SmthCurve Bonding Curve Specification
1.1 Release Edition

Something Labs

November 15, 2025

Contents

(*) 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: Delta vs. Archive Edition

This document supersedes `contracts/docs/archive/SmthCurve.tex`. The main updates are:

- **Per-asset quote enablement.** The factory now lets partners launch curves against any ERC20 quote asset whitelisted in `allowedAssets`. The archive version assumed WETH-only issuance.
- **Minimum initial quote enforcement.** A global native threshold and per-asset ERC20 thresholds are enforced at launch to mitigate dust markets.
- **Expanded configuration surface.** Treasury, referral, and migration wallets can be updated live, and referral/treasury splits are programmable through `setTradeFeeSplit`.
- **Permissioned pair creators.** The launch path integrates with `SomePPAccessRegistry` proofs, enabling launchpads and other partners to gate pair creation.
- **Audit-driven fixes.** Reserve accounting, fee routing (treasury/referrer), strict slippage guards, and migration checks were hardened relative to the original document and are reflected here.

The remainder of the document provides a ground-up view of the current implementation that matches the `audit-fixes` branch.

Chapter 1

System Overview

1.1 Actors and Components

- **SmthTokenFactory** (subject of this document) mints the full token supply, runs the bonding curve, and orchestrates migration into SomeSwap pairs.
- **SomeRouter/SomeFactory** provide secondary-market liquidity. The factory locks initial LP via `SomeLiquidityLocker`.
- **Partners / Integrators** may request ERC20 quote assets by submitting an `ISmthCurveConfig.AssetConfig`; this enables launchpads to denominate sales in their native stablecoins or wrappers.
- **Migration Authority** is allowed to trigger finalization once the curve completes; proofs from `SomePPAccessRegistry` ensure only approved creators spawn pairs.

1.2 Lifecycle Summary

1. **Configuration**: the owner whitelists quote assets, sets fee splits, and assigns treasury/referral/migration wallets.
2. **Launch**: creators call `launchToken` (preset fees) or `launchTokenWithCustomFees`.

   They supply a target gross quote raise (`initialAmmQuoteAmount`), the AMM split ratio, and the desired quote asset.
3. **Trading**: buys/sells follow a constant-product curve with virtual reserves (v_S, v_T) and real reserves (r_S, r_T). Fees are split between the treasury and referral wallet; net proceeds either accumulate as quote or reduce inventory.

4. **Migration**: once the curve inventory empties, the migration authority seeds a SomeSwap pair. Liquidity is locked to `SomeLiquidityLocker` and fee claimers are recorded.

5. **Post-migration**: users interact with SomeSwap pairs directly; residual quote reserves (including migration fees) can be claimed by the designated wallets.

1.3 State Anatomy

Each launched token is tracked inside `SmthTokenFactory.TokenInfo`. Key fields:

- Metadata (creator, token address, symbol, URI), deterministic salt, fee preset/custom params.
- Virtual and real reserves, AMM seed allocation, migration fee escrow, and flags describing completion/migration status.
- Fee claimers, locker initialization flag, and the asset config snapshot so that historical migration/trade fee parameters remain immutable even if the whitelist changes.

Chapter 2

Configuration Layer

2.1 Asset Whitelist

Per-asset quote enablement: `SmthCurveConfig.approveAsset` stores `AssetConfig`:

- `approved / frozen` toggles.
- `migrationFeeNumerator` (m in WAD) and `tradeFeeBpsNumerator`.
- `minInitialQuoteAmount` – the per-asset floor enforced during launch.

Frozen assets fall back to WETH at launch time, allowing the owner to pause a quote asset without blocking creators.

2.2 Minimum Initial Quote

Minimum initial quote enforcement:

- **Native ETH**: `minInitialQuoteNative()` is globally enforced whenever `quoteAsset` equals WETH. The default is 0.1 ETH and can be updated via `setMinInitialQuoteNative`.
- **ERC20 quote**: `AssetConfig.minInitialQuoteAmount` enforces a floor that integrators can tune per stablecoin or partner asset.

Both checks revert with `SmthTokenFactory__InitialQuoteBelowMinimum` to surface the required threshold to frontends.

2.3 Fee Splits and Wallets

Programmable treasury/referral splits:

- Treasury, referral, migration, and migration authority wallets are mutable but must never be the zero address.

