# BUTANE: A SECURE EUTXO SYNTHETICS PROTOCOL

v. dc0cd1998f4d180e1d8b177685ee6af3e77e6aad

Micah Kendall
Butane Team
micah@mcomp.tech

Jay Taylor
Butane Team
jay@mcomp.tech

Abstract. The EUTXO paradigm as implemented on the Cardano Babbage-era ledger provides a concurrent, fully sharded, turing-complete model of computation (generalised state machine), with untyped plutus core as a restricting validating component of state transitions. Transactions transition states, while each UTxO is an independent piece of sharded state. Butane implements Synthetics, a fungible derivative financial instrument, in a coherent manner, playing into the strengths of a sharded ledger. Here we outline its design and implementation.

## 1. Introduction

Synthetic assets are tokenized derivatives that give users exposure to an underlying asset's price without owning it directly.

The Butane Protocol ("Butane") is a permissionless, non-custodial synthetics protocol built on Cardano. This document specifies the behavior implemented by the protocol contracts. Protocol tokenomics and certain details of off-chain services (including the price feed oracles) are out of scope. This paper will be updated as the protocol changes.

1.1. **Why Cardano**. Cardano's EUTXO model, as implemented on the Babbage-era ledger, is well-suited to synthetics. Butane uses deterministic and sharded execution to reduce contention and fees, and represents instruments as lightweight Cardano native tokens ("CNTs") that compose cleanly with the wider Cardano ecosystem.

## 2. The Butane Protocol

Butane facilitates the creation of synthetic assets through Collateralized Debt Positions ("CDPs"). A CDP is opened by locking collateral and minting synthetics against it, subject to collateral constraints. If a position falls below required thresholds, liquidations enforce solvency and preserve the overcollateralization invariant. Redemptions, CDP interest, and treasury interactions are used to manage supply and support a soft peg.

2.1. **Configuration**. Each synthetic asset is parameterized by a set of three UTxOs: a base parameters UTxO, an interest parameters UTxO, and a staking parameters UTxO. The former is controlled by governance actions, while the latter two are managed by a dedicated rates controller (either a public key or a script), which can be different for each synthetic.

2.1.1. *Base Parameters.* These parameters define the core mechanics of a synthetic asset, mainly concerning how CDPs are created for the asset. It also governs certain requirements for how the interest and staking rates are managed.

**collateral_assets:** The sorted list of accepted collateral assets.

**weights:** The list of "weight" parameters for each collateral asset. Used in the calculation of a CDPs "Health Factor".

**denominator:** The shared denominator for all collateral weights.

**minimum_outstanding_synthetic:** For each CDP which mints this synthetic

asset, the minimum amount the CDP is required to mint.

**max_proportions:** The list of the "Maximum Proportion" parameters for each collateral token, where $\mathrm{MP}_i$ indicates that the $i$th collateral token can be used to collateralize at most $\mathrm{MP}_i$% of the outstanding debt.

**max_liquidation_return:** In the event of a liquidation, the maximum proportion of the outstanding debt that can be claimed from the CDP owner.

**treasury_liquidation_share:** In the event of a liquidation, what % of the excess claimed value is sent to the treasury as a fee.

**redemption_share:** In the event of a CDP redemption, what % of the repaid value the redeemer can claim from the CDP.

**fee_token_discount:** Optional. If set, this is the % discount when paying eligible CDP interest with the protocol fee token instead of synthetics. If unset, BTN interest payment is disabled for that synthetic.

**minimum_rate_duration:** The minimum time that must pass before an interest rate can be updated again.

**maximum_interest_increase:** The maximum amount by which the interest rate can be increased in a single update.

**maximum_interest:** The global maximum interest rate that can be charged for this synthetic.

**underlying:** A list of underlying asset baskets that can back direct underlying minting and redemption.

**underlying_redemption_fee:** When redeeming synthetics for underlying assets, the percentage charged as an extra synthetic burn.

**rates_controller:** The credential (key or script) that is authorized to sign interest and staking rate changes.

2.1.2. *Interest Parameters.* These parameters govern the interest accrued on outstanding debt in CDPs.

**current_rate:** The current interest rate being charged to debt positions.

**last_updated:** The timestamp of the last time the `current_rate` was changed.

**start_time:** The timestamp when interest began to accrue for this synthetic. Generally the timestamp of the params UTxOs creation.

**accumulated_interest:** A list representing the total accumulated interest for one unit of the synthetic, calculated as a Riemann sum. Each element corresponds to a whole-percentage maximum-interest cap (e.g. 0%, 1%, 2%, ...).

2.1.3. *Staking Parameters.* These parameters manage the interest earned by users who stake synthetics.

**current_rate:** The current savings rate being earned by stakers.

**last_updated:** The timestamp of the last time the staking `current_rate` was changed.

**start_time:** The timestamp when savings began to accrue. Generally the timestamp of the params UTxOs creation.

**accumulated_interest:** The total accumulated savings earned by one unit of the staked synthetic, calculated as a Riemann sum.

**start_time_offset:** The amount of time that must pass before a staked position can start earning rewards.

2.2. **Price Feeds**. Oracle price feeds provide the collateral pricing data used for CDP validation and treasury redemptions. Each feed includes a validity interval. The transaction interval must be contained within that feed interval, and the feed interval must be shorter than one day. The price feed verification key remains upgradable through governance.

## 3. BTN

The Butane fee and governance token ("BTN") is a CNT which can be owned by any user. The tokenomics of BTN are outside of the scope of this document. Within the protocol, BTN has the following use cases:

**Collateral Token:** BTN may be used as a collateral token for opening CDPs if it is an accepted collateral asset in a synthetic's parameters.

