What is TrueFi?

TrueFi is modular infrastructure for on-chain credit.

TrueFi connects lenders, borrowers, and portfolio managers via smart contracts governed by the TRU token.

Since its launch in November 2020, TrueFi has originated more than $1.7bn in loans to >30 borrowers and paid more than $40mm in interest to protocol participants.

Borrowers include both leading crypto-focused institutions and "real world" firms, such as fintech companies, trading firms, and credit funds.

Lend

Getting started as a Lender on TrueFi

Why lend on TrueFi?

Transactions on TrueFi infrastructure are **transparent** and **publicly auditable,** enabling lenders to track every dollar (or token) lent and borrowed, as well as the status and structure of every loan.

Lending opportunities on TrueFi span multiple sectors and various risk/return profiles.

> TrueFi partner Archblock can help institutional lenders onboard and interact with TrueFi.
>
> For more info, visit archblock.com/investors.

How does lending on TrueFi work?

TrueFi's smart contract infrastructure helps lenders, managers, and borrowers interact transparently and seamlessly.

For lenders, the process is as follows:

1. **Find opportunities** via app.truefi.io
2. **Onboard / KYC** *(if necessary)*
3. **Lend funds** with instant settlement 24/7
4. **Monitor activity & track returns**
5. **Redeem / Withdraw funds** when liquidity is available

How do I learn about portfolio managers on TrueFi?

Portfolio Managers ("PMs") are onboarded through TrueFi governance. You can find posts from PMs and additional information on the TrueFi forum.

For the most up-to-date information on PMs, please see the TrueFi app for a list of active managers and links to their materials.

What types of activity does TrueFi support?

TrueFi smart contracts support multiple structures for various market participants.

- **Credit Vaults** support unitranche and multi-tranche on-chain credit deals.
- **Asset Vaults** support off-chain credit, or "Real World Asset" (RWA), activity.
- **Lines of Credit** enable borrowers to source capital directly from lenders.

What are the fees for using TrueFi contracts?

Pools pay a protocol fee to the TrueFi DAO treasury and an optional fee to the portfolio's manager.

Fees are stated on a per annum basis, accrued continuously and paid periodically by the portfolio smart contract. Read more here for more detail and to see an example.

To find current fee rates, see vaults listed in the TrueFi app.

Are TrueFi contracts ERC-4626 compliant?

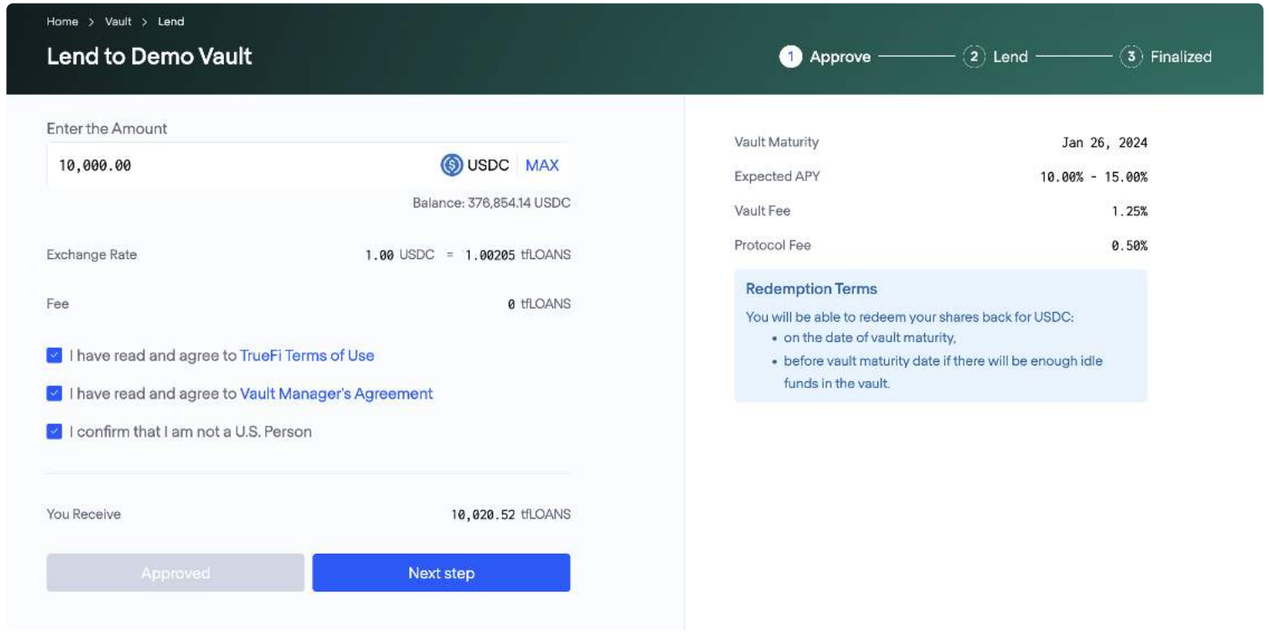
Yes. TrueFi pools follow ERC-4626 standards.

How to lend

Once a user is ready to lend, the user is required to complete two transactions:

1. `approve` : approves the vault smart contract to transfer up to a certain allowance of the asset. (Note: This is a typical interaction on the Ethereum network. Read more about the 'approve' function here).
2. `deposit` : lends funds to the vault. In return, the lender receives lending pool tokens ("LP tokens").

> Note that for some pools, lender funds are locked until the pool's maturity date. Please review and confirm details before lending.

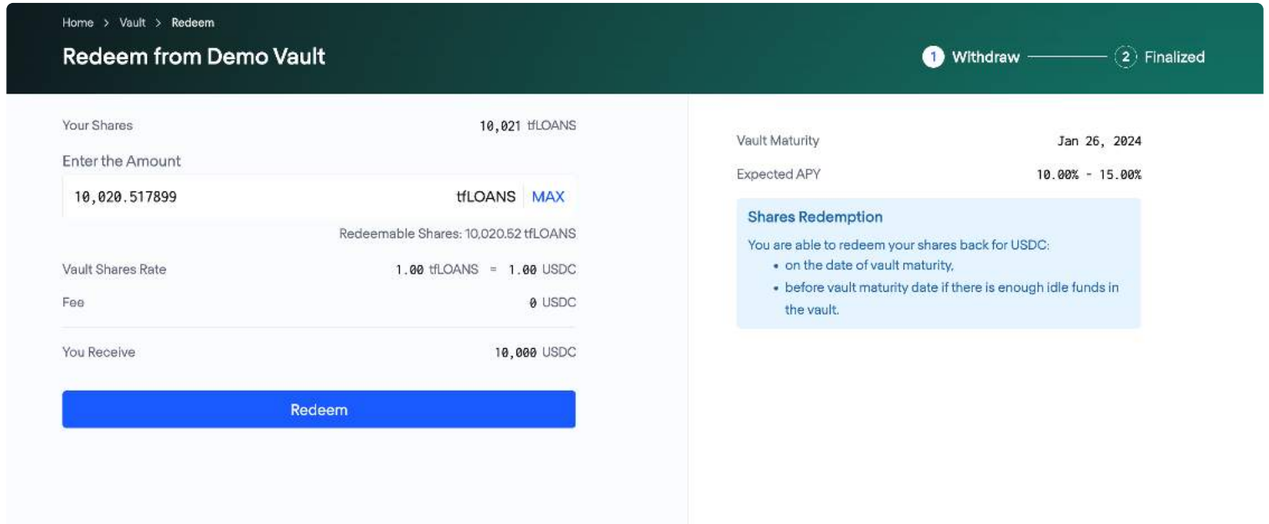


How to withdraw

Lenders can redeem funds once a vault has reached its maturity date (or depending on the PM's configuration, once liquid funds are available).

When redemptions are available, a lender can redeem LP tokens for underlying tokens by calling `withdraw` on the token smart contract.

> Note that for some vaults, redemptions may be processed at the manager's discretion or not available until the vault's maturity date.
>
> Please review and confirm details on each vault before lending.



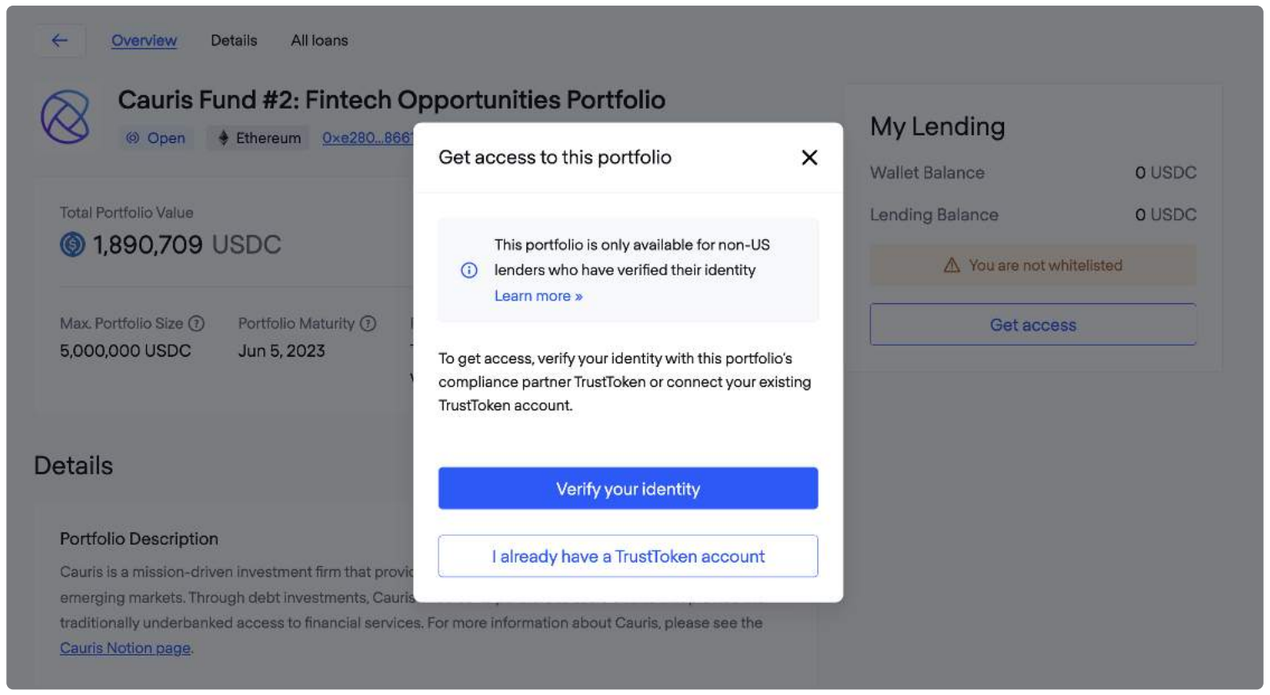
Onboarding / KYC (for permissioned pools)

Portfolio Managers can configure permissioned pools on TrueFi. Accordingly, some vaults on TrueFi may require identity verification before lenders can gain access.

> TrueFi partner Archblock can help institutional lenders onboard and interact with TrueFi.
>
> For more info, visit archblock.com/investors.

How to access permissioned pools

Lenders can get access to permissioned pools by clicking on the "**Get access**" button on the vault's page. This will route the user to verify their identity with an external provider, as defined by the portfolio manager's policy.



Example: How to get access to permissioned portfolios

Why do users need to verify their identity?

In order to be compliant with regulations in certain jurisdictions, Portfolio Managers must verify a user's identity before and while doing business with them.

This is in an effort to maintain the integrity of the pool. The KYC process is a first line of defense against fraud, sanctions evasion, and terrorist financing. Compliance with global KYC standards helps ensure the pool against reputational risk, regulatory fines, and even cease and desist orders.

Users are required to provide credentials that prove their identity and address. Verification credentials can include ID card verification, face verification, biometric verification, and/or document verification. For proof of address, utility bills and bank statements are examples of acceptable documentation.

This process is important for determining users' risk and whether they can meet the Portfolio Manager's requirements to use their services. Moreover, it's also a legal requirement to comply with Anti-Money Laundering (AML) laws in certain jurisdictions.

Why are some pools for Non-US lenders only?

US regulations specify certain requirements, including licenses, that Portfolio Managers need to meet in order to offer investment opportunities to US persons (citizens or legal residents) and entities.

Failing to comply with such regulations would make Portfolio Managers subject to liability, which would severely impact the pool.

Borrow

Getting started as a Borrower on TrueFi

Why borrow on TrueFi?

Borrowers can find capital at competitive rates from a network of global liquidity on TrueFi.

Using TrueFi's infrastructure, borrowers can access capital markets 24/7 with faster settlement than traditional finance systems.

Borrowers on TrueFi also have an opportunity to build their public "on-chain" financial reputation and position themselves to access more capital at better rates moving forward.

Who can borrow on TrueFi?

Borrowers on TrueFi include leading institutions in the crypto industry, as well as "real world finance" borrowers, such as fintechs and credit funds.

See below for recent press on TrueFi borrowers:

Mexican FinTech Uses DeFi to Provide Loans
PYMNTS.com

Getting started as a borrower

Borrowers follow a simple process to receive their first loan on TrueFi:

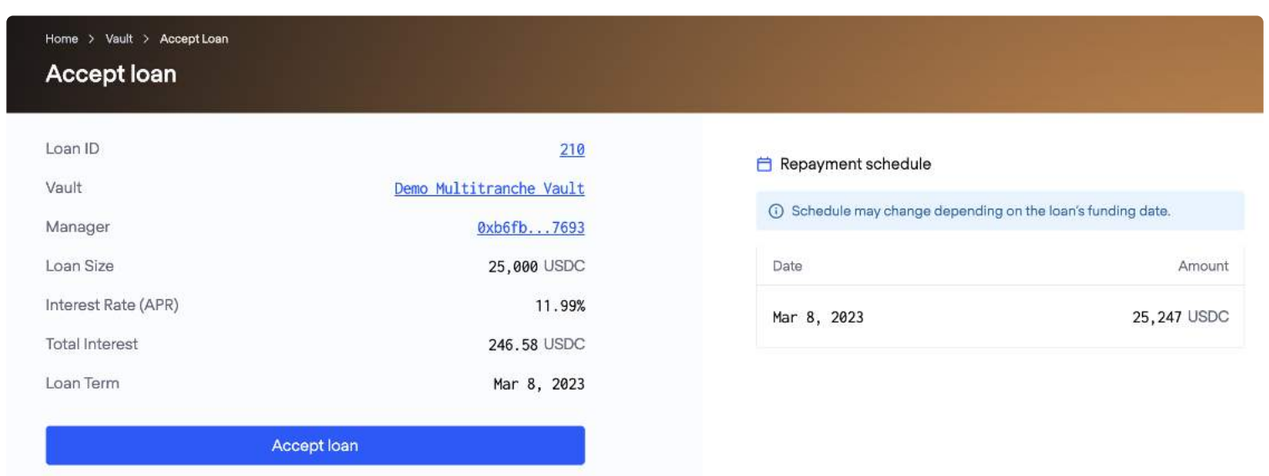
1. **Onboard with a portfolio manager.**
    - Borrowers can communicate with managers directly, or apply via the form at truefi.io/borrow to get connected with potential managers in the TrueFi ecosystem.
    - Complete KYB and negotiate loan terms with a portfolio manager.
2. **Receive a loan.**
3. **Repay a loan.**

Receiving a loan

> If you are a prospective borrower, you can request information or post a borrow request at https://forum.truefi.io/.

In the typical borrower experience, borrowers follow the steps below:

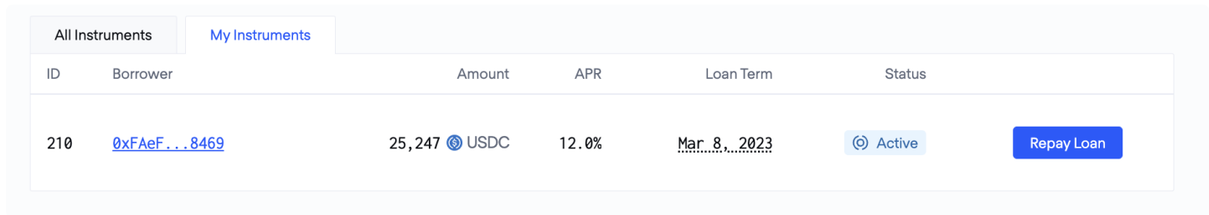
1. PM and Borrower negotiate loan terms offline. PM then submits terms on-chain for review.
2. Borrower accepts terms by executing an on-chain transaction (see screenshot below).
3. PM disburses funds to borrower's address. The borrower receives funds in their wallet.
4. Borrower repays loan at, or before, time of loan maturity.