- The trade-fee split uses a denominator/numerator pair; sums must equal the denominator, otherwise `SmthTokenFactory__InvalidTradeFeeBpsNumerator` reverts.
- The migration authority fee is expressed as a WAD fraction of the quote amount moved into the AMM.

Chapter 3

Launch Flow

3.1 Parameters

- **Name/Symbol/URI**: passed through to the deployed token.
- **InitialAmmQuoteAmount**: gross quote expected to be raised before migration; drives the virtual reserve slope.
- **InitialRatio**: BPS denominator describing how much supply seeds the AMM versus the curve.
- **Quote Asset**: either WETH or a partner ERC20. If the chosen asset is frozen, the factory transparently switches to WETH and emits `UseConfig`.
- **Fee Preset**: fixed presets (0.25%, 1%, 3%) or custom IF the owner toggled `allowCustomFees`.

3.2 Validation

1. Checks initialization and pause state.
2. Fetches the asset config, falls back to default if frozen, and emits the config snapshot.
3. Validates trade/migration fee bounds, minimum quote thresholds, and ratio bounds.
4. Builds a `LaunchCache` record used by `_finalizeLaunch`.

Permissioned pair creators leverage `SomePPAccessRegistry` proofs whenever a launchpad restricts who may instantiate the eventual SomeSwap pair.

3.3 ERC20 vs. Native Launches

- When using WETH, creators may optionally attach ETH to immediately buy tokens (the contract wraps it).
- When using ERC20 quote assets, creators transfer the gross amount directly; no ETH may be sent (`SmthTokenFactory__NativeValueNotAllowed`).
- Partners can pre-approve custom fee controllers that live inside SomeSwap. The factory stores the `FeeParams` for migration-time pair initialization.

Chapter 4

Trading Semantics

4.1 Buys

Audit-driven guardrails keep slippage validation and reserve accounting aligned with the post-audit code.

- **Quote handling**: native buys wrap ETH into WETH; ERC20 buys rely on `transferFrom`. Referral wallet receives its fee share immediately, treasury receives the rest.
- **Curve math**: net quote extends v_S; new v_T is computed via floor division to keep dust inside the protocol. Output tokens first consume `rTokenReserves`; if an order exceeds inventory the factory executes the *buy-all-remaining* branch to empty the curve deterministically.
- **Slippage guard**: users specify `minTokensOut`; the factory reverts with `SmthTokenFactory__SlippageExceeded` when violated.

4.2 Sells

- Users must approve the factory (or use permit variants).
- Gross quote is limited by available r_S; running out

  reverts with `SmthTokenFactory__InsufficientFundsInProtocol`.
- Native payouts unwrap WETH, while ERC20 payouts transfer the quote token back.

4.3 Partial Fills

Both buy and sell routines support partial fills to guarantee deterministic execution. When a buyer requests more quote than needed to empty the curve,

the remainder is refunded. When a seller requests more quote than the protocol has, the trade is rejected.

Chapter 5

Migration and Liquidity Locking

Double-proofed migrations via SomePPAccessRegistry ensure only authorized launchpads can seed liquidity.

5.1 Trigger Conditions

Only the migration authority can call `finalizeAndMigrate` and only when:

- `info.rTokenReserves == 0`.
- The bonding curve is marked completed and liquidity has not yet migrated.

The function wires fee claimers, ensures SomeSwap permissions via double proofs (token and quote paths), adds liquidity through `SomeRouter`, and deposits LP tokens into `SomeLiquidityLocker`.

5.2 Fee Accounting

- Migration fee (`migrationFee`) is escrowed inside the factory until after liquidity is added.
- Trade fees are continuously streamed to the treasury/referral wallets; no sweeping step is required.
- Migration authority may take an additional fee (`migrationAuthorityFee`) to compensate orchestrators.

Chapter 6

Multi-Asset Quote Support

Multi-asset quote support lets partners denominate launches in arbitrary ERC20 assets.

6.1 Motivation

Integrators (launchpads, third-party treasuries) often need to denominate the bonding curve in an ERC20 quote asset such as USDC, stables, or wrapped staking tokens. Previously the factory only accepted WETH, which forced additional `router.swapExactETHForTokens` hops and introduced slippage.

6.2 Implementation

- **Whitelist**: Owner calls `approveAsset` with quote asset metadata and minimum launch sizes. The helper enforces light ERC20 compliance by probing `totalSupply`, `balanceOf`, and `allowance`.
- **Per-launch selection**: Creators pass the desired quote asset into `launchToken`. If the asset is frozen, the factory automatically swaps to WETH to avoid bricking the launch transaction.
- **Accounting**: `TokenInfo.quoteIsWeth` drives payout logic. Reserve math is asset-agnostic because it reasons in generic "quote units," so trading functions stay unchanged.