**Interest Payment:** For synthetics where `fee_token_discount` is set, BTN can be burned in place of synthetics to pay eligible CDP interest at a discount.

When BTN interest payment is enabled for a synthetic, burning BTN to repay interest is deflationary because total BTN supply decreases over time.

## 4. Collateralized Debt Positions

CDPs are single UTxOs that hold locked collateral and track outstanding debt. Users retain staking rights over any ADA collateral, and so can continue earning staking rewards on locked ADA. Key CDP metrics are described below.

Let $c_i, p_i, W_i$ be the amount, price (in terms of the synthetic), and weight of collateral token $i$ in the CDP which mints $d$ amount of synthetic token $s$. Then the "Borrowing Capacity" and "Health Factor" of the CDP are given as:

$$\mathrm{BC} := \sum_{1 \le i \le n} min\left\{ \frac{c_i \cdot p_i}{W_i}, d \cdot \mathrm{MP}_i^s \right\}$$

$$\mathrm{HF} := \frac{\mathrm{BC}}{d}$$

, where $\mathrm{MP}_i^s$ is the *maximum_proportion* parameter for the $i$'th collateral token of $s$. A position is considered "unhealthy" when $\mathrm{HF} < 1$. Similarly, the "Collateral Value" and "Collateral Ratio" of a CDP is given as:

$$\mathrm{CV} := \sum_{i=1}^{n} c_i \cdot p_i$$

$$\mathrm{CR} := \frac{\mathrm{CV}}{d}$$

The following CDP actions are supported on the protocol:

**Create CDP:** A position is opened by a user minting some amount of a synthetic asset and locking sufficient collateral value such that $\mathrm{HF} \ge 1$. When creating a CDP, the user specifies a maximum interest rate they are willing to pay, which cannot be lower than the current interest rate for that synthetic.

**Repay CDP:** A CDP owner repays the outstanding debt of one of their CDPs and closes the position. Locked collateral is returned to them.

**Liquidate CDP:** A position with $\mathrm{HF} < 1$ has its debt partially or fully repaid by any user, and a substantial portion of the locked collateral is claimed from it. In the event of a partial liquidation (i.e., where not all outstanding debt is repaid by the liquidator), a new position with $\mathrm{HF} \ge 1$ is created with the same owner as the liquidated CDP.

**Redeem CDP:** A position's debt is partially or fully repaid by any user, and equivalent collateral value is claimed from the position by them minus a premium. CDPs can only be redeemed when their maximum interest rate is below the current global interest rate for that synthetic.

CDP actions are entirely composable, permitting more complexity with seamless user experience. A "CDP Adjustment", whereby an existing CDP has either its collateral portfolio or outstanding debt altered, can be achieved via a composition of a "Repay" and "Create" action within a single transaction.

### 4.1. **Interest**.

CDPs accrue interest based on a dynamically adjusting interest rate. Interest functions as a mechanism for the protocol to control the incentives for creating and closing CDPs, hence influencing the supply of synthetics and providing an avenue for price corrections in the event of a depeg. To calculate interest, the protocol uses a Riemann sum-based approach where a global `accumulated_interest` value is updated with each rate change.

The `accumulated_interest` list is indexed by whole-percentage maximum-interest caps. The rates themselves are still basis-point values. Let $A_i(t)$ be the total accumulated interest for a synthetic asset at time $t$ for bucket $i$, where $r_i = i \cdot 100$ basis points. Let $r(t)$ be the global interest rate and $s(t)$ be the staking rate at time $t$. Then:

$$A_i(t) := A_i\big(t_{\text{prev}}\big) + \int_{t_{\text{prev}}}^{t} \min(r(t), \max(r_i, s(t)))\,\mathrm{d}t,$$

Where $t_{\text{prev}}$ is the time of the previous rate change. When interest rates update at time $t$, this list is updated with the new values, and we store $t_{\text{prev}} := t$. Similarly, when a CDP is created, it stores $A_{\text{CDP}}$, which is how much interest the

position is *not* obligated to pay. In particular, $A_{\text{CDP}}$ is the amount of interest accrued since the synth parameters were created to the time of the CDP's opening, and so $A_{\text{CDP}} = A_{i_{\max}}(t_{\text{open}})$, where $i_{\max}$ is the CDP's maximum interest rate in basis points and $t_{\text{open}}$ is the time of the CDP's opening.

We can then find the total interest accrued by a CDP at time $t$ as

$$\begin{aligned} I(t) &= A_{i_{\max}}(t) - A_{\text{CDP}} \\ &= A_{i_{\max}}(t_{\text{prev}}) + \int_{t_{\text{prev}}}^{t} r_{\text{CDP}}\,\mathrm{d}t - A_{\text{CDP}}, \end{aligned}$$

Where $r_{\text{CDP}} \coloneqq \min(r(t), \max(r_{\max}, s(t)))$

Accumulated interest is added to a CDP's total outstanding debt. It is paid when a CDP is closed. The default payment path is the burning of synthetics. If `fee_token_discount` is set for that synthetic, part of the eligible interest can instead be paid by burning BTN (see Section 4.1.1).

The total outstanding debt of a CDP at closing time is hence:

$$D_{\text{close}} \coloneqq d(1 + I(t_{\text{close}}))$$

, where $d$ is the initial debt amount. The health factor at closing is therefore:

$$\text{HF}_{\text{close}} \coloneqq \frac{\text{BC}}{d(1 + I(t_{\text{close}}))}$$

4.1.1. *Paying Interest in BTN.* A portion of accrued CDP interest can be paid by burning BTN only when `fee_token_discount` is set for that synthetic. If `fee_token_discount` is unset, BTN interest payment is disabled and all interest is paid in synthetics. When enabled, the amount payable in BTN is discounted according to `fee_token_discount` and is capped. Because synths can be minted through staking (see Section 8), at least the amount that could have been earned by staking the borrowed amount over the same period must still be paid in the synthetic.

