V2.3 LBE Specification

1. Overview

The Permissionless Liquidity Bootstrapping Event (LBE) is a major initiative by Minswap, supported by Project Catalyst. The goal of the LBE is to facilitate a fair-price liquidity bootstrapping event for projects that wish to launch a token on Cardano.

For more information, refer to:

- Minswap Liquidity Bootstrapping for DAOs.
- Introducing the Minswap Launch Bowl.

2. The LBE Parameters:

Various parameters are to be determined for projects to configure before launching their tokens. These will depend on the project's needs, and must also reflect their communities' needs.

Therefore, we propose the following parameters:

- **Base Asset:** `(require)` Asset project aims to list. Example: MIN, FLDT, ...
- **Reserve Base**: `(require)` The amount of *Base Asset* that project provided. The main goal of this Reserve Base is to create an AMM Pool.
- **Raise Asset:** `(require)` Asset project aims to raise. Example: ADA, USDC, ...
- **Start Time**: `(require)` The start time of the discovery phase.
- **End Time**: `(require)` The end time of the discovery phase.
- **Owner**: `(require)` The address of project's owner.
- **Receiver**: `(require)` Address for receiving the project's LP tokens.
- **Receiver Datum**: `(optional)` The Output's Datum after returning the LP token to the project.
- **Pool Allocation**: `(require)` The rule defines how many funds will go to pool.
- **Minimum Order Raise**: `(optional)` Minimum amount in a single order.
- **Minimum Raise**: `(optional)` The minimum amount expected to raise. If the amount of *Raise Asset* is lower than this threshold, the LBE will be canceled.
- **Maximum Raise**: `(optional)` The maximum amount of raise asset expected to raise. If the amount of *Raise Asset* exceed this threshold, the excess amount should be returned to the users.
- **Penalty Config**: `(optional)` Penalty on participants for early withdrawal.
- **Revocable**: `(optional)` The project owner has the ability to cancel the LBE during the discovery phase.
- **Pool Base Fee**: `(require)` The Numerator of Trading Fee on AMM Pool.

3. Specifications

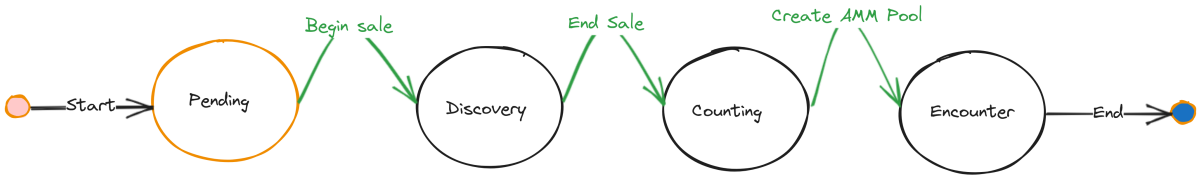
3.1. Architecture

3.1.1 LBE State Transition:

- Happy-case scenario: the LBE is successful:
  1. The new LBE has been initialized (Pending Phase).
  2. During the discovery phase, users have the ability to create orders (Discovery Phase).
  3. Once the discovery phase concludes, order creation is no longer possible. At this point, the system starts gathering Manager, Seller, and Order UTxOs for calculation, as well as funds for setting up the

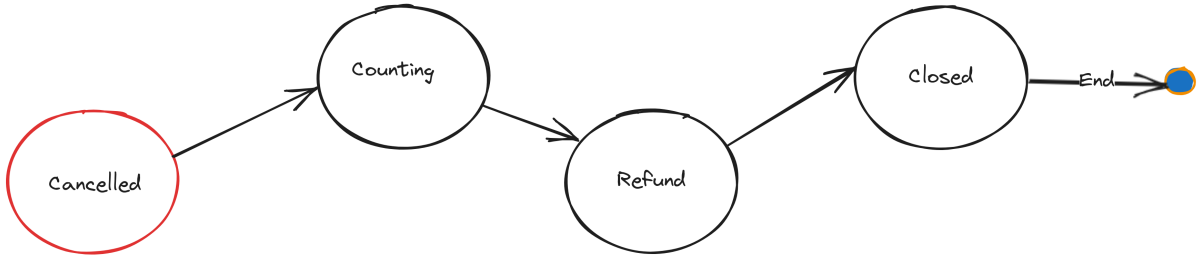
AMM pool (Counting Phase).

4. Upon successful creation of the pool, users start receiving their LP tokens (Encounter Phase).



(Happy-case) LBE State Transition

- Cancelled-case scenario: the LBE is cancelled:
  1. The LBE has been cancelled (Cancelled).
  2. After the LBE is cancelled, it's no longer possible to create orders. The system then begins to gather Manager, Seller, and Order UTxOs for computations, along with funds intended for refunding to users (Counting).
  3. Refunding to users (Refund).
  4. Once all users have received their refunds, the project owner is able to close the LBE (Close).

