# StonkBrokers Smart LP (Volatility Farming) Developer Integration

Version: 2026-09-30 (rev 5 - SB Security review round 3: holder aware previewWithdraw, zap limit on the admitting band, zero output fills return instead of reverting; rev 4 - review round 2 folded into the V2 notes; rev 3 - Arbitrum One fleet added, live V1 vs staged V2 split made explicit, fresh vault counts, HTTP activity feed)

Chains: Robinhood Chain (chainId 4663, explorer https://robinhoodchain.blockscout.com) and Arbitrum One (chainId 42161, explorer https://arbiscan.io).

Audience: wallets, dashboards, yield aggregators, keeper and crank bots, and smart account (ERC-6551 / AA) integrations wiring deposits, withdrawals, reads, and indexing for the Safety Deposit Box Smart LP vaults.

Downloads (from https://stonkbrokers.io/integration/), byte exported from the deployed artifacts:

- SmartLpVault.abi.json - the LIVE vault generation, all reads and writes
- SmartLpRegistry.abi.json - vault discovery (same on both chains)
- SmartLpLens.abi.json - one call grid reads and withdraw previews (live)
- SmartLpFeeManager.abi.json - harvest mode fee engine (live, optional)
- SmartLpVault.v2.abi.json, SmartLpLens.v2.abi.json, SmartLpFeeManager.v2.abi.json - the STAGED V2 generation (section 0)
- StonkBrokers-SmartLp.pdf - this guide
- SmartLp-Integration.md - this guide as markdown

## 0. Which bytecode you are talking to

There are two Smart LP bytecode generations and only ONE of them is on chain today.

| Generation | Status | How to recognise it |
|---|---|---|
| V1 | LIVE on both chains. Every vault the registries list (179 on Robinhood Chain, 22 on Arbitrum One at the time of writing) runs it. Audited 2026-09-08/09 (Hashlock Certified, Smart LP Volatility Farming report). | Runtime code is exactly 23,718 bytes, links SmartLpLib 0x4A2dA8e70b51595Bf564082A86336AbDDc62E32F, blanked code hash 0x1c32f129...eb2e3 (section 2.1). canSettle() and basket() revert with EMPTY data. |
| V2 | STAGED, NOT DEPLOYED. The remediation of the SB Security review (2026-09-25). Ships as NEW vault addresses listed on the same registries; contracts are immutable, nothing is upgraded in place. | canSettle() returns a bool, basket() returns two uints, the lens exposes pairAmount / previewDeposit / uncollectedFees. |

Everything in this guide describes the LIVE V1 behaviour unless a sentence or table row is tagged V2. V2 rows are here so you can prepare; calling a V2 selector on a live vault reverts with empty data, which wallets render as a generic failure. When V2 lands the V1 vaults keep working forever (withdrawals included), so keep serving the positions you already track and feature probe per vault instead of per chain.

Recommended feature probe: eth_call basket() on the vault. A decoded pair means V2, an empty revert means V1. Cache the answer per address forever (code is immutable post deploy). (canSettle moved to the lens in V2 for EIP-170 headroom: lens.canSettle(vault) works on both generations.)

## 1. What Smart LP is

Each Smart LP vault is an immutable ERC-20 share token over exactly ONE Uniswap v3 position in one canonical Uniswap v3 pool, plus any idle balances. No proxy, no upgrade path, no admin path to principal. Withdrawals are proportional, in kind, and can NEVER be paused or gated, not by the owner, not by a guard trip, not by a stale oracle.

Three range modes, immutable per vault (mode()):

| mode | Name | Behavior |
|---|---|---|
| 0 | FULL_RANGE | v2 style min to max range. Never recenters. |
| 1 | BALANCED_BAND | Symmetric band around TWAP; keeper recenters under on chain guards. |

| | | |
|---|---|---|
| 2 | ASK (single sided) | Base token only, placed above spot. Sold progressively as price rises. While price sits inside the band the position is a live Uniswap range order, so a retrace inside the band re buys the in band slice. On V1 isSettled() is derived from spot and flips back on a retrace (the converted quote stays inside the range order). V2: once spot AND TWAP have crossed the whole band the permissionless settle() pulls the position to idle quote and latches the vault, and no path can mint a band again. |

Every vault names its base (the risk asset, e.g. the stock or meme token) and quote (the other pool side: USDG or WETH on Robinhood Chain, USDC or WETH on Arbitrum One) via baseIsToken0(). Share accounting is quote denominated: at genesis 1e18 shares equals about 1 human quote unit, and share price rises as fees compound and withdraw retentions accrue. Never present shares as token amounts; show USD and underlying values from the lens instead.

Vault token symbols follow sdSYMBOL-FR / -BB / -ASK (e.g. sdNVDA-FR). Arbitrum vaults quoted in USDC carry a -USDC suffix (e.g. sdRAAPL-FR-USDC).