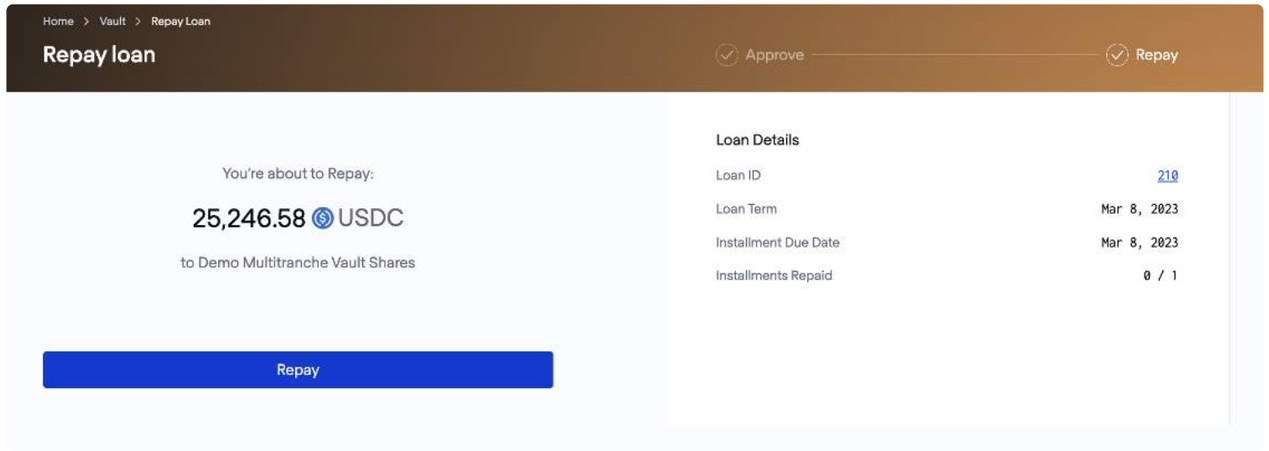
Repaying a loan

How to repay a loan:

1. Borrower goes to app.truefi.io
2. Borrower clicks 'Repay' button shown on loan details



3. Borrower is first prompted to approve transfer of funds (must complete `approve()` on-chain transaction).
4. Borrower is then prompted to `repay` transaction. This transaction transfers funds back to the portfolio and repays the full amount owed on the loan.
    - Once this transaction is complete, the loan is marked ‘repaid’ automatically and no further action by the borrower is needed.



Manage

Getting started as a Portfolio Manager on TrueFi

Portfolio Managers ("PMs") can use TrueFi infrastructure to run their own credit fund on the Ethereum blockchain and additional networks.

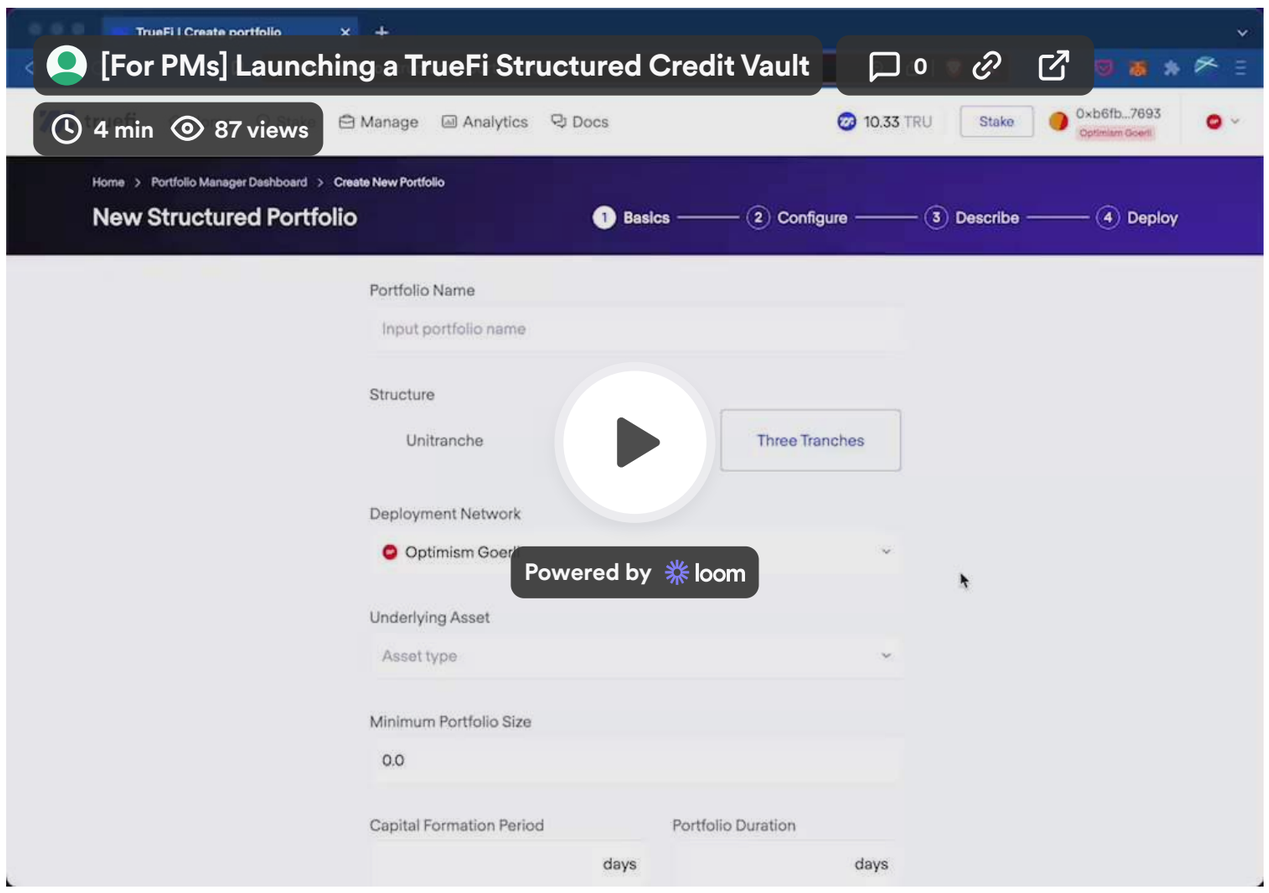
In this section, PMs can learn how to launch and manage a fund on TrueFi. Below is a step-by-step guide to running a fund on TrueFi:

1. **Onboarding for managers**: New managers introduce themselves to the TrueFi community and are vetted by TrueFi governance. During this process, potential lenders can communicate with and learn more about PMs.
2. **Creating a vault**: PMs deploy their fund (called a "vault") via the TrueFi app.
3. **Disbursing loans**: Once a vault is live, PMs can disburse funds to borrowers. The TrueFi app makes it easy for PMs and borrowers to create loans, disburse funds, and repay loans.
4. **Continuous fund management:** TrueFi smart contracts streamline the processing of lender inflows/outflows, as well as principal and interest payments, in real-time 365/24/7.

> TrueFi's partner Archblock can help institutional users onboard and operate on TrueFi.
>
> For more info, visit archblock.com/managers.

How does it work?

For a brief overview, see the demo video below:



When creating a portfolio the PM configures policies for fees, min/maximum portfolio size, portfolio maturity dates/tenor, and other parameters.

Additionally, PMs have the option to run permissioned portfolios, where they can enforce KYC/KYB policies for lenders. Read more here on how PMs can set such policies.

PMs are able to share information and due diligence materials with lenders via the TrueFi app.

Does TrueFi support "real world" use cases?

Yes, TrueFi offers infrastructure to support "real world" financing as well as crypto-centric financing deals. TrueFi smart contracts give PMs the ability to create instruments such as fixed rate loans, lines of credit, amortizing loans, and multi-tranche facilities.

What are potential benefits of using TrueFi?

- **Reduced overhead cost**, by servicing otherwise difficult and expensive components of fund management on-chain.
- **Access TrueFi's global liquidity network,** leveraging DeFi rails to interact with lenders across the world 24/7/365.
- **Best-in-class infrastructure** enables PMs to configure pools to their specifications.

Onboarding for managers

Portfolio Managers ("PMs") can launch and run their own funds on TrueFi, using TrueFi credit infrastructure.

How to onboard (as a Portfolio Manager)

To deploy a portfolio, managers must be approved by TrueFi governance via the following steps:

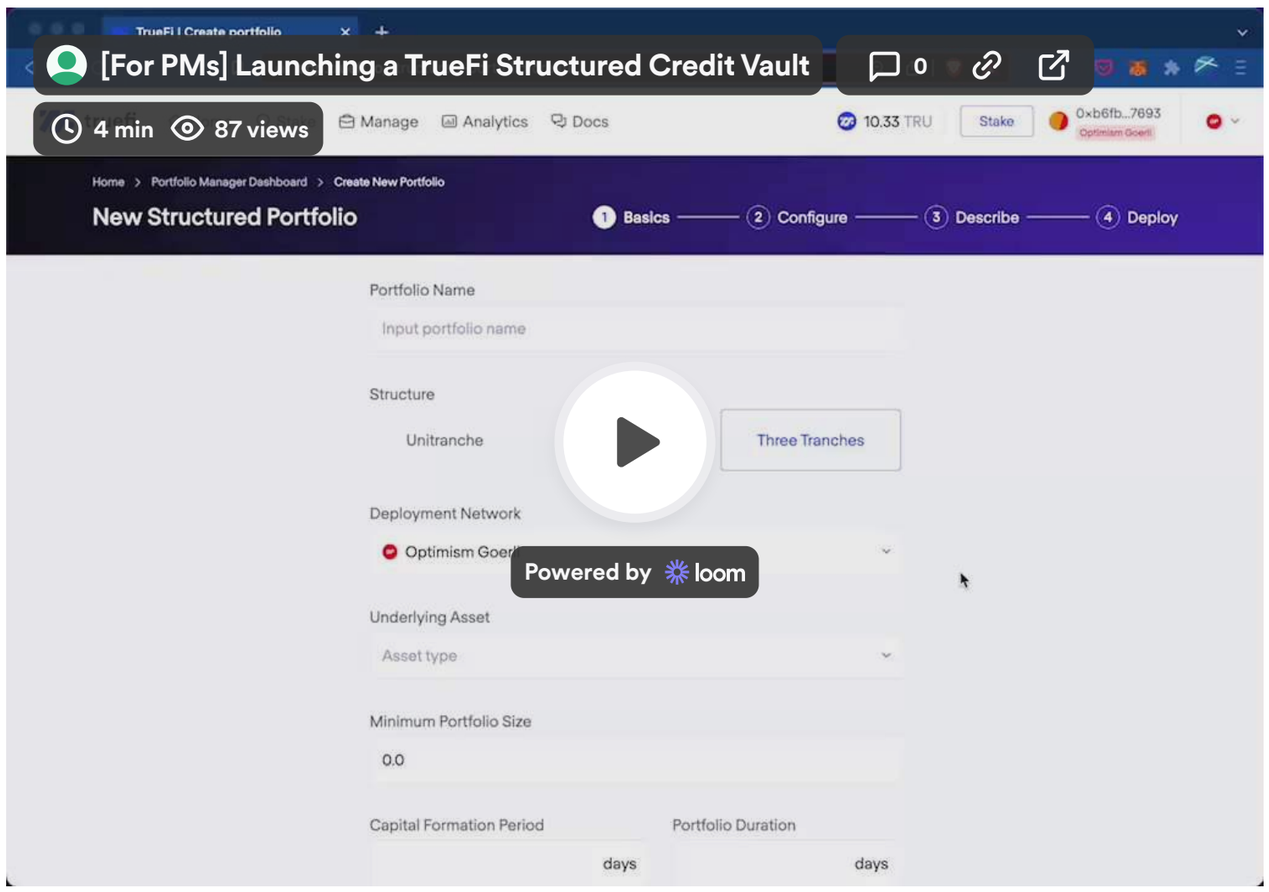
1. PM posts forum request at https://forum.truefi.io/.
    - This forum request should outline the PM's background and plans, along with the PM's Ethereum (or Optimism) address.
    - For previous examples, please see here.
2. After governance approval, TrueFi governance adds the PM's address to an allowlist.
    - This enables the PM to deploy vaults from a factory contract.
3. PM deploys vault
    - Vaults are supported on Ethereum and Optimism.
    - For more details on how vaults work, visit **Credit Vaults** and **Asset Vaults**.

Creating a vault

Once successfully onboarded, a portfolio manager can launch a new vault.

PMs can easily create a fund by clicking "Create New Portfolio" at https://app.truefi.io/managers. To test and demo the product, see the guide here: **Credit Vault tutorial**.

The TrueFi application walks PMs through the vault creation process, as shown in the demo below:



Demo of vault creation

Vault creation parameters and details

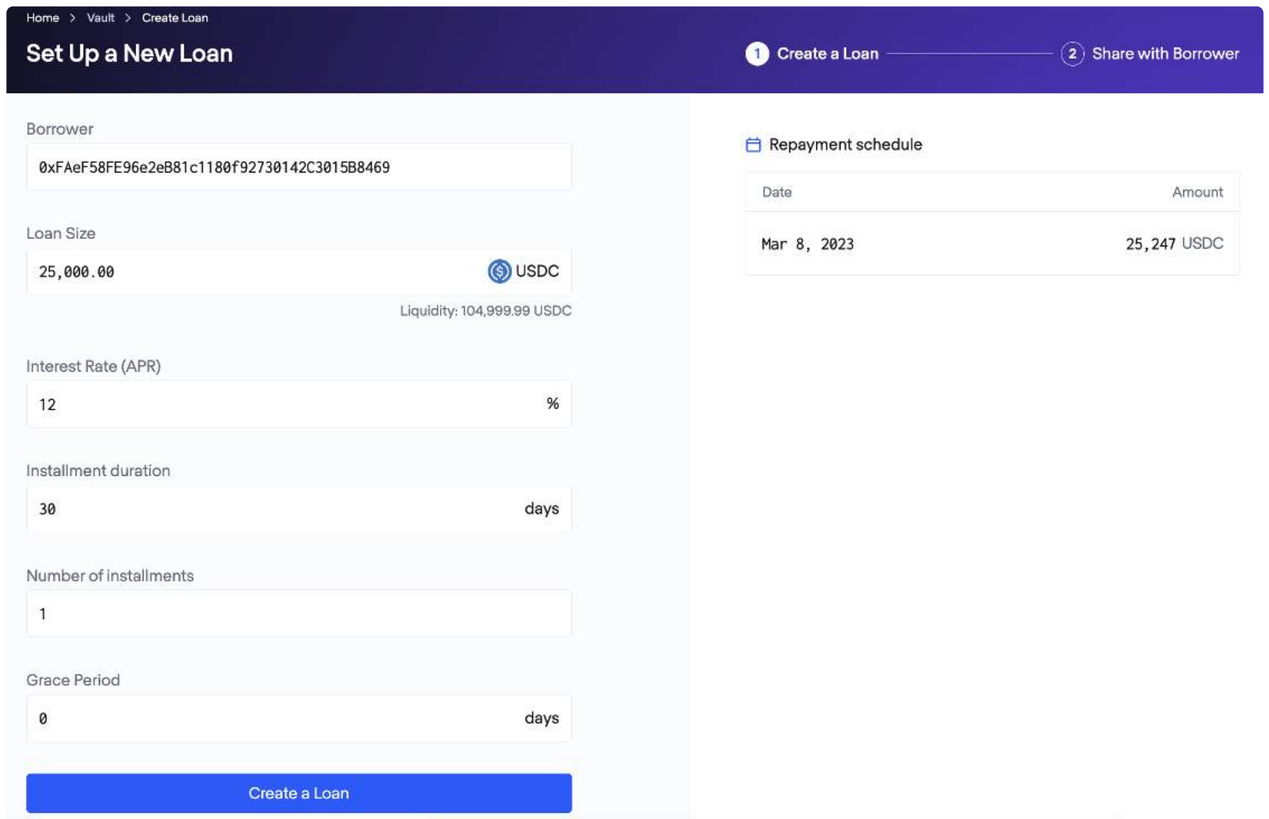
In vault creation, PMs are able to configure the following settings (among others):

- Capital formation rules
  - During the **Capital Formation period**, lenders can commit funds with assurance that the smart contract will return funds (with no fees) if certain requirements are not satisfied. For more details, read here.
    - Example: PM configures a 30 day capital formation period and minimum vault size of 1M USDC. If the fund has not raised ≥1M USDC by day 30, then principal will be returned to lenders with no fees taken. If the fund raises ≥1M USDC within 30 days, then the PM can move the vault to 'Live' status and deploy loans from the vault.
- Tranche configurations
  - Number of tranches: single tranche / 2 tranches / 3 thranches
  - Minimum subordination ratios for each tranche
  - *For more details on tranche mechanics, read here.*
- Vault duration / maturity date
  - Lender funds will be locked up until the vault's maturity date
  - All loans within the vault must mature before, or on the maturity date
- Fee structure
  - Vault fee: in basis points, on annualized basis
  - *For more details on fees, read here.*
- Lender permissions / restrictions
  - i.e. manager can require KYC or set custom list of allowed lenders
  - *For more details on permissions, read here.*

Disbursing loans

How to disburse a loan

To disburse a loan, click 'Disburse a loan' on the portfolio page and follow the on-screen prompts (see below for more detail):



Parameters required for each loan:

- Borrower address
- Principal Amount
- Loan term (in days)
- Number of installments
- Interest rate (APR)

Once loan terms are accepted by borrower, the manager can disburse funds to the borrower's address:

Managing KYC/KYB requirements

PMs can enforce restrictions on who can lend to their vaults by using TrueFi's **Controllers** architecture. PMs can easily configure permissioned vaults that meet their specs.