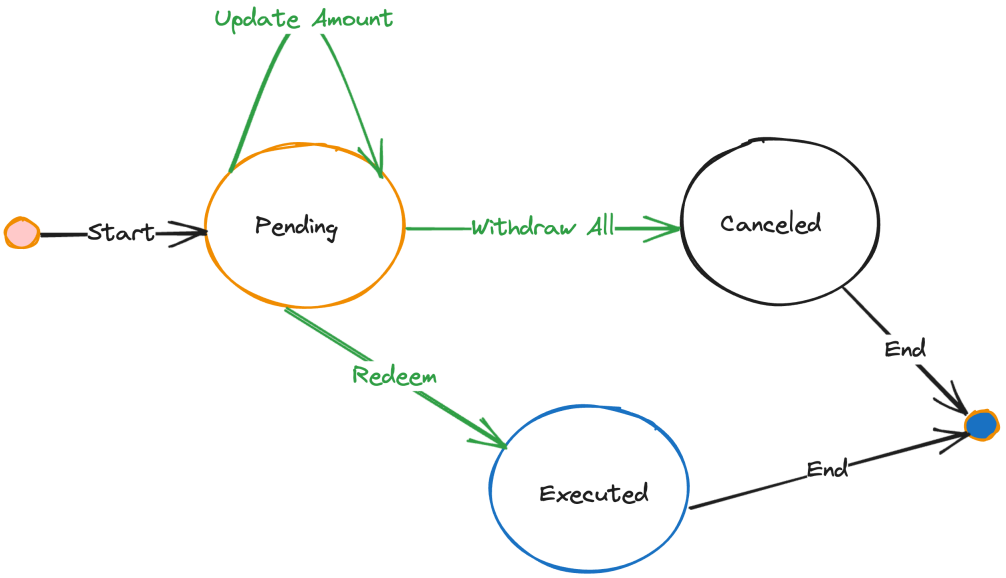


(Cancelled-case) LBE State Transition

- Scenarios leading to the cancellation of LBE:
  1. Prior to the discovery phase, the project owner intends to cancel LBE.
  2. During the discovery phase, the project owner plans to terminate the LBE.
  3. Before the encounter phase, the AMM Pool has already been created.

3.1.2 Order State Transition

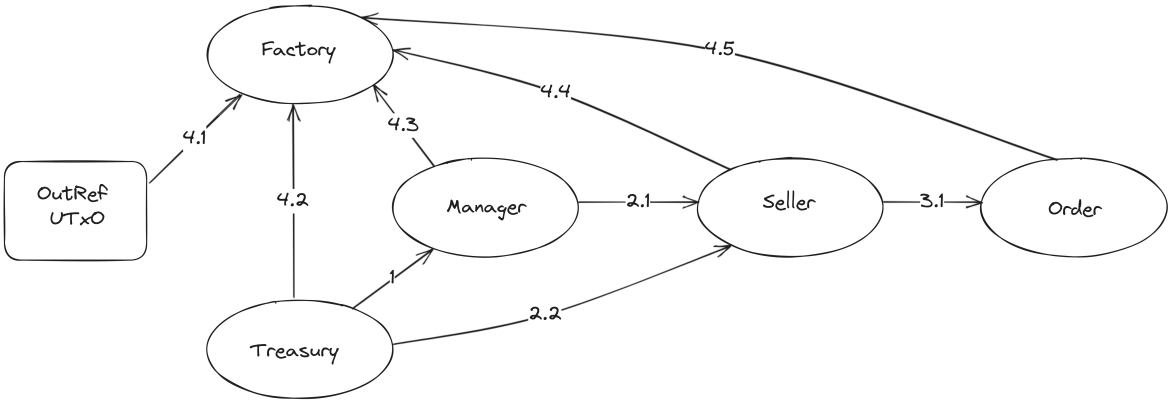
- Please refer to the following diagram for each deposit:



Order State Transition

3.1.3 Parameterized Validators

The LBE validators consist of a collection of parameterized validators. It's crucial to apply these parameters to each validator in a specific sequence.



- Order Validator:

```
-- ValidatorHash of Seller Validator
seller_hash: ValidatorHash
```

  - Each Order UTxO functions as a purchase receipt for a user. It logs the order details, secures the user's funds, and stands ready for the Seller to collect.

- Seller Validator:

```
-- ValidatorHash of Manager Validator
manager_hash: ValidatorHash
-- ValidatorHash of Treasury Validator
treasury_hash: ValidatorHash
```

  - Seller is similar to a seller giving a bill to a customer. The Seller understands how much Raise Asset they've gotten and the quantity of the penalty.

    - Take responsibility for validate logic create | update order.
- Manager Validator:

```
-- ValidatorHash of Treasury Validator
treasury_hash: ValidatorHash
```

    - Every LBE has 1 Manager UTxO.
    - Manager is similar to a supervisor overseeing sellers. They're aware of the total number of sellers, and after counting all the sellers, they can determine the total funds raised and the total penalty amount.
    - Take responsibility for validate logic for add more sellers and collect sellers.
- Treasury Validator:
    - Each Treasury UTxO present for different LBE, 1 LBE has 1 Treasury UTxO.
    - The Treasury Datum holds LBE parameters.
    - With Treasury Datum, we can know the state of LBE.
    - In discovery phase, Treasury UTxO is a reference input, ensure Manager, Sellers work correctly.
    - Treasury UTxO use as input to create AMM pool.
    - Act like a Treasury. Ensure the use of these funds is correct.
- Factory Validator:

```
-- a reference of an UTxO,
-- ensure redeemer `Initialization` be called only one.
out_ref: OutputReference
-- ValidatorHash of Treasury Validator
treasury_hash: ValidatorHash
-- ValidatorHash of Manager Contract
manager_hash: ValidatorHash
-- ValidatorHash of Seller Validator
seller_hash: ValidatorHash
-- ValidatorHash of Order Validator
order_hash: ValidatorHash
```

    - Each `Factory` UTxO is a node of `Factory Linked List` (https://github.com/Anastasia-Labs/aiken-linked-list?tab=readme-ov-file)
    - Factory is a Minting Policy, that response for minting all Authenticate Tokens.
    - Factory Spend Validator will validate logic Create or Close a LBE.
    - Additionally, the Factory is a representative that handles the logic of checking the correctness of *Collecting* | *Redeeming* | *Refunding* Orders.

