

A two-in-one SDK for **four.meme** (TokenManager V1/V2/Helper3) and **Flap Protocol** (Portal/CDPV2). This comprehensive guide is designed for **complete beginners** – no prior experience with these platforms required. We'll explain every concept, method, and parameter in detail.

Why This SDK?

This SDK helps you:

- **Launch tokens** on four.meme or Flap Protocol with just a few lines of code
- **Trade tokens** using bonding curves (buy/sell before DEX listing)
- **Query token state** and estimate prices
- **Understand two DeFi platforms** that revolutionize token launches

Runtime Requirements

- **Node.js**: Version 18 or higher (uses native fetch API)
- **Module Type**: ESM (NodeNext)
- **Dependencies**:
  - `ethers@^6` - For Ethereum interactions

Installation

```
npm i four-flap-meme-sdk ethers
```

Optional (use Pinata for IPFS uploads)

```
npm i pinata
```

Table of Contents

Platform Overview

What is four.meme?

four.meme is a token launchpad that uses a **bonding curve** mechanism for price discovery before listing on decentralized exchanges (DEX).

Key Concepts:

1. **Bonding Curve Phase**:
    - When a token is first created, it's NOT on a DEX
    - Instead, prices are determined by a mathematical formula (bonding curve)
    - As more people buy, the price automatically increases
    - As people sell, the price automatically decreases
2. **Migration to DEX**:
    - Once enough funds are raised (reaches threshold), the token "graduates"
    - It gets listed on PancakeSwap (for BSC) with a liquidity pool
    - After this, it trades like a normal DEX token
3. **Versions**:
    - **V1 (TokenManager)**: Original version, still in use
    - **V2 (TokenManager2)**: Improved version with more features
    - **Helper3**: A utility contract that helps estimate prices and route trades

Why Use four.meme?

- **Fair Launch**: Everyone buys at the same bonding curve price
- **No Rug Pulls**: Liquidity is automatically added to DEX
- **Early Access**: Buy before DEX listing at potentially lower prices

---

What is Flap Protocol?

Flap Protocol is another token launch platform with advanced bonding curve features (CDPV2 curve).

Key Features:

1. **Advanced Bonding Curve (CDPV2)**:
    - More flexible curve parameters: `r`, `h`, `k`
    - Better price discovery
    - Customizable migration thresholds
2. **Multi-Chain Support**:
    - BSC (BNB Chain)
    - Base
    - X Layer
    - Morph
3. **Quote Token Options**:
    - Can launch with native tokens (BNB, ETH)
    - Can launch with quote tokens (USDT, etc.)

4. **Tax & Migrator Options**:
    - Set custom tax rates
    - Choose V2 or V3 migrators for DEX

Understanding Bonding Curves

What is a Bonding Curve?

A **bonding curve** is a mathematical formula that determines token price based on supply.

Simple Example:

Imagine a token with this rule:

- First 1000 tokens: $0.01 each
- Next 1000 tokens: $0.02 each
- Next 1000 tokens: $0.03 each
- And so on...

Bonding curves do this smoothly with a formula instead of steps.

The Formula (CDPV2):

```
price = k / (1,000,000,000 + h − supply)²
```

Where:

- `k`: A constant that affects overall price level
- `h`: Height adjustment (shifts the curve)
- `supply`: Current circulating supply

Why It Matters:

- **Early buyers**: Get lower prices
- **Price increases**: Automatically as supply grows
- **Predictable**: Anyone can calculate the price
- **No manipulation**: Can't be front-run like DEX orders

Internal Market vs External Market

In token launch platforms, there are two important trading phase concepts:

🔵 Internal Market (Bonding Curve Phase)

Definition: The phase where tokens trade on the bonding curve before migrating to a DEX.

Characteristics:

- ✅ **Formula-based pricing**: Uses mathematical curves (like CDPV2) to calculate prices
- ✅ **Locked liquidity**: All funds locked in the platform contract
- ✅ **Rug-pull prevention**: Developers cannot withdraw liquidity
- ✅ **Fair price discovery**: Everyone buys/sells at the same curve price
- ✅ **Instant settlement**: No order book or counterparty needed
- ✅ **Predictability**: Can precisely calculate buy/sell prices

On four.meme and Flap Protocol:

- Tokens start in the internal market phase when created
- Trade using `FourClient` or `FlapPortalWriter`
- Prices adjust automatically with buys/sells
- Example: `status = 1` (tradable state)

Internal Market Trading Example:

```
// Buy tokens on the internal market (bonding curve)
const txHash = await writer.swapExactInput({
  inputToken: ZERO_ADDRESS,
  outputToken: tokenAddress,
  inputAmount: parseEther('1.0'),
  minOutputAmount: minAmount,
  to: yourAddress
});
```

🟢 External Market (DEX Phase)

Definition: The phase where tokens have "graduated" and migrated to a decentralized exchange (DEX).

Characteristics:

- ✅ **Market pricing**: Price determined by supply/demand, no longer using curves
- ✅ **DEX liquidity pools**: Trade on PancakeSwap, Uniswap, etc.
- ✅ **Free trading**: Can trade on any supported DEX and aggregator
- ✅ **Deeper liquidity**: Usually deeper liquidity with lower slippage
- ✅ **Standard ERC20**: Trades like any normal token

Migration Conditions:

- four.meme: When raised funds reach a specific threshold (e.g., 24 BNB)
- Flap Protocol: When reserve reaches `dexThresh` (configurable)

After Migration:

- Platform contract automatically adds liquidity to DEX
- Bonding curve closes, can no longer trade through platform
- Token status becomes `status = 4` (migrated to DEX)

External Market Trading Example:

```
// Token has migrated to DEX, use DEX router for trading
// For example, using Uniswap SDK or PancakeSwap SDK
import { SwapRouter } from '@uniswap/v3-sdk';
// ... use standard DEX trading methods
```

📊 Internal Market vs External Market Comparison

Feature	Internal Market (Bonding Curve)	External Market (DEX)
Pricing Mechanism	Mathematical formula (bonding curve)	Market supply/demand
Liquidity Source	Platform contract	DEX liquidity pool
Trading Method	Through platform contract	Through DEX (PancakeSwap/Uniswap)
Price Predictability	High (can calculate precisely)	Low (market volatility)
Slippage	Depends on curve parameters	Depends on liquidity depth
Phase	Early stage	Mature stage
SDK Usage	`FourClient` / `FlapPortalWriter`	DEX SDK (Uniswap/PancakeSwap)
Token Status	`status = 1`	`status = 4`

🔄 Migration Process from Internal to External Market

1. **Internal Market Phase**: Token trades on bonding curve
2. **Threshold Reached**: Reserve reaches migration threshold
3. **Automatic Migration**: Platform contract automatically triggers migration
4. **Add Liquidity**: Adds reserve and tokens to DEX
5. **External Market Phase**: Token freely trades on DEX

How to Check Which Phase a Token is In?

```
import { FlapPortal } from 'four-flap-meme-sdk';

const portal = new FlapPortal({ chain: 'BSC', rpcUrl });
const state = await portal.getTokenV5(tokenAddress);

if (state.status === 1) {
  console.log('✅ Internal Market: Token trading on bonding curve');
  console.log('Reserve:', state.reserve);
  console.log('Migration threshold:', state.dexSupplyThresh);
} else if (state.status === 4) {
  console.log('✅ External Market: Token has migrated to DEX');
  console.log('Please trade on PancakeSwap/Uniswap');
}
```

Important Notes:

- ⚠ After token migrates to DEX, cannot trade through platform contract anymore
- ⚠ Migration is one-way and irreversible
- ⚠ After migration, must use DEX trading tools (like PancakeSwap, Uniswap)

🔎 Inspect token LP (auto-detect in-house vs DEX)

Use `inspectTokenLP(token, opts)` to detect whether a token is traded on four (bonding curve), Flap (bonding curve), or Pancake V2/V3 (DEX). For DEX, it returns WBNB/USDT pairs and reserves; for in-house platforms, it returns curve reserves/supply so the UI can display pool-like info.

Minimal usage (BSC)

```
import { inspectTokenLP } from 'four-flap-meme-sdk';

const info = await inspectTokenLP('0xToken', {
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org',
  flapChain: 'BSC'
});
console.log(info);
```

Optionally resolve factories via Routers

```
const info = await inspectTokenLP('0xToken', {
  chain: 'BSC',
  rpcUrl,
  routerV2: '0x10ED43C718714eb63d5aA57B78B54704E256024E', // Pancake V2
Router
  routerV3: '0x13f4EA83D0bd40E75C8222255bc855a974568Dd4', // Pancake V3
Router
  // or pass factoryV2/factoryV3 directly
});
```

Return examples

```
// four (in-house)
{ platform: 'FOUR', four: { helper: '0x...', reserveNative: '123456...',
offerTokens: '7890...', lastPrice: '1234...' } }

// flap (in-house)
{ platform: 'FLAP', flap: { quoteToken: '0x...', reserveNative: '...',
circulatingSupply: '...', price: '...' } }

// Pancake V2
{ platform: 'PANCAKE_V2', v2: {
```

```
  wbnbPair: { address: '0x...', reserveToken: '...', reserveWBNB: '...' },
  usdtPair: { address: '0x...', reserveToken: '...', reserveUSDT: '...' }
}}

// Pancake V3 (multiple fee tiers)
{ platform: 'PANCAKE_V3', v3: [
  { base: 'WBNB', fee: 500, pool: '0x...', tokenBalance: '...',
baseBalance: '...' },
  { base: 'USDT', fee: 2500, pool: '0x...', tokenBalance: '...',
baseBalance: '...' }
]}
```

four.meme Platform Guide

four.meme Architecture

four.meme has **three main components**:

1. **REST API** (Backend)

- User authentication (nonce + signature)
- Image upload for token icons
- Token metadata management
- Create parameter generation

2. **Smart Contracts** (On-Chain)

- **TokenManagerV1**: Original token manager
- **TokenManagerV2**: Improved token manager with more features
- **TokenManagerHelper3**: Helper contract for price estimation and routing

3. **This SDK** (Your Interface)

- Simplifies all REST API calls
- Wraps smart contract interactions
- Provides unified interfaces for V1/V2

Token Launch Process

Launching a token on four.meme involves **5 steps**:

```
1. Generate Nonce     → Get a random string from server
2. Sign Message       → Sign with your wallet to prove ownership
3. Login              → Receive access token
4. Upload Image       → Upload token logo/icon
5. Create On-Chain    → Deploy the token contract
```

Why This Process?

- **Authentication**: Proves you own the wallet
- **Security**: Access token prevents unauthorized actions
- **Metadata**: Server stores token info off-chain
- **Decentralization**: Actual token is on-chain, not controlled by server

---

One-Click Token Launch

The Easiest Way

Use `createTokenFlow()` to handle all 5 steps automatically:

```
import { createTokenFlow } from 'four-flap-meme-sdk';

const result = await createTokenFlow({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...your-private-key',
  networkCode: 'BSC',

  // Optional: provide pre-uploaded image URL
  imgUrl: 'https://...',

  // OR: upload new image
  image: imageBlob,

  payload: {
    name: 'My Awesome Token',        // Full name
    shortName: 'MAT',                // Symbol (ticker)
    desc: 'A revolutionary token',   // Description
    label: 'Meme',                   // Category
    preSale: '0',                    // Pre-purchase amount: '0' = no buy,
'0.5' = buy 0.5 BNB
    onlyMPC: false,                  // MPC-exclusive mode (explained
later)

    // Optional social links
    webUrl: 'https://example.com',
    twitterUrl: 'https://twitter.com/...',
    telegramUrl: 'https://t.me/...',
  }
});

console.log('Token created!');
console.log('Transaction:', result.txReceipt.transactionHash);
console.log('Your address:', result.accountAddress);
```

Parameter Explanations:

Parameter	Type	Required	Description
`rpcUrl`	string	✅	RPC endpoint for the blockchain (BSC, Arbitrum, etc.)
`privateKey`	string	✅	Your wallet private key (KEEP SECRET!)
`networkCode`	'BSC'	✅	Currently only BSC is supported
`baseUrl`	string	❌	four.meme API URL (default: https://four.meme/meme-api)
`image`	Blob	❌ *	Image file to upload (*required if no imgUrl)
`imgUrl`	string	❌ *	Pre-uploaded image URL (*required if no image)

Payload Fields:

Field	Type	Required	Description
`name`	string	✅	Token full name (e.g., "Bitcoin")
`shortName`	string	✅	Token symbol (e.g., "BTC")
`desc`	string	✅	Token description
`label`	string	✅	Category: 'Meme', 'AI', 'Defi', 'Games', 'Infra', 'De-Sci', 'Social', 'Depin', 'Charity', 'Others'
`preSale`	string	✅	BNB amount to buy when creating (pre-purchase). Examples: '0' = no buy, '0.1' = buy 0.1 BNB, '1' = buy 1 BNB
`onlyMPC`	boolean	❌	If true, only MPC wallets can buy (anti-bot)
`launchTime`	number	❌	Launch timestamp in milliseconds (default: now)
`webUrl`	string	❌	Project website
`twitterUrl`	string	❌	Twitter/X profile
`telegramUrl`	string	❌	Telegram group/channel

Return Value:

```
{
  accountAddress: string,        // Your wallet address
  accessToken: string,           // JWT token for future API calls
  imgUrl: string,                // Uploaded image URL
  api: {
    createArg: string,           // Encoded parameters sent to contract
    signature: string,           // Server signature for verification
  },
  txReceipt: any                 // Transaction receipt from blockchain
}
```

Step-by-Step Launch

For Advanced Users

If you need more control, use individual methods:

Step 1: Generate Nonce

```
import { FourClient } from 'four-flap-meme-sdk';

const four = new FourClient();
const nonce = await four.generateNonce({
  accountAddress: '0x...your-address',
  verifyType: 'LOGIN',
  networkCode: 'BSC'
});
```

What it does: Gets a random nonce (number used once) from server to prevent replay attacks.

