SomeSwap
1.1 Release Edition

Something Labs

November 15, 2025

Contents

Foreword: Changes Since archive/SomeSwap.tex **3**

1 Architecture Overview **4**
1.1 Moving Pieces 4
1.2 Repository Guide 4

2 Factory and Registry **5**
2.1 Pair Creation 5
2.2 Fee Presets 5

3 Dynamic Fee Controller **6**
3.1 Reserve-Coupled Impulse 6
3.2 State Machine 6

4 Router Mechanics **7**
4.1 Strict vs. Supporting Paths 7
4.2 Liquidity Provision 7

5 SomeLpToken **8**
5.1 Reward Accounting 8
5.2 Permit + Delegation 8

6 Liquidity Locker **9**
6.1 Lock Model 9

7 Permissioned Launchpads **10**
7.1 SomePPAccessRegistry 10

8 Partner Integration Playbook **11**
8.1 Stakeholders and Contacts 11
8.2 Prerequisites for Launchpads 11
8.3 Onboarding Steps 12
8.4 Launch and Migration Checklist 12
8.5 Ongoing Responsibilities 12
8.6 Integrator Checklist (Technical Artifacts) 13

8.7 Launchpad Authorization and Signing Requirements 13

9 Interface Snapshot **16**
9.1 ISomeLiquidityLocker 16
9.2 ISomeLpToken 17
9.3 ISomePair 17
9.4 ISomeFactory 18
9.5 ISomeRouter 19

(*) Unless stated otherwise, references to "ETH" denote the native gas token of the active EVM network (e.g., ETH on Mainnet, MON on Monad).

Foreword: Changes Since `archive/SomeSwap.tex`

- **Reserve-aware dynamic fee.** The impulse coefficients (b_1, b_2) in `DynamicFee.sol` now scale with live reserves instead of default constants, removing stale-parameter griefing vectors.
- **Donation tolerance.** Strict swap paths reject front-run donations on intermediate hops; new precision tests codify the acceptance criteria for input vs. output donations.
- **Permissioned launchpad flow.** `SomePPAccessRegistry` signs pair-creator requests so launchpads can prevent pool frontrunning.
- **Liquidity locker parity.** The locker now exposes explicit `permanent` flags in public methods and the Router honors them when seeding SmthCurve migrations.

Chapter 1

Architecture Overview

1.1 Moving Pieces

- **SomeFactory**: deploys `SomePair` contracts via CREATE2, stores fee presets, tracks pair metadata, and wires treasury/locker addresses.
- **SomeRouter**: canonical swap/liquidity interface; contains strict-path, supporting-fee-on-transfer, and ETH-aware variants.
- **SomePair**: constant-product AMM with directional fee split, donation-resistant reserve accounting, and hooks into the dynamic fee controller.
- **SomeLpToken**: ERC20 + Permit LP token that tracks per-token fee indices for dual-asset rewards.
- **SomeLiquidityLocker**: escrow for LP tokens with arbitrary unlock timestamps or permanent locks; integrates with SmthCurve migrations.
- **Abstract__SomeFeeController + DynamicFee**: pluggable fee controllers keyed by `feeConfigId`; the canonical implementation is a reserve-relative S-curve.
- **SomePPAccessRegistry**: optional permission proving system for restricted fee IDs and pair creators.

1.2 Repository Guide

Tests live under `contracts/tests` with both "full_coverage" Foundry suites and "solidity_tests" harnesses. Scripts such as `InitReserves.s.sol` mirror the math shown in SmthCurve docs and are referenced whenever a migration occurs.

Chapter 2

Factory and Registry

2.1 Pair Creation

1. Caller submits $(tokenA, tokenB, feeParams)$ plus optional permissions (pair creator proof, quote asset proof) checked by `SomePPAccessRegistry`.
2. Tokens are ordered to $(token0, token1)$ and the fee config ID is derived via `SomeSwapHelpers.computeFeeConfigId`.
3. CREATE2 deploys the pair and LP token; addresses are cached under $(token0, token1, feeId)$.
4. The first LP tokens (MINIMUM_LIQUIDITY) are deposited permanently into `SomeLiquidityLocker` to enforce forever-locked seed liquidity.