3.2 Interaction Flow for Validators

3.2.1 Happy-Case Flow

1. Project Owner start LBE by creating Treasury, Manager, Sellers and adding 1 node to Factory Linked List.
2. Users create Orders.
3. Manager collects all Sellers.
4. Treasury collects Manager.
5. Treasury collects all Orders.

6. Treasury create AMM Pool, also return LP Tokens to Project.
7. Treasury return LP Tokens to Users.

3.2.2 Cancelling Flow

1. Project Owner updates Treasury `is_cancelled == True`
2. Manager collects all Sellers.
3. Treasury collects Manager.
4. Treasury collects all Orders if any.
5. Treasury refund to Users if any.
6. Project Owner close event.

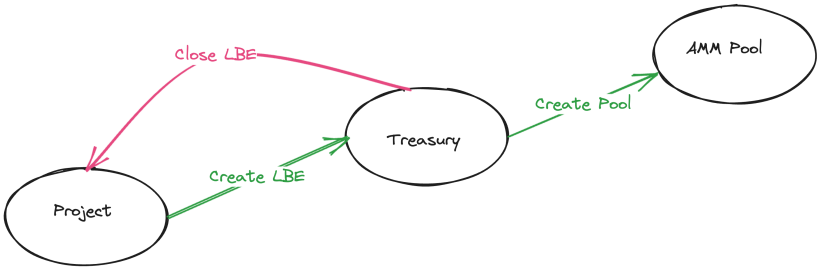
3.3 Tokens

Notes:

- There are 5 types of authenticate tokens corresponding exactly to 5 validators.
- Each Legitimate UTxO must always contains exactly 1 Authenticate Token. Example: Order UTxO contains 1 `Order Token.`

Token	Policy ID	Asset Name	Explain
`Factory Token`	`Factory Policy ID`	`factory_auth_an`	check legitimate `Factory` UTxO
`Treasury Token`	`Factory Policy ID`	`treasury_auth_an`	check legitimate `Treasury` UTxO
`Manager Token`	`Factory Policy ID`	`manager_auth_an`	check legitimate `Manager` UTxO
`Seller Token`	`Factory Policy ID`	`seller_auth_an`	check legitimate `Seller` UTxO
`Order Token`	`Factory Policy ID`	`order_auth_an`	check legitimate `Order` UTxO
`LP Token`	`AMM Authen Policy ID`	`lp asset name`	LP Tokens for redeeming to users
`AMM Pool Authen Token`	`AMM Authen Policy ID`	`amm_pool_auth_asset_name`	check legitimate `AMM Pool` UTxO
`AMM Factory Authen Token`	`AMM Authen Policy ID`	`amm_factory_auth_asset_name`	check legitimate `AMM Factory` UTxO

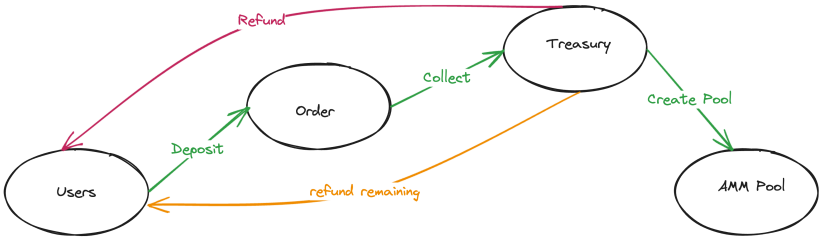
- Besides Authentication tokens, *Base Asset, Raise Asset, LP Token* are essential to successfully setting up an LBE. The three diagrams that follow illustrate the cash flow associated with these tokens:
  - Base Asset:
    - The project deposits their *Base Asset* funds into the Treasury to create a LBE.
    - Pass-Case: Treasury use *Base Asset* funds to create AMM Pool.
    - Fail-Case: Treasury return *Base Asset* funds to Project to close LBE.



Base Asset Cash-Flow Diagram

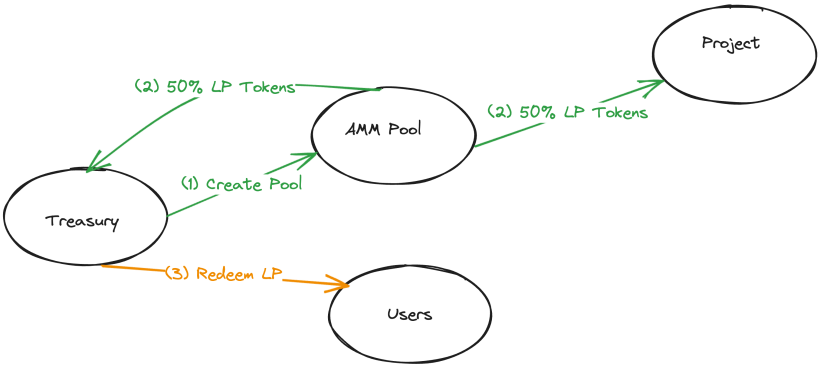
  - Raise Asset:

- Users deposit their *Raise Asset* funds into Order to create an order.
- After discovery phase or the LBE is cancelled, Treasury will collect funds from Orders.
- Pass-Case: Treasury use *Raise Asset* funds to create AMM Pool.
- Fail-Case: Treasury return *Raise Asset* funds to Users.
- There is one edge case where users may receive a remaining *Raise Asset* amount. This occurs when the total raised exceeds the *Maximum Raise* amount. In such instances, each user will receive the same percentage of their deposit as LP Tokens, and the remaining *Raise Asset* will be returned to them. For instance, if project A aims to raise a maximum of 10M ADA and users deposit a total of 11M ADA, then 10M ADA will be used to create a pool. The remaining 1M ADA will be returned to the users.



