Perpgame Launchpad

A NAV-Anchored Bonding Curve for Tokenized Baskets of Leveraged Perpetuals

Perpgame

Technical Whitepaper — v0.1 (June 19, 2026)

Abstract

The Perpgame Launchpad lets a creator deploy a tradeable ERC-20 (AGENT) whose value is mechanically anchored to the net asset value (NAV) of an actively rebalanced basket of Hyperliquid leveraged tokens (LTs). Each launched market is two contracts: an `AgentTreasury` that custodies the LT basket and executes rebalances, and an `AgentCurve` that is simultaneously the AGENT ERC-20 and its primary market. The curve uses a continuous, supply-elastic bonding mechanism: there is no fixed supply, no premine, and no graduation event. Buyers mint AGENT at a premium-loaded multiple of NAV-per-share; sellers burn AGENT and are paid their pro-rata slice of the basket back in USDC (or, optionally, in kind) at the NAV floor. A small protocol fee is charged on each buy and sell. The buy/sell premium spread accrues directly to the NAV per share of existing holders. Markets are minted by a `TreasuryFactory` that deploys each treasury behind an upgradeable beacon proxy and spawns its curve atomically. This document specifies the token, the pricing curve, the treasury and rebalance machinery, the trust model, and the protocol invariants.

Contents

1 Overview

A *launched market* on Perpgame is minted by a singleton `TreasuryFactory` and consists of two per-market contracts wired together atomically at deploy:

1. **`TreasuryFactory`** — the protocol-level singleton (an `Ownable2Step` owner) that deploys markets, holds the global fee parameters and the global pause switch, and owns the upgrade beacon shared by every treasury. `deployTreasury(params, permit)` applies the creator's offline USDC permit, `CREATE2`-deploys the treasury as a `BeaconProxy`, and initializes it.

2. **`AgentTreasury`** — custodian of a weighted basket of leveraged tokens, deployed behind an upgradeable beacon proxy. Holds the LTs and any idle USDC, computes NAV, and is the only contract permitted to deploy capital into LTs or return it to redeemers. A *rebalancer* role sets target weights and steps the portfolio toward them.

3. **`AgentCurve`** — the AGENT ERC-20 *and* its bonding-curve market in one contract. It mints AGENT on buys, burns AGENT on sells, and drives the treasury accordingly.

Binding happens inside `AgentTreasury.initialize`: the treasury sets its initial target portfolio, pulls the seed USDC via the permit, deploys it into the basket, then *spawns* its `AgentCurve` with `new AgentCurve(…)` and records its address (`CurveSet`). There is no separate creator wiring step — the curve is the sole mutator of treasury LT positions through buys and sells from genesis, while the rebalancer adjusts *composition* without changing total NAV.

Role	Held by	Capabilities
`owner`	factory owner (`Ownable2Step`)	`deployTreasury`, set `feeBps`/`feeRecipient` (fee ≤ `MAX_FEE_BPS`), global `setPaused`, beacon upgrade
`CREATOR`	immutable per-market address	recipient of swept dust / stranded LT balances (recovery only)
`rebalancer`	rotatable address (two-step)	set targets, step rebalance, `cancelRedeem`, `migrateLt`, `sweepDust`, propose successor
`curve`	spawned `AgentCurve`	`deployUsdc`, `withdrawLtsTo`

Table 1: On-chain roles.

2 Leveraged Tokens (lts)

The atomic building block is a *leveraged token* (LT) — a Bounce Tech ERC-20 on HyperEVM that wraps a leveraged perpetual position, parameterized by (coin, direction, leverage). For example `HYPE5L` is a 5× long on `HYPE`; `BTC3L` is a 3× long on `BTC`. Each LT:

- is itself an ERC-20 with an internal `exchangeRate`;
- exposes `mint(to, baseAmount, minOut)` to enter (paying USDC base) and `redeem(to, ltAmount, minBaseAmount)` to exit;
- provides value conversion helpers `ltToBaseAmount` and `baseToLtAmount` (LT units ↔ USDC value);
- may pause minting (`mintPaused`) and may settle redemptions either *atomically* (from an on-chain buffer) or *asynchronously* (the LT issuer owes USDC that arrives later, tracked via `userCredit`).