2.2 Fee Presets

Factory stores:

- base fee in bps,
- protocol vs. LP split,
- dynamic fee configuration (caps, impulse multipliers, decay half-lives).

Each preset can be marked *permissioned* requiring `hasAccess[msg.sender]` or an access proof.

Chapter 3

Dynamic Fee Controller

3.1 Reserve-Coupled Impulse

Reserve-aware dynamic fee: the updated impulse applies:

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

where b_1 and b_2 are functions of the current reserves (see `DynamicFee.sol`) rather than global constants. This parallels the audit request to let the activity threshold adapt to pool depth.

3.2 State Machine

- Activity decays exponentially between swaps.
- The controller clamps the dynamic add-on so that `base + dynamic < 10_000 bps` and the dynamic component never exceeds 50% of the configured base fee.
- Fee settings are updated atomically from the factory; pairs only read from immutable pool storage to avoid cross-contract reentrancy.

Chapter 4

Router Mechanics

4.1 Strict vs. Supporting Paths

Donation tolerance checks keep strict routes aligned with the audit-fixes behavior.

- **Strict paths** (`swapExactTokensForTokens`) assert that each hop returns *at least* the pre-quoted amount. Front-ran donations on either the input or output token will now revert because the equality check compares reserves before/after each hop.
- **Supporting FoT paths** skip intermediate checks and only enforce the final `amountOutMin`, making them suitable for taxed tokens at the cost of lower MEV protection.

Precision tests in `SomeRouterPrecision.t.sol` lock down the behavior for donation scenarios referenced in the audit report.

4.2 Liquidity Provision

- ERC20/ERC20 and ETH/ERC20 variants exist, each with a supporting-FoT counterpart.
- Router instructs the locker to deposit LP tokens on behalf of the pair (permanent) or the end-user (time delayed) depending on context.
- Permit-enabled remove liquidity functions let UIs streamline UX without raw approvals.

Chapter 5

SomeLpToken

5.1 Reward Accounting

- Two accumulators (`feePerToken0/1`) store fees per unit of LP supply using 1e18 precision.
- Accounts store checkpoints so that claiming rewards only costs $O(1)$ regardless of history.
- Authorized contracts (pairs, lockers) can mark addresses as *excluded* to block fee accrual (e.g., permanently locked seed liquidity).

5.2 Permit + Delegation

LP token implements EIP-2612 permit and allows lockers to account for delegated balances when calculating claimable fees, aligning with launchpad locking requirements.

Chapter 6

Liquidity Locker

Liquidity locker parity keeps router/locker semantics synced with the audit-fixes contract changes (explicit permanent locks, delegated balances).

6.1 Lock Model

- Locks store `amount`, `unlockTime`, `permanent`.
- `deposit` and `depositFor` accept the permanent flag directly (a change from the archive doc where it was implied).
- Withdrawals require either an expired timestamp or a non-permanent flag; permanent locks can only be released by governance hooks (unused by default).

Chapter 7

Permissioned Launchpads

Permissioned launchpad flow is enforced via `SomePPAccessRegistry`.

7.1 SomePPAccessRegistry

- Maintains per-authority configs describing whether first/all pairs are restricted, expected template hashes, and the trusted signer.
- `PairCreatorAuthorization` carries (authority, token, caller, creator, nonces, deadlines) and must be provided for both the token creator and the quote asset when a restriction is active.
- SmthTokenFactory wires these proofs during migration so that the deployer of the bonding-curve token cannot be front-run on SomeSwap.

Chapter 8

Partner Integration Playbook

Expanded documentation scope now includes a practical guide for integrators.

8.1 Stakeholders and Contacts

- **Protocol integrator**: launchpad, wallet, or project supplying liquidity and requesting a dedicated quote asset or fee preset.
- **Something Labs integration manager**: coordinates access registry onboarding, documentation, and review of smart-contract changes.
- **Security liaison**: ensures external auditors (or partner teams) align with SomeSwap's audit-fixes expectations.

Partners should reach out via the designated integration manager (Discord/Telegram/email per engagement) and provide technical PoCs plus a mainnet test plan before deployment.