Raise Asset Cash-Flow Diagram

- LP Token:
  - Treasury use *Base Asset, Raise Asset* funds to create Pool.
  - 50% LP Tokens return to Project and 50% remains back to Treasury.
  - Treasury redeem LP Tokens to Users.



LP Token Cash-Flow Diagram

3.4 Validator

3.4.1 Order Validator

3.4.1.1 Structure

- Parameters: *(described above)*
- Purpose: Spend
- ▼ Redeemer:

```
pub type OrderRedeemer {
  // Update Order Amount
  UpdateOrder
  CollectOrder
```

```
  RedeemOrder
}
```

▼ Datum:

```
OrderDatum {
  // ValidatorHash of Factory Validator
  factory_policy_id: PolicyId,
  // Asset aims to list
  base_asset: Asset,
  // Asset aims to raise
  raise_asset: Asset,
  // Update order require owner authorize
  // System will redeem | refund to owner address (should be Pubkey Address)
  owner: Address,
  // Amount of Raise Asset
  amount: Int,
  // Default is False. Become True after Order is collected by Treasury
  is_collected: Bool,
  // Amout of Raise Asset being penalty.
  // Default is zero.
  penalty_amount: Int,
}
```

3.4.1.2 Notes

- Everyone can deposit *Raise Asset* into the Order Validator Address to participate in the LBE.
- The Order Validator manages user's order information, including funds and details. The Order UTxO maintains the user's funds, while the Order Datum stores details such as "Owner, Amount, Total Penalty".
- Users can create or update orders during the *discovery phase* as long as the LBE is not *cancelled.*

▼ 3.4.1.3 Validation

- `UpdateOrder`:
  - This redeemer is invoked when a user wants to update their orders. Users can either increase or decrease the order amount. The transaction will use the previous order input and create a new order output with the updated amount. If a user wishes to withdraw the entire amount, no new order output is created. If a user reduces the order amount after the *Penalty Start Time* begins, a penalty will be applied.
  - The validation is delegate to `Seller` Validator. This transaction must include a Seller UTxO and Seller redeemer `UsingSeller` should be invoked → Check Seller Validation for details.
- `CollectOrder`:
  - This transaction involves looking up uncollected orders and using them to accumulate *Raise Asset* funds into the Treasury Value. The process is complete when all orders have been collected. This is confirmed when `TreasuryDatum` ensures that `collected_fund == reserve_raise + total_penalty`.
  - The validation is delegate to `Factory` Withdrawal Validator → Check Factory Validation for details.
- `RedeemOrder`:
  - This transaction will return *Raise Asset* to Order's Owner if LBE is cancelled.
  - This transaction will return *LP Tokens* to Order's Owner if LBE is successful and AMM pool is created.
  - The validation is delegate to `Factory` Withdrawal Validator → Check Factory Validation for details.

3.4.2 Seller Validator

3.4.2.1 Structure

- Parameters: *(described above)*
- Purpose: `Spend`

▼ Redeemer

```
pub type SellerRedeemer {
  // use a seller for create | update orders
  UsingSeller
  // Collect Seller UTxOs
  CountingSeller
}
```

▼ Datum

```
pub type SellerDatum {
  // ValidatorHash of Factory Validator
  factory_policy_id: PolicyId,
  // Fee will be return to seller's owner (PubKey Address)
  owner: Address,
  // Asset aims to list
  base_asset: Asset,
  // Asset aims to raise
  raise_asset: Asset,
  // Total amount this seller manage. It can be negative
  amount: Int,
  // Total penalty amount this seller manage. It can be negative
  penalty_amount: Int,
}
```

3.4.2.2 Notes

- Every action that changes orders, such as creating new ones or updating, must go through Seller Validation.
- The `amount` and `penalty_amount` values in Seller Datum can be negative.
- Total Reserve Raise calculate by collecting all Sellers:

$$\text{total\_reserve\_raise} = \sum_{s \in \text{sellers}} (s.\text{amount} + s.\text{penalty\_amount})$$

3.4.2.3 Validation

- `UsingSeller`:
  - The LBE is not cancelled.
  - Validate Output:
    - Paying Order Outputs correctly!
    - Paying Seller Output correctly!
  - Find exactly 1 treasury reference input to get information about LBE(Is it in discovery phase)
    - seller, orders have the same LBE ID with treasury's LBE ID.
    - Valid time range of transaction in discovery phase.