Let $I_{\text{total}}$ be the total interest accrued on a CDP, and $E_{\text{stake}}$ be the maximum theoretical staking rewards that could have been earned by the borrowed amount during the same period. When BTN interest payment is enabled, the amount payable in BTN is:

$$I_{\text{BTN}} = I_{\text{total}} - E_{\text{stake}}$$

If BTN interest payment is disabled for that synthetic, then $I_{\text{BTN}} = 0$.

4.2. **Ownership**. CDP ownership is not necessarily tied to a specific user but to a key or some constraint. Specifically, the following are possible definitions of CDP ownership:

- The payment key credential of a Cardano address
- The current holder of a specific NFT
- The ability to withdraw stake from a specified script
- Authorization via a signature from a specified signing key

This allows for CDPs to be owned by scripts, and for the ownership of a CDP to be transferred to another party via the transfer of an NFT bond, allowing for future interoperability and extension.

## 5. Liquidations

A CDP is subject to liquidation when it is unhealthy. This involves a liquidator burning some amount of outstanding debt and claiming value of locked collateral up to some value greater than the debt that was repaid.

The *max_liquidation_return* parameter defines the limit on the value a liquidator can claim. $\text{MLR} = 120\%$ means a liquidator who fully repaid a CDP can claim up to $120\%$ of the repaid debt value from collateral. If they repaid half, they could claim at most $60\%$.

Following liquidation, the debt must be either 0 (the CDP closes) or at least the *minimum_outstanding* parameter. If it closes, remaining collateral (if any) is sent to a collection script for the CDP owner. If debt remains, leftover debt and collateral are moved into a new CDP with the same owner. This ensures partial liquidations do not leave unresolved unhealthy debt. A liquidator cannot increase any collateral token amount in the leftover CDP, which prevents changing token composition to extract more of a particular token.

The *treasury_liquidation_share* parameter indicates what % of the excess collateral claimed by the liquidator gets sent to the treasury as a fee. The value of this fee can be determined:

$$\text{LF} = \text{TS}\left(1 - \frac{1}{\text{CR}}\right) \cdot V$$

, where $V$ is the value claimed from the CDP owner during the liquidation and TS is the *treasury_liquidation_share* parameter.

The liquidator can choose which assets to claim, as long as they satisfy *max_liquidation_return* and, for partial liquidations, keep the leftover CDP healthy. The treasury fee uses the same token proportions as the claimed value.

## 6. Redemptions

CDP redemptions involve a user repaying some amount of a CDP's outstanding debt and claiming an equivalent amount of collateral value from it (minus a premium), based on the mark prices provided by the oracle price feed. Redemptions act as a depeg protection mechanism and as a way to enforce the protocol's monetary policy. In the event of a synthetic's market price falling below its mark price, arbitrage opportunities arise for users to redeem synthetics at values near their mark price, reducing supply and creating buy pressure. A CDP becomes eligible for redemption by any user if its owner-defined *maximum interest rate* falls below the current global interest rate for that synthetic. This incentivizes CDP owners to set a competitive maximum rate.

Redemptions do not make the CDP owner lose total value, because outstanding debt decreases by an amount greater than or equal to the collateral value claimed. This is configured by *redemption_share*, which scales collateral claim per token burned. For example, $\mathrm{RS} = 97\%$ means a redeemer can claim $97\%$ of the repaid value from CDP collateral.

The redeemers pay interest on behalf of the CDP owner; the total amount of synthetics they burn includes the principal debt plus any accrued interest, which is treated as additional debt. Redemptions can be partial. The redemption amount must be at least the synthetic's minimum outstanding debt parameter. In the case of a partial redemption, the original CDP is replaced by a new one with a reduced debt and collateral amount, and the leftover debt must be at least the minimum outstanding amount. If a CDP is fully redeemed (i.e., all of its outstanding debt is paid), any leftover collateral is sent to a collection script for the owner's reclamation.

## 7. Underlying Assets

Synthetics can be parameterized with a list of accepted *underlying baskets*. A user mints through this path by depositing an accepted basket of underlying assets into a `TreasuryUnderlying` UTxO. The UTxO records the synthetic and the amount of synthetic obligation it backs. This obligation, not the raw token amount, is what limits later withdrawals.

Each active basket defines the assets and conversion rates that may back the synthetic. The protocol allows the deposited basket value to back less synthetic than the conversion rates would strictly permit. This is useful for small positions and minimum-ADA edge cases.

The *underlying_redemption_fee* is charged on redemption. If the conversion rate is 1:1, a deposit of 100 ADA into an "ADAb" market can mint up to 100 ADAb. With a 1% redemption fee, redeeming 100 ADAb of obligation requires burning 101 ADAb. The extra 1 ADAb is the protocol fee. Deposits are fee-free.

## 8. Staking

Users can stake their synthetic assets to earn a yield based on a variable savings rate. Staked positions have an enforced unlock gate: the `start_time` of a staked position must be at least `start_time_offset` after the upper bound of the staking transaction's validity range (see Section 2.1.3). i.e. if `start_time_offset` is 1 day, then the position would only start earning staked rewards after 1 day has passed. Unstaking is only valid once the transaction time is strictly after `start_time`. The ownership of staked position is defined by the same credential system as CDPs (see Section 4.2). The logic for rewards calculation is a simplified version of the CDP interest model, requiring only one "accumulated" value instead of a list.