8.2 Prerequisites for Launchpads

1. **Asset Standards**: Quote assets must be ERC20-compliant (proper `totalSupply`, `balanceOf`, `allowance`, non-reverting transfers).
2. **Security Artifacts**: Share audit reports or internal reviews for custom routers, lockers, or tokens.

8.3 Onboarding Steps

Partner Side

1. Submit an **integration dossier** describing the quote asset, fee preferences, desired dynamic fee profile, and whether pools must be permissioned.
2. Provide **SomePPAccessRegistry** signer details (acceptable deadlines, per-creator limits). For launchpads, this usually means deriving a dedicated EOA or contract wallet to sign pair-creation proofs.
3. Coordinate dry runs on a fork (or testnet) showing token launch, swap activity, and migration into SomeSwap.

SomeSwap Side

1. Approve the partner's quote asset via **SmthCurveConfig.approveAsset** and configure minimum initial quote thresholds to match the partner's needs.
2. Register partner-specific fee presets in **SomeFactory** if new dynamic fee curves are required. Mark them permissioned until the partner's access proofs are active.
3. Configure **SomePPAccessRegistry** entries describing which authorities can create pairs, whether first/all pairs are restricted, and which template hashes are accepted.

8.4 Launch and Migration Checklist

- **Before Launch**: verify whitelists, fee splits, locker setup.
- **During Launch**: monitor emitted `UseConfig` and `SmthTokenFactory__TokenLaunch` events. Partners should validate that creator/pair addresses match their expected CREATE2 derivations.
- **Migration**: when `rTokenReserves == 0`, the migration authority supplies both token and quote proofs to SomeSwap. Ensure the locker receives LP tokens (permanent for seed liquidity) and that fee claimers are registered.
- **Post-Migration**: partners confirm router swaps, LP deposits, and reward accrual operate with their quote asset.

8.5 Ongoing Responsibilities

- **Upgrades**: when requesting fee or config changes, submit change windows and regression tests. Something Labs will update presets and access rules accordingly.

8.6 Integrator Checklist (Technical Artifacts)

- **Quote Asset Details**: contract address, decimals, and min launch liquidity.
- **Access Registry Inputs**: list of authorized pair creators, public keys of signing authorities, expected template hashes for token deployments, and desired proof expiration windows.
- **Fee Preset Requirements**: preferred base fee, protocol/LP split, dynamic fee controller parameters (caps, decay, impulse multipliers), and whether the preset must remain permissioned.
- **Locker/LP Workflow**: addresses that should receive permanent locks, cliff schedules for user deposits, and any custom locker contracts or adapters involved.
- **Migration Expectations**: target timelines, migration authority address (if different from creator), and configuration for fee claimers (`feeClaimer1/2`) post migration.
- **Operational Contacts**: on-call email/handles for engineering and security leads, escalation path, and notification channels for incident response.

8.7 Launchpad Authorization and Signing Requirements

Manager Checklist

1. **Configure authority**: call `configureAuthority(authority, AuthorityConfigInput)` with the launchpad's parameters:
   - `restrictFirstPair` / `restrictAllPairs`: set to `true` when every pair must be permissioned until unlock.
   - `unlockDelay`: seconds before restrictions expire automatically (0 = never auto-unlock).
   - `isPairCreatorAuthority`: usually `true`; enables CREATE2/pair-creator validation.
   - `tokenTemplateHash`: `keccak256(init code)` of the launchpad's pair-creator contract.
   - `deployer`: CREATE2 deployer contract that spawns pair-creator tokens.
   - `saltTypeHash`: optional custom salt domain. Use `DEFAULT_PAIR_CREATOR_SALT_TYPEHASH` if not overriding.
   - `trustedSigner`: *EOA* enforced by SomeSwap. This wallet must sign every migration authorization.

2. **Allow callers**: via `setAuthorityCallers(authority, callers, allowed)`, whitelist contracts that will present signatures (SmthTokenFactory, migration helpers, QA routers).

3. **Register tokens**: for each SmthCurve deployment that should remain permissioned, call `registerPermissionedToken(PermissionedTokenInput)` with the token, authority, and (if required) CREATE2 salt + deployer metadata.