        - penalty config of this LBE.
    - Find n Order inputs then calculate the total amount and penalty in all of those inputs.
    - Find m Order outputs then calculate the total amount and penalty in all of those outputs.
    - The Validator will calculate the amount deposited or withdrawn from a transaction by comparing the total amounts in order inputs and outputs.
        - This delta is used to update the amount in the Seller Datum.
        - The same delta is also applied to calculate the penalty (determined by the amount of LBE withdrawn during penalty time) generated in this transaction.
    - Check the number of `Order Tokens` minted or burned based on the number of Order Input and Order Output.
- `CountingSeller`:
    - In this action, the Manager will collect the Sellers. Upon doing so, the Manager will gather information obtained from the Seller.
    - The validation is delegate to `Manager` Validator. This transaction must include a Manager UTxO and Manager redeemer `ManageSeller` should be invoked → Check Manager Validation for details.

3.4.3 Manager Validator

3.4.3.1 Structure

- Parameters: *(described above)*
- Purpose: `Spend`

▼ Redeemer:

```
pub type ManagerRedeemer {
  // Add more Seller UTxOs
  AddSellers
  // Collect Seller UTxOs
  CollectSellers
  // Collect Manager UTxO
  SpendManager
}
```

▼ Datum:

```
pub type ManagerDatum {
  // ValidatorHash of Factory Validator
  factory_policy_id: PolicyId,
  // Asset aims to list
  base_asset: Asset,
  // Asset aims to raise
  raise_asset: Asset,
  // The current number of seller utxo
  seller_count: Int,
  // The total amount the manager collected from sellers. \
  //  Initially is zero. Changes when collecting sellers.
  reserve_raise: Int,
  // The total penalty amount the manager collected from sellers. \
  //  Initially is zero. Changes when collecting sellers.
```

```
  total_penalty: Int,
}
```

3.4.3.2 Notes

- The Manager creates the Seller before the discovery phase ends, collects all Sellers, calculates their total raise and penalty. After collection, the Manager has the total raise and penalty for this LBE.

3.4.3.3 Validation

- `AddSellers`:
  - `seller_count` increase
  - Mint `Seller Token`
- `CollectSellers`:
  - Burn `Seller Token`
  - `seller_count` decrease:

$$seller\_count\_out = seller\_count\_in - seller\_input\_count$$

  - The manager will aggregate the amounts from the Seller to know the total raise of LBE:

$$reserve\_raise\_out = reserve\_raise\_in + \sum(seller.amount)$$

  - The manager will calculate the seller's total penalty amount to determine the overall penalty of the LBE:

$$total\_penalty\_out = total\_penalty\_in + \sum(seller.penalty\_amount)$$

- `SpendManager`:
  - With this redeemer, after collecting all Sellers, the Treasury will collect the Manager to know the total amount of raise and penalty of this LBE.
  - Validate there is exactly 1 Treasury Input using the `CollectManager` redeemer. This action will delegate to the Treasury Validator with the redeemer being `CollectManager`.

3.4.4 Treasury Validator

3.4.4.1 Structure

- Parameters: *(described above)*
- Purpose: `Spend`

▼ Redeemer:

```
pub type TreasuryRedeemer {
  // collect manager to update reserve_raise and total_penalty
  CollectManager
  // collect amount, penalty_amount from orders
  CollectOrders
  // Create AMM Pool
  CreateAmmPool
  // redeem LP Tokens or refund if cancel
  RedeemOrders
  // Close event when LBE is cancelled and finished couting, refund.
  CloseEvent
  // Cancel the LBE base on reason
  CancelLBE { reason: CancelReason }
```

```
  // update LBE Parameters before discovery phase
  UpdateLBE
}
```

▼ Datum:

```
pub type TreasuryDatum {
  // Protocol Info:
  // ValidatorHash of Factory Validator
  factory_policy_id: PolicyId,
  // ValidatorHash of Manager Validator
  manager_hash: ValidatorHash,
  // ValidatorHash of Seller Validator
  seller_hash: ValidatorHash,
  // ValidatorHash of Order Validator
  order_hash: ValidatorHash,
  // --------------------------------
  // LBE Parameters
  // Asset aims to list
  base_asset: Asset,
  // Asset aims to raise
  raise_asset: Asset,
  // The start time of discovery phase
  start_time: Int,
  // The end time of discovery phase
  end_time: Int,
  // Project Owner Address
  owner: Address,
  // Address for receiving the project's LP tokens.
  receiver: Address,
  // The Datum of the receiver's output
  receiver_datum: ReceiverDatum,
  // The rule defines how many funds will go to pool
  pool_allocation: Int,
  // Minimum amount in a order.
  minimum_order_raise: Option<Int>,
  // The minimum amount expected to raise.
  // If raised amount less than this threshold, the LBE will be cancelled.
  minimum_raise: Option<Int>,
  // The maximum amount expected to raise.
  // If raised amount exceed this threadhold, \
  //   the excess amount should be returned to the users.
  maximum_raise: Option<Int>,
  // Amont of Base Asset use to create Pool.
  reserve_base: Int,
  penalty_config: Option<PenaltyConfig>,
  // POOL DATUM
  // - asset_a, asset_b, reserve_a, reserve_b will be validated by treasury depend
  //    how much fund it raise and treasury config
  // - pool_batching_stake_credential, total_liquidity, fee_sharing_numerator_opt,
  // =>  we just need 1 base_fee for base_fee_a_numerator and base_fee_b_numerator
  pool_base_fee: Int,
  // Indicates whether project's owner can cancel LBE during discovery phase.
  revocable: Bool,
```

```
  // --------------------------------
  // Treasury State:
  // Total fund accumulate by collecting orders (amount, penalty).
  // It increases when collect orders, descreases when redeem LP or refund
  collected_fund: Int,
  // Total Order Amount. Increasing when collect orders.
  reserve_raise: Int,
  // Total Order's penalty amount. Increasing when collect orders.
  total_penalty: Int,
  // Amount LP Tokens will distribute to users.
  total_liquidity: Int,
  // Indicates whether the LBE has cancelled.
  is_cancelled: Bool,
  // Indicates whether the manager has collected
  is_manager_collected: Bool,
}
```