## 2. Live addresses

| Chain | Contract | Address |
|---|---|---|
| Robinhood Chain (4663) | SmartLpRegistry (discovery, source of truth) | 0xE8749183Fbf6A657EB58B3a4D3E4B9Cc09560146 |
| Robinhood Chain (4663) | SmartLpLens (all reads) | 0x754Bf8479630bbC22aA7b5E9742156ce89dD3D4d |
| Robinhood Chain (4663) | SmartLpLib (linked library, audited) | 0x4A2dA8e70b51595Bf564082A86336AbDDc62E32F |
| Robinhood Chain (4663) | Chainlink ETH/USD (for WETH quoted USD display) | 0x6091E64eb7138EEF066a80FD3A0d7427B91f2721 |
| Arbitrum One (42161) | SmartLpRegistry | 0xFB2eA53b16C07011d4390A2e449385180A54eD5e |
| Arbitrum One (42161) | SmartLpLens | 0x1243c8FB4099568460ba0619aaB24DC15758887F |
| Arbitrum One (42161) | SmartLpLib (linked library, same address as Robinhood) | 0x4A2dA8e70b51595Bf564082A86336AbDDc62E32F |
| Arbitrum One (42161) | Chainlink ETH/USD | 0x639Fe6ab55C921f74e7fac1ee960C0B6293ba612 |

The registry is the ONLY source of truth for vault addresses on each chain. 179 vaults are listed on Robinhood Chain and 22 on Arbitrum One at the time of writing (all seeded and active); the lineup grows, so never hardcode vault addresses. Delisting only removes a vault from the UI surface. The vault itself keeps working forever (withdrawals included), so keep serving positions in delisted vaults you already track. Vaults never move between chains: a registry row belongs to the chain you read it from.

Arbitrum One specifics: the wave 1 fleet covers WETH/USDC, USDT/USDC, WBTC, ARB, APE, GMX, PENDLE, PEAR (WETH quoted, TWAP only, no Chainlink cross check) and the Reality Protocol stock tokens rAAPL / rSPCX / rHOOD (USDC quoted). USDC and USDT are 6 decimals. Everything below applies unchanged.

All contracts are BUSL-1.1 licensed and source verified on Blockscout, Arbiscan and Sourcify. A full security audit of the deployed fleet (2026-09-09, 90 on chain probes against live bytecode) confirmed: no owner, keeper, or guardian path can reach depositor principal; deposit caps and guard bounds cannot be tightened against holders; and plain withdraw succeeds even with deposits paused or feeds stale.

### 2.1 Verifying a vault is genuine (recommended)

Registry listing is a curation signal, not a cryptographic one. To verify a vault carries the audited bytecode, fetch its runtime code (eth_getCode) and check:

1. Code length is exactly 23,718 bytes.
2. Every linked library slot (the vault delegatecalls SmartLpLib; its 20 byte address appears at 15 fixed sites in the code) carries the audited library 0x4A2dA8e70b51595Bf564082A86336AbDDc62E32F.
3. Blank the constructor immutables, the 15 library slots, and the trailing 32 byte solc metadata hash to zero, then keccak256 the result. The audited generation hashes to 0x1c32f1291fb1038d8c6506800210a217c649195efb2dd0134869d8776d1eb2e3.

The same three checks pass on BOTH chains (verified 2026-09-28 against a Robinhood vault and an Arbitrum vault: same length, same library address, same blanked hash). The stonkbrokers.io frontend applies exactly this check to every registry row before

rendering it (app/lib/smartLpCodePin.ts in the public site bundle is a working reference implementation, including the immutable slot offsets). Contract code is immutable post deploy, so a pass can be cached per address forever. The three first generation AMZN rehearsal vaults on Robinhood Chain (0x5019...f17a, 0xc7f7...4dd0, 0x9bb1...16ad) run an older audited build and hash differently; treat them by raw code hash or skip them. All other listed vaults are the current generation. V2 vaults will publish a new length, library and hash here when they deploy.

## 3. Discovery and reads

One call renders the whole floor:

```
// SmartLpRegistry
count() returns (uint256)
vaults(uint256 index) returns (address)
all() returns (address[])
isListed(address vault) returns (bool)

// SmartLpLens (live)
viewAll(address registry) returns (VaultView[])
viewVault(address vault) returns (VaultView)
previewWithdraw(address vault, uint256 shares) returns (uint256 out0, uint256 out1)

// SmartLpLens (V2 additions)
pairAmount(address vault, address tokenIn, uint256 amountIn) returns (uint256)
previewDeposit(address vault, uint256 a0, uint256 a1) returns (uint256 shares, uint256 used0, uint256 used1)
uncollectedFees(address vault) returns (uint256 fees0, uint256 fees1)
recenterOutlook(address vault) returns (...)
previewWithdraw(address vault, uint256 shares, address holder) returns (uint256 out0, uint256 out1)
```