## 9. Governance

Changes to Butane are handled through a governance mechanism. A governance NFT dictates conditions for issuing governance actions, and ownership can be a key or a script. Governance NFT ownership is transferable, so governance control itself is upgradable.

9.1. **Actions**. The following are the primary governance actions:

- **New Parameters:** Defining a synthetic by creating its base, interest, and staking parameter UTxOs.
- **Parameters Update:** Updating governance-controlled parameters for a synthetic.
- **Treasury Spend:** An arbitrary spend from the treasury, defined by a governance-specified output.
- **Treasury Mint:** The minting of a synthetic token via the treasury. Primarily used for minting previously-burned interest.
- **Treasury Debt Creation:** Creating redeemable treasury debt for a synthetic.
- **Treasury Stake Update:** Updates the treasury's stake delegation.
- **Compatibility Upgrade:** Authorizing compatibility wrapping for a later protocol version.
- **External Script:** A governance action handled by an external script.
- **Text Proposal:** An arbitrary proposal containing text to be voted on. It has no programmatic effect on the protocol.

## 10. Validator Design

(See also Section Appendix A for a full protocol type specification.) Butane's contracts use EUTXO-specific design patterns to reduce transaction cost and improve composition.

10.1. **Validators**. The current contracts are split into a router structure plus dedicated endpoint validators:

- **`synthetics`:** Primary router for all CDP actions, including creation, repayment, adjustment, and liquidation.
- **`synthetics_aux`:** Simple router contract for all other protocol actions.
- **`external_params`:** Governance-controlled synthetic parameter creation, update, and retirement.
- **`external_rates`:** Interest and savings rate updates.
- **`external_gov`:** Creation and consumption of governance actions, including protocol upgrade actions.
- **`external_treasury_actions`:** Treasury-facing governance actions such as spend, redeem, mint, and debt handling.
- **`external_treasury_ops`:** Treasury state operations.
- **`external_staking`:** Synthetic staking and unstaking.
- **`external_underlying`:** Underlying minting and redemption.
- **`price_feed`:** Verifies oracle signatures over price-feed payloads.
- **`leftovers`:** Collection of leftover collateral from liquidations and redemptions.
- **`migration`:** V1-to-V2 state migration.
- **`bad_debt`:** Treasury absorption of bad CDP debt.
- **`voided_synth`:** Owner collection from CDPs whose params have been voided.
- **`next_compat`:** Locking and unlocking assets for compatibility with a future protocol version.

The dependency structure is mostly encoded through withdrawal validators. The main synthetics validator either validates CDP actions directly or routes to `synthetics_aux`. The auxiliary router then requires the matching endpoint validator withdrawal for the selected redeemer family, and some endpoints require an additional price-feed, owner, governance, rates-controller, certificate, or reward-withdrawal dependency.

10.2. **General Patterns and Optimizations**. In internal benchmark tests, Butane reached up to 50 CDP creations and 42 CDP repayments in a single transaction. These figures come primarily from the implementation details below.

10.2.1. *The "Withdraw 0" Trick*. The ledger permits transactions that withdraw 0 rewards from a script staking credential. This allows one-off validation logic to run in the same transaction. Butane uses this heavily for CDP validation. Instead of duplicating heavy checks in each CDP spend, most validation runs through a stake withdrawal validator invoked whenever a Butane UTxO is spent or a protocol token is minted. The spend-side checks stay minimal.

This pattern also simplifies how extra data is attached to transactions. For price feeds, the feed data is passed through a price-feed withdrawal

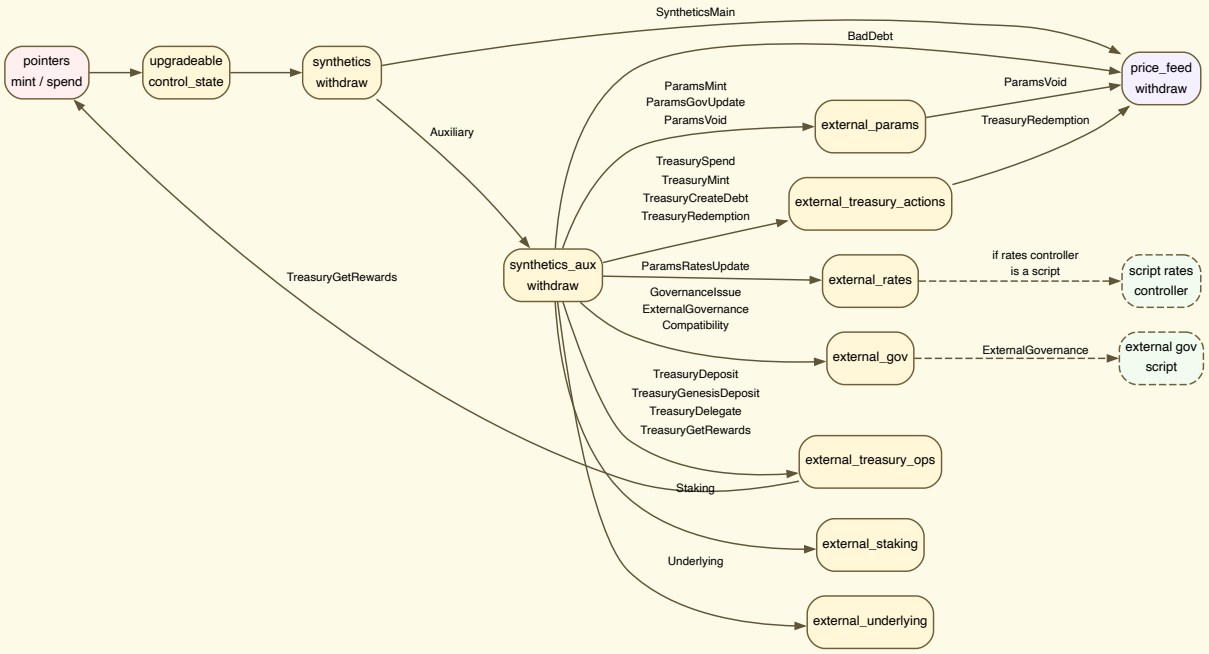


Figure 1: Validator and redeemer dependencies for the synthetics router and legacy upgrade-control layer. Solid edges are required validator withdrawals with the expected redeemer. Dashed edges are dynamic script withdrawals determined from a parameter or governance datum.

redeemer and then read by other scripts in the same transaction.