3.4.4.2 Notes

3.4.4.3 Validation

- `CollectManager`:
  - All Sellers must be collected before collecting Manager ⇒ no need to check time range
  - Manager Input, Treasury Input must be the same LBE ID
  - The Treasury Values have no changes
  - The Treasury Output Datum must update `reserve_raise, total_penalty, is_manager_collected`
  - Burn 1 Manager Token
- `CollectOrders`:
  - Must collect at least `minimum_order_collected` orders per transaction or collect all remaining orders.
  - All Sellers, Manager have been collected.
  - All Inputs and Outputs must be legitimate by containing an Authen Token.
  - Treasury Input, Order Inputs must share the same LBE ID.
  - The length of Order Inputs equals the length of Order Outputs.
  - Transaction mint nothing.
  - *Notes:*
    - *Order Outputs should correspond to Order Inputs.*
    - *Collecting orders and cancelling LBE are independent operations.*
    - *There's no need to check the time range since it should collect sellers first.*
- `CreateAmmPool`:
  - Not cancelled yet.
  - Collected all orders.
  - Only mint AMM tokens related.
  - Paying 50% total LP Tokens to `Receiver` with correct `ReceiverDatum` .
  - AMM Pool reserve Token A base on formula:

$$\text{pool\_reserve\_a} = \frac{\text{lbe\_reserve\_a} \times \text{pool\_allocation}}{100}$$

- AMM Pool reserve Token B base on formula:

$$\text{pool\_reserve\_b} = \frac{\text{lbe\_reserve\_b} \times \text{pool\_allocation}}{100}$$

- Total LP Tokens distribute to `Receiver` :
  - `(poolDatum.total_liquidity - 10)` : *Total LP Tokens of LBE.*
  - `pool_allocation` : *the percentage of total funds to create AMM pool.*
  - `50%` : *the percentage of total funds of Users.*
  - `pool_allocation - 50` : the percentage of project's owner to create AMM Pool.

$$\text{receiver\_lp\_tokens} = (\text{pool\_allocation} - 50) \times \frac{(\text{poolDatum.total\_liquidity} - 10)}{\text{pool\_allocation}}$$

- `RedeemOrders` :
  1. Redeem LP Tokens:
     a. User will receive:
        i. `order_minimum_ada` *Lovelace*.
        ii. `total_lp * amount / reserve_raise` *LP Tokens*.
        iii. *If total raise funds exceed Maximum Raise threshold, user will receive amount of LP Tokens bellows:*

$$\left(\frac{\text{reserve\_raise} + \text{total\_penalty} - \text{maximum\_raise}}{\text{reserve\_raise}}\right) \times \text{amount}$$

  2. Refund:
     a. User will receive `order_minimum_ada` *Lovelace* + `(amount + penalty_amount)` *Raise Asset*.
- `CloseEvent` :
  - Ensure transaction has 2 Factory Inputs.
  - The validation is delegate to `Factory` Spend Validator → Check Factory Validation for details.
- `CancelLBE { reason: CancelReason }` :
  - The LBE has not cancelled yet.
  - Transaction mint nothing.
  - Ensure there's no cancellation after successfully creating an AMM Pool with LBE.
  - There are 3 reasons to cancel a LBE:
    1. Cancel by pool exist: showing that AMM Pool have been created.
    2. Cancel by not reach minimum raise: after discovery phase, the reserve raise amount not reach target.
    3. Cancel by owner:
       a. Before discovery phase start: Project's Owner can cancel LBE if need.
       b. After discovery phase start and `revocable == true` : Project's Owner can cancel LBE before discovery phase ended.
- `UpdateLBE` :
  - The LBE can only be updated before the start of the discovery phase.
  - The LBE can only be updated if it has not been cancelled.

- The Project Owner must authorize the transaction.
- Project Owner can update LBE Parameter fields except for the *Base Asset, Raise Asset* field. This is because the LBE ID cannot be updated once the LBE has been created.
- Transaction mint nothing.
- The new Treasury Output must follow the same validation as creating a new LBE (value, datum).

3.4.5 Factory Validator

3.4.4.1 Structure

- Parameters: *(described above)*
- Purpose: `Spend`

▼ Redeemer:

```
pub type FactoryRedeemer {
  Initialization
  CreateTreasury { base_asset: Asset, raise_asset: Asset }
  CloseTreasury { base_asset: Asset, raise_asset: Asset }
  MintManager
  MintSeller
  BurnSeller
  MintOrder
  MintRedeemOrders
  // Withdrawal
  ManageOrder
}
```

▼ Datum:

```
pub type FactoryDatum {
  head: ByteArray,
  tail: ByteArray,
}
```

3.4.4.2 Notes

- The Factory Minting Policy is responsible for minting authentic tokens.
- The Factory Spending Validator manages the Factory Linked List, used to create or close an LBE.
- The Factory Withdrawal Validator is an optimized solution.