VaultView fields:

| Field | Meaning |
|---|---|
| vault, pool, token0, token1 | Addresses; token0/token1 are the pool's own ordering |
| symbol0/1, decimals0/1 | Token metadata (USDG, USDC and USDT are 6 decimals; never assume 18) |
| mode | 0 FULL_RANGE / 1 BALANCED_BAND / 2 ASK |
| baseIsToken0 | Which side is the risk asset |
| poolFee | Pool fee in hundredths of a bip (3000 = 0.3%) |
| tickLower/tickUpper/spotTick | Live range vs spot for band visuals |
| positionId | NFPM token id; 0 = no position minted yet |
| totalSupply | Share supply |
| totalValueQuote, tvOk | Vault TVL in quote units at the pool TWAP; tvOk false = TWAP read failed, display "unavailable" |
| capQuote, maxCapQuote | Live deposit cap and its immutable ceiling (quote units) |
| perfFeeBps, withdrawFeeBps | 1000 = 10% performance fee; 10 = 10 bps withdraw retention |
| depositsPaused | Guardian pause. Deposits only, never withdrawals |
| settled, allBase | ASK lifecycle flags. On V1 settled is derived from spot and can flip back on a retrace. V2: settled is the one way settle() latch and allBase is always false once settled |
| lastRecenterAt | Last keeper recenter / trail timestamp |
| feeManager | 0 = compound mode; nonzero = harvest mode (section 8.1) |
| canSettle | ASK: settle() would succeed right now (computed by the lens from spot, TWAP and the band, so it reads on V1 too; on V1 there is no settle() to call). Show a "Settle" button when true on a V2 vault |
| earlyExitFeeBps, earlyExitWindow | V2: the early exit retention and its window (both 0 on V1 and on V2 vaults that never armed it). vault.exitFeeBpsFor(holder) is the rate a withdraw pays right now for that holder |

The lens is a display rail: every per vault read is try/catch wrapped so one broken vault or a stale oracle can never blank the grid. Transaction paths on the vault itself stay strict.

previewWithdraw returns the holder's in kind claim net of the withdraw retention, safe to display as "you receive". On V1 it prices the position at the pool TWAP and does not include uncollected position fees, so the real withdraw output can be a hair higher; use it as min0/min1 with a small margin, never as an exact figure. V2 makes it an exact mirror of withdraw (the pro rata liquidity is taken first and priced at the real sqrt price, uncollected fees folded in and capped at what the pool position actually pays, net of fee): use it as min0/min1 directly, at most one wei of pool rounding apart. lens.previewDeposit prices on lens.basketAfterCollect(vault) because a real deposit collects the position's fees first, so its shares figure IS the mint.

V2 rev 5: the preview knows the early exit retention. Call the three argument overload previewWithdraw(vault, shares, holder): it nets vault.exitFeeBpsFor(holder), which is exactly what withdraw charges that holder right now (on V1 bytecode it falls back to the flat withdrawFeeBps). Use that figure as min0/min1 on every harvest mode vault, since setFeeManager requires the early exit retention to be armed and every fresh depositor is stamped inside the window. The two argument overload stays for callers without a holder: it is conservative and adds earlyExitFeeBps whenever earlyExitWindow is nonzero, so it can read low for a holder past the window but never high.

Useful direct vault reads: twapTick(), spotTick(), totalValueQuote(), isSettled(), isAllBase(), baseToken(), quoteToken(), minFirstDepositQuote(), guards(), positionAmountsAt(sqrtPriceX96), valueInQuote(a0, a1, sqrtPriceX96). V2 adds basket(), depthGateMult(), recenterPending(), exitFeeBpsFor(holder) (withdraw retention plus the early exit retention while the holder's time weighted receipt stamp sits inside earlyExitWindow()), pendingOwner(); and on the lens canSettle(vault), askDepositCountsQuote(vault) (ASK: the next depositSingle prices the idle quote in and zaps part of the deposit into it), observationRing(vault) (ring health for keepers).

## 4. Deposits

All deposit paths are nonReentrant, run the TWAP/feed guard (section 7), enforce the on chain cap (CapExceeded), and on the LIVE fleet price the minted shares off the pool TWAP tick, so a same block spot push cannot mint cheap shares. Pending pool fees are collected BEFORE share pricing on every deposit, so a newcomer never captures fees earned before it arrived.

| Function | Modes | Live (V1) behavior |
|---|---|---|
| `deposit(amount0, amount1, minShares, to) payable` | FR, BB | Pair deposit. The ratio need not match: unmatched value stays idle and is still counted by share valuation, but match the pool ratio for full deployment (the stonkbrokers.io desk pre balances from spotTick / tickLower / tickUpper with V3 math). |
| `zapDeposit(tokenIn, amountIn, minShares, to) payable` | FR, BB | Single token entry. The vault swaps toward the position ratio through its OWN pool under a TWAP floored minOut. The depositor bears the pool fee. V2 rev 5: the swap's price limit is the same band the guard admitted the deposit under (maxSpotVsTwapBps, or maxSpotVsTwapStaleBps when the feed is stale), so an admitted deposit always has room to fill. |
| `depositSingle(amountBase, minShares, to) payable` | ASK | Base token only. Reverts DepositsClosed once price has entered the band or the vault reads settled. V2: when the vault holds idle quote above the dust floor the entry zaps part of the base into quote so the depositor pays for that quote instead of taking it from holders; that zap runs on the admitting band too (rev 5). |