Below are some notes on standard configurations available. PMs can also implement their own custom policies (as decribed here).

- *KYC users only:* manager delegates access list management to external KYC provider, such as Archblock, Verite, or others.
- *Allowlisted lenders only:* manager limits access to a set of allowlisted wallet addresses.
- *Non-US lenders only:* only non-US IP addresses can access portfolio; lenders must attest that they are non-US persons

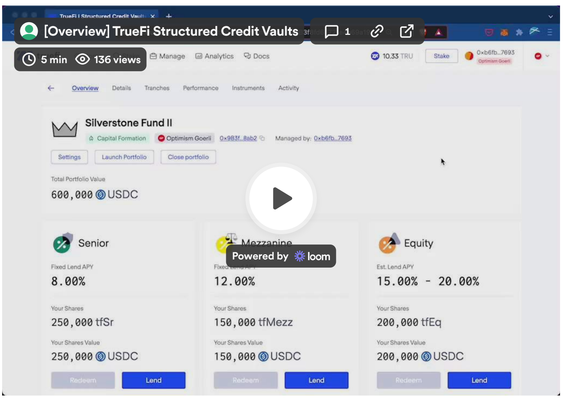
Credit Vaults

What are Credit Vaults?

Credit Vaults are TrueFi smart contracts that coordinate the lending, borrowing, and management of on-chain credit.

Credit Vaults enable on-chain credit, helping portfolio managers (PMs) configure modular lending pools that meet their specs (single or multiple tranches, permissioned access, etc).

For a brief demo video showing their capabilities, see below:



Demo: Strucuted Credit Vaults

Credit Vaults can be made up of a single tranche or multiple tranches, creating the opportunity for lenders to participate in distinct "slices" of a given vault, each with their own risk/return profile. To learn more, see `#` **What are tranches?** below.

Credit vaults also introduce the concept of a *capital formation period,* in which lenders can commit capital to a smart contract with assurances that if the vault does not meet certain requirements within a given time period, funds will be returned to lenders with no fees incurred. To learn more, see `#` **What is the capital formation period?** below.

How do Credit Vaults work?

For technical docs, see **Credit Vault contract overview**.

What types of activity do Credit Vaults support?

Credit Vaults support both on-chain and off-chain "Real World Asset" ("RWA") credit activity.

PMs can use Credit Vaults to make simple fixed-yield loans to borrowers.

For PMs who want to manage more complex off-chain/RWA activities, please see **Asset Vaults**. Asset Vaults are specialized smart contracts that offer more flexibility to support complex off-chain use cases.

What are tranches?

A tranche (from French, a "*slice*" or "*portion*") refers to a financial product that can be split into distinct pieces that can be offered to buyers or lenders, each with their own risk-reward profile.

Credit vaults can be created with up to three tranches, meaning they can also support two-tranche or single tranche deals.

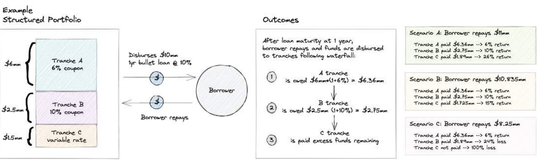
How do interest & repayments flow to lenders?

Let's take a vault with 3 tranches: A/B/C, with A being most senior.

In this example, `Tranche A` has a fixed interest rate of 6%, `Tranche B` has a fixed interest rate of 10%, and `Tranche C` receives any excess funds in the vault after `Tranches A and B` have been paid amounts owed. `Tranche A` has $6mm in principal, subordinated by `Tranche B` ($2.5mm principal), and `Tranche C` ($1.5mm principal).

Over the life of the vault, each tranche linearly accrues interest owed based off its coupon rate. For instance, after 30 days `tranche A` with 6% interest rate is owed $6.03mm `=$6mm*[1+(6%*(30/365))]` and after 365 days it is owed $6.36mm `=$6mm*(1+6%)`.

When funds are repaid to the vault, they are distributed by a waterfall where the most senior tranche's amount owed must be repaid before the subordinated tranche can withdraw. The following illustration shows how such a vault would handle various scenarios:



Illustrative scenarios for Credit Vaults

What is the capital formation period?

During the capital formation period, lenders can commit funds to one or more tranches of the vault, with assurance that the vault smart contract will return funds if certain requirements are not satisfied. If requirements are not satisfied by the end of the period, then capital is returned to lenders with no additional fees.

In this way, lenders can stipulate that deals must reach a specific minimum size and/or with specific ratios in order to go live.

For example, a credit vault may launch with a capital formation period of 30 days, with requirements that the total deal size is at least $5mm with at least 30% of funds in the junior tranche. If the vault does not meet this criteria at day 30, the vault will return funds to lenders and move to 'Closed' status.

How are lender restrictions / permissions managed?

Portfolio managers ("PMs") can define lenders access rules / restrictions for each individual tranche. Like other pools on TrueFi, PMs can set their own policies for permissioned or permissionless pools.

For example, a PM could make the equity tranche open to only one specific wallet address, while enabling all ID verified addresses to participate in the junior and senior tranches.

Lender restrictions (as well as redemption policies, fee structures, and more) can be configured to a PM's spec by using **Controllers**, customizable logic that helps TrueFi vaults meet the varied needs of financial users.

What are the fees on Credit Vaults?

Vaults pay a protocol fee to the TrueFi DAO treasury. Fees accrue are quoted on a per annum basis, accrue block-by-block, and are paid upon each smart contract interaction (lend/withdraw/disburse loan/repay loan).

The example below illustrates how the protocol fee works:

> **Protocol Fee example**
>
> Take an example vault *Verum Fund,* which holds 1,000,000 USDC worth of loans and assume protocol fee = 50 bps per annum (0.50%).
>
> Assuming the value of Verum Fund grows linearly from 1,000,000 USDC to 1,100,000 USDC over the course of 30 days (avg. value of 1,050,000 USDC), the vault would pay a protocol fee of 431.51 USDC for this time period:
>
> ```
> Protocol fee = 1,050,000 USDC * 0.50% * (30/365) = 431.51 USDC
> ```

Additionally, PMs can set an optional Portfolio Fee. Portfolio Fees are paid to the PM, and can be configured such that they are accrued linearly over time, or paid as a flat fee at time of deposit and/or withdrawal.

For up-to-date fee rates on each vault, please see vault pages at https://app.truefi.io/.

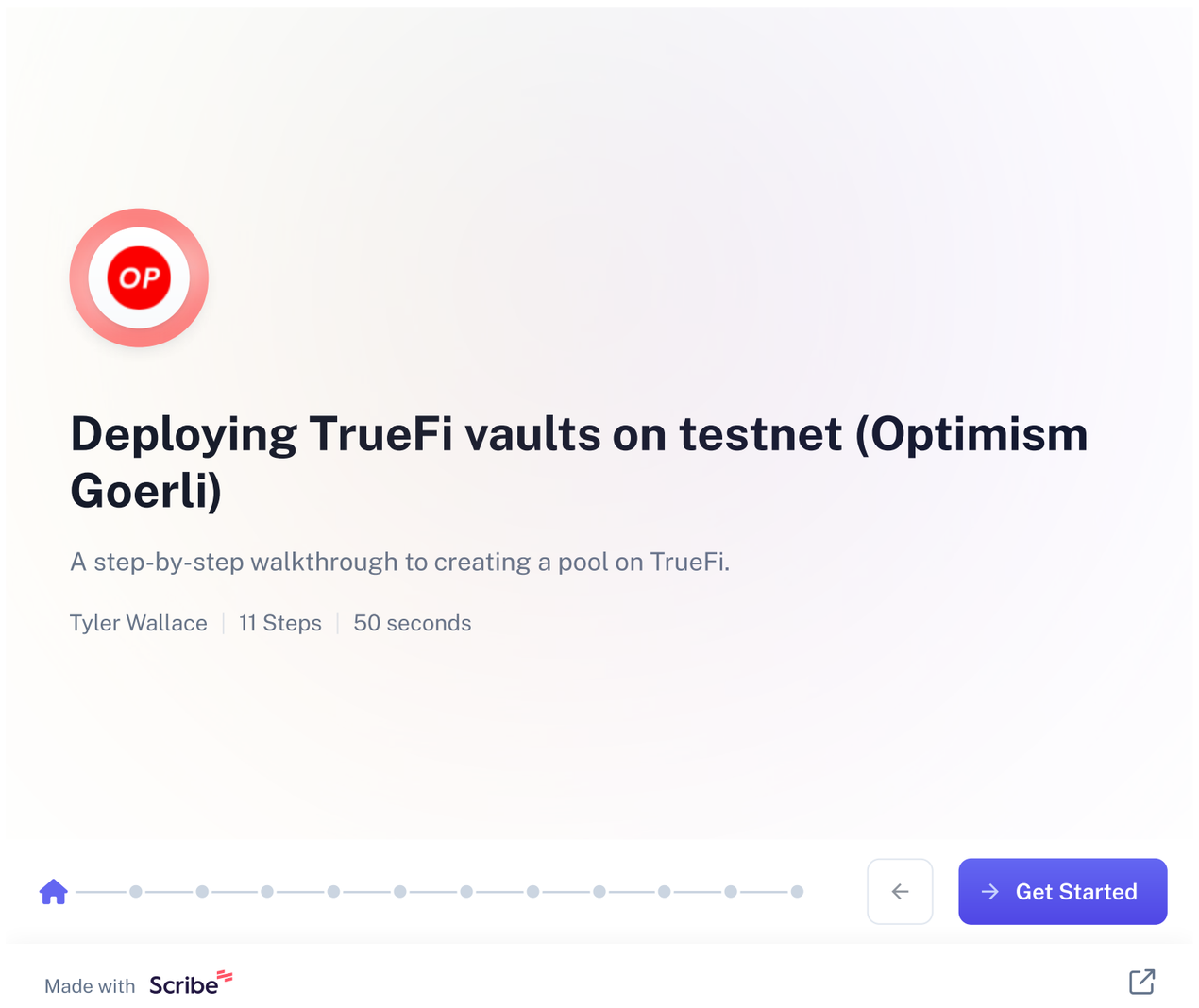
Credit Vault tutorial

Users can test Credit Vaults by creating their own demo vault on Optimism Goerli using this link.

> Before beginning, you will need to switch to the Optimism Goerli test network and ensure your wallet is funded with test ETH and test USDC assets.
>
> To complete these steps, follow this step-by-step guide.

Follow the Scribe tutorial below or view written instructions here.

Step-by-step walkthrough (using Scribe app)



Instructions:

1. First, switch network to Optimism Goerli (testnet)
    - To add network to wallet, navigate to https://chainid.link/?network=optimism-goerli and click 'Connect'
2. Next, fund your wallet with testnet ETH and testnet USDC
    - Get 0.01 test ETH by using https://isomorph.loans/faucet or other faucet
    - Mint "mock" USDC `0x5f1c3c9d42f531975edb397fd4a34754cc8d3b71` via Etherscan to use for lending/borrowing in test vaults
        - Connect wallet and mint desired amount (6 decimals)

3. Finally, navigate to https://app.truefi.io/structured-credit-portfolio/create or click 'Create new portfolio' -> select 'Credit Vault' in TrueFi app and follow the flows:
    - For a demo and more details, see **Creating a vault**
    - For technical documentation, see **Credit Vaults**

Credit Vault technical details

✅ Intro

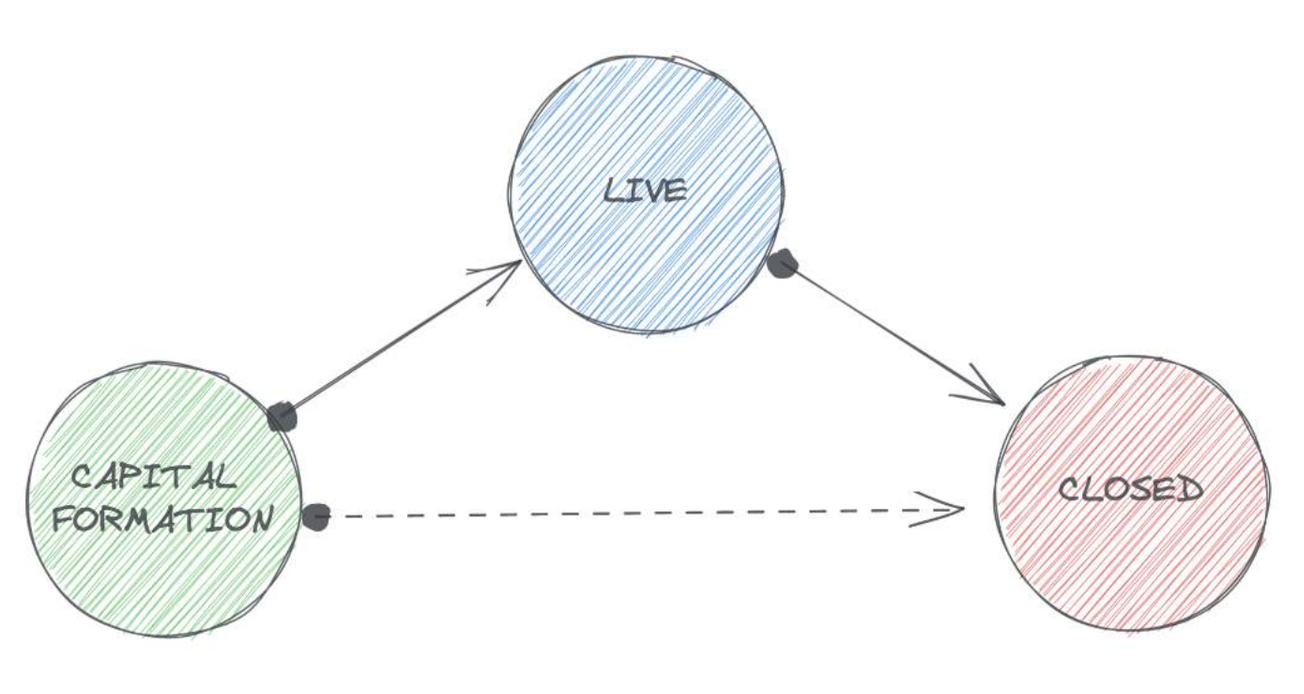
Credit Vaults are multi-vault portfolios allowing different types of investors to deposit into one bucket of funds managed by the portfolio manager, while having different risk-return profiles. There are three tranches:

- A / "Senior" (fixed rate)
- B / "Junior" (fixed rate)
- C / "Equity" (variable rate)

✅ Portfolio Life Cycle

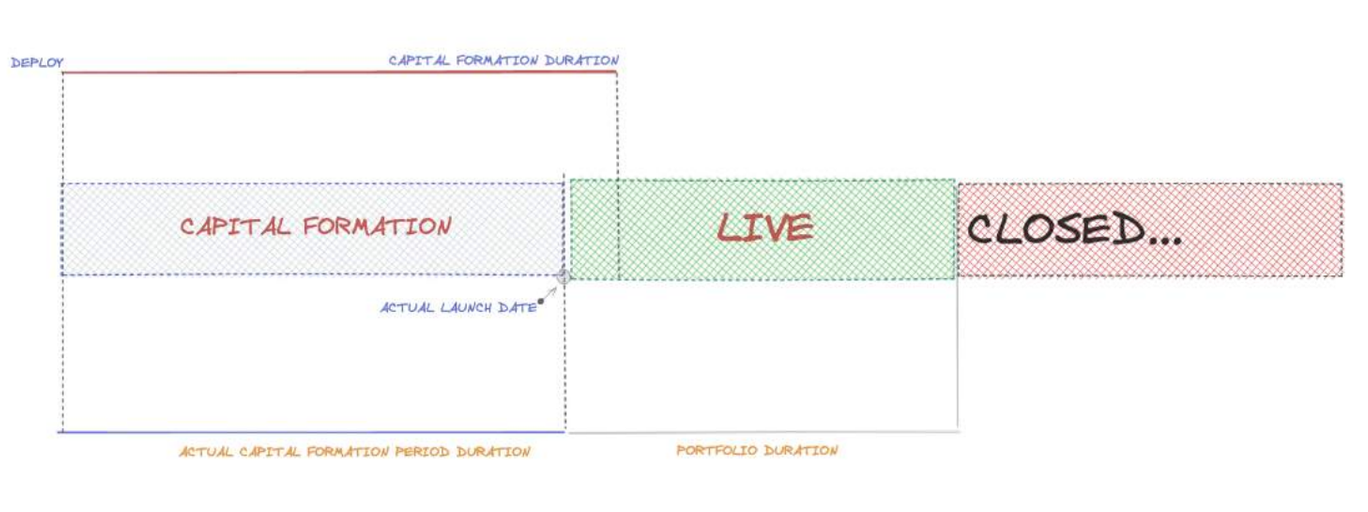
Vault (or "Portfolio") can be in one of three states - Capital Formation, Live and Closed.

Portfolio starts its life cycle in the Capital Formation state. From Capital Formation state, Portfolio can be transitioned into Live state or Closed state.