6.3 Partner Workflow

1. Partner requests the owner to approve their ERC20 with custom min-quote and fee parameters.
2. Partner launches tokens pointing at their asset. The curve raises the specified ERC20 directly.

3. During migration the factory treats the ERC20 as the quote leg when calling `SomeRouter.addLiquidity`.

Chapter 7

Security and Testing Notes

7.1 Guards

- Non-reentrancy and pause modifiers protect every state-changing entry point.
- Launch slippage checks ensure misconfigured ratios or minimum quotes fail fast.
- Signature-gated pair creation prevents malicious actors from frontrunning a permissioned launchpad.

7.2 Testing Strategy

The repository now enforces Level 2+ coverage:

- **Functional suites**: `BuyToken.t.sol`, `SellToken.t.sol`, `LaunchToken.t.sol`, `MigrateToken.t.sol`, `SmthCurveConfig.t.sol`.
- **Precision suites**: `SomeRouterPrecision.t.sol` reproduces donation/front-run scenarios from the audit report.
- **Coverage harnesses**: "full_coverage" folders assert every branch, revert, and fee path.

When extending the factory, new invariants must be mirrored in these suites before code merges.

Chapter 8

Interface Snapshot

Listing 8.1: Selected SmthTokenFactory surface

```
function launchToken(
    string calldata name_,
    string calldata symbol_,
    string calldata uri_,
    uint256 initialAmmQuoteAmount_,
    uint256 initialRatio_,
    address quoteAsset_,
    FeePercent feePercent_
)
    external
    payable
    returns (address tokenAddress);

function launchTokenWithCustomFees(
    string calldata name_,
    string calldata symbol_,
    string calldata uri_,
    uint256 initialAmmQuoteAmount_,
    uint256 initialRatio_,
    address quoteAsset_,
    IFeeController.FeeParams calldata feeParams_
) external payable returns (address tokenAddress);

function buyWithNative(
    address token_,
    uint256 minOut,
    uint256 deadline
) external payable returns (uint256 tokensOut);

function buyWithQuote(
    address token_,
    uint256 grossQuote,
```

```
    uint256 minOut,
    uint256 deadline
) external returns (uint256 tokensOut);
```

Listing 8.2: Quote enforcement helpers

```
function setMinInitialQuoteNative(uint256 amount) external
    onlyOwner;
function approveAsset(address asset, AssetConfig calldata
    config) external onlyOwner;
function minInitialQuoteNative() external view returns
    (uint256);
function allowedAssets(address asset) external view returns
    (AssetConfig memory);
```

8.1 ISmthTokenFactory

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;

import "./ISmthTokenErrors.sol";
import {SomePPAccessRegistry} from
    "@something/someswap/contracts/factory/Abstract__SomePPAccessRegistry.sol";
import {IFeeController} from
    "@something/someswap/interfaces/IFeeController.sol";
import {ISmthCurveConfig} from
    "@something/interfaces/ISmthCurveConfig.sol";
import {IERC20} from
    "@openzeppelin/contracts/token/ERC20/IERC20.sol";

/// @title ISmthTokenFactory
/// @notice Interface for the Smth Token Factory: launching
    ERC20-like tokens,
///         tracking bonding-curve style virtual/real reserves,
    handling buys/sells,
///         and migrating liquidity to AMMs.
interface ISmthTokenFactory is ISmthTokenErrors {
    // raw error type wrapper
    struct CustomErrorWrapper {
        bytes payload;
    }

    /// @notice Global configuration used on token launch and fee
        accounting.
    // -------------------- Config --------------------

    /// @notice Token decimals (for display/metadata; the math
        assumes 18 decimals).
    function tokenDecimals() external view returns (uint8);

    /// @notice Whether the config has been initialized.
    function isInitialized() external view returns (uint8);
```