10.2.2. *Ordering*. The ledger orders transaction inputs and reference inputs lexicographically, and orders the withdrawals map by script hash. That can affect execution cost because data order is not fully controlled. Transaction outputs, however, are ordered deterministically by the transaction builder. Butane uses that behavior in the following ways:

- The order of synthetic parameters (which are referenced inputs) is determined as the transaction is built.
- The price feed redeemer is constructed in the same order as the synthetic parameters.
- Created CDP outputs are grouped by synthetic type and ordered the same as the synthetic parameters.
- In partial liquidations, spent CDPs with remaining outputs keep those outputs at the same index in the Butane output list as in the Butane input list.
- The same index-preserving rule applies to liquidations with leftovers.
- CDP repayment and liquidation actions are detailed in a list of records in the stake withdrawal redeemer in the same order as the CDP inputs.
- It is impossible to fully control both transaction input order and reference input order.
- Each record therefore includes the associated index of the parameter UTxO for that CDP's synthetic type.

These orderings minimize the need for searches and sorts in the validation logic.

10.2.3. *Burning*. Creating extra outputs is expensive, since each output increases transaction budget and must carry minimum ADA (see CIP-55). Butane uses token mint and burn flows to avoid unnecessary outputs. In particular, CDP interest is repaid by burning synthetics, and for synthetics with BTN interest payment enabled, by burning BTN for the eligible portion, instead of creating immediate treasury-interest outputs. The treasury can mint synthetics later when needed.

10.3. **Upgradability**. Butane supports full protocol upgrades. Governance itself is upgradable because governance NFT ownership is transferable, and the protocol supports compatibility wrapping for upgraded token sets via governance action. This allows a 1:1 swap between old and new synthetics.

In the event of major protocol changes, a state migration may be necessary. This can be done by setting the upgrade controller to a temporary migration script, where protocol UTxOs can be converted to the new version in batch transactions, followed by a final switch to the new protocol script.

## 11. Further Work

Butane focuses on atomic composition of protocol actions under the EUTXO model. This document has focused on composition within Butane itself, but the same approach can be used for cross-protocol interoperability on Cardano.

Future work is to formalize these composition patterns into practical interoperability standards that other protocols can adopt.

## Appendix A. Protocol Definitions

A1. **Validator Definitions**.

| **Validator Name** | **Parameters** |
|---|---|
| Leftovers → Leftovers → Spend | |
| PriceFeed → PriceFeed → Withdraw | `VerificationKey`: ByteArray, |
| PriceFeed → PriceFeed → Publish | `VerificationKey`: ByteArray, |
| Synthetics → DatumType → Withdraw | |
| Synthetics → ExternalGov → Withdraw | `GovNft`: butane/types.AssetClass,<br>`MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalGov → Publish | `GovNft`: butane/types.AssetClass,<br>`MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalParams → Withdraw | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash,<br>`FeeToken`: butane/types.AssetClass,<br>`PriceFeedScriptHash`:<br>aiken/crypto.ScriptHash, |
| Synthetics → ExternalParams → Publish | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash,<br>`FeeToken`: butane/types.AssetClass,<br>`PriceFeedScriptHash`:<br>aiken/crypto.ScriptHash, |
| Synthetics → ExternalRates → Withdraw | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalRates → Publish | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalStaking → Withdraw | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalStaking → Publish | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalTreasuryActions → Withdraw | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash, |

| | |
|---|---|
| | `PriceFeedScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalTreasuryActions → Publish | `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, `PriceFeedScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalTreasuryOps → Withdraw | `GovNft`: butane/types.AssetClass, `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, `FeeToken`: butane/types.AssetClass, |
| Synthetics → ExternalTreasuryOps → Publish | `GovNft`: butane/types.AssetClass, `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, `FeeToken`: butane/types.AssetClass, |
| Synthetics → ExternalUnderlying → Withdraw | `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → ExternalUnderlying → Publish | `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, |
| Synthetics → Synthetics → Withdraw | `FeeToken`: butane/types.AssetClass, `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, `LeftoversScriptHash`: aiken/crypto.ScriptHash, `PriceFeedScriptHash`: aiken/crypto.ScriptHash, `SyntheticsAuxHash`: aiken/crypto.ScriptHash, |
| Synthetics → Synthetics → Publish | `FeeToken`: butane/types.AssetClass, `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, `LeftoversScriptHash`: aiken/crypto.ScriptHash, `PriceFeedScriptHash`: aiken/crypto.ScriptHash, `SyntheticsAuxHash`: aiken/crypto.ScriptHash, |
| Synthetics → SyntheticsAux → Withdraw | `MintScriptHash`: aiken/crypto.ScriptHash, `SpendScriptHash`: aiken/crypto.ScriptHash, `ParamsScriptHash`: aiken/crypto.ScriptHash, `RatesScriptHash`: aiken/crypto.ScriptHash, `GovScriptHash`: aiken/crypto.ScriptHash, `TreasuryActionsScriptHash`: aiken/ |