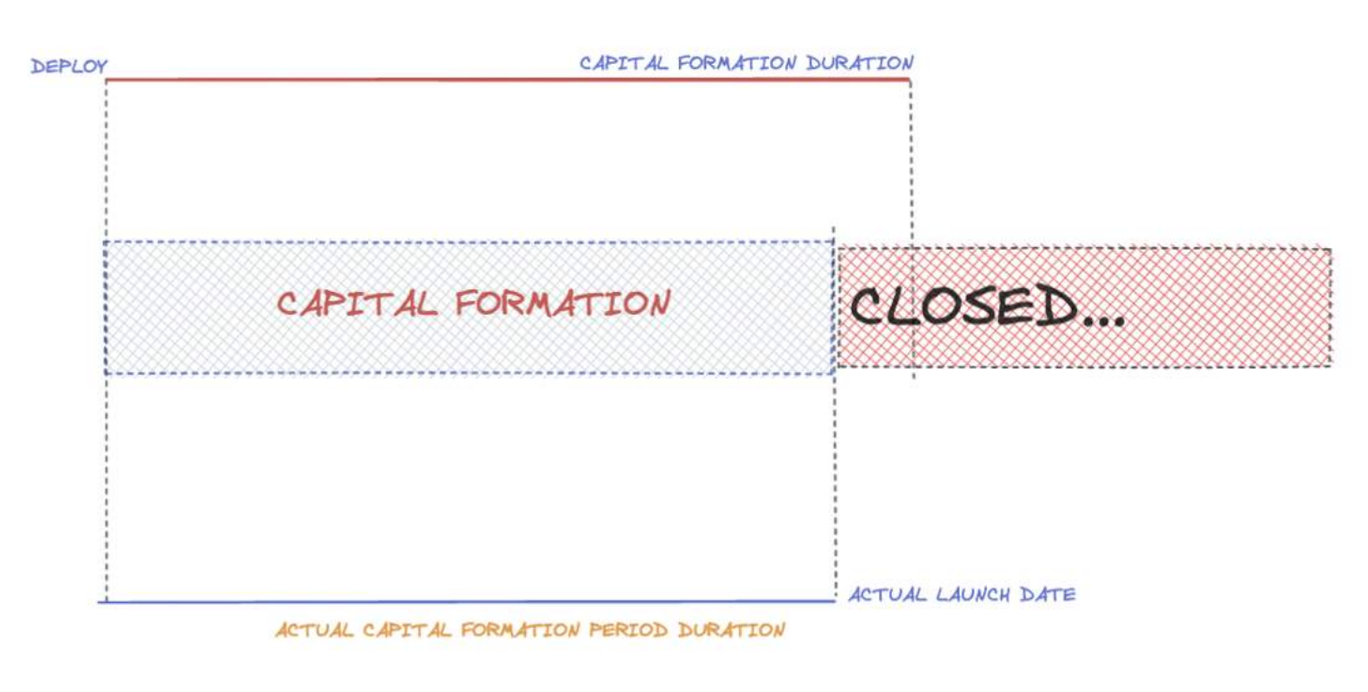


✅ Standard Portfolio Flow

Portfolio can be launched at any time, from Capital Formation - into Live state. While in Live state, the portfolio can only be transitioned into Closed state. Closed state is the final state of the portfolio.

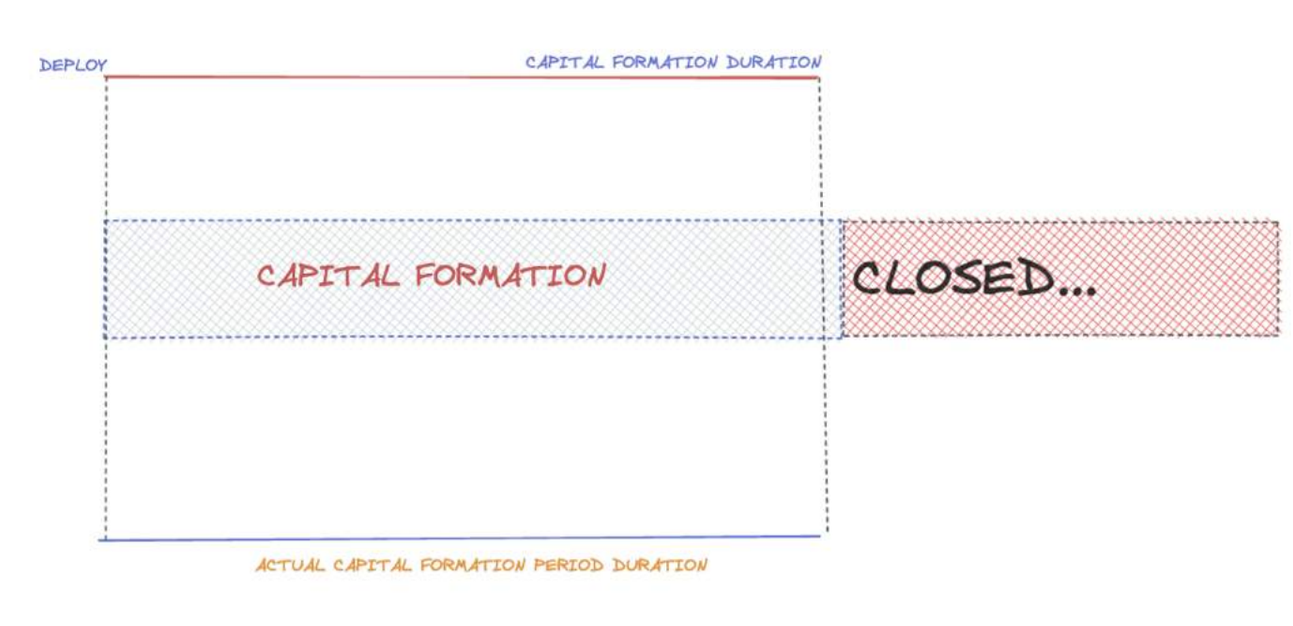


Portfolio Manager can decide to Close the Portfolio before it hits end of Capital Formation.



✅ Limiting Capital Formation phase

Length of the Capital Formation phase prevents the portfolio from staying in the Capital Formation state forever. If the deadline timestamp is met, the portfolio can be transitioned from Capital Formation state to Closed using a public function. It is a safety mechanism ensuring that funds will not remain stuck in the portfolio i.e. in case of PM losing his private key and not being able to manage the portfolio anymore.



✅ Emergency close of Portfolio in Live state

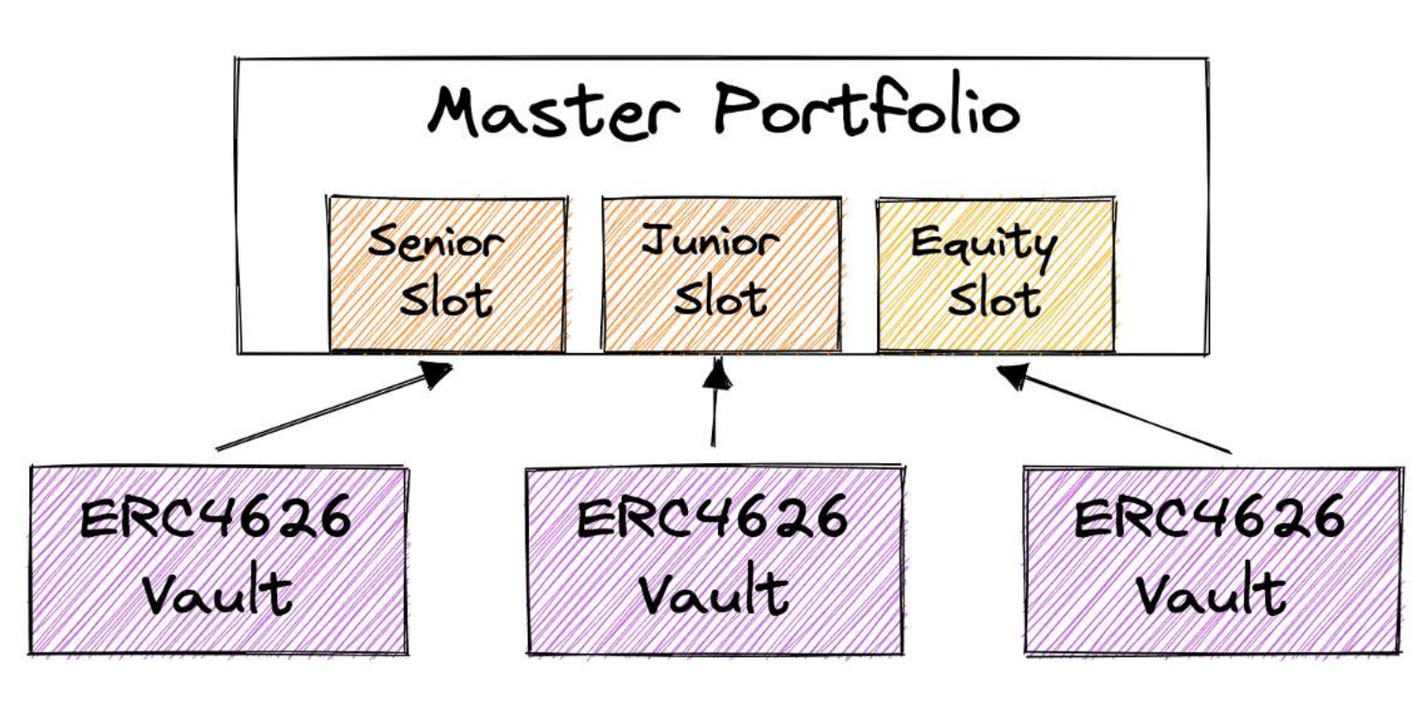
Public transition from Live to Closed should be possible as well. If all loans get repaid or default and the portfolio is past the end date, any address can close the portfolio.

✅ Early closing

Manager can close portfolio anytime, when portfolio is Live. From this time on, essentially anyone can start withdrawing, manager stops earning fees and cannot disburse any more loans. **Additionally the total portfolio value has to consist of 100% cash if it is being closed before maturity date.**

✅ Architecture

Portfolio consists of four main smart contracts - Master Portfolio and three ERC4626 Vaults. Master Portfolio is a central contract responsible for most of the calculations. It also serves as a main capital pool and a terminal for portfolio managers to manage capital deployments. ERC4626 Vaults serve as ports for investors to deposit into or withdraw from a particular tranche. They store the capital when the portfolio is NOT in the Live state (so in the Capital Formation state and in the Closed state). In the Live state they only handle interactions between the investors and the portfolio. Additionally each Vault issues a token which represents shares of a particular investor in a particular tranche. These main four smart contracts are most likely going to be supported by a set of auxiliary smart contracts like Portfolio Factory or Protocol Config.



✅ Roles / Actors

There are three main actors in the system:

- Portfolio Manager (PM) - an actor responsible for portfolio configuration and capital deployments. They run an asset management business and earn fees as returns. It's highly probable that they will also be lenders in the equity tranche.
- Borrower - an actor that is a target of Portfolio Managers' capital deployment operations. Receives capital on loan disbursement and is obligated to return it in the schedule defined in the used debt instrument.
- Lender - an actor that invests their funds into a portfolio seeking returns. Lenders might have different risk appetite and different return requirements. For the purpose of this document they are divided into three main categories:
  - Senior Lender - is a lender whose priority is safety and high returns are a secondary goal.
  - Junior Lender - is a lender who wants to balance exposure to various risks and opportunity to earn an attractive yield.
  - Equity Lender - is a lender who is comfortable with exposure to highly volatile assets that may present different performance in different market conditions. Their capital will serve as first loss capital in case of default or portfolio underperformance.
- DAO - an entity that oversees all portfolio operations, provides infrastructure and earns fees.

✅ Lending

✅ Deposit

Depositing is allowed when a portfolio is in the Capital Formation state or in the Live state.

✅ Capital Formation State Deposits

As the deposits in the Capital Formation state happen only before any capital deployments are made, or any other types of risks applied, for each deposit lender receives a number of LP tokens exactly equal to the number of tokens deposited. Portfolio token has the same number of decimals as the portfolio asset token (important note: there is always only one underlying asset token for each portfolio). For 1 wei of the underlying token deposited, a lender will receive 1 wei of the LP token. By default there are no restrictions on the sizes of the deposits. The only requirement that needs to be met is not depositing more into a particular tranche than is defined in the tranche ceiling parameter.

✅ Capital Formation State Withdrawals

Withdrawals in the Capital Formation state are allowed. LP tokens are burned 1:1 to assets (at least in the vanilla, default strategy)

The default controller should require the withdrawal flag to be on, and funds to not drop below tranche floor level (controller setting).

✅ Live State Deposits

When deposits happen in the Live state, the value of a LP token is dynamic. It changes because of the interest accrual mechanisms, portfolio value fluctuations etc. In order to make a deposit in the Live state the total value of the portfolio and the value of a particular LP token needs to be calculated in the depositing transaction. Because initially the value of one underlying asset wei was set to one LP token wei, very small deposits (deposits worth way below $0.01) may be mispriced. Additionally when making deposits in the Live state, both tranche ceilings and tranche ratios have to be respected.

✅ Live State Withdrawals

When withdrawals happen in the Live state, the total value of the portfolio is dynamic. The total value of the portfolio, and then the value of a particular LP token tranche needs to be calculated in the withdrawing transaction. When withdrawals are made in the Live state, tranche ratios and tranche floors have to be respected.

✅ Closed State Withdrawals

When withdrawals happen in the Closed state, the total value of the portfolio is not subject to any changes anymore, LP tokens have a stable value. This value is calculated when the PM closes the portfolio and might change in case of recovery of any defaulted assets. When withdrawals are made in the closed state, neither tranche ratios, nor tranche floors need to be respected.

✅ Portfolio Management

✅ Factory Whitelisting

A PM creates a portfolio by calling a proper method on the Portfolio Factory. In order to create a portfolio a potential PM needs to be on the factory whitelist. In order to be whitelisted a PM needs to go through a process defined by the DAO. DAO is a final decision maker and executor when it comes to the whitelisting logistics.

✅ Alternative Deployments

If the PM wishes to use only 1, or 2 out of 3 available tranches - it is possible. There has to be a way to deploy only the necessary number of contracts. If PM launches a portfolio with only 2 (or 1) tranches, only 2 (or 1) of them should be deployed. This can be solved with a proper parametrisation of factory, different function for portfolio creation in factory or having multiple factories using the same factory whitelist. If portfolio is once deployed as a 1 or 2 tranche portfolio, more tranches cannot be added later on. If PM ultimately wants to run 3 tranches, but wants to start with 1, they still need to deploy all 3 of them, and using controllers block access to the ones that are not needed.

✅ Withdraw Lever and Deposit Lever

Each tranche has two flags a.k.a levers ****(they should be inside proper default Controllers) and determine whether deposits or withdrawals are allowed. By default, portfolio is created with the Deposit Lever set to `allowed` and Withdraw Lever set to `disallowed`. Then manager can set the levers as they please. When portfolio exits Capital Formation state and enters Live state, both levers are automatically set to `disallowed`. When portfolio enters Closed state, Deposit Lever is automatically set to `disallowed` and Withdraw Lever is automatically set to `allowed` and manager can no longer change them.

✅ Waterfall

At any point of time when the current value of any subsequent tranche needs to be calculated a waterfall calculation needs to be performed. Waterfall calculation first evaluates the total value of the portfolio. Sums up the value of the cash sitting in the portfolio and then assigns chunks of that value to particular tranches in the seniority order.

Structured Credit Vaults support only one type of Waterfall: learn more here.

✅ Portfolio Creation

Once whitelisted, a PM can create a portfolio. In order to create a portfolio they need to pass the following parameters. Some of them are editable (**E**) during the portfolio lifecycle or fixed (**F**) and cannot be changed after the portfolio creation.

- A / Senior tranche parameters
  - Senior tranche name (**F**)
  - Senior tranche symbol (**F**)
  - Senior tranche manager fee (**F**)
  - Target senior interest rate (**F**)
  - First Loss Capital buffer for Senior (**F**)
- B / Junior/Mezzanine tranche parameters
  - Mezzanine tranche name (**F**)
  - Mezzanine tranche symbol (**F**)
  - Mezzanine tranche manager fee (**F**)
  - Target senior interest rate (**F**)
  - First Loss Capital buffer for Mezzanine (**F**)
- C / Equity tranche parameters
  - Equity tranche name (**F**)
  - Equity tranche symbol (**F**)
  - Equity tranche manager fee (**F**)
  - Minimum expected Equity interest rate (**E**)
  - Maximum expected Equity interest rate (**E**)
- Administrative parameters
  - Portfolio name (**F**)
  - Portfolio duration (**E**)
  - Capital formation duration (**E**)
  - Minimum portfolio size (**F**)

There might be a need to pass additional parameters that were not mentioned here.

✅ Capital Formation

Once a portfolio is created it enters the Capital Formation state. In the Capital Formation state deposits of new capital by investors are available, but withdrawals (by default - they can be allowed) and loan disbursements are blocked. Particular deposits do not need to be larger than tranche floor, but they cannot overflow above tranche ceiling. PM is free to change tranche sizes during that period.