Step 2: Sign Login Message

```
import { buildLoginMessage } from 'four-flap-meme-sdk';
import { Wallet } from 'ethers';

const wallet = new Wallet(privateKey);
const message = buildLoginMessage(nonce); // "You are sign in Meme
{nonce}"
const signature = await wallet.signMessage(message);
```

What it does: Creates a signed message proving you own the wallet.

Step 3: Login to Get Access Token

```
const accessToken = await four.loginDex({
  region: 'WEB',
  langType: 'EN',
  walletName: 'MetaMask',
  verifyInfo: {
    address: '0x...your-address',
    networkCode: 'BSC',
    signature: signature,
    verifyType: 'LOGIN'
  }
});
```

What it does: Exchanges your signature for an access token (like a session cookie).

Step 4: Upload Image

```
const imgUrl = await four.uploadImage(accessToken, imageBlob);
```

What it does: Uploads token logo to four.meme's CDN, returns URL.

Step 5: Get Create Parameters

```
const { createArg, signature } = await four.createToken(accessToken, {
  name: 'My Token',
  shortName: 'MTK',
  desc: 'Description',
  imgUrl: imgUrl,
  launchTime: Date.now(),
  label: 'Meme',
  lpTradingFee: 0.0025,        // Fixed at 0.25%
  preSale: '0',
  onlyMPC: false,
  webUrl: 'https://...',
  twitterUrl: 'https://...',
  telegramUrl: 'https://...',
});
```

What it does: Server generates and signs the parameters for on-chain token creation.

Step 6: Create Token On-Chain

```
import { createTokenOnChain } from 'four-flap-meme-sdk';

const receipt = await createTokenOnChain({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org',
  signerPrivateKey: privateKey,
  args: createArg,
  signature: signature,
});

console.log('Token created at tx:', receipt.transactionHash);
```

What it does: Sends transaction to TokenManager2 contract, deploying your token.

Trading on four.meme

Once a token is created, anyone can buy and sell it using the bonding curve.

Trading Flow Overview

```
1. Estimate Price (Optional)  → See how much you'll get
2. Check Approval (Sell Only) → Allow TokenManager to spend your tokens
3. Execute Trade              → Buy or sell
```

Price Estimation

Before trading, you should estimate the cost/return:

Estimating Buy Price

```
import { tryBuy } from 'four-flap-meme-sdk';

// Option 1: Buy with specific BNB amount (funds)
const estimate = await tryBuy(
  'BSC',
  rpcUrl,
  tokenAddress,
  0n,                     // amount = 0 means "buy by funds"
  1n * 10n ** 18n         // 1 BNB
);

console.log('You will receive approximately:', estimate.estimatedAmount);
console.log('Cost:', estimate.estimatedCost);
console.log('Fee:', estimate.estimatedFee);

// Option 2: Buy specific token amount
const estimate2 = await tryBuy(
  'BSC',
  rpcUrl,
  tokenAddress,
  10_000n * 10n ** 18n,    // Buy 10,000 tokens
  0n                       // funds = 0 means "buy by amount"
);

console.log('It will cost:', estimate2.estimatedCost, 'wei');
```

Return values explained:

```
{
  tokenManager: string,      // Which TokenManager contract to use (V1 or
V2)
  quote: string,             // Quote token address (usually WBNB)
  estimatedAmount: bigint,   // Tokens you'll receive
  estimatedCost: bigint,     // Total cost in wei (including fees)
  estimatedFee: bigint,      // Trading fee in wei
```

```
  amountMsgValue: bigint,    // BNB to send as msg.value
  amountApproval: bigint,    // Amount to approve (if using quote token)
  amountFunds: bigint        // Actual funds to use in contract call
}
```

Estimating Sell Price

```
import { trySell } from 'four-flap-meme-sdk';

const estimate = await trySell(
  'BSC',
  rpcUrl,
  tokenAddress,
  1_000n * 10n ** 18n      // Sell 1,000 tokens
);

console.log('You will receive:', estimate.funds, 'wei');
console.log('Fee:', estimate.fee, 'wei');
```

Return values:

```
{
  tokenManager: string,      // Which TokenManager to use
  quote: string,             // Quote token address
  funds: bigint,             // BNB you'll receive (after fees)
  fee: bigint                // Trading fee
}
```

Buying Tokens

There are **three ways** to buy tokens:

Method 1: Simple Buy (Recommended)

Use `tradeBuy()` which **auto-detects** V1 or V2:

```
import { tradeBuy } from 'four-flap-meme-sdk';

// Buy with specific BNB amount
await tradeBuy(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  {
```

```
    type: 'funds',           // Buy by specifying funds
    funds: 1n * 10n ** 18n,  // 1 BNB
    minAmount: 0n,           // Minimum tokens to receive (slippage
protection)
    to: yourAddress          // Optional: send tokens to different address
  }
);

// Buy specific token amount
await tradeBuy(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  {
    type: 'amount',             // Buy specific amount
    amount: 10_000n * 10n**18n, // 10,000 tokens
    maxFunds: 5n * 10n ** 18n,  // Maximum BNB to spend (slippage
protection)
    to: yourAddress
  }
);
```

Parameters explained:

- `type: 'funds'`: You specify how much BNB to spend
- `type: 'amount'`: You specify how many tokens to buy
- `minAmount`/`maxFunds`: Slippage protection (prevents bad trades if price changes)
- `to`: Optional recipient address (defaults to your address)
- `origin`: Optional origin code for referral tracking (default: 0n)

Method 2: Direct Buy (Advanced)

If you already know it's V2, use `buyTokenWithFunds()`:

```
import { tryBuy, buyTokenWithFunds } from 'four-flap-meme-sdk';

// First, estimate
const estimate = await tryBuy('BSC', rpcUrl, tokenAddress, 0n, 1n * 10n **
18n);

// Then buy
await buyTokenWithFunds(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  estimate.amountFunds,    // Use estimated funds
  0n,                      // minAmount (slippage)
  yourAddress              // recipient
);
```

Method 3: Using TM1/TM2 Classes (Expert)

For maximum control:

```
import { TM2 } from 'four-flap-meme-sdk';

// Use chain enum (SDK handles contract address automatically)
const tm2 = TM2.connectByChain('BSC', rpcUrl);

// Buy by amount
await tm2.buyToken(
  tokenAddress,
  1_000n * 10n ** 18n,     // Buy 1,000 tokens
  2n * 10n ** 18n          // Max 2 BNB
);

// Buy by funds (AMAP = As Much As Possible)
await tm2.buyTokenAMAP(
  tokenAddress,
  1n * 10n ** 18n,         // Spend 1 BNB
  0n                       // Min 0 tokens (no slippage protection)
);
```

Selling Tokens

Selling requires **token approval** first (letting TokenManager spend your tokens).

Complete Sell Flow

```
import { ensureSellApproval, tradeSell } from 'four-flap-meme-sdk';

const tokenAddress = '0x...';
const amountToSell = 1_000n * 10n ** 18n;  // 1,000 tokens

// Step 1: Approve TokenManager to spend your tokens
await ensureSellApproval(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  yourAddress,
  amountToSell
);

// Step 2: Sell
await tradeSell(
  'BSC',
```

```
  rpcUrl,
  privateKey,
  tokenAddress,
  {
    type: 'direct',
    amount: amountToSell,
    minFunds: 0n              // Minimum BNB to receive
  }
);
```

Why approval?

ERC20 tokens require **two transactions** to sell:

1. **Approve**: Give permission to TokenManager
2. **Sell**: Actually transfer and receive BNB

This is a security feature of ERC20 tokens.

Router Sell (For Third-Party Apps)

If you're building a router/aggregator:

```
await tradeSell(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  {
    type: 'router',
    from: userAddress,          // Sell from user's wallet
    amount: amountToSell,
    minFunds: 0n,
    feeRate: 100n,              // 1% fee (in basis points: 100 = 1%)
    feeRecipient: routerAddress // Where to send the fee
  }
);
```

Auto-Routing

What is auto-routing?

four.meme has two versions (V1 and V2) with **different interfaces**. The SDK provides unified methods that:

1. Detect which version a token uses
2. Call the correct contract
3. Handle parameter differences

Using Auto-Routing

```
import { tradeBuy, tradeSell } from 'four-flap-meme-sdk';

// Works for BOTH V1 and V2 tokens
await tradeBuy('BSC', rpcUrl, privateKey, token, params);
await tradeSell('BSC', rpcUrl, privateKey, token, params);
```

Behind the scenes:

1. Calls `Helper3.getTokenInfo(token)`
2. Checks `version` field (1 or 2)
3. Routes to TM1 or TM2
4. Adapts parameters automatically

Error Handling

Trading on four.meme can fail for various reasons. The SDK provides error code parsing:

```
import { parseFourError } from 'four-flap-meme-sdk';

try {
  await tradeBuy(rpcUrl, privateKey, helper3, token, params);
} catch (error) {
  const { code, message } = parseFourError(error);

  if (code === 'Slippage') {
    console.error('Price moved too much! Try increasing slippage
tolerance');
  } else if (code === 'More BNB') {
    console.error('Not enough BNB sent:', message);
  } else {
    console.error('Error:', code, message);
  }
}
```

Error Codes Reference

Code	Description	Solution
`GW`	Amount not aligned to GWEI	Round to GWEI (divide by 10^9)
`ZA`	Target address is zero address	Provide valid recipient address
`TO`	Cannot send to PancakePair	Don't use LP address as recipient
`Slippage`	Price changed too much	Increase minAmount/maxFunds tolerance
`More BNB`	Insufficient BNB in msg.value	Send more BNB or check estimate
`FR`	Fee rate > 5%	Router fee rate must be ≤ 5%

Code	Description	Solution
`SO`	Order too small	Increase trade amount

MPC-Exclusive Tokens

What is MPC-exclusive?

Some tokens on four.meme are marked as **"MPC-only"**, meaning:

- Only **MPC wallets** (multi-party computation wallets) can buy them
- This is an **anti-bot** feature
- Regular EOA (externally owned account) wallets are blocked

Why MPC-Exclusive?

- **Prevents bots**: Bots usually use simple EOA wallets
- **Fair launch**: Gives human users a better chance
- **Less manipulation**: Harder to snipe or front-run

How to Detect MPC-Exclusive Tokens

On-Chain Detection (Recommended)

```
import { isExclusiveOnChain } from 'four-flap-meme-sdk';

const isMPCOnly = await isExclusiveOnChain({
  chain: 'BSC',              // SDK automatically uses the correct proxy
contract
  tokenAddress: '0x...',
  rpcUrl: rpcUrl
});

if (isMPCOnly) {
  console.log('This token is MPC-exclusive!');
  console.log('You need an MPC wallet to trade it');
}
```

How it works: Checks the token's `template` field. If `template & 0x10000 > 0`, it's MPC-only.