```
28
29      // -------------------- TokenInfo --------------------
30
31      /// @notice Per-token accounting tracked by the factory for
            bonding-curve math and migration.
32      /// @dev vQuoteReserves * vTokenReserves = constant product
            (virtual reserves).
33      struct TokenInfo {
34          /// @notice Original token creator address.
35          address creator;
36          /// @notice Launched token address (ERC20-like).
37          address tokenAddress;
38          address quoteAsset;
39          // Changes from address(0) to the valid address only after
                pair created
40          address pairAddress;
41          address predictedPairAddress;
42          /// @notice Token metadata (not enforced on-chain here).
43          string name;
44          string symbol;
45          string uri;
46          /// @notice Total supply and decimals (decimals kept for
                completeness).
47          uint256 tokenTotalSupply;
48          uint256 tokenDecimals;
49          /// @notice PairCreator nonce & salt used for
                deterministic token deployment.
50          uint256 launchNonce;
51          bytes32 launchSalt;
52          // @notice Pair fee config
53          FeePercent feePercent;
54          bool useCustomPairFee;
55          bytes32 feeConfigId;
56          IFeeController.FeeParams feeParams;
57          /// @notice Virtual QUOTE reserve (vS) used by the
                bonding-curve formulas.
58          uint256 vQuoteReserves;
59          /// @notice Virtual token reserve (vT) used by the
                bonding-curve formulas.
60          uint256 vTokenReserves;
61          /// @notice Real QUOTE reserve (accumulated QUOTE the
                protocol can pay out on sells).
62          uint256 rQuoteReserves;
63          /// @notice Real token reserve (remaining token inventory
                for sales; increases on user sells).
64          uint256 rTokenReserves;
65          /// @notice Initial token reserve in v2 amm protocol.
66          uint256 ammTokenReserves;
67          /// @notice Absolute migration fee kept aside from
                rQUOTEReserves during liquidity migration.
68          uint256 migrationFee;
69          /// @notice True once the bonding-curve is completed
                (e.g., after final mint/migration).
70          bool isCompleted;
71          bool liquidityMigrated;
72          bool liquidityLocked;
73          bool pairCreated;
```

```
        bool quoteIsWeth;
        // --- fee claimers / locker integration ---
        address feeClaimer1;
        address feeClaimer2;
        bool lockerInitialized; // whether initializePairOnce()
            already called for this pair
        ISmthCurveConfig.AssetConfig assetConfig;
    }

    struct MigrateCtx {
        uint256 tokensToPair;
        uint256 requiredQuoteAmount;
        uint256 migrationFeeAmount;
        uint256 availableQuoteReserves;
        uint256 executedTokenAmount;
        uint256 executedQuoteAmount;
        uint256 lpTokensMinted;
        uint256 factoryTokenBalance;
        uint256 routerTokenUsed;
        uint256 routerQuoteUsed;
        uint256 routerLiquidityMinted;
        bytes approveErrorData;
        bytes addLiquidityErrorData;
        bool addLiquiditySuccess;
        bool approveSuccess;
        bytes32 poolFeePresetId;
        IERC20 baseToken;
        address quoteToken;
    }

    struct BuyCtx {
        uint256 quoteNetMax;
        uint256 newS;
        uint256 newT;
        uint256 outAmt;
        uint256 quoteNetNeeded;
        uint256 quoteGrossNeeded;
        uint256 refund;
        uint256 R;
    }

    struct LiquidityExecution {
        uint256 tokenAmount;
        uint256 quoteAmount;
        uint256 lpMinted;
    }

    struct TryAddLiuqidityCtx {
        uint256 t1;
        uint256 t2;
        uint256 e1;
        uint256 e2;
        uint256 t1Min;
        uint256 t2Min;
        uint256 e1Min;
        uint256 e2Min;
        bool ok1;
```

```
        bytes err1;
        uint256 aT1;
        uint256 aE1;
        uint256 L1;
        bool ok2;
        bytes err2;
        uint256 aT2;
        uint256 aE2;
        uint256 L2;
    }

    struct LaunchCache {
        address creator;
        address usedQuoteAsset;
        ISmthCurveConfig.AssetConfig usedConfig;
        uint256 initialAmmQuoteAmount;
        uint256 initialRatio;
        FeePercent feePercent;
        bool useCustomFeeParams;
        IFeeController.FeeParams customFeeParams;
    }

    enum FeePercent {
        P_0_25,
        P_1_0,
        P_3_0,
        CUSTOM
    }

    // -------------------- Events --------------------

    /// @notice Emitted once a token is launched and reserves are
        initialized.
    event SmthTokenFactory__TokenLaunch(
        address indexed token,
        string name,
        string symbol,
        string uri,
        uint256 vReserveQuote,
        uint256 vReserveToken,
        uint256 rReserveQuote,
        uint256 rReserveToken,
        uint256 initialRatio,
        uint256 initialAmmQuoteAmount,
        address indexed creator,
        address quoteAsset,
        FeePercent feePercent
    );