✅ Portfolio Start

In order to start the portfolio, or in other words change its state from Capital Formation to Live, the only action that needs to be taken is calling a proper method. This method would only be callable by the PM and would require sum of all tranche sizes to be above defined total minimum portfolio size. Additionally all the tranches need to fulfil the tranche ratio requirements (so the Junior tranche cannot be too large compared to Equity tranche and Senior tranche cannot be too large compared to Junior tranche and Equity tranche combined). Calling this method will enable loan disbursements, start fixed interest accruing.

✅ Capital Deployment

Mechanism of loan disbursement will not be discussed in this doc. Tranched portfolio should use the current TrueFi state of the art loan mechanism. Most certainly it's going to be based on Fixed Interest Only Loans, which can be configured either in a way that they will force the borrower to repay interest periodically and then principal at the loan end date, or as bullet loans. All issued debt instruments need to have their end date before the end date of the portfolio. Though this is only an UI-enforced requirement. We need to remove this requirement from smart contracts.

✅ Portfolio Closing

In order to close a portfolio before the maturity date, the portfolio cannot have any outstanding assets. Portfolio can always be closed, if it's after the maturity date. When closing the portfolio a waterfall needs to be calculated and the value of each of the particular tranches set. After the portfolio is closed, deposits are blocked and capital deployments are blocked, but withdrawals are allowed. Withdrawals are made at the LP token price preset in the waterfall calculation. Tranches' floors are not blocking withdrawals. Default controller flags indicating whether deposits and withdrawals are allowed are set, so deposits are not allowed anymore and withdrawals are allowed - this is being set automatically and cannot be changed by the manager anymore.

✅ Defaulted Assets Recovery

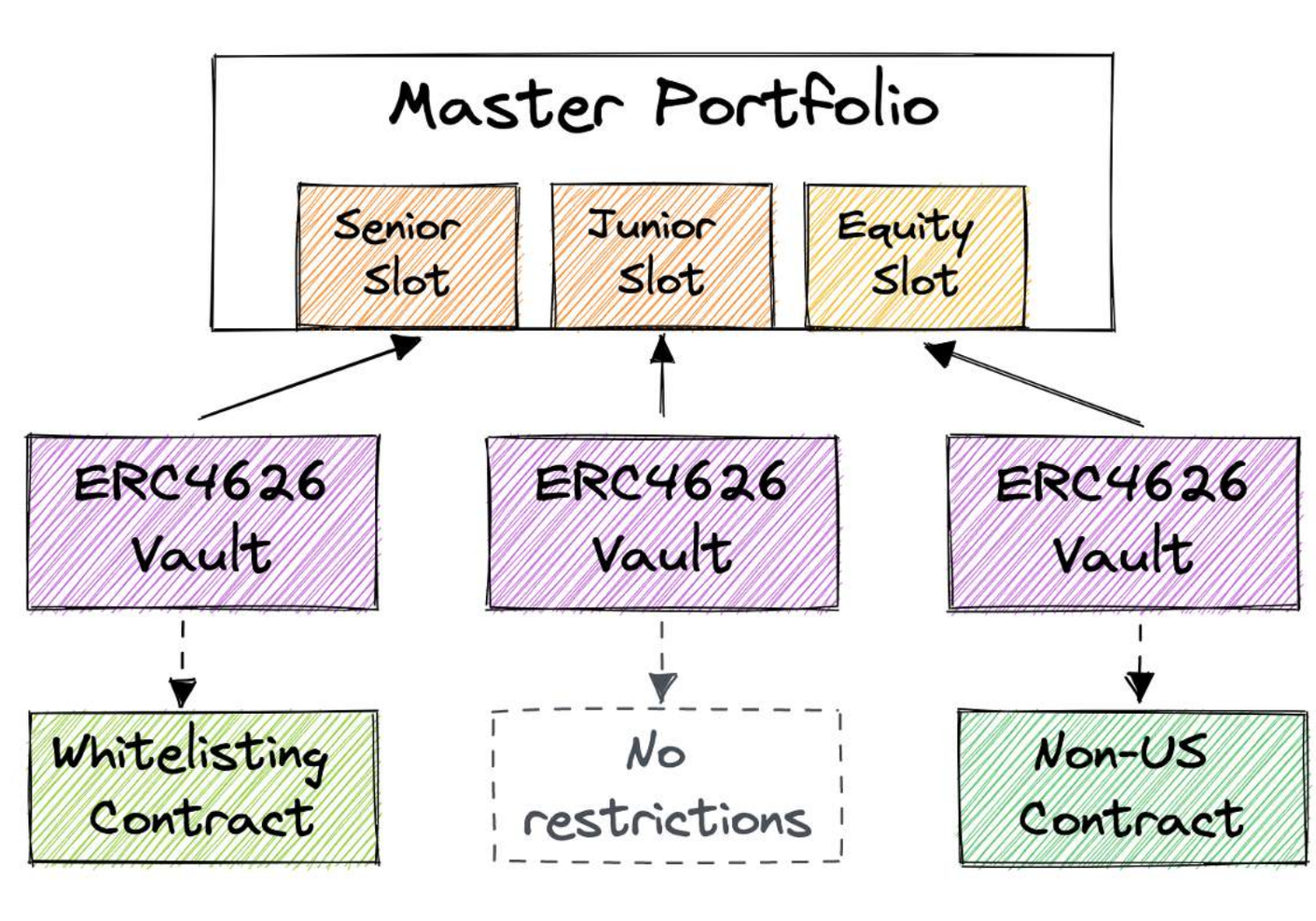
It is possible that some repayments from overdue instruments are gonna be made after the portfolio is officially closed. If this happens, Waterfall needs to be recalculated as the overall value of the portfolio changes.

✅ Controllers

For more on the concept of controllers, read more here.

✅ Deposit Controller

In order to allow utilisation of different lender restriction strategies, without breaking the ERC4626 interface used for tranche vaults, a special mechanism needs to be put in place. Whenever a user would be about to create portfolio LP tokens, a special auxiliary smart contract is asked about the results of a particular deposit or mint. This contract should take all the parameters provided in the `deposit()` or `mint()` call, plus the message sender of the call and return a pair of values - amount of LP tokens that will be minted on deposit, or amount of asset tokens that need to be sent in order to mint desired amount of LP tokens, plus the amount of asset tokens that need to be sent as a fee to the management fee beneficiary address. In the simplest case, this contract might be just a whitelist that only checks its internal state and facilitates standard, proportional deposits or mints, but this interface allows implementation of more sophisticated strategies. If there are any other pieces of data that need to be passed, they would need to be passed directly to the controller contract, before interacting with the vault. So for example the whitelist doesn't necessarily need to be managed manually by the PM, but might automatically add anyone that provides a signature of some particular piece of data. Each vault has to have its own Deposit Controller.



✅ Controlled Functions

`deposit(assets, receiver)`

`onDeposit(msg.sender, assets, receiver)`

Returns:

- `shares` - how many shares are gonna be minted to the `receiver`
- `depositFee` - how much is going to be subtracted from `assets` and sent to the manager beneficiary address

`mint(shares, receiver)`

`onMint(msg.sender, shares, receiver)`

Returns:

- `assets` - how many assets are going to be sent from `receiver` to the portfolio
- `mintFee` - how many assets are going to be sent from `receiver` to the manager fee beneficiary

Additionally Deposit Controller needs to handle all the necessary view functions. Their interfaces are identical to the original ERC-4626 interfaces.

- `previewDeposit(assets)` - returns how many LP will the depositor get for the assets
- `maxDeposit(receiver)` - returns max amount of assets that can be put into deposit (before fee subtraction)
- `previewMint(shares)` - returns how many assets does user need to send to mint particular number of shares (actually deposited + fee)
- `maxMint(receiver)` - returns max amount of shares that can be minted

✅ Withdrawal Controller

Analogically to the deposits and mints there is a controller, responsible for handling withdrawals and redeems. Withdrawal controller, on each withdrawal or redeem attempt, will receive all call arguments, plus the message sender and determine the amount of assets that is being taken out of the vault, or amount of LP tokens burned, plus fee for the manager.

✅ Controlled Functions

`withdraw(assets, receiver, owner)`

`onWithdraw(msg.sender, assets, receiver, owner)`

Returns:

- `shares` - how many shares are gonna be burned from the `owner`
- `withdrawFee` - how much is going to withdrawn from the vault on top of the `assets` and sent to the manager beneficiary address

`**redeem(shares, receiver, owner)**`

`onRedeem(msg.sender, shares, receiver, owner)`

Returns:

- `assets` - how many assets are going to be sent from the vault to the `receiver`
- `mintFee` - how many assets are going to be taken from the vault on top of the `assets` and sent to the manager fee beneficiary

Additionally Withdrawal Controller needs to handle all the necessary view functions. Their interfaces are identical to the original ERC-4626 interfaces.

- `previewWithdraw(assets)` - returns how many LP will one need to burn in order to get desired amount of assets
- `maxWithdraw(owner)` - returns max amount of assets that can be withdrawn (after paying fee)
- `previewRedeem(shares)` - returns how many assets will user get (after paying fee) for a particular redeem
- `maxRedeem(receiver)` - returns max amount of shares that can be burned to redeem

✅ Transfer Controller

Transfer controller determines whether particular ERC-20 vault LP token can be transferred. Transfer controller does not reimplement the whole logic of the transfer, but in this case simply returns a boolean determining, whether a particular transfer is allowed or not. In order to return this value transfer controller would receive all parameters of the transfer - sender, recipient, amount and the message sender.

✅ Controlled Functions

`transfer(to, value)`

`transferFrom(from, to, value)`

`onTransfer(msg.sender, from, to, value)`

Returns:

- `isTransferAllowed` - boolean determining whether a particular transfer is allowed or not

✅ Fees

Fees can be configured separately for each tranche of the vault. Below are all of the fee types that can be applied to vaults:

1. Block-By-Block continuous fees
   - Protocol fee *(required, set by protocol)*
   - Management fee *(optional, set by PM)*
2. Additional optional fees
   - LP Deposit fees *(optional, set by PM)*
   - LP Redemption fees *(optional, set by PM)*
   - Instrument origination and/or repayment fees *(optional, set by PM)*

Protocol fees & Management fees *(Block-By-Block continuous)*

Protocol fees accrue block-by-block during the Live state and in the Closed state of a vault. **Management fees** also accrue block-by-block, but only during the Live state of a vault.

Protocol fees and Management fees are paid on each interaction with the vault, including lender deposits, lender redemptions, disbursements made by the PM, repayments from borrowers, or updates to the vault value ("NAV updates").

Whenever an interaction happens, the following sequence is executed by the vault smart contract:

1. Check following parameters:
   - Current TVL (before the action is executed)
   - Time elapsed since previous interaction
   - Fee rate (basis points per annum)
2. Calculate fee accrued for the period between the current and previous interaction
3. If there are sufficient funds available in the vault, send fees (incl. any previously unpaid fees) directly to fee beneficiary addresses
   - Protocol fee is sent to TrueFi protocol treasury
   - Management fee is sent to an address set by the PM
   - If there are not sufficient funds available, save the value of fees that couldn't be paid as unpaid fees

Additional fees *(optional, configurable by PM)*

LP deposit fees and redemption fees are handled entirely by Deposit Controller and Withdrawal Controller contracts, respectively. Vaults can charge custom management fees upon deposit or redemptions if respective controllers implement proper logic and return non-zero fee values.

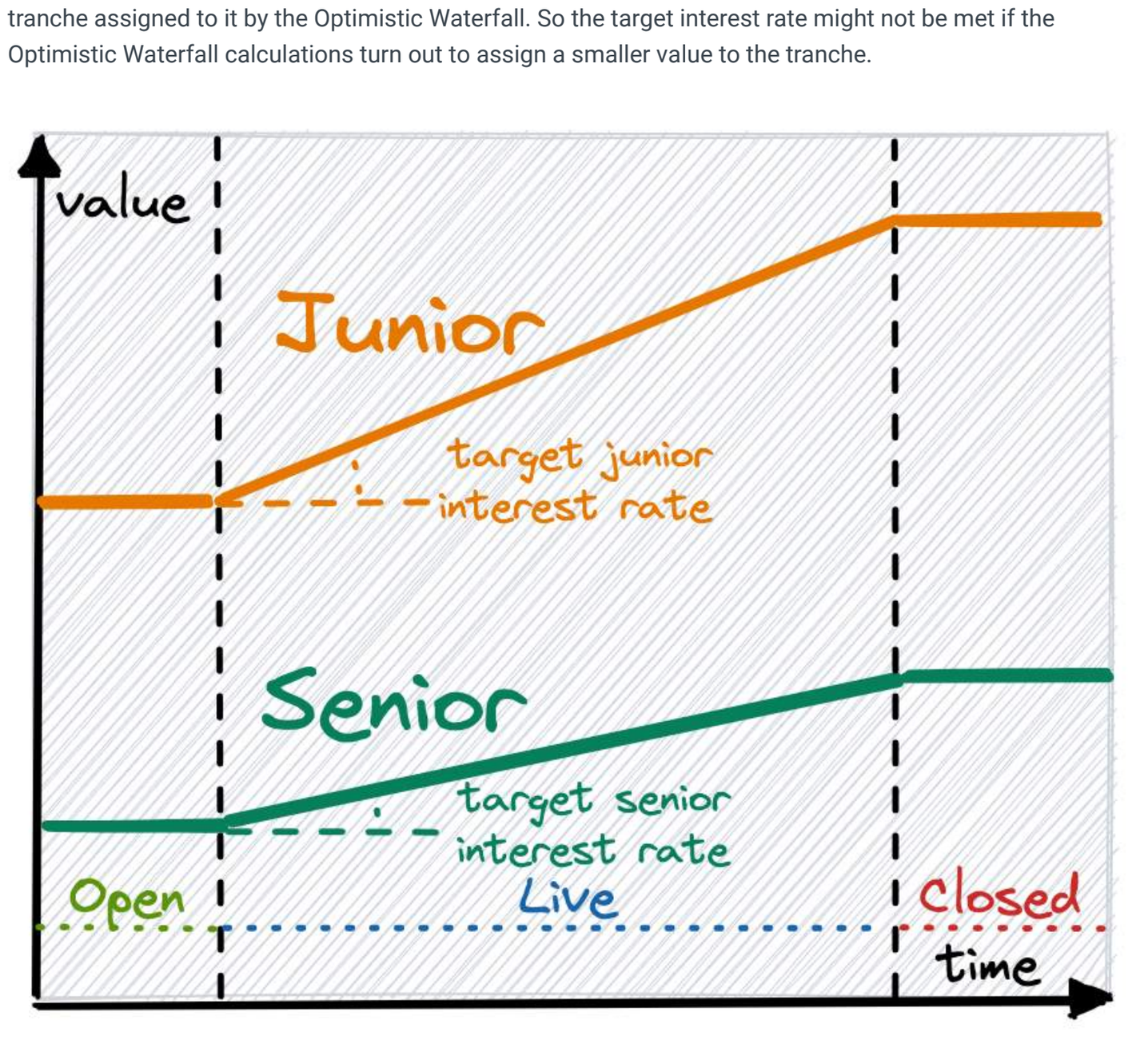
Additionally, instrument origination fees and/or instrument repayment fees can be implemented within the mechanism of an instrument itself. The vault smart contract itself does not implement any mechanism that facilitates charging such fees.

✅ Interest Accrual

Value of each subsequent tranche can be fetched by calling a proper method on a proper ERC4626 Vault. It works differently for different tranches.

✅ Senior & Junior Tranches

Value of Senior and Junior tranches is flat and remains unchanged when a portfolio is in an Capital Formation or Closed state. When it is in Live state, Senior and Junior tranches' values increase linearly at the pace defined by the target interest rate parameters. This value should be capped by the value of the tranche assigned to it by the Optimistic Waterfall. So the target interest rate might not be met if the Optimistic Waterfall calculations turn out to assign a smaller value to the tranche.