Off-Chain Detection (Using API)

```
import { isExclusiveOffChain, FourClient } from 'four-flap-meme-sdk';

const four = new FourClient();
const isMPCOnly = await isExclusiveOffChain(
  (addr) => four.getTokenByAddress(addr),
```

```
  tokenAddress
);
```

How it works: Checks if `version === 'V8'` in API response.

---

four.meme API Reference

FourClient Class

Constructor

```
const four = new FourClient({
  baseUrl?: string  // Default: 'https://four.meme/meme-api'
});
```

Authentication Methods

generateNonce()

Gets a random nonce for login signature.

```
const nonce = await four.generateNonce({
  accountAddress: string,      // Your wallet address
  verifyType: 'LOGIN',         // Fixed value
  networkCode: 'BSC'           // Fixed value (for now)
});
```

Returns: `Promise<string>` - The nonce string

---

buildLoginMessage()

Builds the message to sign for login.

```
import { buildLoginMessage } from 'four-flap-meme-sdk';

const message = buildLoginMessage(nonce);  // Returns: "You are sign in
Meme {nonce}"
```

---

loginDex()

Exchanges signature for access token.

```
const accessToken = await four.loginDex({
  region: 'WEB',                    // Fixed value
  langType: 'EN' | 'ZH',            // Language preference
  walletName: string,               // e.g., 'MetaMask', 'WalletConnect'
  verifyInfo: {
    address: string,                // Your wallet address
    networkCode: 'BSC',             // Fixed value
    signature: string,              // Signature from signing the nonce
message
    verifyType: 'LOGIN'             // Fixed value
  }
});
```

Returns: `Promise<string>` - JWT access token

Token Management Methods

uploadImage()

Uploads token image to CDN.

```
const imgUrl = await four.uploadImage(
  accessToken: string,  // From loginDex()
  file: Blob            // Image file (JPEG, PNG, etc.)
);
```

Returns: `Promise<string>` - Image URL

---

createToken()

Gets signed parameters for on-chain token creation.

```
const { createArg, signature } = await four.createToken(
  accessToken: string,  // From loginDex()
  {
    name: string,                 // Token full name
    shortName: string,            // Token symbol
    desc: string,                 // Description
    imgUrl: string,               // Image URL from uploadImage()
    launchTime: number,           // Timestamp in milliseconds
    label: string,                // Category (Meme, AI, etc.)
    lpTradingFee: 0.0025,         // Fixed at 0.25%
    preSale: string,              // Pre-sale amount (e.g., '0', '0.1')
    onlyMPC?: boolean,            // MPC-exclusive flag
    webUrl?: string,              // Optional website
    twitterUrl?: string,          // Optional Twitter
```

```
    telegramUrl?: string          // Optional Telegram
  }
);
```

Returns: `Promise<{ createArg: string, signature: string }>`

---

getTokenByAddress()

Gets token metadata by contract address.

```
const tokenInfo = await four.getTokenByAddress(
  address: string,           // Token contract address
  accessToken?: string       // Optional access token
);
```

Returns: Token metadata object with fields like `name`, `symbol`, `imgUrl`, `version`, etc.

---

getTokenById()

Gets token metadata by token ID.

```
const tokenInfo = await four.getTokenById(
  id: string | number,       // Token ID from four.meme
  accessToken?: string       // Optional access token
);
```

---

getPublicConfig()

Gets four.meme platform configuration.

```
const config = await four.getPublicConfig();
```

On-Chain Methods

createTokenOnChain()

Deploys the token contract on-chain.

```
import { createTokenOnChain } from 'four-flap-meme-sdk';
```

```
const receipt = await createTokenOnChain({
  chain: 'BSC',                      // Chain name
  rpcUrl: string,                    // Blockchain RPC endpoint
  signerPrivateKey: string,          // Your private key
  args: BytesLike,                   // From four.createToken()
  signature: BytesLike,              // From four.createToken()
  valueWei?: bigint                  // Optional BNB to send (for pre-
sale)
});
```

Returns: Transaction receipt

Trading Methods

tryBuy()

Estimates buy cost without executing.

```
import { tryBuy } from 'four-flap-meme-sdk';

const estimate = await tryBuy(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // Chain name
  rpcUrl: string,
  token: string,           // Token address
  amount: bigint,          // Token amount (0 = buy by funds)
  funds: bigint            // BNB amount (0 = buy by amount)
);
```

Returns: `TryBuyResult` object with estimation details

---

trySell()

Estimates sell return without executing.

```
import { trySell } from 'four-flap-meme-sdk';

const estimate = await trySell(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // Chain name
  rpcUrl: string,
  token: string,
  amount: bigint           // Token amount to sell
);
```

Returns: `TrySellResult` with funds and fee

---

buyTokenWithFunds()

Buys tokens with specific BNB amount (V2 AMAP method).

```
import { buyTokenWithFunds } from 'four-flap-meme-sdk';

await buyTokenWithFunds(
  chain: 'BSC',                    // Chain name
  rpcUrl: string,
  signerPrivateKey: string,
  token: string,
  funds: bigint,                   // BNB amount to spend
  minAmount: bigint,               // Minimum tokens to receive
  to?: string,                     // Optional recipient
  origin?: bigint                  // Optional origin code (default: 0n)
);
```

---

sellToken()

Sells tokens for BNB (V2 method).

```
import { sellToken } from 'four-flap-meme-sdk';

await sellToken(
  chain: 'BSC',                    // Chain name
  rpcUrl: string,
  signerPrivateKey: string,
  token: string,
  amount: bigint,                  // Token amount to sell
  minFunds: bigint,                // Minimum BNB to receive
  origin?: bigint                  // Optional origin code (default: 0n)
);
```

Note: Requires prior token approval (use `ensureSellApproval` first)

---

tradeBuy()

Auto-routing buy method (works for V1 and V2).

```
import { tradeBuy } from 'four-flap-meme-sdk';

await tradeBuy(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // Chain name
  rpcUrl: string,
  privateKey: string,
```

```
  token: string,
  params: {
    type: 'funds' | 'amount',
    funds?: bigint,          // If type === 'funds'
    minAmount?: bigint,      // If type === 'funds'
    amount?: bigint,         // If type === 'amount'
    maxFunds?: bigint,       // If type === 'amount'
    to?: string,
    origin?: bigint
  }
);
```

---

tradeSell()

Auto-routing sell method (works for V1 and V2).

```
import { tradeSell } from 'four-flap-meme-sdk';

await tradeSell(
  chain: 'BSC' | 'BASE' | 'ARBITRUM_ONE',  // Chain name
  rpcUrl: string,
  privateKey: string,
  token: string,
  params: {
    type: 'direct' | 'router',
    amount: bigint,
    minFunds?: bigint,
    from?: string,           // If type === 'router'
    feeRate?: bigint,        // If type === 'router', in basis points
    feeRecipient?: string    // If type === 'router'
  }
);
```

Utility Methods

Token Approval Methods

The SDK provides **multiple authorization methods** to support different TokenManager versions (V1/V2) and platforms (four.meme/Flap Protocol).

Four.meme Token Approval

For V1 Tokens:

```
import { checkSellApprovalV1, ensureSellApprovalV1 } from 'four-flap-meme-
sdk';
```

```
// Check V1 approval status (read-only)
const status = await checkSellApprovalV1(
  'BSC',                    // Supported: 'BSC' | 'BASE' | 'ARBITRUM_ONE'
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// Ensure V1 approval (sends tx if needed)
const result = await ensureSellApprovalV1(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

For V2 Tokens:

```
import { checkSellApprovalV2, ensureSellApprovalV2 } from 'four-flap-meme-
sdk';

// Check V2 approval status (read-only)
const status = await checkSellApprovalV2(
  'BSC',                    // Supported: 'BSC' | 'BASE' | 'ARBITRUM_ONE'
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// Ensure V2 approval (sends tx if needed)
const result = await ensureSellApprovalV2(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

Universal Methods (defaults to V2):

```
import { checkSellApproval, ensureSellApproval } from 'four-flap-meme-
sdk';
```

```
// Check approval (defaults to V2)
const status = await checkSellApproval(
  'BSC',
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// Ensure approval (defaults to V2)
const result = await ensureSellApproval(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

Flap Protocol Token Approval

```
import { checkFlapSellApproval, ensureFlapSellApproval } from 'four-flap-
meme-sdk';

// Check Flap approval status (read-only)
const status = await checkFlapSellApproval(
  'BSC',                    // Supported: 'BSC' | 'BASE' | 'XLAYER' |
'MORPH'
  rpcUrl,
  tokenAddress,
  ownerAddress,
  amount
);

// Ensure Flap approval (sends tx if needed)
const result = await ensureFlapSellApproval(
  'BSC',
  rpcUrl,
  privateKey,
  tokenAddress,
  ownerAddress,
  amount
);
```

Return Values:

All `check*` methods return:

```
{
  isApproved: boolean,       // true if already approved
  currentAllowance: bigint,  // Current approval amount
  requiredAllowance: bigint  // Required approval amount
}
```

All `ensure*` methods return:

```
{
  alreadyApproved: boolean,  // true if no tx was needed
  currentAllowance: bigint,
  requiredAllowance: bigint,
  txReceipt?: TransactionReceipt  // Only if approval tx was sent
}
```

Complete Workflow Examples:

Four.meme V2 Token Sell:

```
// 1. Check approval status (fast, read-only)
const status = await checkSellApprovalV2('BSC', rpcUrl, token, wallet,
amount);

if (!status.isApproved) {
  // 2. Send approval transaction if needed
  console.log('Sending approval...');
  await ensureSellApprovalV2('BSC', rpcUrl, privateKey, token, wallet,
amount);
}

// 3. Execute sell
await tradeSell('BSC', rpcUrl, privateKey, token, { type: 'direct',
amount, minFunds: 0n });
```

Flap Protocol Token Sell:

```
// 1. Check approval status
const status = await checkFlapSellApproval('BSC', rpcUrl, token, wallet,
amount);

if (!status.isApproved) {
  // 2. Approve Flap Portal
  await ensureFlapSellApproval('BSC', rpcUrl, privateKey, token, wallet,
amount);
}
```

```
// 3. Execute sell
const writer = new FlapPortalWriter({ chain: 'BSC', rpcUrl }, privateKey);
await writer.swapExactInput({
  inputToken: token,
  outputToken: ZERO_ADDRESS,
  inputAmount: amount,
  minOutputAmount: 0n,
  permitData: '0x'
});
```

UI Integration Example:

```
// Update UI based on approval status
const status = await checkSellApprovalV2('BSC', rpcUrl, tokenAddr,
walletAddr, sellAmount);

if (status.isApproved) {
  showButton('Sell Now', 'green');
} else {
  showButton('Approve First', 'orange');
  showInfo(`Current: ${status.currentAllowance}, Need:
${status.requiredAllowance}`);
}
```

Important Notes:

- ⚠ **V1 and V2 use different proxy contracts** - must approve the correct version
- ⚠ **Four.meme and Flap use different contracts** - cannot mix authorization methods
- ✅ **Universal methods default to V2** for backward compatibility
- ✅ **Recommend using explicit version methods** (`V1`/`V2`) for new code

---

parseFourError()

Parses four.meme error codes.

```
import { parseFourError } from 'four-flap-meme-sdk';

const { code, message } = parseFourError(error);
```

Returns: `{ code?: FourErrorCode, message: string }`

---

isExclusiveOnChain()

Checks if token is MPC-exclusive (on-chain).

```
import { isExclusiveOnChain } from 'four-flap-meme-sdk';

const isExclusive = await isExclusiveOnChain({
  chain: 'BSC',  // Only BSC supported (TokenManagerV2)
  tokenAddress: string,
  rpcUrl: string
});
```

---

isExclusiveOffChain()

Checks if token is MPC-exclusive (via API).