3.4.4.3 Validation

- `Initialization`: Run once, initialize the factory linked list.
- `MintManager`: Mint or burn Manager Token.
- `MintSeller`: Mint Seller Token.
- `BurnSeller`: Burn Seller Token.
- `MintOrder`: Mint Order Token.
- `MintRedeemOrders`: Burn Order Tokens.
- `ManageOrder`: Optimize for collecting orders and redeeming orders.
- `CreateTreasury { base_asset: Asset, raise_asset: Asset }`: Create a new LBE, add node to Factory Linked List.
- `CloseTreasury { base_asset: Asset, raise_asset: Asset }`: Close LBE, remove node from Factory Linked List.

3.5 Transaction

Notes:

- *All Validator Inputs, Outputs require containing 1 Authenticate Token (for example, Order Input Value must contains 1 `Order Token`). Therefore, the documentation below will omit the representation of Authenticate Token in the Value.*
- *All Validator Inputs and Outputs must contain a default ADA amount (for example, Order UTxO requires `order_minimum_ada` to be locked). Therefore, the documentation below will omit the representation of default ADA amount in the Value.*
- *All Validator Inputs and Outputs have `Reference Script is None`. This prevents UTxO from becoming heavy.*

3.5.1 Initialization

Actor	Inputs	Outputs	Redeemer	Mint
`Admin`	OutRef UTxO	Change UTxOs		
`Authen`				
`Factory`		Factory Output	`Initialization`	+1 `Factory Token`

▼ Factory Output

- Datum:

```
FactoryDatum {
head: "00",
tail: "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00",
}
```

3.5.2 Create LBE

Notes:

- *"n" is `minimum_number_seller`, defined in the code-base.*
- *`lp_asset_name` is asset name of AMM Pool LP Token compute from `base_asset, raise_asset`.*
- *`default_manager_datum` is default `ManagerDatum`, check function `build_default_manager_output` in code-base for details.*
- *`default_seller_datum` is default `SellerDatum`, check function `build_default_seller_output` in code-base for details.*

Actor	Inputs	Outputs	Redeemer	Mint
`Project's Owner`	Project Input	Change UTxOs		
`Factory`	Factory Input	Factory Output 1 Factory Output 2	`CreateTreasury { base_asset, raise_asset }`	+1 `Factory Token` +1 `Treasury Token` +1 `Manager Token` +*n* `Seller Token`
`Manager`		Manager Output		
`Seller`		*n* Seller Outputs		
`Treasury`		Treasury Output		

▼ Project Input

- Value contains `reserve_base` amount of `Base Asset`

▼ Factory Input

- Datum:

    - `head`
    - `tail`
  - Condition: `head < lp_asset_name < tail`

▼ Factory Output 1

- Datum:
  - `head`
  - `lp_asset_name`

▼ Factory Output 2

- Datum:
  - `lp_asset_name`
  - `tail`

▼ Manager Output

- Datum equals `default_manager_datum`

▼ Seller Outputs

- `default_number_seller` Seller Outputs
- Datum equals `default_seller_datum`

▼ Treasury Output

- Value contains `reserve_base` amount of `Base Asset`
- Datum: check code-base for details.

3.5.3 Update LBE

Actor	Inputs	Outputs	Redeemer	Notes
`Project's Owner`				require authorize
`Treasury`	*Treasury Input*	*Treasury Output*	`UpdateLBE`	

3.5.4 Cancel LBE

3.5.4.1 Cancel By `CreatedPool`

Actor	Ref Input	Inputs	Outputs	Redeemer
`AMM Pool`	*AMM Pool Reference Input*			
`Treasury`		*Treasury Input*	*Treasury Output*	`CancelLBE(CreatedPool)`

3.5.4.2 Cancel By `NotReachMinimum`

Actor	Inputs	Outputs	Redeemer
`Treasury`	*Treasury Input*	*Treasury Output*	`CancelLBE(NotReachMinimum)`

3.5.4.3 Cancel By `ByOwner`

Actor	Inputs	Outputs	Redeemer	Notes
`Project's Owner`				require authorize
`Treasury`	*Treasury Input*	*Treasury Output*	`CancelLBE(ByOwner)`	

3.5.5 Close LBE

Actor	Inputs	Outputs	Redeemer	Mint
`Treasury`	*Treasury Input*	*Treasury Output*	`CloseEvent`	
`Factory`	*Factory Input 1* *Factory Input 2*	*Factory Output*	`CloseTreasury`	-1 `Factory Token` -1 `Treasury Token`

3.5.6 Add new Seller

Actor	Ref Input	Inputs	Outputs	Redeemer	Mint
`Treasury`	*Treasury Ref Input*				
`Manager`		*Manager Input*	*Manager Output*	`ManageSeller`	
`Factory`				`MintSeller`	+n `Seller Token`
`Seller`			*n Seller Outputs*	`UsingSeller`	

3.5.7 Using Seller

Actor	Ref Input	Inputs	Outputs	Redeemer	Mint	Notes
`Treasury`	*Treasury Ref Input*					Require
`Factory`				`MintSeller`	+n `Order Token`	Optional
`Seller`		*Seller Input*	*Seller Output*	`UsingSeller`		Require
`Order`		*Order Inputs*		`UpdateOrder`		Optional
`User`		*User Inputs*	*User Outputs*			Optional

3.5.8 Collect Sellers

Actor	Ref Input	Inputs	Outputs	Redeemer	Mint
`Treasury`	*Treasury Ref Input*				
`Manager`		*Manager Input*	*Manager Output*	`ManageSeller`	
`Factory`				`MintSeller`	-n `Seller Token`
`Seller`		*n Seller Inputs*		`CountingSeller`	
`Seller Owners`		*n Seller's Owner Outputs*			