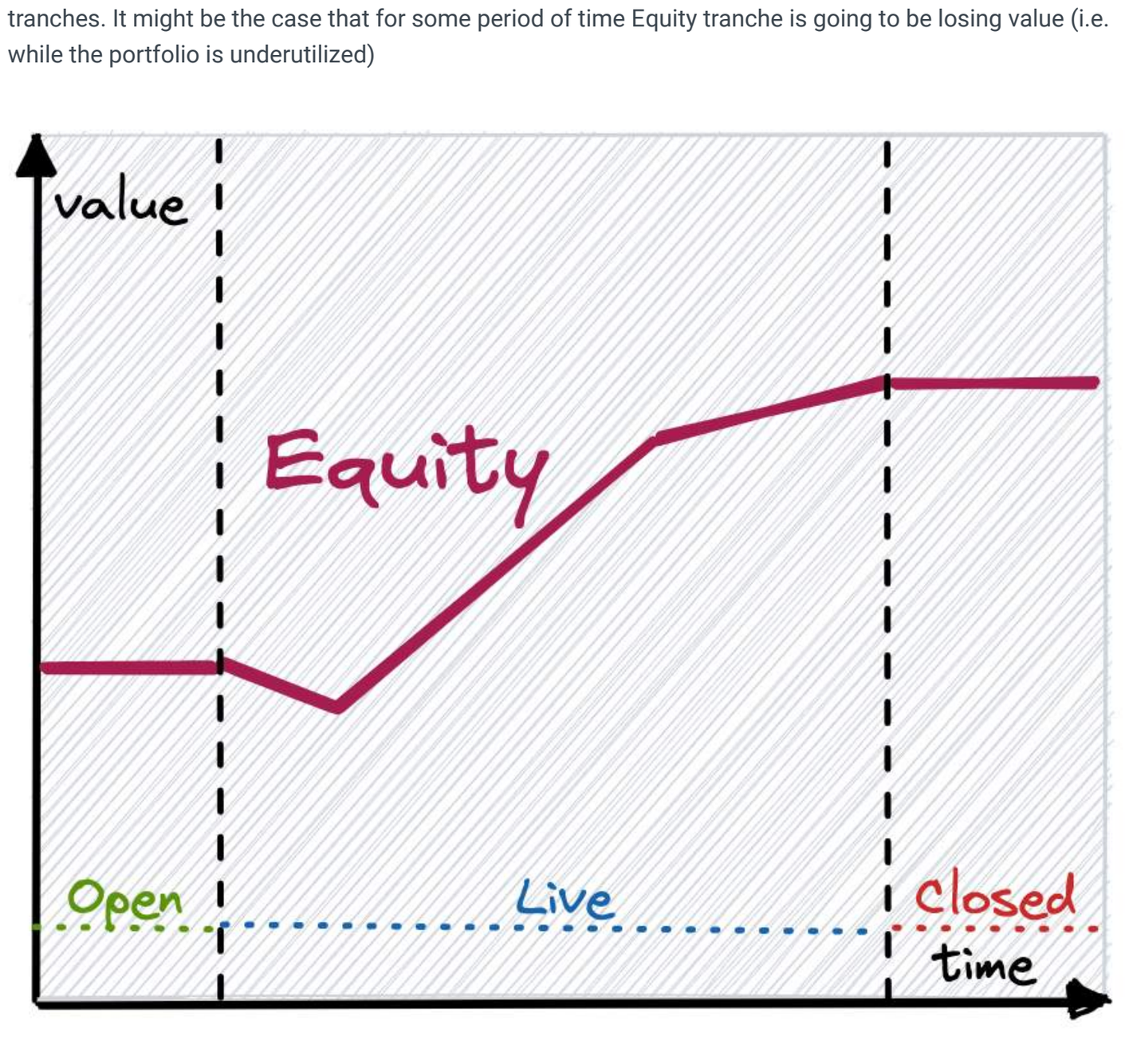


✅ Compounding effect

Whenever the value of the LP token is repriced in the Live state (i.e. because of deposit or withdrawal), already accrued interest is accounted towards the total value of the portfolio and now is also accruing interest. So the interest is compounding with each "tick" of the portfolio. For portfolios that do not last very long or do not have very high interest rates, the compounding effect is insignificant. You can check this Compounding vs Simple doc to see example calculations comparing simple interest versus compounding interest.

✅ Equity Tranche

Equity tranche, similarly to Junior and Senior tranches remains flat in the Capital Formation and Closed states. While a portfolio is in the Live state the value of the Equity tranche might be highly volatile. It is calculated as the remaining value of the total portfolio value after subtracting values of all of the other tranches. It might be the case that for some period of time Equity tranche is going to be losing value (i.e. while the portfolio is underutilized)



✅ Clarifications

In order to make the portfolio more attractive for Senior lenders it is possible to first onboard equity capital (or Equity and Junior capital), deploy it and then, once the PM has proven that they will perform sensible capital deployments, onboard Senior capital. PM can set the ceiling of the Senior tranche to 0 at the beginning and only start gathering Junior and Equity capital. Once enough funds are collected they can switch the Portfolio state to Live, and disburse the first batch of loans. Once Senior lenders can clearly see what PM is investing in, PM opens Senior deposits, by increasing senior tranche ceiling. Now, each newly collected Senior deposit would be deployed into the same instruments PM already used. If implemented, PM can also use the "Warehousing Line" product to seed the Portfolio.

Credit Vault contract overview

Credit Vaults are multi-tranche vaults allowing lenders to deposit into one or more specific buckets of funds ("tranches") managed by the portfolio manager. Credit Vaults can have up to 3 tranches, enabling each tranche to deliver unique risk-return profiles.

Below is an overview of Credit Vault contracts.

Contracts

StructuredPortfolio

`StructuredPortfolio` is a contract responsible for loan management and Tranche value calculations. It might hold any number of tranches (limited by gas) but at launch, credit vaults are intended to contain 1, 2, or 3 tranches.

`StructuredPortfolio` extends `LoansManager` contract.

`StructuredPortfolio` has 3 states: `CapitalFormation`, `Live` and `Closed`.

The credit vault goes into `Live` state when the `start()` method is called. It transfers all funds from tranches to the credit vault and enables the ability to create and fund loans. Each tranche value is now calculated using the debt waterfall algorithm.

Creating and funding loans is only possible in `Live` state.

When the vault's end date passes, anyone can close the vault. Manager can close a vault prematurely if there are no ongoing loans. This will transfer funds back to tranches according to the waterfall algorithm.

StructuredPortfolioFactory

`StructuredPortfolioFactory` is a factory contract that creates `StructuredPortfolio` contracts along with its `TrancheVault` contracts and controllers.

TrancheVault

`TrancheVault` is the contract that allows users to deposit/withdraw funds and to manage the vault. `TrancheVault` creates checkpoints on every action that changes total tranche value. This allows the vault to calculate linear growth since the last change in waterfall calculations.

`TrancheVault` handles paying fees to the manager and to the protocol. Fees are calculated continuously but are transferred on every checkpoint update.

`TrancheVault` also manages `DepositController`, `WithdrawController` and `TransferController`. `TrancheVault` supports ERC20, ERC165 and ERC4626 interfaces.

ProtocolConfig

The `ProtocolConfig` contract holds parameters necessary for fee accruals.

These parameters are:

- `defaultProtocolFeeRate`: the protocol fee (in basis points) that will be charged on each vault
- `protocolAdmin`: the address of the protocol owner
- `protocolTreasury`: the address where all fees are transferred
- `pauserAddress`: address of developer multisig for emergency pausing the protocol
- `customFeeRates`: custom fee model that can be defined by the manager

All of these parameters are settable by the protocol admin.

LoansManager

`LoansManager` is an abstract contract that is an adapter to `FixedInterestOnlyLoans` that allows to add/fund/cancel/repay them.

Controllers

Controllers regulate different aspects of how TrueFi products work.

DepositController

`DepositController` checks whether a lender is allowed to deposit and checks maximum deposit amounts for each lender. Managers can choose to enable or disable deposits.

WithdrawController

`WithdrawController` manages whether a lender can withdraw and checks maximum withdrawal amounts for each lender. Managers can choose to enable or disable withdrawals prior to the vault's maturity date. This controller is similar to `DepositController` but for withdrawals.

TransferController

`TransferController` has a single method `onTransfer` that is called when a tranche's ERC20 transfer is made.

The basic implementation of this controller always returns true (all transfers enabled). This could be swapped for a different controller to add custom functionality (e.g. only transfers between specific addresses allowed, no transfers allowed, etc).

Debt Waterfall

In credit vaults, the senior tranches are entitled to receive principal plus accrued interest (fixed rate *target interest*) in higher priority over the more junior tranches.

Thus in a three-tranche credit vault, the waterfall of repayments works as follows:

First, Tranche A receives principal plus target interest, then Tranche B receives principal plus target interest, and then Tranche C receives the remainder of the funds in the vault.

In summary, juniormost tranches absorb performance-based volatility within the vault, while more senior tranches feel losses only if losses exceed the size of subordinated junior tranches.

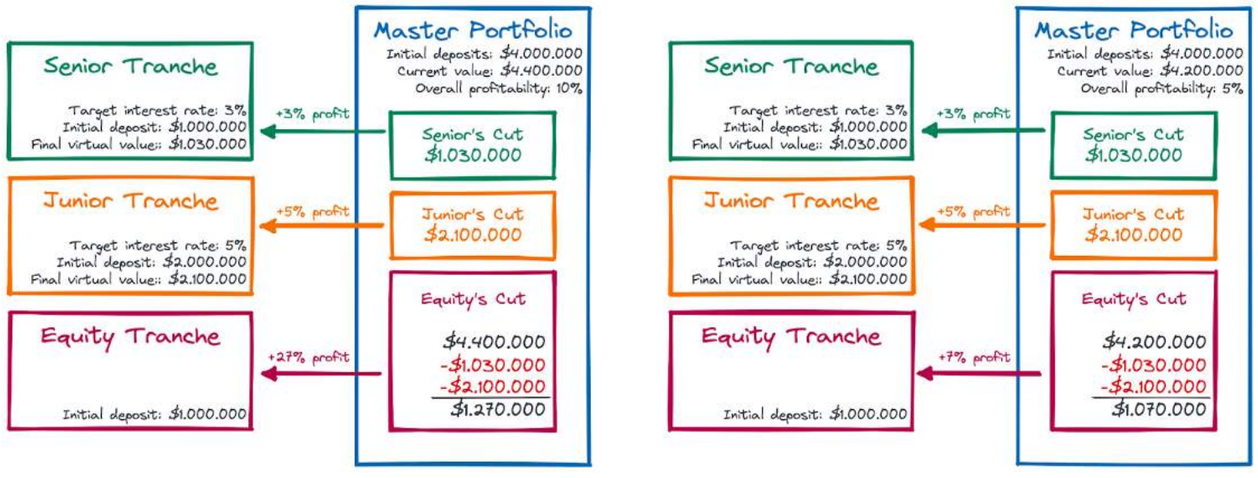
It is important to note that in `CapitalFormation` and `Closed` states, the vault consists only of idle funds. In `CapitalFormation` and `Closed` states, there are no assumptions about the future performance of deployed capital and thus the vault is only calculating how to split idle funds between different lenders.

Waterfall details

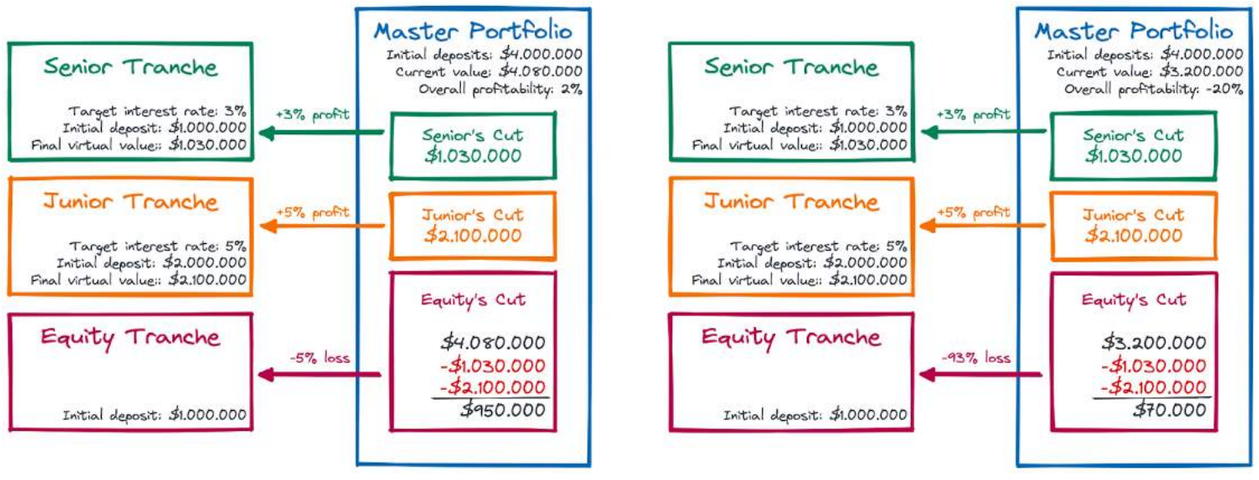
First, the senior tranche would get its principal plus target interest, then junior would get its principal plus target interest and then equity would get the rest. So basically the equity tranche will absorb most of the performance-based volatility, while the junior tranche will be absorbing default losses if the total losses are going to be larger than the equity tranche.

Important to notice is that in the Open and the Closed states, the portfolio consists only of cash, which means that there are no assumptions about future performance of currently deployed capital, but it's only a way of calculating how to split the cash between different lenders.

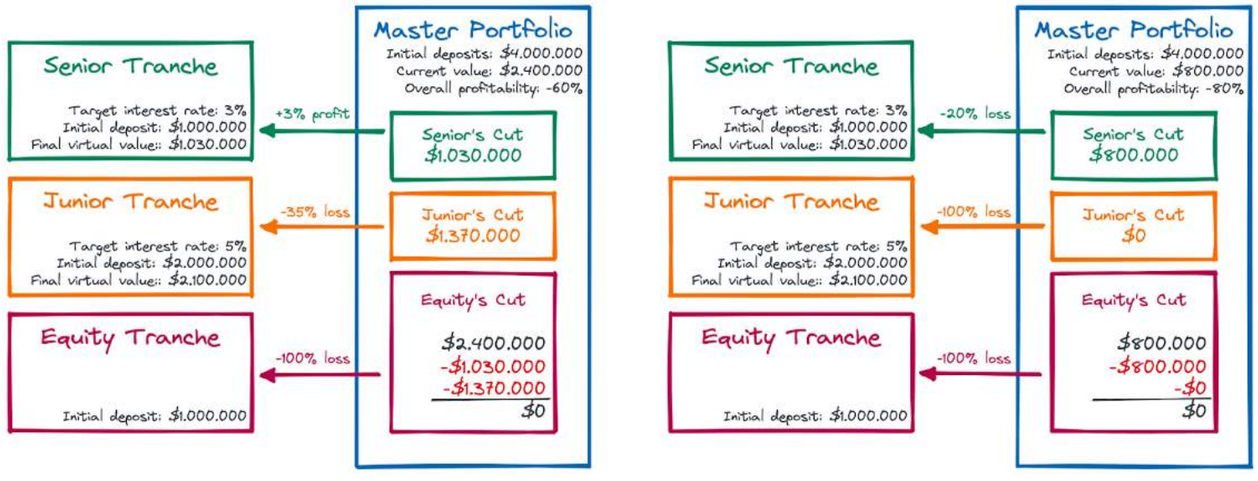
See a few examples below:



Examples of waterfall in well performing portfolios.



Examples of waterfall in poorly performing portfolios.



Examples of waterfall in default scenarios.

Asset Vaults

> **Asset Vaults are currently in *TrueFi Labs* beta.**
>
> Smart contracts are available for builders to use and experiment with. If you would like to learn more, send a message here.

What are Asset Vaults?

Asset Vaults are TrueFi vaults that facilitate off-chain credit, or "Real World Asset" (RWA), activity.

Why are Asset Vaults useful?

Asset Vaults represent off-chain instruments by using on-chain attestations, or Asset *Reports*.

By using this structure, Asset Vaults can facilitate diverse and complex RWA activities, such as:

- *Deploy capital to off-chain uses:* portfolio managers (PMs) can use Asset Vaults to create representations of off-chain debt instruments, ETFs, etc.
- *Support 100+ loans/instruments in a single vault:* Asset Vaults enable PMs to represent many loans or off-chain instruments with low gas costs.
- *Floating rate loans:* PMs can create loans that reference off-chain benchmark rates (e.g. SOFR + 200).
- *Amortizing loans:* PMs can create loans that support complex repayment schedules, following existing structures found in the traditional finance world.

How do Asset Vaults work?

Asset Vaults function similarly to **Credit Vaults**, with the exception that portfolio managers disburse funds using on-chain asset reports, rather than disbursing funds to on-chain loans.