```
import { isExclusiveOffChain } from 'four-flap-meme-sdk';

const isExclusive = await isExclusiveOffChain(
  getTokenInfo: (addr: string) => Promise<{ version?: string }>,
  tokenAddress: string
);
```

Flap Protocol Guide

Flap Architecture

Flap Protocol is a next-generation token launch platform with these key features:

Core Components:

1. **Portal Contract**: Main entry point for all operations
2. **CDPV2 Bonding Curve**: Advanced mathematical curve with three parameters (r, h, k)
3. **Multi-Network Support**: BSC, Base, X Layer, Morph
4. **Flexible Configuration**: Custom taxes, quote tokens, migration settings
5. **Configurable Fees**: Different chains can have different buy/sell fee rates

Token Lifecycle on Flap:

```
1. Create Token      → Deploy with CDPV2 curve
2. Trading Phase     → Buy/sell on bonding curve
3. Progress Tracking → Monitor reserve vs. threshold
4. Migration         → Auto-migrate to DEX when threshold reached
```

Reading Token State

Flap has **four versions** of token state reading methods:

Version Comparison

Method	Returns	Use Case
`getTokenV2()`	Basic state (7 fields)	Legacy compatibility
`getTokenV3()`	+ quote token info (9 fields)	Quote token support
`getTokenV4()`	+ extension ID (10 fields)	Extension support
`getTokenV5()`	+ full CDPV2 params (12 fields)	Complete information (recommended)

Reading Token State (V5 - Recommended)

```
import { FlapPortal } from 'four-flap-meme-sdk';

// Use chain enum (SDK handles addresses and default fee rates
automatically)
const portal = new FlapPortal({
  chain: 'BSC',  // Supports: 'BSC' | 'BASE' | 'XLAYER' | 'MORPH'
  rpcUrl: 'https://bsc-dataseed.binance.org'
});

const state = await portal.getTokenV5('0x...token-address');

console.log('Token Status:', state.status);          // 0=Invalid,
1=Tradable, 4=DEX
console.log('Reserve (wei):', state.reserve);        // Current BNB in
curve
console.log('Supply:', state.circulatingSupply);     // Tokens in
circulation
console.log('Price:', state.price);                  // Current price in
wei
console.log('Curve Params:', state.r, state.h, state.k);  // CDPV2
parameters
console.log('DEX Threshold:', state.dexSupplyThresh);     // Supply needed
for migration
console.log('Quote Token:', state.quoteTokenAddress);     // 0x0 = native
token
```

State Fields Explained

```
{
  status: number,              // 0=Invalid, 1=Tradable, 2=InDuel,
3=Killed, 4=DEX
  reserve: bigint,             // Current reserve in curve (in wei)
  circulatingSupply: bigint,   // Tokens in circulation (in wei, 18
decimals)
```

```
  price: bigint,               // Current price (in wei per token)
  tokenVersion: number,        // Token implementation version
  r: bigint,                   // CDPV2 parameter r (reserve offset)
  h: bigint,                   // CDPV2 parameter h (height adjustment)
  k: bigint,                   // CDPV2 parameter k (curve constant)
  dexSupplyThresh: bigint,     // Supply threshold for DEX migration
  quoteTokenAddress: Address,  // 0x0 for native, or ERC20 address
  nativeToQuoteSwapEnabled: boolean,  // Can swap native to quote
  extensionID: bytes32         // Extension identifier
}
```

CDPV2 Bonding Curve

The **CDPV2** (Constant Dynamic Price V2) curve is the core of Flap's price mechanism.

Understanding CDPV2 Parameters

Parameter `r` (Reserve Offset)

- **What it is**: Starting reserve amount
- **Effect**: Higher `r` = higher initial price
- **Typical value**: 0.1 ETH (0.1 × 10^18 wei)

Parameter `h` (Height Adjustment)

- **What it is**: Shifts the supply curve
- **Effect**: Adjusts where the curve starts
- **Typical value**: 0 (no shift)

Parameter `k` (Curve Constant)

- **What it is**: Determines price growth rate
- **Effect**: Higher `k` = higher prices overall
- **Typical value**: r × 10^9 (1 billion × r)

The CDPV2 Formulas

```
Supply (S) = 1,000,000,000 + h − k / (R + r)
Reserve (R) = k / (1,000,000,000 + h − S) − r
Price (P) = k / (1,000,000,000 + h − S)²
```

Where:

- S = circulating supply
- R = reserve (BNB/ETH in curve)
- P = price per token

Using CDPV2 Class

```
import { CDPV2 } from 'four-flap-meme-sdk';

// Create a curve with r=0.1, h=0, k=1e8
const curve = CDPV2.getCurve(0.1, 0, 1e8);

// Calculate supply needed for 10 ETH reserve
const supply = curve.estimateSupply('10');
console.log('Supply at 10 ETH:', supply.toString());

// Calculate reserve needed for 800M supply
const reserve = curve.estimateReserve('800000000');
console.log('Reserve at 800M supply:', reserve.toString());

// Get current price at 800M supply
const price = curve.price('800000000');
console.log('Price at 800M supply:', price.toString());

// Get FDV (Fully Diluted Valuation) at 800M supply
const fdv = curve.fdv('800000000');
console.log('FDV:', fdv.toString());
```

Calculating Progress

Progress = How close the token is to DEX migration

```
import { FlapPortal } from 'four-flap-meme-sdk';

const portal = new FlapPortal({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org'
});

const state = await portal.getTokenV5(tokenAddress);
const { price, progress } = portal.computePriceAndProgress(state);

console.log('Current Price:', price, 'ETH per token');
console.log('Progress:', progress);  // '0.0000' to '1.0000' (0% to 100%)
```

What progress means:

- `0.0000`: Just launched
- `0.5000`: Halfway to DEX migration
- `1.0000`: Ready for DEX migration
- Progress = (current reserve) / (reserve needed at threshold)

Flap Price Quotes

The SDK provides **two quote methods** with different advantages:

Feature	Offline Quote	On-Chain Quote
Method	`quoteBuy()` / `quoteSell()`	`quoteExactInput()`
Speed	⚡ Instant (local computation)	🐢 Slower (RPC call)
Accuracy	📊 High (99.9%+)	🎯 100% exact
Cost	💰 Free	💸 Uses RPC quota
Use Case	Real-time UI display	Final confirmation before trade
Dependency	Requires token state	Only requires token address

Method 1: Offline Quote (Recommended for UI) ⚡

Advantages:

- ⚡ **Instant**: Local computation, zero latency
- 💰 **Free**: No RPC quota consumption
- 📊 **Accurate**: Uses same formulas and fee rates as on-chain
- 🎨 **Real-time**: Perfect for live UI updates as user types

Methods: `quoteBuy()` / `quoteSell()`

How it works:

- Based on CDPV2 bonding curve formula
- Automatically deducts platform fees (configurable)
- 99.9%+ consistent with on-chain results

```
import { FlapPortal, FLAP_DEFAULT_FEE_RATES } from 'four-flap-meme-sdk';

const portal = new FlapPortal({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org',
  // Configure fee rates (optional, default is chain-specific)
  buyFeeRate: FLAP_DEFAULT_FEE_RATES.BSC.buy,   // 1% = 0.01
  sellFeeRate: FLAP_DEFAULT_FEE_RATES.BSC.sell  // 1% = 0.01
});

// 1. Get token state (once)
const tokenAddress = '0x...' as any;
const state = await portal.getTokenV5(tokenAddress);

// 2. Offline buy quote: How many tokens for 1 BNB?
const tokenAmount = portal.quoteBuy(state, '1.0');
console.log('💰 Buy Quote:');
console.log('  Pay: 1.0 BNB');
console.log('  Fee: 0.01 BNB (1%)');
console.log('  Used to buy: 0.99 BNB');
```

```
console.log('  Expected to receive:', tokenAmount, 'tokens');

// 3. Offline sell quote: How much BNB for 1000000 tokens?
const ethAmount = portal.quoteSell(state, '1000000');
console.log('\n💸 Sell Quote:');
console.log('  Sell: 1000000 tokens');
console.log('  After 1% fee deduction');
console.log('  You receive:', ethAmount, 'BNB');
```

Real-world Use Case - Live UI Updates:

```
// Scenario: User types amount in input box, show quote in real-time
let cachedState: TokenStateV5;

async function init() {
  // Fetch token state once on initialization
  cachedState = await portal.getTokenV5(tokenAddress);
}

function onUserInputChange(inputValue: string) {
  // Instantly calculate and display when user types (no delay)
  const outputAmount = portal.quoteBuy(cachedState, inputValue);
  updateUI(`You will receive ~${formatNumber(outputAmount)} tokens`);

  // Optional: Refresh cachedState periodically (e.g. every 10s) for
accuracy
}
```

Method 2: On-Chain Quote (Exact)🎯

Advantages:

- 🎯 **100% exact**: Directly simulates on-chain contract
- ✅ **Real-time state**: Uses latest on-chain data
- 🔒 **Reliable**: Final confirmation before trade

Disadvantages:

- 🦕 Requires RPC call (slower)
- 💸 Consumes RPC quota

Methods: `quoteExactInput()` / `previewBuy()` / `previewSell()`

```
const portal = new FlapPortal({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org'
});

// Method A: General quote (supports any input/output token combination)
```

```
const outputAmount = await portal.quoteExactInput({
  inputToken: '0x0000000000000000000000000000000000000000' as any,  //
Native token (BNB/ETH)
  outputToken: tokenAddress,         // Token you want
  inputAmount: 1n * 10n ** 18n      // 1 BNB (in wei)
});
console.log('Exact quote - you will receive:', formatEther(outputAmount),
'tokens');

// Method B: Specialized buy quote (more concise)
const tokenAmount = await portal.previewBuy(
  tokenAddress,
  1n * 10n ** 18n  // 1 BNB
);
console.log('Buy quote - you will receive:', formatEther(tokenAmount),
'tokens');

// Method C: Specialized sell quote (more concise)
const ethAmount = await portal.previewSell(
  tokenAddress,
  1000n * 10n ** 18n  // 1000 tokens
);
console.log('Sell quote - you will receive:', formatEther(ethAmount),
'BNB');
```

📋 Recommended Workflow

Best practice flow:

```
import { FlapPortal, FlapPortalWriter, ZERO_ADDRESS } from 'four-flap-
meme-sdk';
import { parseEther, formatEther } from 'ethers';

const rpcUrl = 'https://bsc-dataseed.binance.org';
const tokenAddress = '0x...';  // Token address
const zeroAddress = ZERO_ADDRESS;

// Create read-only instance for quotes
const portal = new FlapPortal({ chain: 'BSC', rpcUrl });

// 1️⃣ UI display phase: Use offline quote (fast)
const state = await portal.getTokenV5(tokenAddress);
const estimatedOutput = portal.quoteBuy(state, userInput);
showToUser(`Expected: ${estimatedOutput} tokens`);

// 2️⃣ User clicks "Confirm Trade": Use on-chain quote (exact)
const exactOutput = await portal.quoteExactInput({
  inputToken: zeroAddress,
  outputToken: tokenAddress,
  inputAmount: parseEther(userInput)
});
```

```
showConfirmDialog(`Exact amount: ${formatEther(exactOutput)} tokens`);

// 3️⃣ Execute trade - Create Writer instance
const writer = new FlapPortalWriter(
  { chain: 'BSC', rpcUrl },
  'YOUR_PRIVATE_KEY'
);

await writer.swapExactInput({
  inputToken: zeroAddress,
  outputToken: tokenAddress,
  inputAmount: parseEther(userInput),
  minOutputAmount: exactOutput * 99n / 100n  // Allow 1% slippage
});
```

Fee Rate Configuration:

Offline quote fee rates depend on `FlapPortal` configuration:

```
// Use default fee rates (recommended)
const portal1 = new FlapPortal({
  chain: 'BSC',
  rpcUrl,
  // Default: buyFeeRate = 0.01 (1%), sellFeeRate = 0.01 (1%) for BSC
});

// Override default fee rates (if needed)
const portal2 = new FlapPortal({
  chain: 'BSC',
  rpcUrl,
  buyFeeRate: 0.015,   // Custom rate 1.5%
  sellFeeRate: 0.015   // Custom rate 1.5%
});
```

Token Swapping

Simple Swap

```
import { FlapPortalWriter, ZERO_ADDRESS } from 'four-flap-meme-sdk';

const writer = new FlapPortalWriter(
  {
    chain: 'BSC',
    rpcUrl: 'https://bsc-dataseed.binance.org'
  },
  '0x...your-private-key'
);
```

```
// Buy tokens with 1 BNB
const txHash = await writer.swapExactInput(
  {
    inputToken: ZERO_ADDRESS,                // BNB
    outputToken: tokenAddress,               // Token to buy
    inputAmount: 1n * 10n ** 18n,           // 1 BNB
    minOutputAmount: 0n,                     // Minimum tokens (slippage)
    permitData: '0x'                         // No permit (optional)
  },
  1n * 10n ** 18n                           // msg.value = 1 BNB
);

console.log('Swap tx:', txHash);
```

Swap with Permit (Advanced)

Permit allows spending ERC20 tokens without prior approval transaction. SDK provides automatic signing:

