SomeSwap Automated Market Maker Specification

Something Labs

October 27, 2025

Contents

Chapter 1

Protocol Overview

SomeSwap is a neutral two-asset AMM that complements SmthCurve by providing the secondary market once the bonding-curve inventory is depleted. The protocol focuses on:

- deterministic CREATE2 deployment of unordered token pairs;
- optional permissioning of fee presets via whitelist control;
- dynamic fee adjustments based on swap activity;
- fee-on-transfer compatibility and permanent LP locking;
- a reward-aware LP token with delegated accrual.

1.1 Core Components

- **SomeFactory**: deploys pairs, manages the registry, and owns dynamic-fee toggles.
- **SomePair**: constant-product market maker with direction-aware fee splitting.
- **SomeRouter**: user entry point for adding/removing liquidity and swapping.
- **SomeLpToken**: ERC20 + Permit LP token that accrues dual-asset rewards.
- **SomeLiquidityLocker**: escrow contract for time-based or permanent LP locks.
- **SomePPAccessRegistry**: permission layer for non-public fee presets.
- **SomeFeeController**: dynamic-fee controller implementing an S-curve model.

Chapter 2

Permissioned Pools

2.1 Fee Presets and Whitelisting

The registry seeds a set of publicly available fee presets. When `SomeFactory.createPair` is called:

- the factory computes `feeConfigId` from `feeParams.baseFeeParams`;
- if the ID is not marked *permissionless*, `hasAccess[msg.sender]` must be true;
- access can be granted by the factory owner through `grantAccess`.

2.2 Pair Registry

- Pairs are keyed by $(token0, token1, feeId)$ where $token0 < token1$.
- CREATE2 ensures deterministic addresses; both orientations are recorded for lookup convenience.

Chapter 3

Fee Mechanics

3.1 Directional Fee Split

For a swap from token0 to token1 the total fee (base + dynamic) is split using weights $w_{0,in}$ (input-side) and $(1 - w_{0,in})$ (output-side). The reverse direction uses $w_{1,in}$.

Protocol and LP shares are computed as

$$fee_{proto} = fee_{tot} \cdot \frac{bps_{proto}}{10{,}000}, \qquad fee_{lp} = fee_{tot} - fee_{proto}.$$

Protocol fees are forwarded to the treasury, while LP fees are accrued on `SomeLpToken.accrue0/1`.

Chapter 4

Dynamic Fee Model

4.1 Impulse and Activity

Given swap input *amountIn* and reserves (R_0, R_1), the impulse is

$$I = b_1 \cdot \frac{amountIn}{\min(R_0, R_1)} + b_2.$$

Activity decays over time using the configurable half-life $T_{1/2}$:

$$A_t = \frac{A_{t-1}}{1 + \ln 2 \cdot \frac{\Delta t}{T_{1/2}}} + I.$$

4.2 S-Curve Fee

Let $x = \max(0, A_t - a_0)$. The dynamic add-on is

$$fee_{dyn} = cap \cdot \frac{x^2}{x^2 + K^2},$$

bounded so that $fee_{base} + fee_{dyn} < 10{,}000$ bps.

Note

Dynamic fees can be enabled only after a valid configuration is set. The factory owner controls set/enable/disable operations.

Chapter 5

Router Mechanics

5.1 Adding Liquidity

- **ERC20/ERC20**: `addLiquidity` maps inputs into token0/token1 order and calls `SomePair.mintLiquidity`.
- **Fee-on-transfer**: `addLiquiditySupportingFeeOnTransferTokens` measures actual balances to support taxed tokens.
- **ETH routes**: `addLiquidityETH` wraps ETH into WETH; the supporting variant tolerates FoT tokens.
- First liquidity add mints `MINIMUM_LIQUIDITY` to the pair and deposits it permanently into the locker with the pair as beneficiary.

5.2 Removing Liquidity

- `removeLiquidity` burns LP tokens and returns assets in the caller's orientation.
- Permit variants handle EIP-2612 signatures for LP tokens.
- ETH variants withdraw WETH and forward native ETH to the recipient.

5.3 Swaps

Strict-path swaps verify each hop against expected reserves. Supporting variants enforce only the final `amountOutMin`, thus accommodating FoT tokens at the expense of intermediate guarantees.

Chapter 6

LP Token and Locker

6.1 SomeLpToken

- ERC20 with Permit support and owner-only mint/burn.
- Tracks reward indices `feePerToken0` and `feePerToken1` with 1e18 precision.
- Supports delegated accrual: lockers can attribute locked LP to beneficiaries.
- Excluded addresses neither earn rewards nor count towards effective supply.

6.2 SomeLiquidityLocker

- Maintains per-user locks (amount, unlock time, permanent flag).
- Enforces permanent locks for excluded beneficiaries (e.g., the pair itself).
- Provides `deposit`, `depositFor`, `withdraw`, and convenience helpers.