V2 share pricing (SB Security #9 / #12): after genesis a deposit is priced against the vault's own redeemable basket, never against the TWAP. basket() returns the position principal at the pool's real sqrt price plus idle balances. Shares are the MINIMUM proportional contribution of the two tokens against that basket, used0/used1 are the amounts the vault keeps, and the remainder of either token is REFUNDED in the same transaction (DepositRefund event). A one sided deposit against a two sided basket mints ZERO and reverts MinShares on V2; pre fill the second leg with lens.pairAmount or use zapDeposit, whose swap runs to a price limit at the edge of the guard band that ADMITTED the deposit (maxSpotVsTwapBps, or maxSpotVsTwapStaleBps on a stale feed; rev 5) and refunds what did not fit. Before rev 5 the limit sat at min(swapFloorBps, maxSpotVsTwapBps), so a vault whose guard band was wider than its swap floor admitted the deposit and then had no room to fill it (MinShares at any size); the ASK depositSingle zap runs under the same admitting band. Only genesis prices at TWAP, because the depositor alone sets the basket.

### 4.1 Native ETH (WETH quoted vaults only)

Vaults whose pool has a WETH side and a wired weth() accept native ETH on every deposit path. CRITICAL: msg.value is ADDED to the declared amount. To deposit 1 ETH via zapDeposit, pass amountIn = 0 and msg.value = 1e18. Passing amountIn = 1e18 AND msg.value = 1e18 declares 2 ETH, and the ERC-20 pull for the second half then reverts on allowance. Same rule for the

WETH side amount0/amount1 in deposit and amountBase in depositSingle. Vaults with weth() == 0 revert NativeNotSupported on any nonzero msg.value; a handful of live meme vaults (PONS, AI, CASHCAT) shipped that way, so probe weth() before offering a native path.

### 4.2 Approvals

ERC-20 legs transferFrom the caller. Approve the VAULT for the exact amount rather than unlimited: wallet risk engines (Blockaid etc.) have near zero coverage of chain 4663 and flag unlimited approvals to unknown contracts. Receipts are balance diff measured, so fee on transfer tokens cannot corrupt accounting (the shortfall is simply what you deposited). USDG note: USDG reverts with its OWN custom InsufficientAllowance() selector 0x13be252b, not the OpenZeppelin shape.

### 4.3 Genesis minimum

While totalSupply() == 0 (or the vault was fully drained) the first deposit must be worth at least minFirstDepositQuote() in quote units (10 USDG / 10 USDC equivalent on the live fleet, 0.004 WETH on WETH quoted vaults) or it reverts FirstDepositTooSmall (0x007426cd). 1000 dead shares are burned at genesis (inflation attack guard). V2 applies the same rule while totalSupply() <= 1000 (only dead shares left), and the genesis deposit sets the basket every later deposit is priced against, so deposit in the pool ratio.

### 4.4 minShares: simulate, then floor

Never sign with minShares = 0 (legal, but leaves the depositor unprotected against share price movement between quote and inclusion). The recommended policy, the one the stonkbrokers.io frontend uses, is:

1. Simulate the exact deposit call via eth_call from the depositor's address. All three deposit functions RETURN the minted share count, so the simulation result IS your quote.
2. Sign the real transaction with minShares = simulated * 98%.

A pool move between simulation and inclusion can then cost at most about 2% before the transaction reverts MinShares instead of filling quietly worse. The simulation doubles as a preflight: any guard or cap revert surfaces with a decodable selector (section 10) before the user signs.

## 5. Withdrawals, always live

| Function | Behavior |
|---|---|
| withdraw(shares, min0, min1, unwrapNative, to) | Proportional in kind. Ignores pause, guard trips, stale feeds, settled state: nothing blocks a burn and exit. No oracle involved. |
| zapWithdraw(shares, tokenOut, minOut, unwrapNative, to) | Single token exit: in kind burn, then the unwanted side swaps through the vault's own pool under a TWAP floored minOut. Runs the guard, so it CAN revert under manipulation or feed divergence; fall back to plain withdraw. |

- Withdraw retention: withdrawFeeBps (10 bps on live vaults) is deducted and STAYS IN THE VAULT, accruing to remaining holders. Surface this in your UI; an unexplained immediate round trip loss reads as a bug to users. previewWithdraw already nets it. V2 BALANCED_BAND vaults require at least half the pool fee (15 bps on a 0.3% pool, 50 bps on a 1% pool).
- Early exit retention (V2): while a holder's receipt stamp sits inside earlyExitWindow, withdraw charges exitFeeBpsFor(msg.sender) = withdrawFeeBps + earlyExitFeeBps. Preview with the holder overload previewWithdraw(vault, shares, holder) (rev 5) and pass its figure as min0/min1; the two argument overload assumes the holder is inside the window whenever one is armed, so it reads low, never high, and a MinOut revert can no longer come from the preview itself.
- Withdrawing also collects and skims pending pool fees first, so a withdrawer is never diluted by uncompounded fees.
- unwrapNative = true on WETH side vaults pays the WETH leg as native ETH to to. V2 sends the ERC-20 leg first and the ETH leg last so a share price read inside a receiver callback is consistent with the burned supply.