The treasury never takes naked perp exposure directly; all leverage is encapsulated inside the LTs. The universe is not fixed: any LT on the Bounce factory allowlist (checked via `BOUNCE_FACTORY.ltExists`) may be used, and a single market holds up to `MAX_ASSETS` = 20 distinct LT legs. A typical basket spans a handful of coins and leverages, e.g. `HYPE5L, BTC3L, ETH3L`.

3 Net Asset Value

NAV is the total USDC-denominated value the treasury controls: the marked value of every LT held plus any idle USDC.

$$\text{NAV} \;=\; \sum_{i=1}^{n} \texttt{ltToBaseAmount}_i\big(\texttt{balance}_i\big) \;+\; \texttt{USDC.balanceOf(treasury)} \tag{1}$$

NAV is denominated in USDC (6 decimals). It is recomputed on demand from live LT exchange rates — there is no stored, stale book value. Because rebalancing only swaps value between LTs (modulo execution slippage and the protocol fee), it does not by itself change NAV; deposits, redemptions, and the price action of the underlying LTs do.

During an in-flight rebalance an LT leg may sit in Bounce's redemption escrow — moved out of `balanceOf` but still owed to the treasury as `userCredit`. `nav()` counts only held balances and idle USDC; an internal `navWithPendingRedemptions` adds escrowed credit back when sizing buy targets mid-rebalance, so a step does not over-allocate against value that is temporarily out in escrow. Idle USDC originating from deposits is tracked separately (`depositIdleUsdc`) so deploy thresholds and fee/redemption accounting can distinguish it from transient redemption proceeds.

4 The agent Token

AGENT is a standard 18-decimal ERC-20 implemented directly by `AgentCurve`. Its supply is fully elastic and entirely determined by curve flow:

- **No free premine, no team allocation, no fixed cap.** The only genesis AGENT is the creator's seed, minted 1:1 against seed USDC that is itself deployed into the basket — fully NAV-backed, not a free allocation. Every other AGENT is minted by a buy.

- **Minted on buy, burned on sell.** `totalSupply()` rises and falls with demand.
- **No graduation.** Unlike pump-style curves, there is no migration to an external AMM at a threshold. The curve *is* the permanent market.
- **Backed, not reserved.** Each AGENT is a pro-rata claim on the LT basket. There is no separate reserve token; the backing *is* the leveraged basket itself.

The economically meaningful quantity is NAV per share:

$$\pi \;=\; \frac{\mathrm{NAV}}{S}, \qquad S \equiv \texttt{totalSupply()}. \tag{2}$$

5 The Bonding Curve

5.1 Price function

The instantaneous AGENT price is NAV-per-share scaled by a supply-dependent *premium* multiplier:

$$\texttt{price}(S) \;=\; \frac{\mathrm{NAV}}{S} \cdot \texttt{premium}(S). \tag{3}$$

The premium ramps quadratically in the supply ratio from 1.0× at $S = 0$ up to a capped ceiling $1 + \Pi$:

$$\texttt{premium}(S) = \begin{cases} 1 + \left(\dfrac{S}{S_{\mathrm{cap}}}\right)^2 \Pi, & S < S_{\mathrm{cap}}, \\ 1 + \Pi, & S \geq S_{\mathrm{cap}}, \end{cases} \tag{4}$$

where S_{cap} (`PREMIUM_CAP_SUPPLY`) is the supply at which the premium multiplier saturates, and Π (`EXTRA_PREMIUM`, a WAD fraction) is the maximum add-on. Note that supply may grow *indefinitely* past S_{cap} — only the *multiplier* stops growing; the token does not.