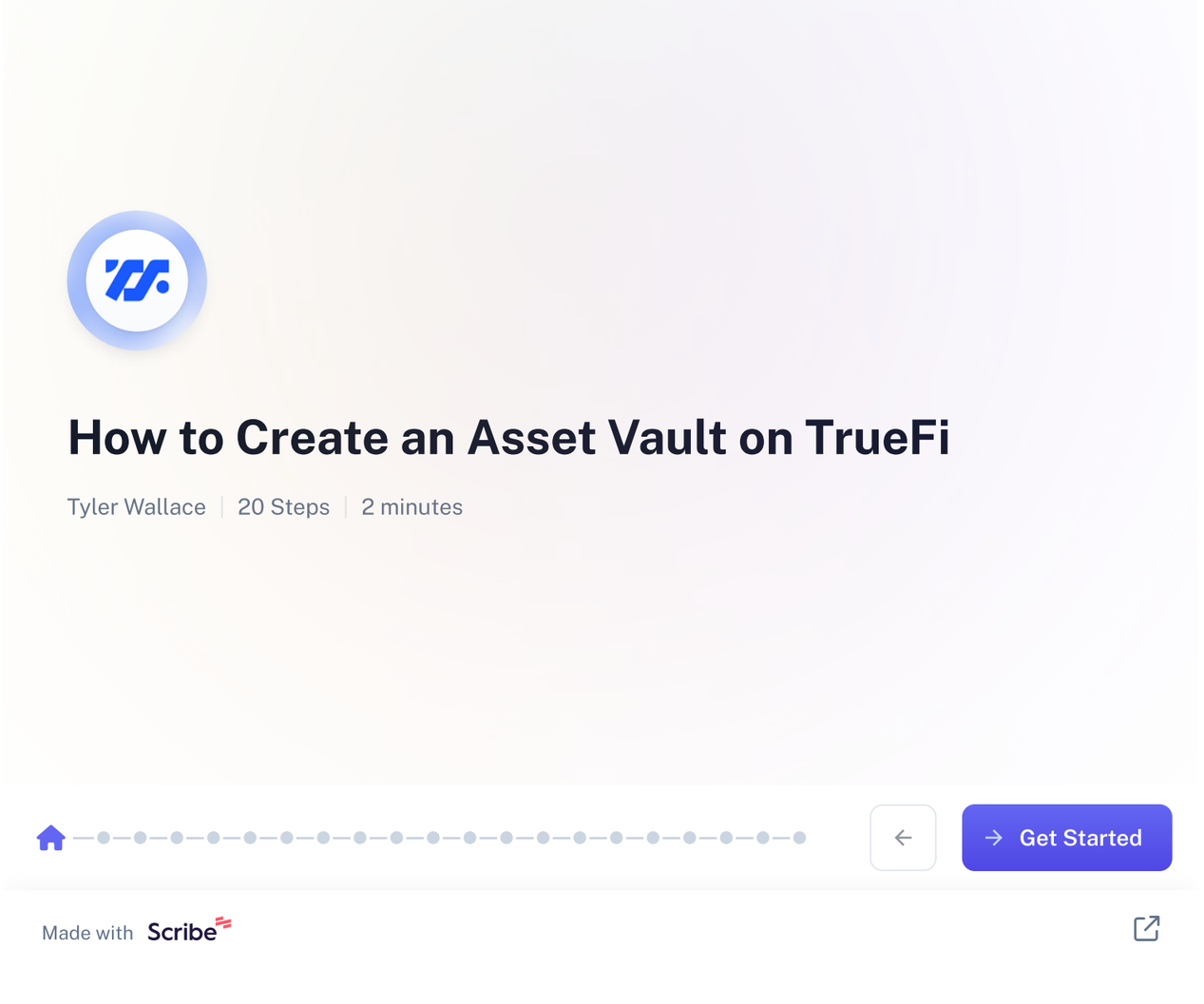
Asset Vaults are made up of modular components, enabling PMs to configure vaults to their unique needs.

Tutorial / demo

Users can test Asset Vaults by creating their own demo vault on Optimism Goerli using this link.

> Before beginning, you will need to switch to the Optimism Goerli test network and ensure your wallet is funded with test ETH and test USDC assets.
>
> To complete these steps, follow this step-by-step guide.

Follow the guide below to get started:



Asset Vault technical details

See documentation here:

contracts-fluorine/docs/interfaces/IStructuredAssetVault.md at main · trusttoken/contracts…

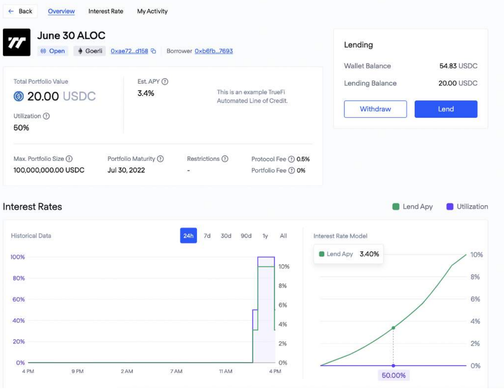
GitHub

Lines of Credit

Automated lending pools governed by supply/demand curve

> For technical docs, see **Lines of Credit technical details**

TrueFi Lines of Credit (also referred to as Automated Lines of Credit or "ALOCs") are lending pools for a single borrower, where the interest rate paid by borrowers is determined by a configurable interest rate curve.



Example of Automated Line of Credit portfolio

FAQs

Why use TrueFi Lines of Credit?

TrueFi Lines of Credit provide borrowers a flexible way to raise capital and lenders a way to deploy capital while maintaining high liquidity.

TrueFi Lines of Credit use supply and demand dynamics to set interest rates, giving lenders incentive to supply capital to pools above target utilization rates and borrowers incentive to maintain optimal liquidity ratios in each pool.

How are interest rates determined?

It's generally (but not always) the case that as utilization increases, the interest rate increases (but not past a ceiling interest rate, chosen by the borrower) and as utilization decreases, the interest rate decreases (but not past a floor interest rate, chosen by the borrower)

From the borrower's perspective: Borrower expresses that they are willing to borrow between [x]% and [y]%, with an ideal utilization rate of [90]% so that some amount of instant liquidity is available to lenders.

From the lender's perspective: Lenders want to deploy capital to loan opportunities without losing the ability to withdraw funds when needed. In return, lenders earn a dynamic rate based on the pool utilization and interest rate curve configuration.

Do lines of credit have a maturity date?

Yes, each line of credit has a specified maturity date (`endDate`). All debt and interest is due by this maturity date. Once maturity has passed, borrowers cannot draw additional capital and lenders cannot lend additional funds to this line of credit.

How does lending to a Line of Credit work?

Once a Line of Credit is created, lenders can put funds into the pool if they meet the lender restriction requirements that have been configured (lines of credit can be permissionless, or permissioned pools).

Lenders can withdraw from the portfolio's idle funds at any time funds are available. As lenders enter or exit the portfolio, the utilization of the portfolio changes, and thus, the lender APY changes as the interest rate paid by the borrower changes.

How does borrowing from a TrueFi Line of Credit work?

Once a line of credit has been created, borrowers can withdraw idle funds at any time before the pool's maturity (`endDate`). After maturity, deposit and borrow actions within the line of credit are disabled, and borrower must repay all principal and interest accrued.

How is interest accrued?

Interest owed by the borrower is accrued block-by-block. Similarly, lenders accrue interest each block as the pool value increases (pool value = principal + interest accrued).

Note that for borrowers, interest owed is calculated upon the principal borrowed (i.e. interest amount is not compounded, **borrower does not "pay interest on interest"**).

How are TrueFi Lines of Credit created?

Borrowers work with TrueFi DAO governance to create a line of credit. This should be done as a standard proposal to TrueFi DAO describing all the parameters involved in the proposal. It is a 2 step process.

1. Borrower requests Line of Credit approval by the TrueFi DAO and recommends parameters (see section below What are the parameters for each Line of Credit?)
2. If vote passes, then the address requesting creation is allowlisted for line of credit creation. Borrower deploys Line of Credit with approved parameters.

What are the parameters for each Line of Credit?

1. Borrowers define base parameters:
   - Underlying token: can denominate pool in any ERC20 (including but not limited to USDC, USDT, WETH, etc)
   - protocolFee (const) = 50 bps paid by lender at withdrawal
   - premiumFee [optional]: = set to 0 by default; can be configured by borrower
   - endDate: all loans must be repaid by end date
   - maxSize: principal in portfolio cannot exceed this amount
   - "Lender restrictions" (aka deposit strategy): portfolio can be permissioned, can use whitelist or signature to determine who can lend
2. Configure Interest rate curve (similar to Aave/Compound/other protocols, borrowing rates are defined by a rate curve). *For a worksheet example, see this spreadsheet.*
   - Set Interest rate min/optimum/max
   - Set interest rate curve "kinks"
   - Note: TrueFi contributors are happy to consult with borrowers to help set these curves.
     - For brief demo of configuration, please see this for video of Lines of Credit interest rate curve configuration: https://twitter.com/TrueFiDAO/status/1511034505028583426

What are the fees on Lines of Credit?

Lines of Credit pay a protocol fee to the TrueFi DAO treasury. Fees are quoted on a per annum basis, accrue block-by-block, and are paid upon each smart contract interaction (lend/withdraw/disburse loan/repay loan).

The example below illustrates how the protocol fee works:

> **Protocol Fee example**
>
> Take an example line of credit *Verum Fund,* which holds 1,000,000 USDC worth of loans and assume protocol fee = 50 bps per annum (0.50%).
>
> Assuming the value of Verum Fund grows linearly from 1,000,000 USDC to 1,100,000 USDC over the course of 30 days (avg. value of 1,050,000 USDC), the line of credit would pay a protocol fee of 431.51 USDC for this time period:
>
> ```
> Protocol fee = 1,050,000 USDC * 0.50% * (30/365) = 431.51 USDC
> ```

For up-to-date fee rates on each vault, please see vault pages at https://app.truefi.io/.

Lines of Credit technical details

`AutomatedLineOfCredit` is a contract that serves as a lending pool of funds managed by the borrower, who is the sole party who can borrow the funds. The borrower can: borrow funds, repay funds, set `AutomatedLineOfCredit` max size, add/remove deposit strategies, add/remove withdraw strategies, and change the transfer strategy. The borrower can also upgrade the AutomatedLineOfCredit implementation.

Contract Name	Address
AutomatedLineOfCreditFactory	0xF48E61510d9f3B46f3752c458c1b93fd051FA9cA

GitHub - trusttoken/contracts-helium: TrueFi smart contracts second release 2022.
GitHub

`AutomatedLineOfCredit` satisfies ERC-4626 requirements, meaning it meets all features described here.

Interest rate calculations

When a borrower repays a given amount of borrowed funds, they owe interest. This interest is added to their balance of the outstanding debt. The amount of interest they owe is based on the utilization rate of the funds in the `AutomatedLineOfCredit` at the time they repay a given amount of debt. The interest rate that the Borrower pays on their funds is a function of the utilization of the `AutomatedLineOfCredit`. This function is parameterized by six values which are determined by the borrower when the `AutomatedLineOfCredit` is created:

- A (1) minimum interest rate utilization threshold up to which point the interest rate is equal to the (2) minimum interest rate,
- An (3) optimum utilization rate and (4) optimum interest rate. The interest rate is determined by linearly interpolating between the minimum interest rate and the optimum interest rate when the utilization rate is between the minimum interest rate utilization threshold and the optimum utilization rate,
- A (5) maximum interest rate utilization threshold beyond which the interest rate is equal to the (6) maximum interest rate. The interest rate is determined by linearly interpolating between the optimum interest rate and the maximum interest rate when the utilization rate is between the optimum utilization rate and the maximum interest rate utilization threshold.

The `utilization` of an AutomatedLineOfCredit is equal to:

```
(borrowed_funds + unpaid_interest) / value
```

The `value` of an AutomatedLineOfCredit is equal to:

```
liquid_funds_in_ALOC + borrowed_funds + unpaid_interest - unclaimed_fees
```

When the *Borrower* borrows from the `AutomatedLineOfCredit`, `utilization` increases. When a Lender deposits into the `AutomatedLineOfCredit`, `utilization` decreases. When a Lender withdraws from the `AutomatedLineOfCredit`, `utilization` increases.

Fees

Whenever an action that changes `AutomatedLineOfCredit` value is performed (`borrow` / `repay` / `deposit` / `mint` / `withdraw` / `redeem` / `updateAndPayFee`), a fee for the TrueFi DAO is calculated and immediately transferred to the TrueFi DAO Treasury address. The fee is deducted from the `AutomatedLineOfCredit` value, so actions like `borrow` / `withdraw` / `redeem` cannot move the funds that are designated as fees. If the accrued fee cannot be repaid at the moment because of the lack of liquidity, additional information about the fee amount is stored in the contract and will be used to make the overdue payment the moment it will be possible.

The `accruedFee` value is equal to:

```
(current_timestamp - last_update_timestamp) / YEAR * protocol_fee_rate * portfolio_value
```

** `portfolio_value` does not take into account the interest that has been accrued since the last update.

`protocol_fee_rate` is the rate taken from the `ProtocolConfig` contract the last time an update was made. This means that if the DAO decides to increase the fees, the higher rate will be applied for `accruedFee` calculation since the next update, not the current one.

Deposit / Withdrawal Strategies

Functions enabling Lenders to deposit/withdraw funds to/from the contract can additionally be limited by Deposit/Withdraw Strategy. These strategies (if set), are called with a hook every time a specific action is performed on the contract. The same calldata as for the initial call is passed to them and then the strategy independently decides if this action can or cannot be performed. The strategies might also write some state to themselves on such hooks, but this is not a mandatory behavior.

AutomatedLineOfCreditFactory

`AutomatedLineOfCreditFactory` serves for deploying new AutomatedLinesOfCredit. All of the `AutomatedLineOfCredit` parameters are chosen by the borrower, except for `ProtocolConfig`.

Index Vaults

TrueFi's "Fund of Funds"

> **Index Vaults are currently in *TrueFi Labs* beta.**
>
> Smart contracts are available for builders to use and experiment with. If you would like to learn more, send a message here.

What are Index Vaults?

Index Vaults enable "fund of funds" activity on TrueFi, by allocating capital across multiple underlying TrueFi products (such as **Credit Vaults** or **Lines of Credit**).

Index Vaults do not allow managers to disburse funds to themselves, or to make loans directly.

How do Index Vaults work?

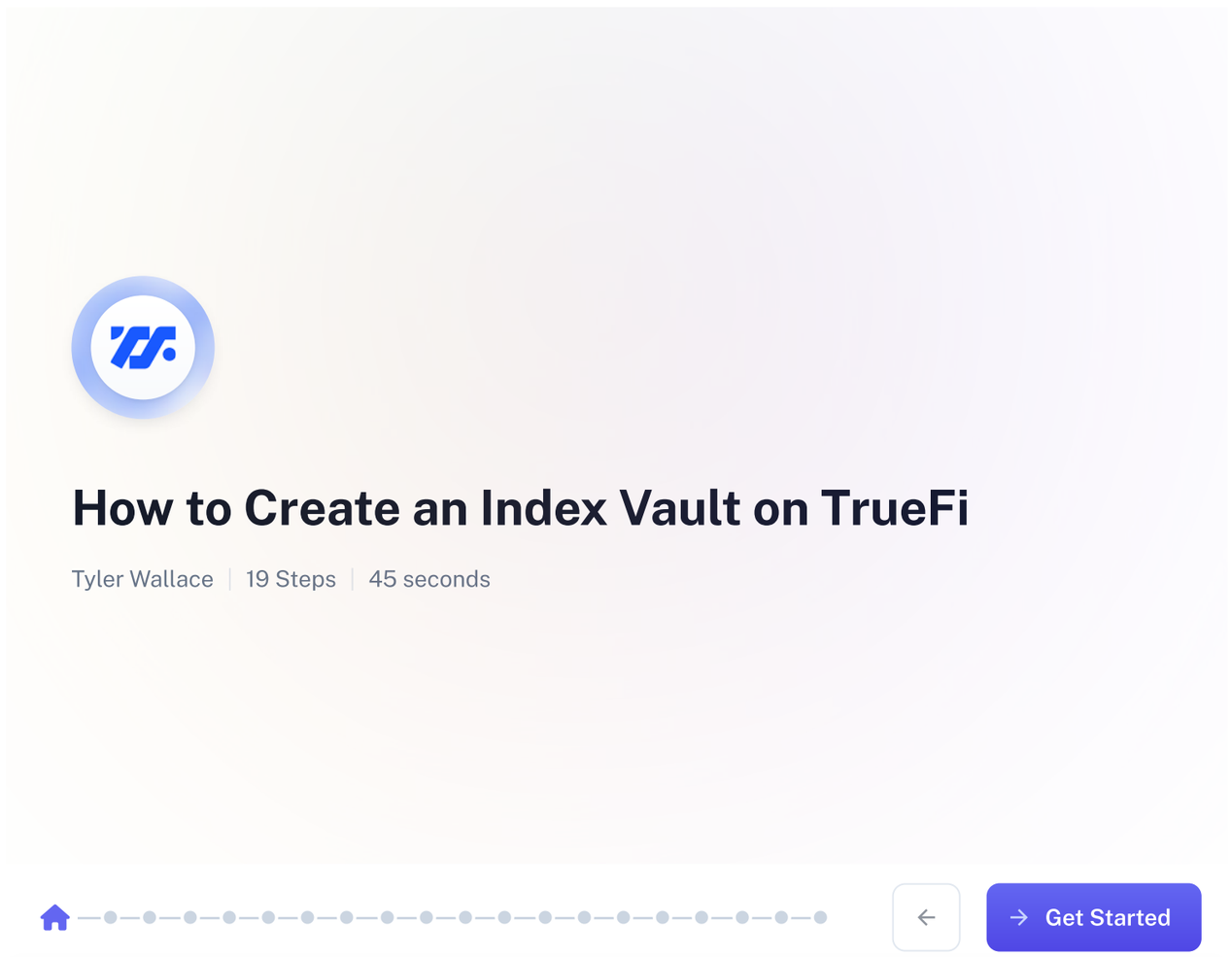
For technical details, see **Index Vault technical details**.

Tutorial / Demo

Users can test Index Vaults by creating their own demo vault on Optimism Goerli using this link.

> Before beginning, you will need to switch to the Optimism Goerli test network and ensure your wallet is funded with test ETH and test USDC assets.
>
> To complete these steps, follow this step-by-step guide.

Follow the guide below to get started:



Index Vault technical details

Technical considerations

Index Vault behavior is very similar to Credit Vaults, with exceptions in the following areas:

Capital Deployment

Instead of loans, IVs utilize an instrument called `Investment` to deploy funds.