## 6. Share math reference

```
sharePriceQuote = totalValueQuote() / totalSupply()          // both live reads
depositShares  ~= depositValueQuote * totalSupply() / tvBefore // live V1 (TWAP priced)
withdrawOut    = pro rata of (position principal + idle) * (1 - withdrawFeeBps/1e4)

// V2 (post genesis)
(h0, h1)       = basket()                                     // redeemable amounts at spot
```

```
depositShares  = min(amount0 * ts / h0, amount1 * ts / h1)
used_k         = ceil(shares * h_k / ts); refund_k = amount_k - used_k
```

totalValueQuote prices the position at the pool TWAP tick, in the QUOTE token's native decimals (USDG / USDC vaults: 6 decimals). For USD display on WETH quoted vaults multiply by your ETH/USD mark (the Chainlink feeds in section 2 are Crypto class and fresh 24/7 on both chains).

## 7. Guards, oracles, and weekend behavior

Every deposit / zap path (and keeper recenter / trail) runs checkTwapBand:

1. TWAP band: pool spot tick must sit within maxSpotVsTwapBps "ticks as bps" of the pool's own TWAP (twapWindow, 30 min on feed backed vaults, 10 min on feed less vaults) or the call reverts TwapDeviation (0x64a78d82). Live bands: 100 ticks on stock vaults, wider (up to 400) on feed less meme vaults; read guards() per vault.
2. Chainlink cross check (only when the feed is FRESH): the pool implied base price must sit within maxVsFeedBps (default 3%) of the feed or the call reverts FeedDeviation (0xc7f602ab).
3. Stale or absent feed = band widening, never a revert: Robinhood equity feeds publish nothing from Friday close to Monday 00:00 UTC. A stale (or missing; meme vaults and every Arbitrum WETH pair have feed = 0) feed only widens the TWAP band to maxSpotVsTwapStaleBps. Deposits and withdrawals stay live all weekend.

Monday open note: when a stock feed resumes after a volatile weekend, the 3% feed check re arms and a drifted pool briefly reverts FeedDeviation on deposit paths until arbitrage converges the pool. Self healing within minutes, by design. Plain withdraw is unaffected. Do not retry loop aggressively; back off and re quote.

Wallets surface guard reverts as opaque gas estimation failures (the generic "3900" style error). Decode the revert selector from the estimation error data and map it to copy; the full selector table is in section 10.

### 7.1 Deposit availability probe (recommended UX)

You can classify whether a vault currently ACCEPTS deposits without any balance or approval: the TWAP/feed guards run BEFORE any token pull, so a dust simulation is a clean probe. eth_call a 1 wei deposit (depositSingle(1, 0, vault) on ASK vaults, deposit(1, 1, 0, vault) otherwise, from any address; the vault's own address works):

- Reverts TwapDeviation or FeedDeviation: the market is CLOSED for deposits right now (show "deposits paused, market moving"; on stock vaults with a paused weekend feed, "reopens Monday 00:00 UTC" is the honest wording). Withdrawals remain live.
- Reverts DepositsClosed: ASK vault inside or past its band.
- V2: reverts DepthGate (0xceee81e7, BALANCED_BAND only): the vault is FULL relative to its pool. V2 band vaults ship with a depth gate armed (depthGateMult(), default 5) that refuses any deposit pushing vault value past that multiple of the quote side pool depth inside the swap floor. Show "vault at capacity for this pool's depth"; withdrawals are never gated.
- Any OTHER revert (allowance, balance, FirstDepositTooSmall): the guards PASSED; the vault is open and the revert is just the dust amount. Success: open.

Poll it every ~30s per rendered vault and disable the deposit form with a plain reason instead of letting the user hit a wallet level estimation error. Keep the last known state through RPC hiccups rather than flapping the form.

## 8. Keeper and permissionless surface

| Function | Access | What it does |
|---|---|---|
| compound() | Permissionless | Collect pool fees, skim the performance fee, reinvest the rest into the position (compound mode; in harvest mode the net fees leave for the fee manager). Safe to crank from any bot. V2: the redeploy runs only while spot sits inside the TWAP band and is skipped (never reverted) otherwise, and the crank never mints a band vault's first position. |

| | | | |
|---|---|---|---|
| recenter() | keeper / owner | BALANCED_BAND only: exit, ratio swap (bounded by maxSwapBps, TWAP floored), re mint centered on TWAP. Cooldown enforced. On V1 a vault that is a large share of its pool can hit SwapFloor permanently; the operator lowers maxSwapBps per vault (owner setGuards) so the swap fits the depth. V2: the swap runs to a price limit and accepts a partial fill, converging over recenterPending tranches. A zero output fill is returned by the lib (rev 5: swapDepthLimited no longer reverts on it) and ratioSwapForRange reverts SwapFloor on used == 0 | out == 0, which is the check that keeps the #16 brick closed. |
| trailDown() | keeper / owner | ASK only: swap free re anchor of the band toward spot while the vault is still 100% base. | |
| settle() | Permissionless, V2 only | ASK: once spot AND the pool TWAP sit past the far edge of the band, collect fees, pull the whole position to idle quote, burn the NFT and latch settled. Reverts NotSettleable while only spot is past the band, AlreadySettled after. Not present on live vaults. | |