```
import { buildPermitPiggybackAuto, FlapPortalWriter } from 'four-flap-
meme-sdk';

// SDK automatically signs Permit with correct Portal proxy address
const permitData = await buildPermitPiggybackAuto(
  'BSC',                    // Chain name (SDK auto-uses Portal address)
  privateKey,               // Your private key
  tokenAddress,             // ERC20 token address
  amount,                   // Amount to permit
  deadline,                 // Deadline timestamp
  nonce                     // Permit nonce (get from token contract)
);

// Then swap with permit (no separate approval tx!)
const writer = new FlapPortalWriter({ chain: 'BSC', rpcUrl }, privateKey);
await writer.swapExactInput({
  inputToken: tokenAddress,
  outputToken: targetToken,
  inputAmount: amount,
  minOutputAmount: 0n,
  permitData                // ✅ Approval + swap in one transaction
});
```

Creating Tokens on Flap

Use IPFS first to upload your metadata and obtain a CID, then pass the CID via the `meta` field to `newTokenV2/newTokenV3`. The helper `uploadTokenMeta(file, meta, apiUrl?)` is provided; note `FLAP_IPFS_API_URL` is a placeholder and may not work. Prefer supplying your own GraphQL endpoint or any third-party IPFS service (Pinata/Infura/Web3.Storage).

newTokenV2 - Standard Creation

```
import { FlapPortalWriter, ZERO_ADDRESS } from 'four-flap-meme-sdk';

const writer = new FlapPortalWriter(
  {
    chain: 'BSC',
    rpcUrl: 'https://bsc-dataseed.binance.org'
  },
  privateKey
);

const receipt = await writer.newTokenV2({
  name: 'My Token',
  symbol: 'MTK',
  meta: cid,                          // IPFS CID for token metadata
  dexThresh: 1,
  migratorType: 0,
  salt: '0x0000...',
  taxRate: 0,
  quoteToken: ZERO_ADDRESS,
  quoteAmt: 1n * 10n ** 18n,
  beneficiary: yourAddress,
  permitData: '0x',
  msgValue: 1n * 10n ** 18n
});

console.log('Token created:', receipt.transactionHash);
```

---

newTokenV3 - With Extension Support

V3 version supports extension features for future protocol upgrades:

```
const receipt = await writer.newTokenV3({
  // Basic info
  name: 'My Token V3',
  symbol: 'MTK3',
  imageFile: imageFile,               // Image file (SDK auto-uploads to
IPFS)
  description: 'A revolutionary token',
  website: 'https://example.com',
  twitter: 'https://x.com/mytoken',
  telegram: 'https://t.me/mytoken',

  // Token settings
  dexThresh: 1,
  salt: '0x0000...',
  taxRate: 0,
  migratorType: 0,
  quoteToken: ZERO_ADDRESS,
  quoteAmt: 1n * 10n ** 18n,
```

```
    beneficiary: yourAddress,
    permitData: '0x',
    msgValue: 1n * 10n ** 18n,

    // V3 new parameters
    extensionID: '0x0000...',           // Extension feature ID
    extensionData: '0x'                 // Extension feature data (optional)
  });

  console.log('V3 token created:', receipt.transactionHash);
```

Parameter Details

dexThresh (Migration Threshold)

Determines at what supply the token migrates to DEX:

Value	Name	Threshold	Description
0	TWO_THIRDS	66.67%	Migrate at 2/3 of max supply
1	FOUR_FIFTHS	80%	Migrate at 4/5 of max supply (recommended)
2	HALF	50%	Migrate at 1/2 of max supply
3	_95_PERCENT	95%	Migrate at 95% of max supply
4	_81_PERCENT	81%	Migrate at 81% of max supply
5	_1_PERCENT	1%	Migrate at 1% of max supply (testing)

migratorType (DEX Version)

Value	Name	DEX Version
0	V3_MIGRATOR	Uniswap V3 (recommended)
1	V2_MIGRATOR	Uniswap V2

taxRate (Transfer Tax)

- **Unit**: Basis points (1 bp = 0.01%)
- **Range**: 0 to 10000 (0% to 100%)
- **Example**: 250 = 2.5% tax on transfers
- **Recommended**: 0 (no tax) for most cases

salt (CREATE2 Salt)

- **Purpose**: Deterministic address generation
- **Type**: 32-byte hex string
- **Use Case**: Vanity addresses (see next section)

Vanity Addresses

Vanity addresses are token addresses with custom patterns, making tokens easier to identify and remember.

What are Vanity Addresses?

Vanity addresses use the CREATE2 mechanism by adjusting the `salt` parameter to generate token contract addresses with specific patterns (usually specific endings).

Flap Platform's Default Vanity Suffixes:

- `8888`: Default vanity suffix for normal tokens (NORMAL)
- `7777`: Default vanity suffix for taxed tokens (TAXED)

These two suffixes are just **Flap platform conventions** for quick token type identification:

- See `8888` ending → likely a normal non-taxed token
- See `7777` ending → likely a token with transfer tax

You can generate any custom vanity address:

- ✅ `888` - Shorter, easier to find
- ✅ `8888` - Flap platform default
- ✅ `88888` - Longer, more rare (requires more computation)
- ✅ `666`, `999`, `abc`, `def` - Any hexadecimal pattern you want
- ✅ `yourname` - Even English words (if you can find them)

Computational Complexity:

- Each additional digit increases difficulty by 16x (hexadecimal)
- `888` (3 digits) ≈ a few thousand iterations
- `8888` (4 digits) ≈ tens of thousands of iterations
- `88888` (5 digits) ≈ millions of iterations
- `888888` (6 digits) ≈ tens of millions of iterations

Predicting Token Address

```
import { predictVanityTokenAddressByChain } from 'four-flap-meme-sdk';

// Use chain name (SDK auto-handles Portal and TokenImpl addresses)
const predictedAddress = predictVanityTokenAddressByChain(
  'BSC',                    // Chain name
  salt                      // 32-byte salt
);

console.log('Token will be deployed at:', predictedAddress);

// For BSC, if creating a taxed token, specify the taxed parameter
const predictedTaxedAddress = predictVanityTokenAddressByChain(
  'BSC',
```

```
  salt,
  true                      // taxed = true
);
```

Finding a Vanity Salt

```
import { findSaltEndingByChain } from 'four-flap-meme-sdk';

// Example 1: Find Flap platform default 8888 ending (normal token)
const result1 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '8888',             // Flap platform default (normal token)
  maxIterations: 1_000_000    // Usually finds within tens of thousands
});
console.log('Found address:', result1.address); // e.g: 0x...8888
console.log('Using salt:', result1.salt);
console.log('Iterations:', result1.iterations);

// Example 2: Find shorter vanity (faster)
const result2 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '888',              // Only 3 digits, easier to find
  maxIterations: 100_000      // Usually finds within thousands
});

// Example 3: Find custom vanity
const result3 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: 'abc',              // Custom suffix
  maxIterations: 100_000
});

// Example 4: Find longer vanity (takes more time)
const result4 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '88888',            // 5 digits, more rare
  maxIterations: 10_000_000,  // May require millions of iterations
  seed:
'0x0000000000000000000000000000000000000000000000000000000000000001'  //
Optional starting salt
});

// Example 5: Find taxed token default vanity (BSC chain)
const result5 = await findSaltEndingByChain({
  chain: 'BSC',
  suffix: '7777',             // Flap platform default (taxed token)
  taxed: true,                // Specify as taxed token
  maxIterations: 1_000_000
});

// Use the found salt when creating token
```

```
import { FlapPortalWriter } from 'four-flap-meme-sdk';

const writer = new FlapPortalWriter(
  {
    chain: 'BSC',
    rpcUrl: 'https://bsc-dataseed.binance.org'
  },
  'YOUR_PRIVATE_KEY'
);

await writer.newTokenV2({
  name: 'My Token',
  symbol: 'MTK',
  // ... other parameters
  salt: result1.salt          // Use the found salt
});
```

Important Notes:

- ⚠ Vanity address generation is **CPU-intensive**
- ⚠ In production, recommend **pre-computing salts offline**
- ⚠ Longer suffixes require **exponentially more** computation time
- ⚠ Recommend using **3-4 digit** suffixes for balance between recognition and speed
- ✅ `8888` and `7777` are just Flap platform conventions, you can use **any suffix**
- ✅ Suffixes are **hexadecimal** (0-9, a-f), case insensitive

---

Uploading Token Metadata

Use `uploadTokenMeta(file, meta, apiUrl?)` helper or any third-party IPFS. The default `FLAP_IPFS_API_URL` is a placeholder and may not work; provide your own GraphQL endpoint if using this helper.

```
import { uploadTokenMeta } from 'four-flap-meme-sdk';

const cid = await uploadTokenMeta(
  imageBlob as File,
  {
    description: 'My amazing token',
    creator: walletAddress,
    website: 'https://example.com',
    twitter: 'https://x.com/mytoken',
    telegram: 'https://t.me/mytoken'
  },
  'https://your-ipfs-graphql.example'
);
```

Using Pinata (optional)

You can also rely on Pinata SDK via this SDK's `PinataClient` to upload files/JSON and quickly obtain a CID. See Pinata Quickstart.

```
import { PinataClient, ZERO_ADDRESS, pinFileToIPFSWithJWT, pinImageByPath,
pinFileToIPFSWithJWTWeb, pinDataURLWithJWTWeb } from 'four-flap-meme-sdk';

// 1) Initialize (prefer server-side env for JWT)
const pinata = new PinataClient({
  jwt: process.env.PINATA_JWT as string,
  gateway: 'example-gateway.mypinata.cloud' // optional
});

// 2) Upload JSON to get CID (or use uploadFile for images)
const { cid: metaCid } = await pinata.uploadJSON({
  image: 'ipfs://<image_cid>',
  description: 'My token description',
  creator: '0xYourAddress',
  website: 'https://example.com'
});

// 3) Pass meta: cid when creating
const receipt = await writer.newTokenV2({
  name: 'My Token',
  symbol: 'MTK',
  meta: metaCid,
  dexThresh: 1,
  salt: '0x...',
  taxRate: 0,
  migratorType: 0,
  quoteToken: ZERO_ADDRESS,
  quoteAmt: 1n * 10n ** 18n,
  beneficiary: '0xYourAddress',
  permitData: '0x',
  msgValue: 1n * 10n ** 18n
});
```

Direct Pinata REST (Node and Web)

```
// Node: file path or stream/Buffer
const { cid } = await pinFileToIPFSWithJWT(PINATA_JWT,
'/abs/path/img.png');
// Or stream:
const stream = fs.createReadStream('/abs/path/img.png');
const { cid: cid2 } = await pinFileToIPFSWithJWT(PINATA_JWT, undefined,
stream, 'img.png');

// Node: path shortcut
const cid3 = await pinImageByPath(PINATA_JWT, '/abs/path/img.png');

// Web: File/Blob
```

```
const input = document.querySelector('input[type=file]') as
HTMLInputElement;
const file = input.files![0];
const { cid: webCid } = await pinFileToIPFSWithJWTWeb(PINATA_JWT, file,
file.name);

// Web: dataURL (e.g. canvas.toDataURL())
const { cid: dataUrlCid } = await pinDataURLWithJWTWeb(PINATA_JWT,
dataUrl, 'image.png');
```

Note: Pass the resulting CID to `newTokenV2/newTokenV3` via the `meta` field.

Manual Metadata Upload (Advanced Users)

If you need to upload metadata separately before creating a token:

```
import { uploadTokenMeta } from 'four-flap-meme-sdk';

// Manual upload (uses Flap official free endpoint)
const cid = await uploadTokenMeta(
  imageFile,
  {
    description: 'My amazing token that does XYZ',
    creator: '0x1234...',
    website: 'https://example.com',
    twitter: 'https://x.com/mytoken',
    telegram: 'https://t.me/mytoken'
  }
);

console.log('Metadata CID:', cid);  // Returns IPFS CID like:
'bafkreicwlkp...'

