Starport: Lending Kernel

Andrew Redden
andrew@astaria.xyz

Joseph Delong
joe@astaria.xyz

Gregory Curtis
greg@astaria.xyz

January 12, 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 data availability 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 modules *Pricing*, *Status*, and *Settlement*.

Starport 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 of `originate` and `refinance`, as well as maintain the record of loan state.

1.2 Custodian

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

The duty of the `Custodian` is to custody the borrower collateral and enforce loan settlement and repayment conditions.

`Custodian` is not a strict implementation. Borrowers and lenders can provide differing implementations of the `Custodian` that support the `Custodian` interface. However, a gas efficient default `Custodian` contract will be 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 the `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. An active status allows a loan to be repaid.

2.3 Settlement

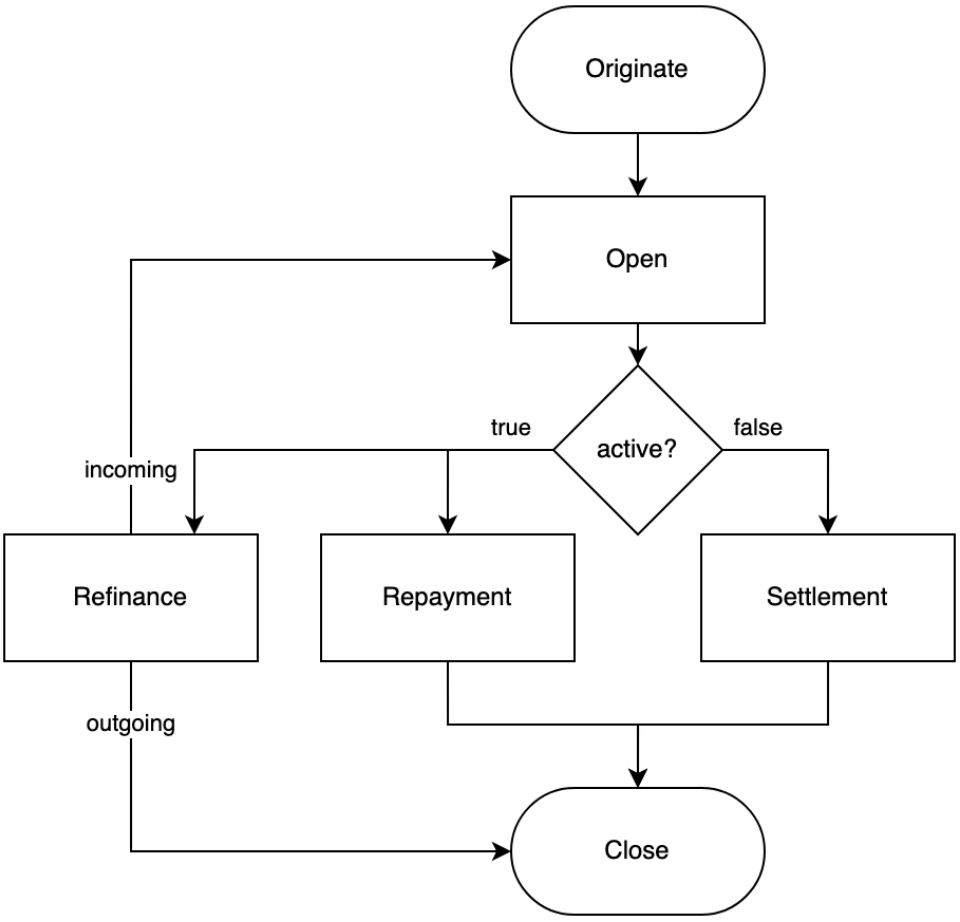
The duty of the `Settlement` module is to provide the conditions of settlement to the `Custodian`. Settlement is more commonly referred to as liquidation, but is a restrictive definition for this framework.

3 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.
3. **Fulfiller** is the user fulfilling the transaction. A fulfiller can be any transacting user including a borrower or lender.

4 Loan Lifecycle



- **Originate** (action)
  - `Originate` is the action of enforcing conditions in order to create a new loan.
  - During `originate`, `Starport` enforces the transfers of the collateral from the borrower to the `Custodian`, debt transfers from the lender to the borrower, and enforcing `additionalTransfers`. When authentication conditions of the borrower and lender are met, a `loanId` is created and the state is transitioned to `Open`.