Default parameters. The reference deployment uses $S_{\mathrm{cap}} = 30{,}000 \cdot 10^{18}$ and $\Pi = 0.3 \cdot 10^{18}$ (+30%), so a buyer at or beyond the cap pays 1.3 × NAV per share. The constructor bounds $\Pi \leq \texttt{MAX\_EXTRA\_PREMIUM} = 4 \cdot 10^{18}$ (a 5× ceiling). This bound guarantees a post-cap buyer always mints at least $1/(1 + \Pi) \geq 1/5$ of the NAV-pro-rata amount, so no launch can be configured to be economically confiscatory.

5.2 Buying

A buyer supplies `usdcIn`; the curve first skims the protocol fee $\texttt{fee} = \texttt{usdcIn} \cdot \phi/10^4$ (where $\phi \equiv \texttt{feeBps}$ is read from the factory) and works with the net amount $\texttt{usdcNet} = \texttt{usdcIn} - \texttt{fee}$. Only `usdcNet` is deployed into the basket *and* used to size the mint, so the fee never enters NAV. The minted amount is the premium-discounted NAV-pro-rata share of the net spend:

$$\texttt{agentOut} \;=\; \frac{\texttt{usdcNet} \cdot S \cdot 10^{18}}{\mathrm{NAV} \cdot \texttt{premium}(S)}. \tag{5}$$

The 10^{18} factor de-scales the WAD premium; the USDC (6-dec) units in `usdcNet` and NAV cancel, and the surviving 18-dec scale comes from S, yielding an 18-dec AGENT amount. Dividing by $\texttt{premium}(S) \geq 1$ mints *fewer* AGENT than a naive NAV-pro-rata buy would — this shortfall is the value accrual to existing holders (§5.4).

Genesis seed. A market is born seeded, not via a public first buy. When the factory deploys the treasury it pulls the creator's seed USDC and the spawned `AgentCurve` constructor mints the genesis supply at a fixed unit:

$$\texttt{agentOut} \;=\; \texttt{usdcSeed} \cdot \texttt{USDC\_TO\_AGENT\_SCALE}, \qquad \texttt{USDC\_TO\_AGENT\_SCALE} = 10^{12}, \tag{6}$$

so 1 USDC ($=10^6$) mints 1 AGENT ($=10^{18}$) to the creator. Because the seed and the curve are created in one atomic `initialize`, the opening anchor cannot be front-run or manipulated by pre-funding the treasury. The curve's internal quote retains an $S = 0$ branch for completeness, but in practice supply is positive from the first block.

Execution. `buy(usdcIn, minAgentOut, minLtOuts, recipient, deadline)` pulls USDC from the buyer, enforces the `deadline`, skims the fee to `feeRecipient`, quotes the net amount against *pre-trade* supply and NAV, enforces $\texttt{agentOut} \geq \texttt{minAgentOut}$ (slippage), forwards `usdcNet` to the treasury via `deployUsdc` (which splits it across the basket by target weights and mints LTs), and finally mints AGENT to `recipient`.

5.3 Selling

Selling burns AGENT and pays the seller their *pro-rata slice of the basket*, by default *in USDC* — there is no premium on exit, only the protocol fee. A share `agentIn` of supply S claims the fraction $\texttt{agentIn}/S$ of every position. `sell` burns first, then calls `withdrawLtsTo(recipient, agentIn, S_before, minUsdcOut, returnLts)`, which:

1. pays the seller their pro-rata slice of *idle* USDC (pro-rata, not idle-first, so a seller cannot exit entirely in stale-marked cash and leave their LT slice — and its pending streaming fee — behind for other holders);
2. covers the remaining notional by redeeming the seller's pro-rata LT slice to USDC. Each leg redeems *atomically* when Bounce's instant-redeem buffer covers it; if a leg cannot fill now (buffer too low, paused, below min size) it is skipped when `returnLts = false` (the seller is paid only what redeemed), or handed to the seller as raw LT (net of fee) when `returnLts = true`, to redeem later;
3. deducts the fee $\texttt{fee} = \texttt{usdcOut} \cdot \phi/10^4$ from the proceeds and forwards the net to the seller, enforcing $\texttt{netOut} \geq \texttt{minUsdcOut}$ (slippage) on the USDC-only path.