| | |
|---|---|
| | crypto.ScriptHash,<br>`TreasuryOpsScriptHash`:<br>aiken/crypto.ScriptHash,<br>`PriceFeedScriptHash`:<br>aiken/crypto.ScriptHash,<br>`StakeScriptHash`: aiken/crypto.ScriptHash,<br>`UnderlyingScriptHash`:<br>aiken/crypto.ScriptHash, |
| Synthetics → SyntheticsAux → Publish | `MintScriptHash`: aiken/crypto.ScriptHash,<br>`SpendScriptHash`: aiken/crypto.ScriptHash,<br>`ParamsScriptHash`: aiken/crypto.ScriptHash,<br>`RatesScriptHash`: aiken/crypto.ScriptHash,<br>`GovScriptHash`: aiken/crypto.ScriptHash,<br>`TreasuryActionsScriptHash`: aiken/crypto.ScriptHash,<br>`TreasuryOpsScriptHash`:<br>aiken/crypto.ScriptHash,<br>`PriceFeedScriptHash`:<br>aiken/crypto.ScriptHash,<br>`StakeScriptHash`: aiken/crypto.ScriptHash,<br>`UnderlyingScriptHash`:<br>aiken/crypto.ScriptHash, |

A2. **Redeemers**.

| **Validator Name** | **Redeemer** |
|---|---|
| Leftovers → Leftovers → Spend | butane/types.CDPCredentialVerifier |
| PriceFeed → PriceFeed → Withdraw | butane/types.PriceFeedRedeemer |
| PriceFeed → PriceFeed → Publish | Data |
| Synthetics → DatumType → Withdraw | butane/types.MonoDatum |
| Synthetics → ExternalGov → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → ExternalGov → Publish | Data |
| Synthetics → ExternalParams → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → ExternalParams → Publish | Data |
| Synthetics → ExternalRates → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → ExternalRates → Publish | Data |

| | |
|---|---|
| Synthetics → ExternalStaking → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → ExternalStaking → Publish | Data |
| Synthetics → ExternalTreasuryActions → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → ExternalTreasuryActions → Publish | Data |
| Synthetics → ExternalTreasuryOps → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → ExternalTreasuryOps → Publish | Data |
| Synthetics → ExternalUnderlying → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → ExternalUnderlying → Publish | Data |
| Synthetics → Synthetics → Withdraw | butane/types.PolicyRedeemer |
| Synthetics → Synthetics → Publish | Data |
| Synthetics → SyntheticsAux → Withdraw | butane/types.AuxilliaryRedeemer |
| Synthetics → SyntheticsAux → Publish | Data |

A3. **Datums**.

| **Validator Name** | **Datum** |
|---|---|
| Leftovers → Leftovers → Spend | butane/types.LeftoversDatum |

A4. **Definitions**.

**Bool** := {

    False
  | True

}

**ByteArray** ByteArray

**Data** Data

**Int** Int

**List<a>** $\{x_n \in \text{a}\}_{n=0}^{\infty}$

**Option<a>** := {

    Some(a)
  | None

}

**Tuple<a, b>** $\{x_n \in \}_{n=0}^{\infty}$

**aiken/crypto.DataHash** ByteArray

**aiken/crypto.ScriptHash** ByteArray

**aiken/crypto.VerificationKey** ByteArray

**aiken/crypto.VerificationKeyHash** ByteArray

**aiken/interval.IntervalBound<a>** := {

```
    IntervalBound {
        bound_type := aiken/interval.IntervalBoundType<a>,
        is_inclusive := Bool,
    }
```

}

**aiken/interval.IntervalBoundType<a>** := {

```
     NegativeInfinity
   | Finite(a)
   | PositiveInfinity
```

}

**butane/types.ActiveParams** := {

```
    ActiveParams {
        collateral_assets := List<butane/types.AssetClass>,
        weights := List<Int>,
        denominator := Int,
        minimum_outstanding_synthetic := Int,
        max_proportions := List<butane/types.BasisPoints>,
        max_liquidation_return := butane/types.BasisPoints,
        treasury_liquidation_share := butane/types.BasisPoints,
        redemption_share := butane/types.BasisPoints,
        fee_token_discount := Option<butane/types.BasisPoints>,
        minimum_rate_duration := butane/types.PosixTime,
        maximum_interest_increase := butane/types.WholePercentageBasisPoints,
        maximum_interest := butane/types.WholePercentageBasisPoints,
        underlying := List<List<butane/types.UnderlyingAsset>>,
        underlying_redemption_fee := butane/types.BasisPoints,
        rates_controller := cardano/address.Credential,
    }
```

}

**butane/types.AssetClass** := {

```
    AssetClass {
        policy_id := cardano/assets.PolicyId,
        asset_name := cardano/assets.AssetName,
    }
```