Owner / guardian levers on the live fleet (setCap, setGuards, setDepositsPaused, setRoles, setFeeManager, sweep of non pool tokens) can never touch principal, tighten a cap below current TVL, or block a withdrawal. guards() returns (twapWindow, maxSpotVsTwapBps, maxVsFeedBps, feedStaleAfter, maxSpotVsTwapStaleBps, recenterCooldown, maxSwapBps, swapFloorBps).

### 8.1 Harvest mode (SmartLpFeeManager)

Vaults with a nonzero feeManager() route net collected fees to an immutable SmartLpFeeManager instead of reinvesting. Most live vaults run compound mode (feeManager == 0); probe per vault. Live (V1) surface:

- harvest() - permissionless crank, no arguments: converts the accumulated fee balances to the manager's immutable target() token under TWAP floors and pays the caller crankRewardBps (max 5%) of the harvest. Returns (targetOut, reward). A profitable crank for bots once fees accumulate.
- pendingOf(holder) - claimable target for a shareholder; claim(to) pays it; checkpoint(from, to) is what the vault calls on every share transfer so claims settle automatically.
- Events Harvested(caller, targetOut, reward, accPerShareX128) and Claimed(holder, to, amount) on the MANAGER address.

V2 keeps the ONE per share index credited at harvest time (the push time unit ledger tried in the first remediation round was withdrawn on the auditors' advice: partial harvests grew the unit count without bound and stranded fees). What changes: the vault notifys every push so the manager converts from TRACKED ledgers (held(k), never balanceOf), harvest(max0, max1) takes cranker sized legs (0 = all held), swaps are price limited so a thin pool converts partially instead of reverting, dust legs are skipped, absorb() books donations, and a stuck leg returns (0, 0). The timing window of harvest time crediting is covered operationally: the keeper harvests on a fixed cadence, and a harvest mode vault MUST have the early exit retention armed (earlyExitFeeBps inside earlyExitWindow, owner setEarlyExit, at most 300 bps / 7 days, cannot be disarmed once the manager is wired). Show exitFeeBpsFor(holder) as the retention a withdraw pays right now. A zero output leg never reverts the harvest: rev 4 briefly made swapDepthLimited revert on it, which would have reverted the whole _convert (no try/catch there), and rev 5 dropped that guard on the auditors' advice, so a dust leg simply returns nothing and stays held.

## 9. Fee map

- Performance fee: perfFeeBps (10%) on COLLECTED POOL FEES only, net revenue, never principal. Split 50% to the StockBooster (Clock In distributions) and 50% to $STONKBROKER buybacks.
- Withdraw retention: withdrawFeeBps (10 bps) stays in the vault for remaining holders (churn deterrent). V2 harvest mode vaults add the early exit retention inside earlyExitWindow; exitFeeBpsFor(holder) is the total a given holder pays right now.
- No deposit fee, no maintenance fee. Zap swaps pay the pool's own fee.

## 10. Revert selectors

Selectors are identical on both chains. Rows tagged V2 cannot fire on live vaults yet.

| Selector | Error | Typical cause / integrator action |
|---|---|---|
| 0xa4875a49 | CapExceeded | Deposit would push TVL over capQuote. Show "vault full". |
| 0x64a78d82 | TwapDeviation | Spot too far from TWAP (active pool or manipulation). Retry later. |
| 0xc7f602ab | FeedDeviation | Pool vs fresh Chainlink diverged more than 3% (Monday opens). Self heals. |