    // remane: Buy
    /// @notice Emitted after a successful buy.
    /// @param token Token address.
    /// @param buyer Buyer address.
    /// @param amount Tokens minted to the buyer.
    /// @param cost QUOTE paid by the buyer (msg.value).
    /// @param vReserveQuote Updated virtual QUOTE reserve.
    /// @param vReserveToken Updated virtual token reserve.
```

```
186     /// @param rReserveQuote Updated real QUOTE reserve.
187     /// @param rReserveToken Updated real token reserve.
188     event SmthTokenFactory__Buy(
189         address indexed token,
190         address indexed buyer,
191         uint256 amount,
192         uint256 cost,
193         uint256 vReserveQuote,
194         uint256 vReserveToken,
195         uint256 rReserveQuote,
196         uint256 rReserveToken
197     );
198
199     // rename: Sell
200     /// @notice Emitted after a successful sell.
201     /// @param token Token address.
202     /// @param seller Seller address.
203     /// @param amount Tokens received from the seller.
204     /// @param refund QUOTE sent to the seller (after fee).
205     /// @param vReserveQuote Updated virtual QUOTE reserve.
206     /// @param vReserveToken Updated virtual token reserve.
207     /// @param rReserveQuote Updated real QUOTE reserve.
208     /// @param rReserveToken Updated real token reserve.
209     event SmthTokenFactory__Sell(
210         address indexed token,
211         address indexed seller,
212         uint256 amount,
213         uint256 refund,
214         uint256 vReserveQuote,
215         uint256 vReserveToken,
216         uint256 rReserveQuote,
217         uint256 rReserveToken
218     );
219
220     /// @notice Emitted whenever the treasury address is updated.
221     event SmthTokenFactory__TreasuryUpdated(address indexed
        treasury);
222     event SmthTokenFactory__ReferralWalletUpdated(address indexed
        referral);
223     event SmthTokenFactory__MigrationWalletUpdated(address indexed
        wallet);
224     event SmthTokenFactory__MigrationAuthorityUpdated(
225         address indexed migrationAuthority
226     );
227     event SmthTokenFactory__MigrationAuthorityFeeUpdated(
228         uint256 migrationAuthorityFee
229     );
230     event SmthTokenFactory__TradeFeeSplitUpdated(
231         uint256 denominator,
232         uint256 treasuryNumerator,
233         uint256 referralNumerator
234     );
235
236     /// @notice Emitted whenever the liquidity locker is updated.
237     event SmthTokenFactory__LiquidityLockerUpdated(address indexed
        locker);
238
```

```
239     /// @notice Emitted whenever the router is updated.
240     event SmthTokenFactory__RouterUpdated(address indexed router);
241
242     /// @notice Emitted after migrating liquidity to an AMM.
243     // migrated
244     event SmthTokenFactory__LiquidityMigrated(
245         address indexed token,
246         address indexed pair,
247         uint256 tokenAmount,
248         uint256 quoteAmount
249     );
250
251     event SmthTokenFactory__CurveCompleted(
252         address indexed token,
253         uint256 vReserveQuote,
254         uint256 vReserveToken,
255         uint256 rReserveQuote,
256         uint256 rReserveToken
257     );
258
259     // RENAME: LP Locked
260     event SmthTokenFactory__LiquidityLocked(
261         address indexed token,
262         address indexed pair,
263         uint256 lockedLiquidity
264     );
265     event SmthTokenFactory__AllowCustomFeesUpdated(bool allowed);
266
267     // -------------------- Views --------------------
268
269     /// @notice Get per-token info by token address.
270     function tokenInfo(address token) external view returns
        (TokenInfo memory);
271
272     /// @notice UniswapV2 router address used for liquidity
        migration.
273     function router() external view returns (address);
274
275     /// @notice UniswapV2 factory address used for pair address
        calc.
276     function factory() external view returns (address);
277
278     /// @notice WETH address for the configured router.
279     function WETH() external view returns (address);
280
281     /// @notice Returns whether custom fee configurations are
        allowed for new launches.
282     function allowCustomFees() external view returns (bool);
283
284     /// @notice Predict deterministic token address for a
        (creator, nonce) pair.
285     function predictPairCreatorToken(
286         address creator,
287         uint256 nonce
288     ) external view returns (address);
289
290     // -------------------- Actions --------------------
```