}

**butane/types.AuxilliaryRedeemer** := {

```
  CollectVoidedCDP {
      verifier := butane/types.CDPCredentialVerifier,
  }
| BadDebt {
      treasury_out_idx := Int,
  }
| ParamsMint {
      oidx := Int,
  }
| ParamsGovUpdate {
      oidx := Int,
      reverse_order := Bool,
  }
| ParamsRatesUpdate {
      oidx := Int,
      action := butane/types.ParamsRatesAction,
      synthetic := cardano/assets.AssetName,
  }
| ParamsVoid
| GovernanceIssue {
      output_idx := Int,
  }
| ExternalGovernance
| TreasurySpend {
      expected_output_idx := Int,
      gov_inp_idx := Int,
      inputs_from_fees := Bool,
  }
| TreasuryMint {
      expected_output_idx := Int,
  }
| TreasuryCreateDebt {
      expected_output_idx := Int,
      synth := cardano/assets.AssetName,
  }
| TreasuryRedemption {
      lock_burned := Option<Bool>,
      inputs_from_fees := Bool,
      synth := cardano/assets.AssetName,
  }
| TreasuryDelegate
| TreasuryGetRewards
| Compatibility {
      inner := butane/types.CompatRedeemer,
  }
| Staking(butane/types.StakingRedeemer)
| Underlying {
      synthetic := cardano/assets.AssetName,
  }
```

```
    | TreasuryDeposit {
        expected_output_idx := Int,
      }
    | TreasuryGenesisDeposit {
        expected_output_idx := Int,
      }
}
butane/types.BasisPoints := {
    BasisPoints {
      value := Int,
    }
}
butane/types.CDPCredential := {
    AuthorizeWithPubKey(aiken/crypto.VerificationKeyHash,aiken/crypto.VerificationKey)
  | AuthorizeWithConstraint(butane/types.Constraint)
}
butane/types.CDPCredentialVerifier := {
    AuthorizingDirectly(butane/types.CDPSubVerifier)
  | AuthorizingOtherWithSignature {
      other := butane/types.ConstraintCredential,
      sub_verifier := butane/types.CDPSubVerifier,
      signature := ByteArray,
    }
}
butane/types.CDPSubVerifier := {
    AuthorizedWithExtraSigs
  | AuthorizedWithInputsOref(cardano/transaction.OutputReference)
  | AuthorizedWithWithdrawal
}
butane/types.CompatRedeemer := {
    CompatLock {
      oidx := Int,
    }
  | CompatUnlock {
      soidx := Option<Int>,
    }
}
butane/types.Constraint := {
    MustSpendToken(butane/types.AssetClass)
  | MustWithdrawFrom(cardano/address.StakeCredential)
}
```

```
butane/types.ConstraintCredential := {
    ConstraintCredential {
        utxo := cardano/transaction.OutputReference,
        interval := cardano/transaction.ValidityRange,
        constraint := butane/types.Constraint,
    }
}
butane/types.FakeOutput := {
    FakeOutput {
        address := cardano/address.Address,
        value := List<(Int, Int)>,
        datum := cardano/transaction.Datum,
        reference_script := Option<ByteArray>,
    }
}
butane/types.FeeType := {
    FeeInSynthetic
  | FeeInFeeToken {
        fee_token_idx := Int,
    }
}
butane/types.Feed<a, b> := {
    Feed {
        data := butane/types.PriceFeed,
        extra := b,
    }
}
butane/types.GovAction := {
    NewParamsAuth {
        params := butane/types.Params,
        interest := butane/types.InterestParams,
        staking := butane/types.StakingParams,
        asset := cardano/assets.AssetName,
    }
  | UpdateParamsAuth {
        asset := cardano/assets.AssetName,
        action := butane/types.ParamsGovAction,
    }
  | TreasurySpendAuth {
        out := butane/types.FakeOutput,
    }
  | TreasuryMintAuth {
        asset := cardano/assets.AssetName,
```

```
        amount := Int,
    }
  | TreasuryCreateDebtAuth {
        debt := butane/types.TreasuryDebt,
    }
  | TextProposalAuth {
        text := ByteArray,
    }
  | TreasuryStakeUpdate {
        delegatee := cardano/certificate.StakePoolId,
    }
  | ExternalScript {
        other_script := aiken/crypto.ScriptHash,
        other_data := Data,
    }
  | GovNewCompat {
        upgrade_policy := cardano/assets.PolicyId,
    }
}
```

**butane/types.InterestParams** := {

```
    InterestParams {
        current_rate := butane/types.BasisPoints,
        last_updated := butane/types.PosixTime,
        start_time := butane/types.PosixTime,
        accumulated_interest := List<butane/types.UncheckedRational>,
    }
}
```

**butane/types.LeftoversDatum** := {

```
    LeftoversDatum {
        owner := butane/types.CDPCredential,
    }
}
```

**butane/types.MonoDatum** := {

```
    ParamsWrapper {
        params := butane/types.Params,
    }
  | CDP {
        owner := butane/types.CDPCredential,
        synthetic_asset := cardano/assets.AssetName,
        synthetic_amount := Int,
        start_time := butane/types.PosixTime,
        interest_debit := butane/types.UncheckedRational,
        max_interest := butane/types.WholePercentageBasisPoints,
        paid_interest := butane/types.UncheckedRational,
    }
```

```
    | GovDatum {
        gov := butane/types.GovAction,
      }
    | TreasuryDatum {
        treas := butane/types.TreasuryDatum,
      }
    | CompatLockedTokens
    | StakedSynthetics {
        owner := butane/types.CDPCredential,
        synthetic_asset := cardano/assets.AssetName,
        start_time := butane/types.PosixTime,
        interest_debit := butane/types.UncheckedRational,
      }
    | TreasuryUnderlying {
        synthetic := cardano/assets.AssetName,
        corresponding_minted := Int,
      }
}
```

**butane/types.Params** := {

```
    LiveParams {
        params := butane/types.ActiveParams,
      }
    | LiveStakingParams {
        params := butane/types.StakingParams,
      }
    | LiveInterestParams {
        params := butane/types.InterestParams,
      }
    | VoidedParams
}
```