| | | |
|---|---|---|
| 0x007426cd | FirstDepositTooSmall | Genesis deposit under minFirstDepositQuote(). |
| 0xdeeb6943 | DepositsPaused | Guardian pause. Withdrawals unaffected. |
| 0x1a8ade30 | DepositsClosed | ASK vault entered its band or reads settled. |
| 0x2a580d6d | WrongMode | Wrong entrypoint for the vault's mode (e.g. deposit on an ASK vault; use depositSingle). |
| 0x54b8210d | MinShares | Share slippage floor hit. Re quote. V2: also an unbalanced pair deposit that would mint zero; pair it with lens.pairAmount or use zapDeposit. |
| 0x168f8aad | MinOut | Withdraw / zap output floor hit. Re quote. |
| 0x0a7287b5 | NativeNotSupported | msg.value sent to a vault without a wired WETH side. |
| 0x15fa6e71 | NotPoolToken | tokenIn / tokenOut is not one of the pool's two tokens. |
| 0x1f2a2005 | ZeroAmount | Nothing pulled. |
| 0x0cb4955b | SwapFloor | Internal swap fill under the TWAP floor. Retry later. On V1 band vaults this can be permanent when the vault is a large share of its pool until the operator lowers maxSwapBps; V2 fills what the depth allows and marks recenterPending. |
| 0xabf0f034 | NoPosition | Keeper action on a vault with no minted position. |
| 0x1bd3ec13 | TrailNotCloser | trailDown that would not move the band toward spot. |
| 0xb0782df7 | Cooldown | Keeper action inside recenterCooldown. |
| 0x98248e64 | EthSendFailed | Native payout to a receiver that rejected ETH. Withdraw with unwrapNative = false. |
| 0xf512b278 / 0x30cd7471 / 0xef6d0f02 | NotKeeper / NotOwner / NotGuardian | Access gates. |
| 0x019af637 | BadParams | Owner setGuards / setCap outside the immutable bounds. |
| 0xb2f71223 | NotSettleable (V2) | settle() before both spot and TWAP are past the band. Retry once the TWAP follows. |
| 0x560ff900 | AlreadySettled (V2) | settle() on a settled vault. Nothing to do. |
| 0xceee81e7 | DepthGate (V2) | BALANCED_BAND deposit refused: vault value would exceed depthGateMult() times the pool's quote side depth. Withdrawals unaffected. |
| 0x146f8be2 | TwapUnavailable (V2) | The pool's observation ring cannot serve the TWAP window right now. Grow the ring with increaseObservationCardinalityNext and wait. |
| 0x7b542bd7 | ObservationRingTooSmall(uint16 have, uint32 need) (V2) | Constructor / setGuards / manager constructor: the ring holds fewer slots than the window has seconds (one slot per second is the floor, so heavier trading cannot cycle the history out). Grow the ring to need first. Keepers watch lens.observationRing(vault). |
| 0x13be252b | USDG InsufficientAllowance | USDG's own custom allowance error; raise the approval. |

Two masquerading failures to rule out before blaming the vault:

- Sender has no ETH for gas: the node's up front gas_limit x maxFee balance check surfaces as a misleading "execution reverted, data: 0x" at estimation. Check the signer's ETH balance first.
- Empty revert data: a V2 selector called on a live V1 vault, or a mislinked / nonstandard token in the path, bubbles an empty revert which wallets render as a generic failure. Simulate via eth_call from the sender to capture real revert data.

## 11. Events (indexing)

All logs land on the VAULT address (library events are emitted via DELEGATECALL). Topics are identical on both chains.

| Event | Topic0 |
|---|---|
| Deposited(address indexed sender, address indexed recipient, uint256 amount0, uint256 amount1, uint256 shares) | 0x8bab6aed5a508937051a144e61d6e61336834a66aaee250a00613ae6f744c422 |

| | |
|---|---|
| `Withdrawn(address indexed sender, address indexed recipient, uint256 amount0, uint256 amount1, uint256 shares)` | `0x3cae9923fd3c2f468aa25a8ef687923e37f957459557c0380fd06526c0b8cdbc` |
| `Compounded(uint256 added0, uint256 added1)` | `0x8082b0dab6d73a8025e5f15e4fdd54b68197a613e361810d4a2446c634b98ada` |
| `FeesCollected(uint256 fees0, uint256 fees1, uint256 skim0, uint256 skim1)` | `0xf5d590414d56d256b8c16b850d0b57f2f5d2ed90686166e150b48a96f0dbdd61` |
| `Recentered(int24 tickLower, int24 tickUpper)` | `0xf50915569476905e05b4ef6c338d995bc8bf49e89258202b9709c6dd6504e3d7` |
| `Trailed(int24 tickLower, int24 tickUpper)` | `0xd606fe2b531f0b729332a3ba354a4acddcdc86720901401c7bbdd72af6556931` |
| `CapRaised(uint256 newCap)` | `0xe0c76087275a8c4223be123d025629c6693fd33f20c0f0f6bf632708bda4329b` |
| `GuardsSet(...) / DepositsPausedSet(bool) / RolesSet(...) / FeeManagerSet(address)` | `0x03243639...4cf2 / 0x9cc20448...c4cd / 0xcbb983ad...0672 / 0xbf5f5806...8e63 (admin config, rare)` |
| `Settled(...) (V2 only)` | `0x7823e479a1a4ebe2418874847436f8a1680c5ee5b17f38bb59dbff28e1b45552` |
| `DepositRefund(address indexed to, uint256 amount0, uint256 amount1) (V2 only)` | `0x67b1105972f389118797e8b84acfdd40581567974b584258e34ff63c296eec46` |
| `FeeSendFailed(address token, address to, uint256 amount) (V2 only)` | `0x6ce6bf31f670ee8a9d72c9af93d48e23fe0e2399c524befb52d77322d10b4381` |
| `EarlyExitSet(uint16 bps, uint32 window) (V2 only)` | `0xdbd92e2c558cec59ba49a1844b9237933b46e0daa49982ed24c7228d768ba4b1` |
| `RecenterSwap(bool zeroForOne, uint256 wanted, uint256 used, uint256 out, bool pending) (V2 only)` | `0x3acf110eec3b98891e15e445963af9bad5cc4ce353682bbff727997171ce19c4` |