Chapter 7

Smart-Contract Interfaces

7.1 Factory

Listing 7.1: ISomeFactory excerpt

```
interface ISomeFactory {
    function WETH() external view returns (address);
    function treasury() external view returns (address);
    function locker() external view returns (address);
    function allPairs(uint256) external view returns (address);
    function allPairsLength() external view returns (uint256);
    function getPair(address tokenA, address tokenB, bytes32 feeId) external view returns (address);

    function createPair(
        address tokenA,
        address tokenB,
        IFeeController.FeeParams calldata feeParams
    ) external returns (address pair);

    function setTreasury(address) external;
    function setLocker(address) external;
    function setFeeManager(address) external;
    function grantAccess(address) external;

    function setDynamicFeeConfig(address pair, IFeeController.DynamicFeeConfig calldata cfg) external;
    function enableDynamicFee(address pair) external;
    function disableDynamicFee(address pair) external;
}
```

7.2 Pair

Listing 7.2: ISomePair excerpt

```
interface ISomePair is IFeeController {
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function lpToken() external view returns (address);
    function feeId() external view returns (bytes32);
    function getReserves() external view returns (uint112,
       uint112, uint32);

    function initialize(address token0, address token1,
       FeeParams calldata feeParams) external;
    function mintLiquidity(address to) external returns
       (uint256 liquidity);
    function burnLiquidity(address to) external returns
       (uint256 amount0, uint256 amount1);
    function swapExact(address to, bool inputIsToken0)
       external returns (uint256 amountOutUser);

    function setDelegatorOnLp(address locker, bool allowed)
       external;

    function getAmountOut1For0In(uint256 amount0In)
       external view returns (uint256);
    function getAmountOut0For1In(uint256 amount1In)
       external view returns (uint256);
    function getAmountIn0ForExact1(uint256 out1Net)
       external view returns (uint256);
    function getAmountIn1ForExact0(uint256 out0Net)
       external view returns (uint256);
}
```

7.3 Router

Listing 7.3: ISomeRouter excerpt

```
interface ISomeRouter {
    struct AddLiquidityParams {
        address tokenA;
        address tokenB;
        uint256 amountADesired;
        uint256 amountBDesired;
        uint256 amountAMin;
        uint256 amountBMin;
        address to;
        uint256 deadline;
        IFeeController.FeeParams feeParams;
    }

    struct RemoveCtx {
        address tokenA;
        address tokenB;
        bytes32 feeId;
        uint256 liquidity;
        uint256 amountAMin;
        uint256 amountBMin;
        address to;
        uint256 deadline;
    }

    function addLiquidity(AddLiquidityParams calldata
        params) external returns (uint256 amountA, uint256
        amountB, uint256 liquidity);
    function
        addLiquiditySupportingFeeOnTransferTokens(AddLiquidityParams
        calldata params) external returns (uint256 amountA,
        uint256 amountB, uint256 liquidity);
    function addLiquidityETH(AddLiquidityEthParams calldata
        params) external payable returns (uint256
        amountToken, uint256 amountETH, uint256 liquidity);

    function removeLiquidity(RemoveCtx calldata ctx)
        external returns (uint256 amountA, uint256 amountB);
    function removeLiquidityETH(RemoveCtx calldata ctx)
        external returns (uint256 amountToken, uint256
        amountETH);

    function swapExactTokensForTokens(uint256 amountIn,
        uint256 amountOutMin, address[] calldata path,
        bytes32[] calldata feeIds, address to, uint256
        deadline) external returns (uint256[] memory
        amounts);
    function
        swapExactTokensForTokensSupportingFeeOnTransferTokens(uint256
        amountIn, uint256 amountOutMin, address[] calldata
        path, bytes32[] calldata feeIds, address to, uint256
        deadline) external;
}
```

7.4 Liquidity Locker

Listing 7.4: ISomeLiquidityLocker excerpt

```
interface ISomeLiquidityLocker {
    struct LockInfo {
        uint256 amount;
        uint256 unlockTime;
        bool permanent;
    }

    function deposit(address lp, uint256 amount, uint256
        unlockTime, bool permanent) external;
    function depositFor(address lp, address beneficiary,
        uint256 amount, uint256 unlockTime, bool permanent)
        external;
    function withdraw(address lp) external;
    function position(address lp, address user) external
        view returns (LockInfo memory);
}
```

Chapter 8

Security Considerations

- All external entry points in `SomePair` and `SomeRouter` are non-reentrant.
- Treasury should be capable of receiving arbitrary ERC20 assets; otherwise protocol-fee transfers can revert and block swaps.
- Permissioned access lists must be maintained carefully because the base implementation lacks revocation.
- Dynamic fee parameters should be tuned and monitored to avoid extreme behaviour (e.g., cap saturation).