// Use custom IPFS endpoint (optional)
const customCid = await uploadTokenMeta(
  imageFile,
  { description: '...', creator: '...' },
  'https://your-custom-ipfs-endpoint.com/graphql'
);
```

Other IPFS Services:

- **Pinata**: https://pinata.cloud/
- **Infura IPFS**: https://infura.io/
- **Web3.Storage**: https://web3.storage/
- **NFT.Storage**: https://nft.storage/

Note: Third-party services require registration and configuration, and may use different API formats.

Flap Error Handling

The SDK provides specialized error parsing to help you understand transaction failures:

```
import {
  parseFlapError,
  getFlapErrorMessage,
  getFlapErrorMessageEn,
  FlapErrorCode
} from 'four-flap-meme-sdk';

try {
  await writer.swapEthForToken({ ... });
} catch (error) {
  const errorCode = parseFlapError(error);
  const message = getFlapErrorMessageEn(errorCode);  // English error
message
  console.error('Transaction failed:', message);

  // Or use Chinese
  const messageCn = getFlapErrorMessage(errorCode);
  console.error('交易失败:', messageCn);
}
```

Common Error Codes

Error Code	Description	Solution
`TOKEN_ALREADY_EXISTS`	Token already exists	Use a different salt or parameters
`TOKEN_NOT_TRADABLE`	Token is not tradable	Check token state, may be migrated
`ALREADY_ON_DEX`	Token is already on DEX	Trade on DEX instead of Portal
`INSUFFICIENT_BALANCE`	Insufficient balance	Increase wallet balance or reduce amount
`SLIPPAGE_EXCEEDED`	Slippage too high	Adjust `minTokenAmount` or retry later
`PERMIT_EXPIRED`	Permit signature expired	Regenerate Permit signature
`INVALID_SIGNATURE`	Invalid signature	Check signature parameters
`INVALID_AMOUNT`	Invalid amount	Verify input amount is correct
`INVALID_QUOTE_TOKEN`	Invalid quote token	Use supported quote token
`INVALID_DEX_THRESH`	Invalid DEX threshold	Use valid threshold 0-5

Flap Constants

The SDK provides predefined constants to simplify configuration:

```
import {
  FlapPortal,
  FLAP_DEFAULT_FEE_RATES,
  FLAP_DEX_THRESHOLDS,
  FLAP_TOTAL_SUPPLY,
  FLAP_IPFS_API_URL,
  FLAP_VANITY_SUFFIX
} from 'four-flap-meme-sdk';

// Create FlapPortal (SDK automatically manages contract addresses)
const portal = new FlapPortal({ chain: 'BSC', rpcUrl: 'https://...' });
// Supported chains: 'BSC', 'BASE', 'MORPH', 'XLAYER'

// Default fee rates
const { buy, sell } = FLAP_DEFAULT_FEE_RATES.BSC;
console.log(`Buy fee: ${buy * 100}%, Sell fee: ${sell * 100}%`);
// Output: Buy fee: 1%, Sell fee: 1%

// DEX migration thresholds (as percentages)
const thresh = FLAP_DEX_THRESHOLDS.FOUR_FIFTHS; // 0.8 (80%)

// Token total supply (1 billion with 18 decimals)
console.log('Total supply:', FLAP_TOTAL_SUPPLY);
// Output: 1000000000000000000000000000n

// IPFS API URL
const ipfsUrl = FLAP_IPFS_API_URL; // 'https://api.flap.sh/graphql'

// Vanity address suffixes
const normalSuffix = FLAP_VANITY_SUFFIX.NORMAL; // '8888'
const taxedSuffix = FLAP_VANITY_SUFFIX.TAXED;   // '7777'
```

Supported Chains

Currently supported chains and their default configurations:

Chain	Buy Fee	Sell Fee
BSC	1%	1%
BASE	2.5%	2.5%
MORPH	2.5%	2.5%
XLAYER	1.5%	1.5%

Important Notes:

- **Buy/Sell Fees**: Trading fees collected by **Flap Protocol platform**
- These fees are automatically deducted on each trade:
  - **On Buy**: Fee is deducted from your BNB/ETH payment, remaining amount buys tokens
  - **On Sell**: Fee is deducted from your BNB/ETH proceeds

- This is different from token `beneficiary` fees:
  - `beneficiary` receives LP revenue share (V3 migrator) after DEX migration or transfer tax (taxed tokens)
  - Platform trading fees go to Flap Protocol for platform operations

💰 **Example Calculations**:

Trading on **BSC** (1% fee rate):

Buy Example:

```
You pay:           1.0 BNB
Platform fee:      0.01 BNB  → Collected by Flap Protocol
Used to buy:       0.99 BNB  → Buys tokens at bonding curve price
```

Sell Example:

```
Tokens worth:      1.0 BNB   (calculated from bonding curve)
Platform fee:      0.01 BNB  → Collected by Flap Protocol
You receive:       0.99 BNB
```

Trading on **BASE** or **MORPH** (2.5% fee rate):

Buy Example:

```
You pay:           1.0 ETH
Platform fee:      0.025 ETH → Collected by Flap Protocol
Used to buy:       0.975 ETH → Buys tokens at bonding curve price
```

Sell Example:

```
Tokens worth:      1.0 ETH   (calculated from bonding curve)
Platform fee:      0.025 ETH → Collected by Flap Protocol
You receive:       0.975 ETH
```

Trading on **XLAYER** (1.5% fee rate):

Buy Example:

```
You pay:           1.0 ETH
Platform fee:      0.015 ETH → Collected by Flap Protocol
Used to buy:       0.985 ETH → Buys tokens at bonding curve price
```

Note: You can override default fees when creating `FlapPortal`:

```
const portal = new FlapPortal({
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org',
  buyFeeRate: 0.015,   // Override: 1.5% buy fee
  sellFeeRate: 0.015   // Override: 1.5% sell fee
});
```

Flap API Reference

FlapPortal Class

Constructor

```
// Use chain enum (SDK handles contract address and fee rates
automatically)
const portal = new FlapPortal({
  chain: 'BSC' | 'BASE' | 'XLAYER' | 'MORPH',  // Chain name
  rpcUrl: string,                               // RPC endpoint
  buyFeeRate?: number,          // Optional: override default buy fee rate
  sellFeeRate?: number          // Optional: override default sell fee
rate
});
```

Reading Methods

getTokenV2()

Gets basic token state (7 fields).

```
const state = await portal.getTokenV2(token: Address);
// Returns: TokenStateV2
```

---

getTokenV3()

Gets token state with quote token info (9 fields).

```
const state = await portal.getTokenV3(token: Address);
// Returns: TokenStateV3
```

---

getTokenV4()

Gets complete token state (12 fields, recommended).

```
const state = await portal.getTokenV5(token: Address);
// Returns: TokenStateV5
```

Calculation Methods

computePriceAndProgress()

Calculates current price and migration progress.

```
const { price, progress } = portal.computePriceAndProgress(state:
TokenStateV5);
// price: string (ETH per token)
// progress: string ('0.0000' to '1.0000')
```

---

quoteBuy()

Off-chain buy quote (fast but approximate).

```
const tokensOut = portal.quoteBuy(
  state: TokenStateV5,
  inputEth: string        // Amount in human-readable format (e.g., '1')
);
// Returns: string (tokens out)
```

---

quoteSell()

Off-chain sell quote (fast but approximate).

```
const ethOut = portal.quoteSell(
  state: TokenStateV5,
  inputToken: string      // Token amount in human-readable format
);
// Returns: string (ETH out)
```

---

quoteExactInput()

On-chain quote via contract simulation (accurate).

```
const outputAmount = await portal.quoteExactInput({
  inputToken: Address,
  outputToken: Address,
  inputAmount: bigint
});
// Returns: Promise<bigint>
```

FlapPortalWriter Class

For write operations (requires private key).

Constructor

```
const writer = new FlapPortalWriter(
  { rpcUrl: string, portalAddress: Address },
  privateKey: Hex
);
```

---

swapExactInput()

Executes a token swap.

```
const txHash = await writer.swapExactInput(
  {
    inputToken: Address,
    outputToken: Address,
    inputAmount: bigint,
    minOutputAmount: bigint,
    permitData?: Hex        // Optional permit data
  },
  msgValue?: bigint         // BNB to send (if buying with native)
);
// Returns: Promise<Hex> (transaction hash)
```

---

newTokenV2()

Creates a new token. SDK automatically handles IPFS upload.

```
const receipt = await writer.newTokenV2({
  name: string,
```

```
  symbol: string,
  imageFile: File,              // Image file (SDK auto-uploads to IPFS)
  description: string,          // Token description
  website?: string,             // Optional website
  twitter?: string,             // Optional Twitter
  telegram?: string,            // Optional Telegram
  dexThresh: number,            // 0-5 (see DexThreshType enum)
  salt: Hex,                    // 32-byte salt
  taxRate: number,              // Basis points (0-10000)
  migratorType: number,         // 0 or 1 (see MigratorType enum)
  quoteToken: Address,          // 0x0 for native
  quoteAmt: bigint,
  beneficiary: Address,
  permitData?: Hex,
  msgValue?: bigint
});
// Returns: Promise<TransactionReceipt>
```

CDPV2 Class

getCurve()

Creates a curve instance.

```
const curve = CDPV2.getCurve(
  r: number,              // Reserve offset (e.g., 0.1)
  h?: number,             // Height adjustment (optional, default: 0)
  k?: number              // Curve constant (optional, default: r * 1e9)
);
```

---

estimateSupply()

Calculates supply for a given reserve.

```
const supply = curve.estimateSupply(reserve: string);
// Returns: Decimal
```

---

estimateReserve()

Calculates reserve for a given supply.

```
const reserve = curve.estimateReserve(supply: string);
// Returns: Decimal
```

---

price()

Calculates price at a given supply.

```
const price = curve.price(supply: string);
// Returns: Decimal
```

---

fdv()

Calculates fully diluted valuation.

```
const fdv = curve.fdv(supply: string);
// Returns: Decimal
```

Utility Functions

uploadTokenMeta()

```
const cid = await uploadTokenMeta(
  apiUrl: string,
  file: File,
  meta: TokenMetaInput
);
// Returns: Promise<string> (IPFS CID)
```

---

buildPermitPiggyback()

```
const permitData = buildPermitPiggyback(
  owner: Address,
  spender: Address,
  value: bigint,
  deadline: bigint,
  v: number,
  r: Hex,
  s: Hex
);
// Returns: Hex
```

Flowchart (Read + Quote)

```
+--------------+        +--------------------+
| FlapPortal   |        | Portal Contract    |
+--------------+        +--------------------+
      | getTokenV5(token)        |
      |------------------------->|
      |    TokenStateV5 tuple    |
      |<-------------------------|
      | computePriceAndProgress  |
      | quoteBuy/quoteSell       |
```

48.club Private Transactions

48SP Modes & Configuration

The SDK supports both regular (non-48SP) and member (48SP) channels.

- Channel types
  - Regular: No 48SP signature; use `sendPrivateTransaction` / `eth_sendBundle` without `48spSign`.
  - Member (48SP): Provide a 48SP signature for premium priority.
- How to configure
  - Most high-level bundle helpers accept a config with:
    - `spMode?: 'none' | 'timestampPersonalSign' | 'concatTxHash'` (default: `'timestampPersonalSign'`)
    - `spVMode?: '27_28' | '0_1'` (optional v normalization)
    - `spPrivateKey?: string` (optional 48SP key; falls back to `privateKeys[0]` if omitted)
  - If `spMode: 'none'`, the SDK skips 48SP generation.
- Low-level usage
  - `Club48Client.sendBundle(params, opts)`
  - `Club48Client.sendPrivateTransactionWith48SP(tx, spKey, opts)`
  - `sendBatchPrivateTransactions(txs, spKey, endpoint, opts)`

```
import { Club48Client } from 'four-flap-meme-sdk';

const client = new Club48Client({ endpoint: 'https://puissant-bsc.48.club'
});

// Regular bundle (no 48SP)
await client.sendBundle({ txs, maxTimestamp: now + 300 }, { spMode: 'none'
```

```
});

// 48SP bundle with timestamp personal_sign (recommended)
await client.sendBundle(
  { txs, maxTimestamp: now + 300 },
  { spMode: 'timestampPersonalSign', spPrivateKey: '0x...spKey', spVMode:
'27_28' }
);

// Single private tx with 48SP
await client.sendPrivateTransactionWith48SP(signedTx, '0x...spKey', {
  spMode: 'timestampPersonalSign',
  spVMode: '27_28'
});
```

Notes

- Use `'timestampPersonalSign'` unless your endpoint explicitly requires the legacy `concatTxHash` mode.
- If you encounter "invalid signature recovery id", try switching `spVMode` between `'27_28'` and `'0_1'`.
- Avoid sending concurrent bundles from the same address (nonce replacement risk). See 48 docs: `https://docs.48.club/`.

What is 48.club?

48.club is a MEV (Maximal Extractable Value) protection service that provides:

- 🔒 **Private Mempool**: Transactions are not visible in public mempool
- ⚡ **Fast Execution**: Direct connection to validators for faster inclusion
- 🛡️ **Front-running Protection**: Prevents bots from front-running your trades
- 📦 **Batch Transactions**: Submit multiple transactions atomically (all succeed or all fail)
- 🎯 **Backrun**: Execute transactions immediately after a target transaction

Private Transaction Benefits

Why Use Private Transactions?

Public Mempool Problems:

- 🔴 Bots can see your transaction before it's mined
- 🔴 Front-runners can copy and execute before you
- 🔴 Your transaction details are visible to everyone
- 🔴 Sandwich attacks can steal your profits

48.club Private Transaction Advantages:

- ✅ **Hidden Until Mined**: Transaction details stay private
- ✅ **No Front-running**: Bots can't see or copy your trade
- ✅ **MEV Protection**: Validators won't reorder against you

- ✅ **Batch Atomicity**: Multiple wallets buy/sell together, all or nothing
- ✅ **48 SoulPoint (48SP)**: Premium service with higher priority

When to Use Private Transactions?

Use private transactions for:

- 🎯 **Large Trades**: Avoid price impact from front-runners
- 💰 **Token Launches**: Buy new tokens without being sniped
- 📦 **Coordinated Buying**: Multiple wallets buy together atomically
- 🤖 **Anti-Bot**: Prevent bot sniping on fair launches
- 🔐 **Privacy**: Keep your trading strategy confidential

Four.meme Private Trading

The SDK provides convenient methods to trade on four.meme using 48.club private transactions.

Single Private Buy

Buy tokens privately without public mempool exposure.