3.5.9 Collect Manager

Actor	Inputs	Outputs	Redeemer	Mint
`Treasury`	*Treasury Input*	*Treasury Output*	`CollectManager`	
`Factory`			`MintManager`	-1 `Manager Token`
`Manager`	*Manager Input*		`SpendManager`	

3.5.10 Collect Orders

Actor	Inputs	Outputs	Redeemer
`Treasury`	*Treasury Input*	*Treasury Output*	`CollectOrders`
`Order`	*n Order Inputs*	*n Order Outputs*	`CollectOrder`

3.5.11 Create AMM Pool

Actor	Inputs	Outputs	Redeemer	Mint
`Treasury`	*Treasury Input*	*Treasury Output*	`CreateAmmPool`	
`Project Owner`		*Project Output*		
`AMM Factory`	*AMM Factory Input*	*AMM Factory Output 1* *AMM Factory Output 2*	*anything*	
`AMM Authen Minting Policy`			`CreatePool`	+1 `AMM Factory Token` +1 `AMM Pool Token` +n `AMM LP Token`
`AMM Pool`		*AMM Pool Output*		

3.5.12 Redeem or Refund Orders

Actor	Inputs	Outputs	Redeemer	Mint
`Treasury`	*Treasury Input*	*Treasury Output*	`RedeemOrders`	
`Order`	*n Order Inputs*	*n Order Outputs*	`RedeemOrder`	
`Factory`			`MintOrder`	-n `Order Token`
`User`		*n User Outputs*		

4. Annex

4.1 Effective Spam Transactions Prevention

- Supporting **Counting Sellers, Collect Orders, Refund Orders, and Redeem Orders** by batch is important for spam prevention.
- Assume we have 10,000 orders waiting to be batched. If you batch just 1 order each time, it takes 10,000 batches, which costs a lot of time and transaction fees.
- Therefore, we stipulate:
  - **Counting Sellers**: Batch at least 20 sellers or batch all remaining sellers.
  - **Collect Orders, Refund Orders, Redeem Orders**: Batch at least 30 orders or batch all remaining orders.
- How to know if all remaining **orders | sellers** have been batched?
  - **Counting Sellers**:
    - Manager datum contains information about the remaining number of sellers. In each counting sellers transaction, `managerDatum.seller_count` decreases. When `managerDatum.seller_count == 0`, it means all sellers have been counted.
    - Note: Collect Manager requires all sellers to have been collected (ref)

```
let ManagerDatum { seller_count, .. } = manager_in_datum
let Transaction { inputs, .. } = transaction
let seller_inputs = get_seller_inputs(inputs)
let seller_input_count = list.length(seller_inputs)
expect(seller_count == seller_input_count)
```

  - **Collect Orders:**
    - Prerequisite: Manager has been collected.
    - Total Raised Fund: `reserve_raise + total_penalty` will be updated when collect manager (https://github.com/minswap/minswap-lbe-

v2/blob/6e83f6b03f163fa893b2ce36faf8e524bd642203/lib/lb_v2/validation.ak#L290)

```
let Transaction { inputs, outputs, .. } = transaction
let treasury_output = find_treasury_output(outputs)
let treasury_datum: TreasuryDatum = get_datum(treasury_output)

let manager_input = find_manager_input(inputs)
let manager_datum: ManagerDatum = get_datum(manager_input)

// update treasury state:
treasury_datum.is_manager_collected = True
treasury_datum.reserve_raise = manager_datum.reserve_raise
treasury_datum.total_penalty = manager_datum.treasury_datum
```

- Treasury Datum does not have the number of orders, but it includes information about the total raised and collected funds. In each collect orders transaction, `collectedFund` will increase. When `collectedFund == reserve_raise + total_penalty`, it means all orders have been collected.

```
let TreasuryDatum { reserve_raise, total_penalty, .. } = treasury_datum
let total_raise = reserve_raise + total_penalty
let remaining_amount = treasury_datum.collected_fund - total_raise
let collect_amount = apply_collecting_orders()
expect(remaining_amount == collect_amount)

// accumulate collected fund
treasury_datum.collected_fund += collect_amount
```

- Note: Create AMM Pool requires all orders have been collected (https://github.com/minswap/minswap-lbe-v2/blob/6e83f6b03f163fa893b2ce36faf8e524bd642203/lib/lb_v2/treasury_validation.ak#L195).

- **Redeem Orders:**
  - Prerequisite: AMM Pool has been created.
  - In each redeem orders transaction, `collected_fund` will decrease. When `collected_fund == 0`, it means all orders have been redeemed.

```
fund_amount = sum(o.amount + o.penalty_amount for o in orders)
assert fund_amount == treasury_datum.collected_fund
treasury_datum.collected_fund -= fund_amount
```

- **Refund Orders:**
  - Prerequisite: Event was canceled and all orders have been collected.
  - In each refund orders transaction, `collected_fund, reserve_raise, total_penalty` will decrease. When `collected_fund == 0`, it means all orders have been refunded.

```
total_orders_amount, total_orders_penalty = apply_refund_orders(orders)
refund_amount = total_orders_amount + total_orders_penalty
assert refund_amount == treasury.collected_fund

# update treasury state
treasury_datum.collected_fund -= refund_amount
```

```
treasury_datum.reserve_raise -= reserve_raise
treasury_datum.penalty_amount -= penalty_amount
```