Registry (Robinhood 0xE874...0146, Arbitrum 0xFB2e...D5e):

| Event | Topic0 |
|---|---|
| `VaultListed(address indexed vault, address indexed pool, uint8 mode)` | `0x3dd1a36f073a1702b0ea7db13a70123c0e21753a1571eaf80acacea471618152` |
| `VaultDelisted(address indexed vault)` | `0x1e50537a8984b347abcf90ec072a917d874c1737aaca4e13ae66aa499977aab1` |

Fee manager (harvest mode vaults, logs on the MANAGER address):

| Event | Topic0 |
|---|---|
| `Harvested(address indexed caller, uint256 targetOut, uint256 reward, uint256 accPerShareX128)` | `0xe48bba143e2a0b557fa6f3234bd6ffc704518cc98c7de6a2385549fae27d1b75` |
| `Claimed(address indexed holder, address indexed to, uint256 amount)` | `0xf7a40077ff7a04c7e61f6f26fb13774259ddf1b6bce9ecf26a8276cdd3992683` |
| `Notified(uint256 amount0, uint256 amount1) (V2 only)` | `0x0fd7bb2818b0aaa42a4064edb68964c3e29810804cf8cce3060a3bac23fb9d79` |

Vault shares are standard ERC-20s; track Transfer for holder accounting. On harvest mode vaults every share transfer also checkpoints the holder's fee claims automatically.

Suggested scan floors: Robinhood Chain block 53,800,000 (first vault listing), Arbitrum One block 505,449,619 (registry deploy, 2026-09-15). Both chains serve eth_getLogs in chunks; the public Robinhood RPC caps a chunk at roughly 100k blocks, the public Arbitrum RPC at roughly 50k.

## 12. HTTP activity feed (optional)

stonkbrokers.io publishes the fleet's automation ledger so dashboards do not have to scan logs themselves:

```
GET https://stonkbrokers.io/api/locker/smartlp-activity?chain=robinhood
GET https://stonkbrokers.io/api/locker/smartlp-activity?chain=arbitrum
```

Per vault it returns compound / collect / rebalance counts, lifetime fees0/fees1 and skims, added0/added1, the last compound and rebalance blocks, a recent event ledger, a fee time series, and a per holder position map; plus the cursor block and updatedAt.

Values are raw token units as strings (apply decimals0/1 from the lens). The route serves the last good snapshot through RPC trouble and is rate limited per IP; poll no more than every 30s. It is a convenience, not a source of truth: the chain is.

## 13. Smart accounts, ERC-6551 wallets, contract callers

There is NO EOA gate anywhere in Smart LP: token bound accounts, Safes, and AA wallets are first class depositors. Two rules learned in production:

- Preflight the inner call. Wrappers like the StonkBrokers executeCall swallow inner revert data ("call failed"). Before asking the user to sign, run the exact vault call via eth_call from the smart account's own address so real selectors (CapExceeded, FirstDepositTooSmall, ...) surface and can be mapped to copy.
- Gas comes from the signer, not the account. A "the vault reverted" report on a smart account deposit is often just the owner EOA holding no ETH. Check the signer's balance before chasing a contract revert.

## 14. Integration checklist

- [ ] Discover vaults from the registry of EACH chain only; refresh the list, handle new listings and delistings, never mix chains.
- [ ] Verify newly listed vaults against the audited bytecode hash (section 2.1) before offering deposits.
- [ ] Feature probe basket() per vault and branch V1 / V2 (section 0).
- [ ] Read the floor through lens.viewAll; treat tvOk == false as "value unavailable", never zero TVL.
- [ ] Respect decimals0/decimals1 everywhere (USDG, USDC, USDT = 6).
- [ ] Native ETH deposits: declared amount EXCLUDES msg.value; probe weth() first.
- [ ] Exact amount approvals; handle USDG's custom allowance selector.
- [ ] Genesis deposits: check minFirstDepositQuote() when supply is 0.
- [ ] Simulate every deposit and sign with a 98% minShares floor (section 4.4).
- [ ] Probe deposit availability with the dust simulation (section 7.1) and gate the form with plain copy.
- [ ] Map the section 10 selectors from gas estimation errors.
- [ ] Show the 10 bps withdraw retention before signing.
- [ ] Expect FeedDeviation windows at Monday 00:00 UTC on stock vaults; back off and re quote.
- [ ] compound() and (on harvest vaults) harvest() are permissionless cranks; bots welcome, harvest() pays a reward share.

Questions / listings: simple@clutch.market, TG simplefarmer69.