The curve exposes two quotes for the seller:

$$\texttt{quoteSellNotional}(\texttt{agentIn}) = \frac{\text{NAV} \cdot \texttt{agentIn}}{S} \cdot \left(1 - \tfrac{\phi}{10^4}\right), \tag{7}$$

$$\texttt{quoteSellUsdc}(\texttt{agentIn}) = \left(\texttt{idlePaid} + \texttt{redeemable}\right) \cdot \left(1 - \tfrac{\phi}{10^4}\right), \tag{8}$$

where `quoteSellNotional` is the optimistic NAV-floor fair value (net of fee) and `quoteSellUsdc` is what is *instantly* withdrawable right now — the pro-rata idle slice plus only those legs whose redemption fits current atomic-redeem buffers (net of fee). A `deadline` and `minUsdcOut` provide slippage / staleness protection. Sells revert while a rebalance is in flight (§6) so a redeemer never receives a slice of a half-rebalanced, USDC-pending basket.

5.4 Value accrual: why the premium matters

Because buyers pay the premium but sellers exit at the NAV floor, the spread is captured permanently as NAV per share. The protocol fee is skimmed *before* the buy touches NAV (and from sell proceeds on the way out), so it is a flat leakage to `feeRecipient` that sits entirely outside the curve — it neither dilutes nor accrues to holders, and the analysis below applies to the net spend. Concretely, consider a buy whose *net* (post-fee) amount is u USDC at premium $p \equiv \texttt{premium}(S) \geq 1$. The buy adds u to NAV and mints $\texttt{agentOut} = \frac{u\,S}{\mathrm{NAV}\,p}$. The post-trade NAV per share is:

$$\pi' \;=\; \frac{\mathrm{NAV} + u}{S + \frac{uS}{\mathrm{NAV}\,p}} \;=\; \pi \cdot \frac{(\mathrm{NAV} + u)\,p}{\mathrm{NAV}\,p + u}. \tag{9}$$

Since $p \geq 1$ implies $(\mathrm{NAV} + u)p = \mathrm{NAV}p + up \geq \mathrm{NAV}p + u$, we have $\pi' \geq \pi$, with equality only when $p = 1$ (i.e. at $S = 0^+$). **Every premium buy strictly increases** NAV **per share for existing holders, and no sale dilutes it** (exits are pro-rata at the floor). The curve thus has a monotone-rising floor under NAV-neutral conditions; only adverse LT price action can lower π.

5.5 Decimal and scaling constants

Constant	Value	Purpose
`ONE`	10^{18}	WAD unit for premium / fixed-point math
`USDC_TO_AGENT_SCALE`	10^{12}	Genesis anchor (6-dec USDC → 18-dec AGENT)
`MAX_EXTRA_PREMIUM`	$4 \cdot 10^{18}$	Upper bound on Π (5× premium ceiling)
`MAX_ASSETS`	20	Max distinct LT legs per treasury
`MAX_FEE_BPS`	500	Upper bound on the protocol fee (5%)
`DEFAULT_FEE_BPS`	100	Reference protocol fee (1%)
`BPS_DENOM`	10,000	Basis-point denominator for weights / fees

Table 2: Fixed-point constants. USDC is 6-dec; AGENT and LTs are 18-dec. The fee parameters `feeBps`/`feeRecipient` live on the `TreasuryFactory` and are shared by every market.

6 The Treasury and Rebalancing

6.1 Target portfolio

The treasury stores an ordered set of assets, each (`symbol`, `lt`, `targetBps`), with weights in basis points that must sum to exactly 10,000. The rebalancer updates targets via `setTargetPortfolio(newPortfolio, reasoningCid)`:

- weights are validated to sum to 10,000 (else `WeightsDoNotSumTo10000`);
- all existing weights are first zeroed, so any asset not re-specified ends at `bps = 0` (slated to be sold off);
- new symbols are appended after an $O(1)$ Bounce-allowlist check (`ltExists`, else `LtNotAllowed`); an existing symbol cannot be re-pointed to a different LT address (`SymbolTokenMismatch`); duplicate symbols or LTs are rejected;