```
import { fourPrivateBuy } from 'four-flap-meme-sdk';

// Buy with regular private transaction
const txHash = await fourPrivateBuy({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',     // Token contract address
  funds: '1.0',              // BNB amount as string (e.g., '1.0' = 1 BNB)
  to: '0x...',               // Optional: recipient address (defaults to
buyer)
});

console.log('Private buy tx:', txHash);
```

With 48 SoulPoint (Premium):

```
// Use 48SP for higher priority and faster execution
const txHash = await fourPrivateBuy({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',
  funds: '1.0',
  spPrivateKey: '0x...',     // 48 SoulPoint private key for premium
service
});
```

Parameters:

- `rpcUrl`: BSC RPC endpoint
- `privateKey`: Your wallet private key
- `tokenAddress`: Token to buy (four.meme uses 0x0 for BNB internally)
- `funds`: BNB amount as string (human-readable, e.g., '0.5', '1.0', '10')
- `to`: Optional recipient address
- `spPrivateKey`: Optional 48SP key for premium service
- `club48Endpoint`: Optional custom 48.club endpoint
- `club48ExplorerEndpoint`: Optional custom 48.club explorer

Single Private Sell

Sell tokens privately.

```
import { fourPrivateSell } from 'four-flap-meme-sdk';

// Sell tokens privately
const txHash = await fourPrivateSell({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',
  amount: '1000',            // Token amount as string (18 decimals)
  minFunds: 0n,              // Optional: minimum BNB to receive (slippage
protection)
});

console.log('Private sell tx:', txHash);
```

With 48SP:

```
const txHash = await fourPrivateSell({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKey: '0x...',
  tokenAddress: '0x...',
  amount: '1000',
  spPrivateKey: '0x...',     // Use 48SP for premium service
});
```

Parameters:

- `amount`: Token amount as string (e.g., '1000', '50000')
- `minFunds`: Optional minimum BNB output (bigint, defaults to 0n)

Note: Make sure you've approved the TokenManager contract before selling.

Batch Private Buy (Multiple Wallets)

Buy tokens from multiple wallets atomically - either all transactions succeed or all fail.

```
import { fourBatchPrivateBuy } from 'four-flap-meme-sdk';

// Batch buy with 3 wallets (requires 48SP)
const success = await fourBatchPrivateBuy({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKeys: [
    '0x...wallet1_key',
    '0x...wallet2_key',
    '0x...wallet3_key',
  ],
  fundsList: [
    '0.5',    // Wallet 1 buys with 0.5 BNB
    '1.0',    // Wallet 2 buys with 1.0 BNB
    '0.3',    // Wallet 3 buys with 0.3 BNB
  ],
  tokenAddress: '0x...',
  spPrivateKey: '0x...48sp_key',  // Required for batch transactions
});

console.log('Batch buy success:', success);
```

Use Cases:

- 🎯 **Coordinated Launch**: Multiple team members buy together
- 💰 **Fair Distribution**: Split buys across wallets atomically
- 🔐 **All-or-Nothing**: If one fails, all revert (no partial execution)

Important:

- ⚠ Batch transactions **require 48 SoulPoint (48SP)** private key
- ⚠ All transactions execute atomically (all succeed or all fail)
- ⚠ `privateKeys.length` must equal `fundsList.length`

Batch Private Sell (Multiple Wallets)

Sell tokens from multiple wallets atomically.

```
import { fourBatchPrivateSell } from 'four-flap-meme-sdk';

// Batch sell with 3 wallets (requires 48SP)
const success = await fourBatchPrivateSell({
  rpcUrl: 'https://bsc-dataseed.binance.org',
  privateKeys: [
    '0x...wallet1_key',
    '0x...wallet2_key',
```

```
    '0x...wallet3_key',
  ],
  amounts: [
    '1000',   // Wallet 1 sells 1000 tokens
    '2000',   // Wallet 2 sells 2000 tokens
    '500',    // Wallet 3 sells 500 tokens
  ],
  tokenAddress: '0x...',
  spPrivateKey: '0x...48sp_key',  // Required for batch
  minFundsEach: 0n,                // Optional: minimum BNB per wallet
});

console.log('Batch sell success:', success);
```

Parameters:

- `amounts`: Array of token amounts as strings
- `minFundsEach`: Optional unified minimum BNB output for all sellers
- `spPrivateKey`: Required 48SP key

Flap Protocol Private Trading

Similar private trading methods for Flap Protocol.

Single Private Buy (Flap)

```
import { flapPrivateBuy } from 'four-flap-meme-sdk';

// Buy on Flap Protocol privately
const txHash = await flapPrivateBuy({
  chain: 'BSC',              // Supported: 'BSC', 'BASE', 'XLAYER',
'MORPH'
  privateKey: '0x...',
  tokenAddress: '0x...',
  amountIn: '1.0',           // Native token amount (BNB/ETH)
  to: '0x...',               // Optional: recipient
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
    club48Endpoint: 'https://puissant-bsc.48.club',  // Optional
  },
});

console.log('Flap private buy tx:', txHash);
```

With 48SP:

```
const txHash = await flapPrivateBuy({
  chain: 'BSC',
```

```
  privateKey: '0x...',
  tokenAddress: '0x...',
  amountIn: '1.0',
  spPrivateKey: '0x...',     // Use 48SP premium service
  config: { rpcUrl },
});
```

Single Private Sell (Flap)

```
import { flapPrivateSell } from 'four-flap-meme-sdk';

// Sell on Flap Protocol privately
const txHash = await flapPrivateSell({
  chain: 'BSC',
  privateKey: '0x...',
  tokenAddress: '0x...',
  amount: '1000',            // Token amount (18 decimals)
  minEth: 0n,                // Optional: minimum ETH/BNB to receive
  config: { rpcUrl },
});

console.log('Flap private sell tx:', txHash);
```

Batch Private Buy (Flap)

```
import { flapBatchPrivateBuy } from 'four-flap-meme-sdk';

// Batch buy on Flap (requires 48SP)
const success = await flapBatchPrivateBuy({
  chain: 'BSC',
  privateKeys: ['0x...key1', '0x...key2', '0x...key3'],
  amountsIn: ['0.5', '1.0', '0.3'],  // BNB amounts
  tokenAddress: '0x...',
  spPrivateKey: '0x...48sp_key',     // Required
  config: { rpcUrl },
});

console.log('Flap batch buy success:', success);
```

Batch Private Sell (Flap)

```
import { flapBatchPrivateSell } from 'four-flap-meme-sdk';

// Batch sell on Flap (requires 48SP)
```

```
const success = await flapBatchPrivateSell({
  chain: 'BSC',
  privateKeys: ['0x...key1', '0x...key2'],
  amounts: ['1000', '2000'],         // Token amounts
  tokenAddress: '0x...',
  spPrivateKey: '0x...48sp_key',     // Required
  minEthEach: 0n,                    // Optional: minimum ETH per wallet
  config: { rpcUrl },
});

console.log('Flap batch sell success:', success);
```

Flap Protocol Bundle Trading

Bundle trading allows you to package multiple transactions into a single atomic operation - either all succeed or all fail. This is particularly useful for **token creation + bundled purchases**.

Why Use Bundles?

✅ **Atomicity**: All transactions succeed or fail together
✅ **Anti-frontrun**: Creation and purchases complete in the same block
✅ **Multi-wallet coordination**: Support for multiple wallets buying simultaneously
✅ **Fair launch**: Prevents bots from sniping immediately after creation

Important Notes

⚠ **BSC Only**: 48.club Bundle service is currently only available on BSC
⚠ **Requires 48SP**: Bundle features require a 48 SoulPoint private key
⚠ **Address prediction required**: Must calculate token address before creation
⚠ **Multiple txs from same address**: Not recommended to send many bundle txs from a single address. The SDK assigns sequential nonces, but acceptance depends on validator policy; prefer multiple addresses or use private batch APIs.

---

1. Create Token + Bundled Purchases

Create a token and immediately have multiple wallets buy atomically.

Step 1: Prepare IPFS Metadata

```
import { PinataSDK } from "pinata-web3";

const pinata = new PinataSDK({
  pinataJwt: process.env.PINATA_JWT,
});

// Upload image
const imageUpload = await pinata.upload.file(imageFile);
const imageCid = imageUpload.IpfsHash;
```

```
// Upload metadata
const metadata = {
  image: imageCid,
  description: "My token description",
  creator: devAddress,
  website: "https://example.com",
  twitter: "https://x.com/mytoken",
  telegram: "https://t.me/mytoken",
  buy: null,
  sell: null
};

const jsonUpload = await pinata.upload.json(metadata);
const metaCid = jsonUpload.IpfsHash;
```

Step 2: Predict Token Address

```
import { predictVanityTokenAddressByChain } from 'four-flap-meme-sdk';

// Get Portal contract's nonce
const portal = new FlapPortal({ chain: 'BSC', rpcUrl });
const nonce = await portal.nonce();

// Predict token address
const predicted = predictVanityTokenAddressByChain(
  'BSC',
  'My Token',
  'MTK',
  nonce,
  0  // taxRate: 0 = normal token, non-0 = taxed token
);

console.log('Predicted address:', predicted.address);
console.log('Using salt:', predicted.salt);
```

Step 3: Create + Bundled Purchase

```
import { flapCreateTokenWithBundleBuy } from 'four-flap-meme-sdk';

const result = await flapCreateTokenWithBundleBuy({
  chain: 'BSC',

  // Private keys array: [creator, buyer1, buyer2, ...]
  privateKeys: [
    '0x...dev_key',
    '0x...buyer1_key',
    '0x...buyer2_key',
    '0x...buyer3_key',
```

```
    ],

    // Buy amounts (BNB): for buyer1, buyer2, buyer3
    buyAmounts: ['0.5', '1.0', '0.3'],

    // Token info
    tokenInfo: {
      name: 'My Token',
      symbol: 'MTK',
      meta: metaCid,  // IPFS CID from step 1
    },

    // Predicted address from step 2
    tokenAddress: predicted.address,

    // Configuration
    config: {
      rpcUrl: 'https://bsc-dataseed.binance.org',
      club48Endpoint: 'https://puissant-bsc.48.club',
      club48ExplorerEndpoint: 'https://puissant-bsc-tx.48.club',
      // 48SP (member) channel — comment out to use regular channel
      spMode: 'timestampPersonalSign',   // or 'none' to disable 48SP
      // spVMode: '27_28',               // optional v normalization
      // spPrivateKey: '0x...48sp_key',  // optional explicit 48SP key
    },
  });

  console.log('Bundle UUID:', result.bundleUuid);
  console.log('Status:', result.status);
  console.log('Create tx:', result.createTx);
  console.log('Buy txs:', result.buyTxs);
```

Execution Flow:

1. Creator sends transaction to create token
2. Buyers 1, 2, 3 simultaneously send purchase transactions
3. All transactions execute atomically as a bundle
4. If any fails, all rollback

Use Cases:

- 🎯 **Fair launch**: Prevents bots from sniping immediately after creation
- 💰 **Team buy-in**: Multiple team members get initial tokens simultaneously
- 🔐 **Coordinated start**: Ensures token has initial liquidity as soon as it's created

---

2. Batch Buy (Existing Token)

Execute multi-wallet atomic purchases for an existing token.

