Starport: Lending Kernel

Andrew Redden
andrew@astaria.xyz

Joseph Delong
joe@astaria.xyz

Gregory Curtis
greg@astaria.xyz

January 1, 2024

Abstract

Starport is a simple kernel framework used for composing lending protocols on the Ethereum Virtual Machine (EVM). Starport itself is not a lending protocol but instead handles the message passing and agreement enforcement that is required for any lending protocol. Any existing or future lending protocol can be constructed by implementing the three primary Starport module interfaces *Pricing*, *Status*, and *Settlement*.

The Starport kernel design explores loan origination as a conditional swap with modular abstractions for settlement. Starport kernel has the capability to support collateralized loans and options for any ERC-20[1], ERC-721[2], or ERC-1155[3] as either collateral or debt. Rebasing tokens are not supported.

1 Core

The Starport kernel is supported by two core contracts `Starport` and `Custodian`.

1.1 Starport

`Starport` is the entry point for origination and refinancing.

The duty of `Starport` is to enforce the agreements `originate`, `refinance`, and to maintain the record of loan state. `Starport` manages the loan state mutations through a `loanId`. Each loan is uniquely identified by a `loanId` that is a `keccak256` hash of the `Starport.Loan` struct.

1.2 Custodian

`Custodian` is the entry point for repayment or settlement.

The duty of the Custodian is to manage loan settlement conditions by acting as a *Seaport*[4] *ContractOfferer* . Borrower collateral is purchased from the Custodian with the terms generated by the modules.

Custodian is not a strict implementation, borrowers and lenders can propose differing implementations of the `Custodian` that support the `Custodian` interface. However, a gas efficient default `Custodian` contract is provided with Starport.

2 Modules

Modules are implementations of the module interfaces that are untrusted by the core. Each module serves a purpose within the lending lifecycle providing parameters and conditions for the core contracts to enforce.

2.1 Pricing

The duty of the `Pricing` module is to provide the conditions of repayment to `Custodian` and the conditions of refinance to the `Starport` contract.

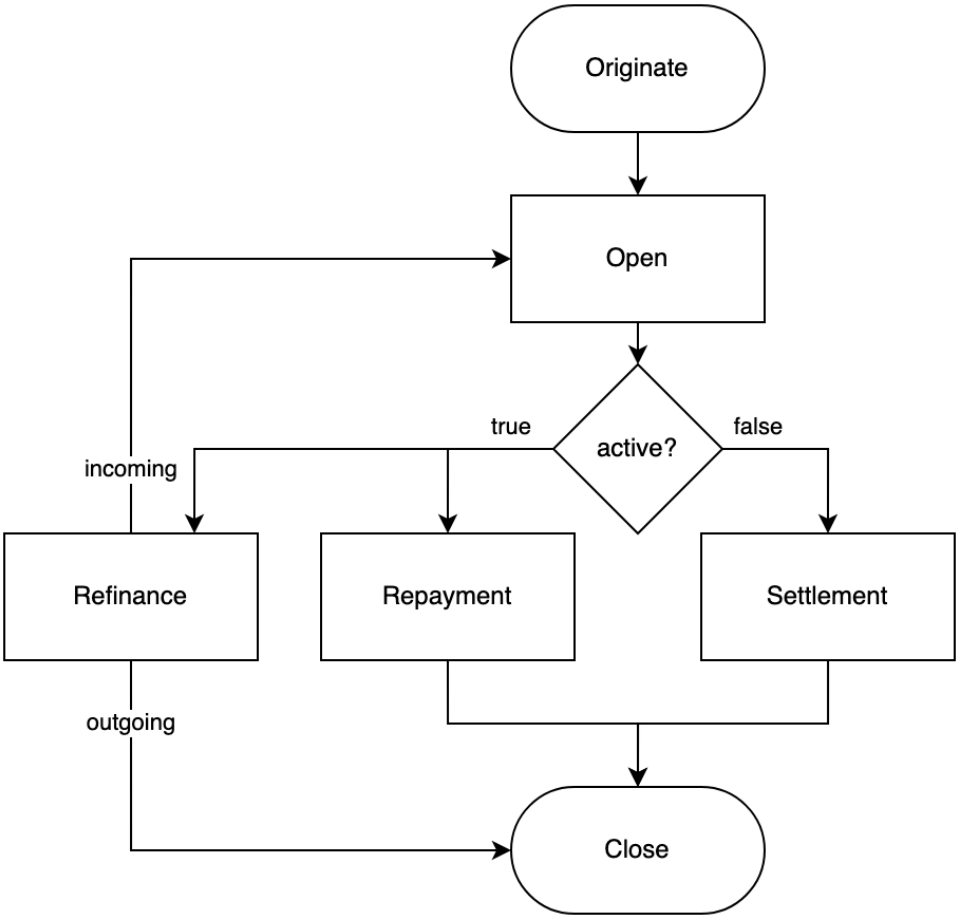
2.2 Status

The duty of the `Status` module is to provide the status of whether a loan is active or inactive. An inactive status allows a loan to be settled by the `Custodian`.

2.3 Settlement

The duty of the `Settlement` module is to provide the conditions settlement to the `Custodian`.

3 Loan Lifecycle



- **Originate** (action)
  - `Starport` loans begin in the `originate` method.