EOA Signature Payload

Launchpads must sign a `PairCreatorAuthorization` for both the token leg and quote leg (if permissioned):

```
struct PairCreatorAuthorization {
    address authority;   // Launchpad authority address
    address token;       // ERC20 being migrated
    address caller;      // Contract executing migration
    address creator;     // Original token creator recorded
        by the launchpad
    uint256 creatorNonce;// Nonce tracked by
        SmthTokenFactory
    uint256 actionNonce; // Launchpad-maintained nonce
    uint256 deadline;    // Expiry timestamp
    bytes signature;     // EIP-712 signature from
        trustedSigner
}
```

Parameter guidance:

- **creatorNonce**: read from contract (per creator) to keep CREATE2 predictions stable.
- **actionNonce**: launchpad increments this every time it signs; SomeSwap rejects reused nonces.
- **signature**: the EOA defined in `trustedSigner` signs the EIP-712 digest defined in `SomePPAccessRegistry`; HSMs or multisig wallets can custody this key.

Launchpad Implementation Notes

- Maintain an on-chain helper or off-chain service that exposes the current `actionNonce` per `creator`. Something Labs queries it before constructing payloads.
- Validate every signature request against your internal allowlist of creators/projects before signing.

Example Migration Flow

1. Integration manager configures the authority and callers; the launchpad shares the signer EOA.
2. A creator launches via SmthCurve using the partner's quote asset; manager registers the token as permissioned.
3. Once `rTokenReserves == 0`, SmthTokenFactory prepares `PairCreatorAuthorization` structs (token + quote) and requests signatures from the launchpad service.
4. The launchpad signs (using the configured EOA), increments `actionNonce`, and returns the payloads.
5. SmthTokenFactory calls `finalizeAndMigrate` with both signed authorizations; SomeSwap validates them and creates the pair without exposing the launch to frontrunning.

Chapter 9

Interface Snapshot

9.1 ISomeLiquidityLocker

Listing 9.1: Locker API

```
interface ISomeLiquidityLocker {
    struct LockInfo {
        uint256 amount;
        uint256 unlockTime;
        uint256 permanentAmount;
    }

    function setFactory(address factory_) external;

    function deposit(
        address pairAddr,
        uint256 amount,
        uint256 unlockTime,
        bool permanent
    ) external;

    function depositPermanent(address pairAddr, uint256 amount)
        external;

    function depositFor(
        address pairAddr,
        address beneficiary,
        uint256 amount,
        uint256 unlockTime,
        bool permanent
    ) external;

    function withdraw(address pairAddr, uint256 amount) external;
    function extendLock(address pairAddr, uint256 newUnlockTime)
        external;

    function getLock(
        address pairAddr,
        address user
```

```
    )
        external
        view
        returns (uint256 amount, uint256 unlockTime, bool
            permanent);
}
```

9.2 ISomeLpToken

Listing 9.2: Reward-aware LP token

```
interface ISomeLpToken is IERC20 {
    function pair() external view returns (address);
    function rewardToken0() external view returns (address);
    function rewardToken1() external view returns (address);

    function feePerToken0() external view returns (uint256);
    function feePerToken1() external view returns (uint256);
    function pending0() external view returns (uint256);
    function pending1() external view returns (uint256);

    function userPaid0(address user) external view returns
        (uint256);
    function userPaid1(address user) external view returns
        (uint256);
    function rewards0(address user) external view returns
        (uint256);
    function rewards1(address user) external view returns
        (uint256);

    function isDelegator(address addr) external view returns
        (bool);
    function setDelegator(address locker, bool allowed) external;
    function increaseDelegated(address beneficiary, uint256
        amount) external;
    function decreaseDelegated(address beneficiary, uint256
        amount) external;
    function setExcluded(address addr, bool excluded_) external;

    function mint(address to, uint256 amount) external;
    function burn(address from, uint256 amount) external;
    function accrue0(uint256 amount0) external;
    function accrue1(uint256 amount1) external;

    function claim() external;
    function claim0() external;
    function claim1() external;
}
```