- the live set — including zeroed legs still lingering in `symbols[]` until pruned — is capped at `MAX_ASSETS` = 20 (`TooManyAssets`);
- `minBps` is recomputed as the smallest non-zero target weight; it drives the minimum deployable USDC so the thinnest leg always clears Bounce's minimum transaction size;
- a `reasoningCid` (an IPFS pointer to the rationale, e.g. produced by the off-chain rebalancing agent) is required and emitted with `RebalanceTargetSet`.

6.2 Deploying buy inflows

On a buy, `deployUsdc(usdcAmount, minLtOuts)` (curve-only, `whenNotPaused`, reverts if a rebalance is in flight) credits the incoming USDC to idle and deploys it. Idle USDC is only deployed once it clears `minDeployUsdc()` — derived from `minBps` so that even the thinnest leg meets Bounce's minimum transaction size; below that the deploy is deferred (`DeployDeferred`) and the USDC waits, fully backing NAV at face value, until a later buy or rebalance. When it deploys, USDC is split across the basket by `targetBps` and minted into each LT; to avoid rounding dust the *last active* leg receives the residual $\texttt{usdcAmount} - \sum_{i<n} \texttt{allocated}_i$. Each mint enforces `minLtOuts[i]`. A leg that is paused or below the minimum size is *skipped* (emitting `MintSkippedPaused` / `MintSkippedBelowMin`) rather than bricking the whole deploy — its USDC stays idle and deploys on a later pass — unless the caller demanded that leg via a non-zero `minLtOuts[i]`, in which case it reverts (`SlippageExceeded`).

6.3 Stepwise rebalance

`executeRebalanceStep(maxRedeemLt, minRedeemUsdc, maxMintUsdc, minMintLt)` (rebalancer-only) moves the realized basket toward the targets. It is *idempotent and callable across blocks* so that asynchronous LT redemptions can settle between steps. A single step:

1. Snapshots NAV_0 = NAV at start, against which the sell loop is sized.
2. **Sell loop.** For each over-target LT, compute the USDC value to shed and convert to LT units (clamped by balance and `maxRedeemLt`; a leg dropped to `targetBps = 0` sheds its full balance). If the LT's on-chain buffer (`LT_HELPER.getLeveragedTokenBufferAssetValue`) covers the expected base out, redeem *atomically* (`AtomicRedeem`); otherwise call `prepareRedeem`, set `rebalanceInFlight = true` (`RebalanceInFlightChanged`), and emit `RedemptionPrepared` — Bounce settles the USDC later. Legs below the minimum transaction size are skipped (`RedeemSkippedBelowMin`).
3. **Re-measure.** Buy targets are sized against `navWithPendingRedemptions` (post-sell value, since atomic redeems pay Bounce fees and checkpoint the LTs, plus escrowed in-flight credit), not the stale pre-sell NAV_0.
4. **Buy loop.** For each under-target LT, mint with available idle USDC (clamped by `maxMintUsdc`). If USDC is short *and* a rebalance is in flight, defer the mint (`MintDeferredForSettlement`) so a later step completes it once redemptions settle; if not in flight, mint what the proceeds actually cover (an atomic same-step sell delivers USDC net of Bounce's redemption fee, so the residual — off-target by that fee — settles next step). Paused / below-min legs are skipped.
5. **Settle & prune.** If in flight and no LT still owes `userCredit`, clear `rebalanceInFlight`; then swap-pop any fully-exited (`bps = 0`, zero-balance, zero-credit) symbols out of `symbols[]`.

Every leg carries explicit slippage bounds (`minRedeemUsdc`, `minMintLt`). The `rebalanceInFlight` flag is the key safety interlock: it gates buys and sells (`deployUsdc` / `withdrawLtsTo` revert `RebalancePending`) so that redeemers never receive a slice of a half-rebalanced, USDC-pending basket.