```
import { flapBatchBuyWithBundle } from 'four-flap-meme-sdk';

const result = await flapBatchBuyWithBundle({
  chain: 'BSC',

  // Private keys array
  privateKeys: [
    '0x...buyer1_key',
    '0x...buyer2_key',
    '0x...buyer3_key',
  ],

  // Corresponding buy amounts (BNB)
  buyAmounts: ['0.5', '1.0', '0.3'],

  // Token address
  tokenAddress: '0x...',

  // Configuration
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
    club48Endpoint: 'https://puissant-bsc.48.club',
    spMode: 'timestampPersonalSign',   // or 'none'
    // spVMode: '27_28',               // optional
    // spPrivateKey: '0x...48sp_key',  // optional
  },
});

console.log('Bundle UUID:', result.bundleUuid);
console.log('Status:', result.status);
console.log('Buy txs:', result.buyTxs);
```

Parameters:

- `privateKeys.length` must equal `buyAmounts.length`
- All transactions execute atomically (all succeed or all fail)

---

3. Batch Sell

Execute multi-wallet atomic sells for a token.

```
import { flapBatchSellWithBundle } from 'four-flap-meme-sdk';

const result = await flapBatchSellWithBundle({
  chain: 'BSC',

  // Private keys array
  privateKeys: [
    '0x...seller1_key',
    '0x...seller2_key',
```

```
    '0x...seller3_key',
  ],

  // Corresponding sell amounts (tokens, 18 decimals)
  sellAmounts: ['1000', '2000', '500'],

  // Token address
  tokenAddress: '0x...',

  // Configuration
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
    club48Endpoint: 'https://puissant-bsc.48.club',
    spMode: 'timestampPersonalSign',   // or 'none'
    // spVMode: '27_28',               // optional
    // spPrivateKey: '0x...48sp_key',  // optional
  },
});

console.log('Bundle UUID:', result.bundleUuid);
console.log('Status:', result.status);
console.log('Sell txs:', result.sellTxs);
```

Note:

- Ensure all addresses have approved the Portal contract before selling
- All transactions execute atomically

Complete Example: Launch from Scratch with Anti-Snipe

```
import {
  FlapPortal,
  predictVanityTokenAddressByChain,
  flapCreateTokenWithBundleBuy
} from 'four-flap-meme-sdk';
import { PinataSDK } from "pinata-web3";

// ========== 1. Upload Metadata ==========
const pinata = new PinataSDK({ pinataJwt: process.env.PINATA_JWT });
const imageUpload = await pinata.upload.file(imageFile);
const metadata = {
  image: imageUpload.IpfsHash,
  description: "Revolutionary token",
  creator: devAddress,
};
const jsonUpload = await pinata.upload.json(metadata);
const metaCid = jsonUpload.IpfsHash;

// ========== 2. Predict Address ==========
const portal = new FlapPortal({
```

```
  chain: 'BSC',
  rpcUrl: 'https://bsc-dataseed.binance.org'
});
const nonce = await portal.nonce();
const predicted = predictVanityTokenAddressByChain(
  'BSC',
  'Revolution Token',
  'REV',
  nonce,
  0
);

// ========== 3. Bundle Create + Buy ==========
const result = await flapCreateTokenWithBundleBuy({
  chain: 'BSC',
  privateKeys: [
    process.env.DEV_KEY!,      // Creator
    process.env.BUYER1_KEY!,   // Buyer 1
    process.env.BUYER2_KEY!,   // Buyer 2
  ],
  buyAmounts: ['0.5', '1.0'],  // Buyer purchase amounts
  tokenInfo: {
    name: 'Revolution Token',
    symbol: 'REV',
    meta: metaCid,
  },
  tokenAddress: predicted.address,
  config: {
    rpcUrl: 'https://bsc-dataseed.binance.org',
  },
});

console.log('✅ Token created successfully!');
console.log('Token address:', result.tokenAddress);
console.log('Bundle status:', result.status);
```

Advantages:

- ✅ Token creation and purchases complete in the same block
- ✅ Prevents bots from sniping immediately after creation
- ✅ Team members fairly get initial tokens
- ✅ If any transaction fails, all rollback (funds are safe)

Bundle Status

```
enum BundleStatus {
  PENDING = 'pending',      // Waiting for execution
  EXECUTED = 'executed',    // Executed (success)
  FAILED = 'failed',        // Execution failed
  TIMEOUT = 'timeout',      // Timed out
```

```
  CANCELLED = 'cancelled',  // Cancelled
}
```

Query Bundle Status:

```
import { Club48Client } from 'four-flap-meme-sdk';

const client = new Club48Client({
  endpoint: 'https://puissant-bsc.48.club'
});

// Wait for bundle completion
const status = await client.waitForBundle(bundleUuid);
console.log('Final status:', status);
```

FAQ

Q: Why does create + buy need address prediction?
A: Because buy transactions need to know the token address, but the token hasn't been created yet. By predicting the address, we can build buy transactions in advance.

Q: What if the predicted address is wrong?
A: Ensure you use the correct `nonce` and parameters. If the nonce changes after prediction (someone else creates a token), the address will be different and the bundle will fail.

Q: Will funds be refunded if the bundle fails?
A: Yes! Bundles are atomic operations - if any transaction fails, all transactions rollback and funds are automatically returned.

Q: Can bundles be used on other chains?
A: Currently only BSC is supported. 48.club may support more chains in the future.

Q: Do bundles require 48SP?
A: Yes, bundle features require a 48 SoulPoint private key.

Low-Level 48.club API

For advanced users who need direct access to 48.club services.

Club48Client Class

```
import { Club48Client } from 'four-flap-meme-sdk';

const client = new Club48Client({
  endpoint: 'https://puissant-bsc.48.club',           // Optional
```

```
  explorerEndpoint: 'https://puissant-bsc-tx.48.club' // Optional
});
```

Send Single Private Transaction

```
// Without 48SP (regular private transaction)
const txHash = await client.sendPrivateTransaction(signedRawTx);

// With 48SP (premium, higher priority)
const txHash = await client.sendPrivateTransactionWith48SP(
  signedRawTx,       // Hex-encoded signed transaction
  spPrivateKey       // 48 SoulPoint private key
);

console.log('Private tx submitted:', txHash);
```

Send Batch Private Transactions

```
import { sendBatchPrivateTransactions } from 'four-flap-meme-sdk';

// Requires 48SP for batch submission
const success = await sendBatchPrivateTransactions(
  [signedTx1, signedTx2, signedTx3],  // Array of signed transactions
  spPrivateKey,                       // 48 SoulPoint private key
  'https://puissant-bsc.48.club'      // Optional endpoint
);

console.log('Batch submitted:', success);
```

Get Minimum Gas Price

```
const gasPrice = await client.getMinGasPrice();
console.log('Minimum gas price:', gasPrice);
```

Use this to ensure your transaction meets the minimum gas price requirement.

Sign 48 SoulPoint

```
import { Club48Client } from 'four-flap-meme-sdk';
```

```
// Sign transactions with 48SP signature
const signature = Club48Client.sign48SoulPoint(
  spPrivateKey,              // 48 SoulPoint private key
  [signedTx1, signedTx2]     // Array of signed transactions
);

console.log('48SP Signature:', signature);
```

Backrun Helper

Execute transactions immediately after a target transaction.

```
import { Club48Client, sendBackrunBundle } from 'four-flap-meme-sdk';

const client = new Club48Client();

const bundleUuid = await sendBackrunBundle(client, {
  backrunTarget: '0xTARGET_TX_HASH',          // Transaction to backrun
  txs: [signedTx1, signedTx2],                // Your transactions
  soulPointSignature: Club48Client.sign48SoulPoint(spPrivateKey,
[signedTx1, signedTx2]),
  maxTimestamp: Math.floor(Date.now() / 1000) + 300  // Deadline (5
minutes)
});

console.log('Backrun bundle UUID:', bundleUuid);

// Wait for bundle result
const status = await client.waitForBundle(bundleUuid);
console.log('Bundle status:', status);
```

Use Cases:

- Execute immediately after a token launch
- React to specific on-chain events
- Guaranteed ordering after target transaction

Permit Piggyback (Auto) – token name resolution

`buildPermitPiggybackAuto` now supports resolving token name from chain, or overriding it.

```
import { buildPermitPiggybackAuto } from 'four-flap-meme-sdk';

const permitData = await buildPermitPiggybackAuto(
  'BSC',
  privateKey,
  tokenAddress,
```

```
  value,
  deadline,
  nonce,
  { rpcUrl, tokenNameOverride: 'MyToken' } // optional
);
```

Appendix

Constants and Addresses

The SDK provides pre-configured addresses for all supported networks:

```
import { CHAIN } from 'four-flap-meme-sdk';

// Use chain enum when calling SDK methods (addresses are handled
internally)
// Example:
// const tm2 = TM2.connectByChain('BSC', rpcUrl);
// const portal = new FlapPortal({ chain: 'BSC', rpcUrl });
// await tryBuy('BSC', rpcUrl, token, amount, funds);

// Chain information
console.log(CHAIN.BSC.chainId);         // 56
console.log(CHAIN.BASE.chainId);        // 8453
console.log(CHAIN.XLAYER.chainId);      // 196
console.log(CHAIN.MORPH.chainId);       // 2818
console.log(CHAIN.ARBITRUM_ONE.chainId); // 42161
```

BigInt and Units

Understanding BigInt

All amounts are in wei (1 ETH = 10^18 wei). Use `bigint` type with `n` suffix.

```
// 1 BNB/ETH in wei
const oneEth = 1n * 10n ** 18n;

// 0.5 BNB/ETH in wei
const halfEth = 5n * 10n ** 17n;

// Convert using ethers.js
import { formatEther, parseEther } from 'ethers';
const readable = formatEther(1000000000000000000n);  // '1.0'
const wei = parseEther('1.0');                        //
1000000000000000000n
```

Frequently Asked Questions

Q: What's the difference between four.meme and Flap Protocol?

A: Both are token launchpads, but:

- **four.meme**: Simpler, BSC-focused, REST API + contracts
- **Flap Protocol**: Advanced CDPV2 curve, multi-chain, more customization

Q: Can I buy my own tokens when creating (preSale)?

A: Yes! Use the `preSale` parameter:

```
const result = await createTokenFlow({
  // ... other params
  payload: {
    name: 'My Token',
    shortName: 'MTK',
    // ... other fields
    preSale: '0.5',  // ✅ Buy 0.5 BNB worth of tokens when creating
  }
});
```

How it works:

1. You set `preSale: '0.5'`
2. SDK automatically sends 0.5 BNB with the creation transaction
3. After contract deploys the token, it immediately buys from the bonding curve using this 0.5 BNB
4. Purchased tokens are automatically sent to your address

Common use cases:

- Prevent sniping (you get initial supply first)
- Test token trading functionality
- Hold initial tokens as project creator

Note: Ensure your wallet has enough BNB (for gas fee + preSale amount)

Q: Why do I need two transactions to sell?

A: ERC20 security. First `approve()`, then `sell()`. Use `ensureSellApproval()` to handle both automatically.

Q: What's a good slippage tolerance?

A: 1-5% for normal markets. Higher (10-20%) for volatile tokens.

Q: Can I trade after DEX migration?

A: No. Once migrated, trade on the DEX (PancakeSwap, Uniswap). The bonding curve closes.

Q: I get "More BNB" error when buying

A: Use `estimate.amountFunds` (not `estimatedCost`) when calling `tradeBuy()`.

Q: Transaction fails with no error when selling

A: You forgot approval. Always call `ensureSellApproval()` before `tradeSell()`.

License & Support

- **License**: MIT
- **GitHub**: Report issues or contribute
- **Documentation**: This README

🎉 **You're now ready to build on four.meme and Flap Protocol!**