9.3 ISomePair

Listing 9.3: Neutral AMM pair

```
interface ISomePair {
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function feeId() external view returns (bytes32);
    function lpToken() external view returns (address);

    function getReserves()
        external
        view
        returns (uint112 reserve0, uint112 reserve1, uint32
            blockTimestampLast);

    function initialize(
        address _token0,
        address _token1,
        IFeeController.FeeParams calldata _feeParams
    ) external;

    function setDelegatorOnLp(address locker, bool allowed)
        external;

    function getAmountOut1For0In(uint256 amount0In)
        external
        view
        returns (uint256 net1Out);

    function getAmountIn0ForExact1(uint256 out1Net)
        external
        view
        returns (uint256 in0Gross);

    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 skim(address to) external;
    function sync() external;
}
```

9.4 ISomeFactory

Listing 9.4: Factory surface

```
interface ISomeFactory {
    function WETH() external view returns (address);
    function locker() external view returns (address);
```

```
4       function treasury() external view returns (address);
5       function authorityAdmin() external view returns (address);
6
7       function getPair(
8           address tokenA,
9           address tokenB,
10          bytes32 feeId
11      ) external view returns (address);
12
13      function allPairsLength() external view returns (uint256);
14      function allPairs(uint256 idx) external view returns (address);
15      function isPair(address pair) external view returns (bool);
16
17      function computeFeeConfigId(
18          IFeeController.BaseFee calldata fee
19      ) external pure returns (bytes32);
20
21      function createPair(
22          address tokenA,
23          address tokenB,
24          IFeeController.FeeParams memory feeParams,
25          SomePPAccessRegistry.PairCreatorAuthorization calldata
                authA,
26          SomePPAccessRegistry.PairCreatorAuthorization calldata
                authB
27      ) external returns (address pair);
28
29      function setTreasury(address _treasury) external;
30      function setAuthorityAdmin(address admin) external;
31      function configureAuthority(
32          address authority,
33          SomePPAccessRegistry.AuthorityConfigInput calldata config
34      ) external;
35      function setAuthorityCallers(
36          address authority,
37          address[] calldata callers,
38          bool allowed
39      ) external;
40      function registerPermissionedToken(
41          SomePPAccessRegistry.PermissionedTokenInput calldata input
42      ) external;
43      function clearTokenPermission(address token) external;
44
45      function skim(address pair, address to) external;
46      function sync(address pair) external;
47  }
```

9.5 ISomeRouter

Listing 9.5: Router highlights

```
1   interface ISomeRouter {
2       struct AddLiquidityParams {
3           address tokenA;
4           address tokenB;
```

```
        uint256 amountADesired;
        uint256 amountBDesired;
        uint256 amountAMin;
        uint256 amountBMin;
        address to;
        uint256 deadline;
        IFeeController.FeeParams feeParams;
        SomePPAccessRegistry.PairCreatorAuthorization authA;
        SomePPAccessRegistry.PairCreatorAuthorization authB;
    }

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

    struct AddLiquidityEthParams {
        address token;
        uint256 amountTokenDesired;
        uint256 amountTokenMin;
        uint256 amountETHMin;
        address to;
        uint256 deadline;
        IFeeController.FeeParams feeParams;
        SomePPAccessRegistry.PairCreatorAuthorization authToken;
        SomePPAccessRegistry.PairCreatorAuthorization authWeth;
    }

    function factory() external view returns (address);
    function WETH() external view returns (address);

    function addLiquidity(
        AddLiquidityParams memory params
    ) external returns (uint256 amountA, uint256 amountB, uint256
        liquidity);

    function addLiquidityETH(
        AddLiquidityEthParams memory params
    )
        external
        payable
        returns (uint256 amountToken, uint256 amountETH, uint256
            liquidity);

    function addLiquiditySupportingFeeOnTransferTokens(
        AddLiquidityParams calldata params
    ) external returns (uint256 amountA, uint256 amountB, uint256
        liquidity);

    function removeLiquidity(RemoveCtx memory ctx)
        external
```

```
        returns (uint256 amountA, uint256 amountB);

    function removeLiquiditySupportingFeeOnTransferTokens(
        RemoveCtx calldata ctx
    ) external returns (uint256 amountA, uint256 amountB);

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

    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] memory path,
        bytes32[] memory feeIds,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);
}
```