Settlement, cancellation, migration. `settleRebalance()` is permissionless and simply rechecks whether the in-flight flag can clear once Bounce has paid out — so the basket unfreezes without waiting for the next rebalance step. `cancelRedeem(symbol)` (rebalancer-only, after Bounce's cancel delay) recovers a pending async redemption that Bounce can no longer settle — e.g. a depreciated leg whose credit fell below Bounce's flat executor fee — pulling the escrowed LT back so the flag is not stuck set forever. `migrateLt(symbol, newLt)` (rebalancer-only) re-points a symbol at a new LT contract after Bounce redeploys an LT address while preserving its symbol; the new LT must be on the allowlist and unused, any pending credit must be settled first, and any stranded balance on the old contract is handed to `CREATOR`.

6.4 Dust and stranded legs

`sweepDust(symbol)` (rebalancer-only) evicts a retired symbol — one with `targetBps = 0` — whose residual LT balance can no longer be drained through Bounce. It reverts if the leg is still active (`SymbolStillActive`), still has a pending async credit (`RedeemPending`), or is still redeemable above the minimum size on a live Bounce LT (`SymbolStillRedeemable`). The guard restricts it to genuinely stuck legs — e.g. a delisted LT, or a 1-wei balance someone sent to pin a symbol in `symbols[]` (which would otherwise tax every NAV/deploy/rebalance iteration forever). Any residual is sent to `CREATOR` to redeem off-protocol; for true dust this is ≈ 0, and for a stranded delisted position it is an intentional, rebalancer-gated recovery that lowers NAV by exactly that leg's value.

7 Trust Model and Invariants

7.1 Access control

- **Curve $\leftrightarrow$ treasury binding.** The treasury spawns its own curve inside `initialize` (one-shot, guarded by `Initializable`); only that curve can call `deployUsdc` / `withdrawLtsTo` (`onlyCurve`). There is no post-deploy wiring step to race or front-run.
- **Rebalancer scope.** The rebalancer can change *composition* and step execution but cannot mint or burn AGENT and cannot move value to arbitrary addresses — recovery paths (`sweepDust`, `migrateLt`) send stranded LTs only to the immutable `CREATOR`. The role rotates via a two-step handshake: `proposeRebalancer` then `acceptRebalancer` by the proposed address, so a typo cannot strand the role on an uncontrolled key.
- **Creator scope.** The creator holds no ongoing custody or mint authority; it is only the recovery recipient for swept/stranded LT balances.
- **Factory owner scope.** The `TreasuryFactory` owner (`Ownable2Step`) deploys markets, sets the global `feeBps` ($\leq$ `MAX_FEE_BPS` $= 5\%$) and `feeRecipient`, can globally pause new deploys/buys/sells (`setPaused`), and owns the beacon that upgrades the shared `AgentTreasury` implementation. It cannot mint, burn, or transfer a treasury's LTs. Treasury storage is append-only across upgrades to preserve layout.

7.2 Safety properties

- **Reentrancy.** `initialize`, `buy`, `sell`, `deployUsdc`, `withdrawLtsTo`, `setTargetPortfolio`, `executeRebalanceStep`, `cancelRedeem`, `migrateLt`, and `sweepDust` are `nonReentrant`.
- **Pre-trade quoting.** Buys quote against pre-trade S and NAV (and the post-fee net spend), so a buy cannot bootstrap its own favorable price within the same call.
- **Genesis anchor protection.** The seed and the curve are created in one atomic `initialize`; the NAV=0 / `NoNav` guard prevents quoting against a zero or poisoned NAV, and there is no public pre-seed window to manipulate the opening anchor.
- **No premium on exit.** Sells settle at the NAV floor (less only the protocol fee), removing any buy/sell round-trip that could extract the premium and making the premium a pure transfer to long-term holders. The seller is paid their *pro-rata* idle slice (not idle-first), so they cannot exit in stale cash and leave their LT slice's streaming fee on remaining holders.
- **Slippage / deadlines.** `minAgentOut` and `minLtOuts` on buys, `minUsdcOut` and a `deadline` on sells, and `minRedeemUsdc` / `minMintLt` on each rebalance leg bound every externally observable execution.
- **In-flight gate.** Both buys and sells revert while `rebalanceInFlight`, so traders always face a fully-settled basket; a permissionless `settleRebalance` can lift the gate as soon as Bounce pays out.
- **Pause & bounded fee.** The factory owner can globally pause deploys/buys/sells (`whenNotPaused`); the fee is hard-capped at `MAX_FEE_BPS` (5%) in `setFeeBps`, so it can never be raised to a confiscatory level.