```
291
292     /// @notice Launch a new token with bonding-curve parameters
            baked into virtual/real reserves.
293     /// @dev The function may mint an initial supply and
            optionally handle an initial buy if msg.value > 0.
294     /// @param name_ Token name (informational).
295     /// @param symbol_ Token symbol (informational).
296     /// @param uri_ Metadata URI (informational).
297     /// @param initialAmmQuoteAmount_  Initial QUOTE amount
                reserved     for AMM math (sets virtuals).
298     /// @param initialRatio_ Ratio used to split token supply
            between meteora/real buckets.
299     function launchToken(
300         string memory name_,
301         string memory symbol_,
302         string memory uri_,
303         uint256 initialAmmQuoteAmount_,
304         uint256 initialRatio_,
305         address quoteAsset_,
306         FeePercent feePercent_
307     ) external payable returns (address tokenAddress);
308
309     /// @notice Launch a token using custom fee parameters
            (requires custom fees to be enabled).
310     function launchTokenWithCustomFees(
311         string memory name_,
312         string memory symbol_,
313         string memory uri_,
314         uint256 initialAmmQuoteAmount_,
315         uint256 initialRatio_,
316         address quoteAsset_,
317         IFeeController.FeeParams calldata feeParams_
318     ) external payable returns (address tokenAddress);
319
320     function finalizeAndMigrate(
321         address token_,
322         SomePPAccessRegistry.PairCreatorAuthorization
323             calldata tokenAuthorization,
324         SomePPAccessRegistry.PairCreatorAuthorization
325             calldata quoteAuthorization,
326         address feeClaimer2
327     ) external;
328
329     /// @notice Buy tokens  for QUOTE against the bonding curve.
330     /// @dev minTokensOut:  for slippage protection.
331     ///      deadline:      for trade operation expiration
            protection.
332     function buyToken(
333         address _token,
334         uint256 minTokensOut,
335         uint256 deadline
336     ) external payable;
337
338     function buyWithNative(
339         address _token,
340         uint256 minTokensOut,
341         uint256 deadline
```

```
342     ) external payable returns (uint256);
343
344     function buyWithQuote(
345         address _token,
346         uint256 quoteGrossAmount,
347         uint256 minTokensOut,
348         uint256 deadline
349     ) external payable returns (uint256);
350
351     /// @notice Sell tokens for QUOTE with permit against the
            bonding curve.
352     function sellTokenWithPermit(
353         address _token,
354         uint256 tokenAmount,
355         uint256 minQuoteOut,
356         uint256 deadline,
357         bytes calldata signature
358     ) external;
359
360     function sellToken(
361         address _token,
362         uint256 tokenAmount,
363         uint256 minQuoteOut,
364         uint256 deadline
365     ) external;
366
367     function sellForNative(
368         address _token,
369         uint256 tokenAmount,
370         uint256 minQuoteOut,
371         uint256 deadline
372     ) external returns (uint256);
373
374     function sellForNativeWithPermit(
375         address _token,
376         uint256 tokenAmount,
377         uint256 minQuoteOut,
378         uint256 deadline,
379         bytes calldata signature
380     ) external returns (uint256);
381
382     function sellForQuote(
383         address _token,
384         uint256 tokenAmount,
385         uint256 minQuoteOut,
386         uint256 deadline
387     ) external payable returns (uint256);
388
389     function sellForQuoteWithPermit(
390         address _token,
391         uint256 tokenAmount,
392         uint256 minQuoteOut,
393         uint256 deadline,
394         bytes calldata signature
395     ) external payable returns (uint256);
396
397     /// @notice Withdraw accumulated protocol fees to `to`.
```

```
    /// @notice Owner-only helper to withdraw mistakenly sent native currency.
    function sweepNative(address payable to, uint256 amount) external;

    /// @notice Set the LiquidityLocker contract. Factory MUST be the owner of the locker.
    function setLiquidityLocker(address locker) external;

    function setRouter(address router_) external;

    function setReferralWallet(address referralWallet_) external;

    function setMigrationWallet(address migrationWallet_) external;

    function setMigrationAuthority(address migrationAuthority_) external;

    function setAllowCustomFees(bool allowed) external;

    function setMigrationAuthorityFee(uint256 migrationAuthorityFee_) external;

    function setTradeFeeSplit(
        uint256 denominator,
        uint256 treasuryNumerator,
        uint256 referralNumerator
    ) external;

    /// @notice Receive ETH.
    receive() external payable;
}
```