`Starport` enforces the transfers of the collateral from the `loan.borrower` to the `Custodian`, debt transfers from the `loan.issuer` to the `loan.borrower`, and transfers the `additionalTransfers`.
When authentication conditions of the `loan.borrower` and `loan.issuer` are met, `loanId` is sent to `Open`.

- **Open** (event)
  - An `Open` event creates a new `loanId` that is hashed from the `Starport.Loan` struct, making each cryptographically unique and representing all data necessary to manage the `loanId` using off chain data availability through representment.
- **Active** (conditional)
  - Active is a conditional provided by `Status.isActive()`. When `isActive()` returns `true`, the repayment or refinance condition is allowed. When `isActive()` returns `false` the settlement condition is allowed. The transition from `isActive()` from `true` to `false` is not a immutable transition. If the condition flips from `true` to `false` it is possible for the condition to return `true` again in the future.
- **Repayment** (action)
  - Repayment condition allows the `loan.borrower` of the associated `loanId` to repay their loan through the *Custodian* as a *Seaport ContractOfferer* on the conditions returned from `Pricing.getPaymentConsideration()`. During which the `loan.issuer` (lender) is repaid and the `loan.originator` is paid carry (if included). Afterwards, the `loanId` is sent to `Close`.
- **Settlement** (action)
  - Settlement conditions are provided by the `Settlement` module. `Settlement.getSettlementConsideration()` returns the authorized and conditions of settlement. If authorized is `address(0)` any fulfiller can provide the settlement conditions, otherwise the settlement fulfiller is restricted to authorized only.
- **Refinance** (action)

  - Refinance conditions are provided by the `Pricing` module. `Pricing.getRefinanceConsideration()` returns the conditions of the refinance. Refinance enforces transfers from the `lender` to the outgoing `loan.issuer`, enforces carry transfers from the `lender` to the outgoing `loan.originator`, and applies those payments to the `loan.debt` array and replaces the `pricingData`. The outgoing `loanId` is sent to `Close` and an incoming `loanId` is sent to `Open`.

- **Close** (event)
  - A `Close` event deletes a `loanId` from `Starport`. A deleted `loanId` cannot be refinanced, repaid, settled, or originated.

4 Actors

Within the Starport system there are three distinct actors borrowers, lenders, and fulfillers.

1. **Borrower** is the user providing collateral in exchange for debt.
2. **Lender** is the user providing debt to the borrower in exchange for future repayment (presumably with interest but module implementation is independent of this fact).
3. **Fulfiller** is the transacting party commonly known as *msg.sender*. A fulfiller can be a borrower, lender, or a different third party.

5 Authentication

5.1 Origination

For the `originate` method, Starport will ensure that the fulfiller is authorized by both the `loan.borrower` (borrower) and the `loan.issuer` (lender).

There are three methods to ensure the fulfiller is authorized by the borrower and the lender.

1. `msg.sender`

- `msg.sender` must be equal to either the borrower or lender

2. approval
    - `msg.sender` is approved by the either the lender or borrower
        - approval must be `ApprovalType.BORROWER` or `ApprovalType.LENDER`, it cannot be both
3. `CaveatEnforcer.SignedCaveats`
    - A signed struct that enforces conditional approval from either the borrower or lender
        - Validates the signature that it is the borrower or lender
        - Makes an external call to a `CaveatEnforcer` with the signed in data

A fulfiller must provide at least one of the following authentication methods for both the lender and the borrower.

5.2 Refinancing

For the *refinance* method within Starport, we need to ensure that the fulfiller is authorized by the refinancing *lender*.

We must authenticate using one of the three methods described in the *originate* section.

6 Additional Transfers

`additionalTransfers` is a concept within `originate` that allows the fulfiller to reimburse themselves or others for cost incurred to perform the transaction or an additional reward for executing the transaction.

`additionalTransfers` is passed to `CaveatEnforcer` and validated conditionally. For instance a borrower signs into their `SignedCaveat` an allowance for an `additionalTransfer` to reimburse the fulfiller for pre-purchasing an ERC-721.

The fulfiller provides `additionalTransfers` and the authentication model either allows the transfers or reverts.

7 Representment

Starport data availability and loan state mutations rely on a concept developed in Astaria v0 called *representment*.

Representment in the context of blockchains is a data availability strategy that reduces storage requirements and as an effect reduces transaction gas consumption.

All data necessary is stored in a `Starport.Loan` struct and hashed using `keccak256` into a single `bytes32` value `loanId`. Any call to the Starport or modules requires the `Starport.Loan` struct be represented, hashed and validated for an open loan before the transaction can proceed.

References

[1] Fabian Vogelsteller, Vitalik Buterin. *ERC-20: Token Standard*. Ethereum Improvement Proposals, 2015.
https://eips.ethereum.org/EIPS/eip-20

[2] William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs. *ERC-721: Non-Fungible Token Standard*. Ethereum Improvement Proposals, 2018.
https://eips.ethereum.org/EIPS/eip-721

[3] Witek Radomski, Andrew Cooke, Philippe Castonguay, James Therien, Eric Binet, Ronan Sandford. *ERC-1155: Multi Token Standard*. Ethereum Improvement Proposals, 2018.

[4] 0age, stephankmin, d1ll0n, emo-eth, horsefacts. *Seaport*. Github, 2022.
https://github.com/ProjectOpenSea/seaport