**butane/types.ParamsGovAction** := {

```
    NewCollateral {
        index := Int,
        collateral_asset := butane/types.AssetClass,
        weight_numerator := Int,
        weight_denominator := Int,
        max_proportion := butane/types.BasisPoints,
      }
    | UpdateWeight {
        collateral_asset_idx := Int,
        weight_numerator := Int,
        weight_denominator := Int,
      }
    | UpdateMinOutstanding {
        min_outstanding := Int,
      }
```

```
    | UpdateMaxProportions {
        max_proportions := List<butane/types.BasisPoints>,
      }
    | UpdateMaxLiquidationReturn {
        max_return := butane/types.BasisPoints,
      }
    | UpdateTreasuryLiquidationShare {
        share := butane/types.BasisPoints,
      }
    | UpdateRedemptionShare {
        share := butane/types.BasisPoints,
      }
    | UpdateFeeTokenDiscount {
        discount := Option<butane/types.BasisPoints>,
      }
    | UpdateUnderlyingRedemptionFee {
        redemption_fee := butane/types.BasisPoints,
      }
    | UpdateUnderlyingAssets {
        underlying := List<List<butane/types.UnderlyingAsset>>,
      }
    | UpdateRatesController {
        rates_controller := cardano/address.Credential,
      }
    | UpdateMaximumInterest {
        max_interest := butane/types.WholePercentageBasisPoints,
      }
    | UpdateMaximumInterestIncrease {
        max_interest_increase := butane/types.WholePercentageBasisPoints,
      }
    | UpdateMinimumRateDuration {
        minimum_rate_duration := butane/types.PosixTime,
      }
}
butane/types.ParamsRatesAction := {
    UpdateInterest {
        interest_rate := butane/types.BasisPoints,
    }
  | UpdateStakingInterest {
        interest_rate := butane/types.BasisPoints,
    }
}
butane/types.PolicyRedeemer := {
    SyntheticsMain {
        spends := List<butane/types.SpendAction>,
```

```
        creates := List<Int>,
    }
  | Auxilliary
}
```

**butane/types.PosixTime** Int

**butane/types.PriceFeed** := {

```
    PriceFeed {
        collateral_prices := List<Int>,
        synthetic := cardano/assets.AssetName,
        denominator := Int,
        validity := cardano/transaction.ValidityRange,
    }
}
```

**butane/types.PriceFeedRedeemer** $\{x_n \in$ butane/types.Feed<butane/types.PriceFeed, ByteArray> $\}_{n=0}^{\infty}$

**butane/types.SpendAction** := {

```
    SpendAction {
        spend_type := butane/types.SpendType,
        params_idx := Int,
        staking_params_idx := Int,
        interest_params_idx := Int,
        fee_type := butane/types.FeeType,
    }
}
```

**butane/types.SpendType** := {

```
    LiquidateCDP
  | PartialLiquidateCDP {
        repay_amount := Int,
    }
  | LeftoversLiquidateCDP
  | RepayCDP(butane/types.CDPCredentialVerifier)
  | RedeemCDP {
        repay_amount := Int,
    }
}
```

**butane/types.StakingAction** := {

```
    StakeSynthetics {
        staked_amount := Int,
    }
  | UnstakeSynthetics {
        verifiers := List<butane/types.CDPCredentialVerifier>,
    }
```

```
}
butane/types.StakingParams := {
    StakingParams {
        current_rate := butane/types.BasisPoints,
        last_updated := butane/types.PosixTime,
        start_time := butane/types.PosixTime,
        accumulated_interest := butane/types.UncheckedRational,
        start_time_offset := butane/types.PosixTime,
    }
}
butane/types.StakingRedeemer := {
    StakingRedeemer {
        synth := cardano/assets.AssetName,
        action := butane/types.StakingAction,
    }
}
butane/types.TreasuryDatum := {
     TreasuryWithDebt {
         debt := butane/types.TreasuryDebt,
         creation_time := Option<butane/types.PosixTime>,
     }
   | TreasuryFromGenesis
   | TreasuryFromFees
   | TreasuryOnlyRedeem
   | TreasuryOnlySpend {
         unlock_time := Option<butane/types.PosixTime>,
     }
}
butane/types.TreasuryDebt := {
    TreasuryDebt {
        amount := Int,
        asset := cardano/assets.AssetName,
    }
}
butane/types.UncheckedRational := {
    UncheckedRational {
        numerator := Int,
        denominator := Int,
    }
}
butane/types.UnderlyingAsset := {
```

```
    UnderlyingAsset {
        asset := butane/types.AssetClass,
        conversion_rate := butane/types.UncheckedRational,
    }
}
butane/types.WholePercentageBasisPoints := {
    BasisPoints {
        value := Int,
    }
}
cardano/address.Address := {
    Address {
        payment_credential := cardano/address.PaymentCredential,
        stake_credential := Option<cardano/address.StakeCredential>,
    }
}
cardano/address.Credential := {
    VerificationKey(aiken/crypto.VerificationKeyHash)
  | Script(aiken/crypto.ScriptHash)
}
cardano/address.PaymentCredential := {
    VerificationKey(aiken/crypto.VerificationKeyHash)
  | Script(aiken/crypto.ScriptHash)
}
cardano/address.StakeCredential := {
    Inline(cardano/address.Credential)
  | Pointer {
        slot_number := Int,
        transaction_index := Int,
        certificate_index := Int,
    }
}
cardano/assets.AssetName ByteArray
cardano/assets.PolicyId ByteArray
cardano/certificate.StakePoolId ByteArray
cardano/transaction.Datum := {
    NoDatum
  | DatumHash(aiken/crypto.DataHash)
  | InlineDatum(Data)
```

```
}
cardano/transaction.OutputReference := {
    OutputReference {
        transaction_id := ByteArray,
        output_index := Int,
    }
}
cardano/transaction.ValidityRange := {
    Interval {
        lower_bound := aiken/interval.IntervalBound<Int>,
        upper_bound := aiken/interval.IntervalBound<Int>,
    }
}
```