- **Open** (event, condition)
  - `Open` is a state transition by which the loan is created. A `loanId` is created and stored representing the loan.
- **Active** (condition)
  - The `Status` module provides whether an `Open` loan is either conditionally active or inactive.
    An inactive condition allows a `Settlement` action to occur. An active condition allows a `Repayment` action to occur. If the condition flips from active to inactive it is possible for the condition to transition to active again in the future (if a `Settlement` did not occur).
- **Repayment** (action)
  - Repayment is the action of a borrower repaying the loan and retrieving their collateral.
  - The repayment action allows the borrower to repay their loan through the `Custodian` on the conditions provided by the `Pricing` module. During which the lender is repaid. Afterwards, the loan state is transitioned to `Close`.
- **Settlement** (action)
  - Settlement is the process of resolving a loan that has transitioned to an inactive status.
  - Settlement conditions are provided by the `Settlement` module. The `Settlement` module returns the authorized fulfiller and conditions of settlement. Specifying the authorized fulfiller is optional. If an authorized fulfiller is not specified any fulfiller can provide the conditions of settlement. Otherwise, settlement is restricted to the authorized fulfiller.
- **Refinance** (action)
  - `Refinance` is the action of replacing an existing lender with an incoming lender with no change to the underlying borrower. Refinancing accrues the current debt and the outgoing loan is replaced with an incoming loan.

  - `Refinance` conditions are provided by the `Pricing` module. `Refinance` enforces transfers from the incoming lender to the existing lender and applies those payments to a new loan and additionally replacing the `Pricing` parameters. The outgoing loan state is transitioned to `Close` and an incoming loan state is transitioned to `Open`.

- **Close** (event, condition)
  - `Close` is a state transition by which the loan is deleted.
  - A `Close` event deletes an `Open` loan by deleting the `loanId` from storage. A deleted `loanId` cannot be refinanced, repaid, settled, and cryptographically secured from being originated.

5 Authentication

There are three methods to ensure the fulfiller is authenticated by the borrower or the lender to transact on their behalf.

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 a borrower or lender, it cannot be both
3. `CaveatEnforcer.SignedCaveats`
   - An EIP-712[4] signed struct that enforces conditional approval from either the borrower or lender
     - Validates the signature of the borrower or lender
     - Makes an external call to the specified `CaveatEnforcer` with the signed in data

5.1 Origination

For origination, `Starport` will ensure that the fulfiller is authenticated by both the borrower and the lender.

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

5.2 Refinancing

For refinancing `Starport` will ensure that the fulfiller is authorized by the refinancing lender.

A fulfiller must provide at least one of the three described authentication methods for the lender.

6 Additional Transfers

`additionalTransfers` is a concept within `originate` that allows the fulfiller to reimburse themselves or other entities in ERC-20, ERC-721, or ERC-1155s for cost incurred to perform the transaction or an additional reward for executing the transaction.

`additionalTransfers` are an array of `AdditionalTransfer` structs naming the `from`, `to`, `amount`, and token details for a transfer.

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

`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.

7 Representment

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

Representment in a process where all data necessary is stored in memory using a single Solidity `struct`. The `struct` is hashed and the resulting hash is stored. An accompanying event is emitted for indexing the `struct` for later use.

Representment is a data availability strategy that reduces storage requirements and as an effect reduces transaction gas consumption.

When a subsequent call is made to the contracts, the transactor represents the `struct`. The `struct` is hashed and the resulting hash is validated against the stored hash before continuing execution.

Within Starport the `Starport.Loan` struct contains all of the values necessary for loan servicing. The `Starport.Loan` struct is hashed using `keccak256` into a single `bytes32` value `loanId`. Any subsequent call to Starport or modules requires the `Starport.Loan` struct be represented, and validated as an `Open` loan.

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.
https://eips.ethereum.org/EIPS/eip-1155

[4] Remco Bloemen, Leonid Logvinov, Jacob Evans. *EIP-712: Typed structured data hashing and signing* Ethereum Improvement Proposals, 2017.
https://eips.ethereum.org/EIPS/eip-712