7.3 Risks

- **Underlying leverage risk.** NAV tracks leveraged perpetuals; adverse moves (and the volatility decay inherent to constant-leverage products) can reduce π below a holder's entry price. The premium floor is *nominal in* AGENT*-per-*NAV, not a guarantee against LT drawdowns.
- **lt liquidity / pause.** A paused or illiquid LT can block deploys and force async redemption paths; deploys and rebalances skip such legs and make progress across steps rather than fail atomically, but a seller wanting pure USDC may receive less than `quoteSellNotional` when buffers are thin (`quoteSellUsdc` surfaces the instantly-withdrawable amount).
- **Rebalancer key.** A compromised rebalancer cannot steal principal but can churn the basket (incurring slippage and fees) and re-weight toward riskier LTs. Rotation is two-step; governance of who holds it is a deployment-time concern.
- **Protocol admin / upgradeability.** The factory owner sets the fee (capped at 5%), can pause trading, and — via the shared beacon — can upgrade the `AgentTreasury` implementation for every market at once. This is a trusted role; a compromised or malicious owner could pause markets or ship a harmful upgrade. It cannot directly mint/burn AGENT or move a treasury's LTs, and the fee bound limits extraction.
- **Protocol fee.** Every buy and sell pays `feeBps` to the `feeRecipient`; this is a round-trip cost borne by traders, not by long-term holders' NAV per share.

- **Oracle / mark dependence.** NAV, and therefore all pricing, depends on each LT's `ltToBaseAmount` mark; the system inherits the correctness of the Bounce LT accounting.

8 Lifecycle Summary

1. **Deploy.** The creator signs an offline EIP-2612 USDC permit for the seed (no gas, no big-block opt-in). The `TreasuryFactory` owner submits `deployTreasury`, which applies the permit, `CREATE2`-deploys the treasury as a `BeaconProxy`, and runs `initialize`: set the target portfolio (+ genesis reasoning CID), pull and deploy the seed into the basket, and spawn the `AgentCurve` (S_{cap}, Π).

2. **Genesis seed.** The curve mints at the 1 USDC = 1 AGENT anchor to the creator; the seed USDC is already deployed across the basket by target weights.

3. **Steady state.** Buys mint AGENT at a premium (net of fee) and grow the basket; sells burn AGENT for a pro-rata USDC exit at NAV (net of fee), or in kind on request. The premium spread compounds into π.

4. **Rebalance.** The rebalancer publishes new targets (with rationale) and steps the basket toward them, using atomic or async LT redemption as liquidity allows, settling in-flight redemptions across steps.

9 Notation

Symbol	Meaning
S	AGENT total supply (`totalSupply()`), 18-dec
NAV	treasury net asset value in USDC, 6-dec
$\pi = \text{NAV}/S$	NAV per share
S_{cap}	`PREMIUM_CAP_SUPPLY`: supply where premium saturates
Π	`EXTRA_PREMIUM`: max premium add-on (WAD)
$p = \texttt{premium}(S)$	current premium multiplier, $1 \le p \le 1 + \Pi$

This document describes contracts `TreasuryFactory.sol`, `AgentTreasury.sol` (`VERSION 2.1.0-upgradeable`), `AgentCurve.sol` (`VERSION 1.0.0`), and the `TreasuryValuation` library as of the current revision. Parameters cited are reference defaults: S_{cap} and Π are set per-launch, while the fee and pause switch are global on the factory. Nothing herein is financial advice.