`Investments` are LP positions at non-IV vaults on TrueFi.

Because TrueFi vaults implement the ERC-4626 interface, interactions with all types of TrueFi vaults work the same way.

Valuation

Instead of an array of loans, IVs hold an array of `investments`.

When calculating IV valuations, the IV iterates through that array and calls ERC-4626 `totalAssets()` on each of the underlying vaults to fetch the value of each `investment`.

Investment Flow

Allocating Capital

Capital allocation happens through proxying one of the following methods:

- `deposit(assets, receiver)`
- `mint(shares, receiver)`

Allocation is only possible if the target vault's token is already registered on the `vaultsRegistry`.

Adding tokens to the vaultsRegistry

Adding a token as a potential `investment` adds the token to the list of tracked asset balances and values.

In order to `register` a token, the token must:

- Be one of the allowlisted types of TrueFi products (i.e. Lines of Credit, Credit Vaults, or Flexible Portfolios)
- Not be registered already

Removing tokens from the vaultsRegistry

`unregister` removes a token from the tracked assets list.

Redeeming / withdrawing

Redeeming and withdrawing happens through proxying one of the following methods:

- `withdraw(assets, receiver, owner)`
- `redeem(shares, receiver, owner)`

Closing Vaults

Index Vaults cannot be closed while holding any active `investment`. Similar to how vaults cannot close before maturity unless all underlying loans are inactive, IVs cannot close unless all `investments` have closed first.

Other concepts

TrueFi infrastructure enables lenders, borrowers, and portfolio managers to deploy capital via lending pools and loans.

Portfolio managers on TrueFi deploy lending pools (or `vaults`) which in turn can deploy capital to borrowers via loans (`instruments`).

The following smart contracts represent vaults and instruments within the TrueFi protocol:

Vaults | Instruments | Other contracts

Credit Vault contract overview

Flexible Portfolio contracts

Lines of Credit technical details

Audits & Security

View audits here.

Controllers

🚀 Intro

Controllers regulate different aspects of how TrueFi products work.

They are responsible for key decision-making on how to perform most of the lender-facing operations. There are three Controllers that TrueFi utilizes:

- Transfer Controller
- Deposit Controller
- Withdrawal Controller

They can implement any logic and freely interact with any external on-chain or off-chain systems.

The details of how these controller work, what are they exactly responsible for and how Vaults interact with them is located below.

How to read interface notation

`whenThisFunctionIsCalledOnTheVault(arguments)`

`vaultCallsThisFunctionOnTheController(msg.sender, arguments)`

Returns:

- `controllersResponse` - description of how the Vault will handle the Controller's response

Parameters of the Vault's methods are described in the ERC-4626 documentation.

🔁 Transfer Controller

Transfer Controller is responsible for setting restrictions around transfers of the share token (LP token). Transfer controller has to be capable of making a binary decision whether a particular transfer is allowed or not.

Interface

`transfer(to, value)`

`transferFrom(from, to, value)`

`onTransfer(msg.sender, from, to, value)`

Returns:

`isTransferAllowed` - boolean determining whether a particular transfer is allowed or not (vault will execute the transfer on `true` and block the revert the transfer on `false`)

Examples

Examples of the most common Transfer Controllers are:

- open - all transfers are allowed
- blocked - all transfers are forbidden and only minting or burning the token is allowed
- restricted - only transfers to farming contract (the one distributing incentives) are allowed

---

⤵️ Deposit Controller

Deposit Controller is responsible for handling all deposits. On a high level it can set:

- Lender restrictions (who can deposit into the Vault and when)
- LP token price at the deposit
- Deposit fee paid to the Portfolio Manager
- Maximum amount that can be deposited
- *Potentially many more via implicit manipulation of the response parameters...*

Interface

`deposit(assets, receiver)`

`onDeposit(msg.sender, assets, receiver)`

Returns:

- `shares` - how many shares are gonna be minted to the `receiver`
- `depositFee` - how much is going to be subtracted from `assets` and sent to the manager beneficiary address

`mint(shares, receiver)`

`onMint(msg.sender, shares, receiver)`

Returns:

- `assets` - how many assets are going to be sent from `receiver` to the portfolio
- `mintFee` - how many assets are going to be sent from `receiver` to the manager fee beneficiary

Additionally, Deposit Controller needs to handle all the necessary view functions. The Vault calls these functions on the Controller, when they are called on the Vault. Their interfaces are identical to the original ERC-4626 interfaces.

- `previewDeposit(assets)` - returns how many LP will the depositor get for the assets
- `maxDeposit(receiver)` - returns max amount of assets that can be put into deposit (before fee subtraction)
- `previewMint(shares)` - returns how many assets does user need to send to mint particular number of shares (actually deposited + fee)
- `maxMint(receiver)` - returns max amount of shares that can be minted

Examples

Examples of the most common Deposit Controllers are:

- Simple - deposits are always allowed, setting deposit ceiling
- Simple with lender restrictions - deposits are allowed for KYCed users, setting deposit ceiling
- Almighty - Portfolio Manager can set custom deposit parameters for each user (LP token price, fee, deposit limit)

⤴️ Withdrawal Controller

Withdrawal Controller is responsible for handling all withdrawals. It can set:

- Who and when can withdraw the Vault
- LP token price at the withdrawal
- Withdrawal fee paid to the Portfolio Manager
- Maximum amount that can be withdrawn
- *Potentially many more via implicit manipulation of the response parameters...*

Interface

`withdraw(assets, receiver, owner)`

`onWithdraw(msg.sender, assets, receiver, owner)`

Returns:

- `shares` - how many shares are gonna be burned from the `owner`
- `withdrawFee` - how much is going to be withdrawn from the vault on top of the `assets` and sent to the manager beneficiary address

`**redeem(shares, receiver, owner)**`

`onRedeem(msg.sender, shares, receiver, owner)`

Returns:

- `assets` - how many assets are going to be sent from the vault to the `receiver`
- `mintFee` - how many assets are going to be taken from the vault on top of the `assets` and sent to the manager fee beneficiary

Additionally, Withdrawal Controller handles necessary view functions. Their interfaces are identical to the original ERC-4626 interfaces.

- `previewWithdraw(assets)` - returns how many LP will one need to burn in order to get desired amount of assets
- `maxWithdraw(owner)` - returns max amount of assets that can be withdrawn (after paying fee)
- `previewRedeem(shares)` - returns how many assets will user get (after paying fee) for a particular redeem
- `maxRedeem(receiver)` - returns max amount of shares that can be burned to redeem

Examples

Examples of the most common Withdrawal Controllers are:

- Simple - withdrawals are always allowed, setting withdrawal floor
- Almighty - Portfolio Manager can set custom withdrawal parameters for each user (LP token price, fee, deposit limit)

💡 General Guidelines

👮 Controller Whitelist

Controllers are approved by DAO Governance. Only approved Controllers are available in the TrueFi interface. Any new Controller proposal should consist not only of smart contract, but also a json file that would configure the controller UI in Portfolio Manager settings, a UI of actions on redeem / deposit button and so on.

Controller Whitelisting Request

A Controller Whitelisting Request should include

- All on-chain addresses necessary to properly integrate the Controller
- Controller source code
- UI JSON file
  - List of fields and options for controller column in the UI
  - Actions to execute upon click on Lender's UI elements

🌓 Portfolio Phases Support

It's important for Deposit and Withdrawal Controllers to support Portfolio lifecycle phases. Behavior of the Controller should match the intentions expressed in the documentation of a particular TrueFi product, that the Controller is aiming to serve.

Examples:

- A good and predictable Withdrawal Controller would remove all restrictions for withdrawing when the Portfolio goes into Closed state.
- A good and predictable Deposit Controller would ban all deposits when the Portfolio goes into Closed state

🗂️ State Management

Controllers typically don't hold any state, so their deployment doesn't require a separate proxy contract for storage. The easiest and the fastest way to onboard a controller is the one that doesn't hold any vault-specific state. It can store a general state and delegate decisions to other, state-holding contracts. Generally, Controllers are free to delegate any logic to other, external contracts.

The process of deploying state-holding controllers is a more complicated, as it requires the deployment of a proxy, but it is still possible and if there is a need to do so, builders are welcome and encouraged to do so.

Lender restrictions

Lender restrictions could be built within the Controller, but it's more common for logic on lender restrictions to be delegated outside of the Controller.

A Controller can utilize another contract to check whether a user is allowed to interact with the Vault by checking an allowlist, NFT ownership, SBT ownership, etc.

Instruments

Instruments are representations of loans and other financial instruments. The initial two instruments on TrueFi are:

FixedInterestOnlyLoan

BulletLoans *(deprecated)*

FixedInterestOnlyLoan

`FixedInterestOnlyLoans` is an ERC-721 contract. Each minted NFT represents a loan that must be paid back to the NFT owner.

GitHub - trusttoken/contracts-helium: TrueFi smart contracts second release 2022.
GitHub

Loan Parameters

Each loan is parametrized by:

- Underlying token with which funds are lent out and repaid (e.g. USDC)
- Principal debt (minting price)
- Period payment (interest paid at each installment)
- Period length (a period when the borrower pays installments)
- Period count (total number of installments)
- End date (date of the last installment to be paid together with the principal, set when the loan is started)
- Recipient's address
- Grace period (time by which borrower can be late in repayment of each installment)
- A canBeRepaidAfterDefault flag that allows a loan to be repaid after a loan was marked as defaulted

Loan states & state transitions

A loan can have one of the possible statuses: `Created`, `Accepted`, `Started`, `Repaid`, `Canceled`, or `Defaulted`.

Upon minting the loan status is set to `Created`. In this state, a borrower can accept a loan by calling `acceptLoan(id)` and the loan status is changed to `Accepted`.

The NFT owner can call `start(id)` on loans whose status is `Accepted`. When a loan is started the loan end date is calculated for the loan and the loan status is changed to `Started`.

The NFT owner can mark loans as canceled whose status is `Created` or `Accepted`.

The NFT owner can mark loans as `Defaulted` whose status is `Started` and the current block time is after a current period `endDate + grace` P`eriod` time.

The NFT owner can update the loan's `gracePeriod` at any time by calling `updateInstrument(id)`. The NFT owner must call `repay(id, amount)` to recalculate repaid periods and the current period end date.

The repaid amount must be equal to the period payment or period payment + principal for the last installment. The loan can be repaid after it was marked as defaulted only if the proper `canBeRepaidAfterDefault` flag was set to true.

BulletLoans

> ⚠ `BulletLoans` is in the process of being sunset.
>
> New TrueFi Capital Markets portfolios deployed after June 2022 use `FixedInterestOnlyLoan`

`BulletLoans` is an ERC-721 contract. Each of the tokens represents a single loan.

contracts-ragnarok/BulletLoans.sol at main · trusttoken/contracts-ragnarok
GitHub

All loan parameters can be read from LoanMetadata struct. `BulletLoans` contract enables loan creation, facilitates loan repayment and allows managing the loan's state and parameters.

Methods

`createLoan( IERC20 _underlyingToken, uint256 _principal, uint256 _totalDebt, uint256 _duration, address _recipient )`

Manager can create loan by passing the principal to be lent, the total debt to repaid, duration of the loan, and the address of the recipient. Total debt to be repaid cannot be less than principal amount.

`repay(uint256 instrumentId, uint256 amount)`

Existing loan can be repaid partially or in full. If loan is paid in full, this function will mark the loan’s status to ‘Fully Repaid’. Note that a loan cannot be overpaid.

`markLoanAsDefaulted(uint256 instrumentId)`

Only the portfolio’s manager can mark a loan as defaulted.

`markLoanAsResolved(uint256 instrumentId)`

Only the portfolio’s manager can mark a loan as resolved. Intended to be used for situations after partial repayment where a loan workout has been agreed to.

`updateLoanParameters( uint256 instrumentId, uint256 newTotalDebt, uint256 newRepaymentDate )`

Manager can modify loan terms, changing maturity date or total debt to be repaid.

`updateLoanParameters( uint256 instrumentId, uint256 newTotalDebt, uint256 newRepaymentDate, bytes memory borrowerSignature )`

Manager can modify loan terms, changing maturity date or total debt to be repaid. In order to change the maturity date to an earlier date or increase the repayment value, the borrower must consent and provide a signature.

View methods

`principal(uint256 instrumentId)`

Returns principal amount of loan.

`underlyingToken(uint256 instrumentId)`

Returns underlying token (e.g. USDC, USDT) of the loan.

`recipient(uint256 instrumentId)`

Returns borrower’s address.

`endDate(uint256 instrumentId)`

Returns maturity date of the loan.

`unpaidDebt(uint256 instrumentId)`

Returns remaining amount to be paid, i.e. total debt less repaid amount.

`getStatus(uint256 instrumentId)`

Returns status of loan (`Issued` , `FullyRepaid` , `Defaulted` , `Resolved`).

[Legacy] DAO pools

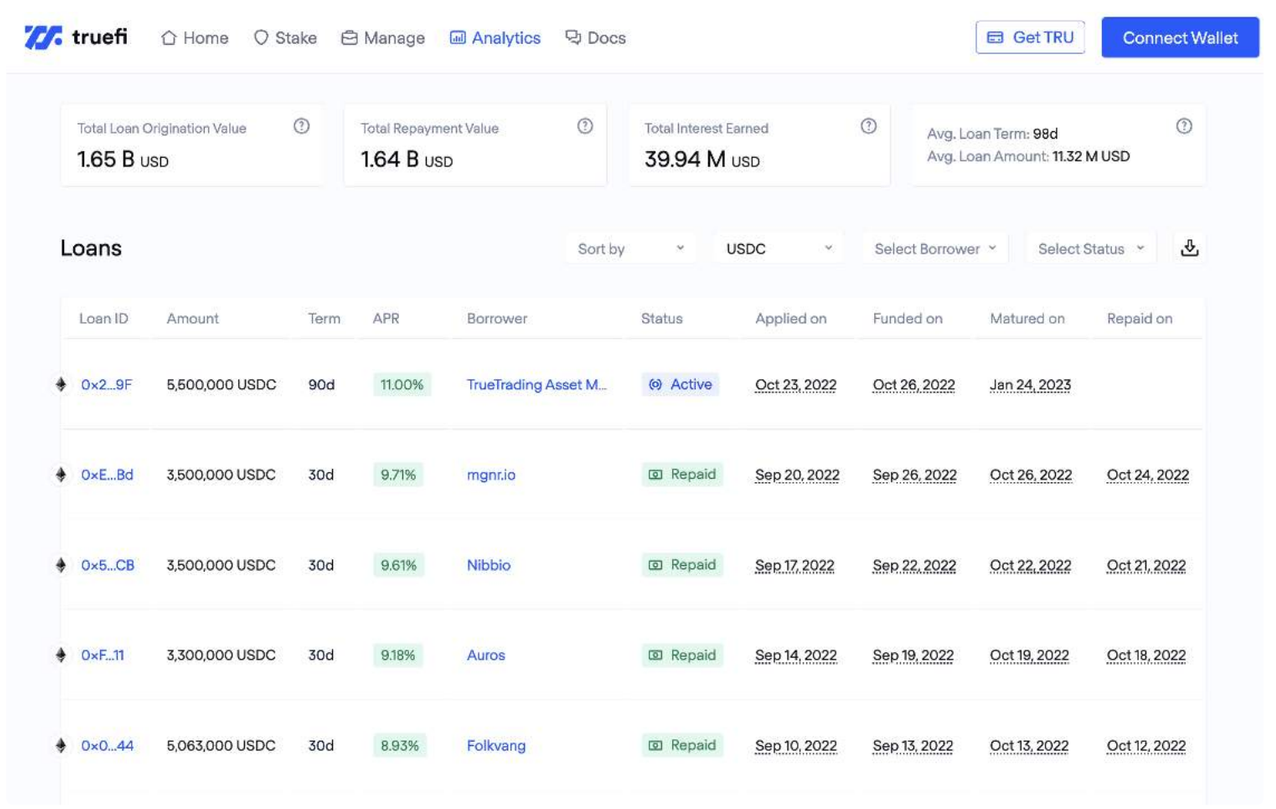
[Legacy] Lending pools governed by TRU stakers

> For developer docs see **Developer docs**

TrueFi DAO-managed lending pools (tfUSDC, tfUSDT, tfTUSD, tfBUSD) lend to institutional crypto borrowers that request loans from the protocol. Loans must be approved by the protocol and meet risk / return criteria set by the protocol.

For Lenders

For Borrowers



TrueFi DAO pool loan lists are available at https://app.truefi.io/loans

delt.ai loan: Jan 2023 airdrop claiming instructions

Airdrop claiming instructions, provided by Archblock

Summary

The airdrop comes in the form of an ERC4626 tokenized vault (0x37C5867ef19DbE096dF8E125F33f895234E02875), with mints and deposits disabled. This contract has undergone an internal security review and an external audit by ChainSecurity.

Shares of the airdrop are proportional to token holders of the tfUSDC lending pool as of January 9, 2023, the original end date of the delt.ai loan. This includes lenders who farmed their tfUSDC holdings, as well as claimable tfUSDC rewards from the stkTRU contract.

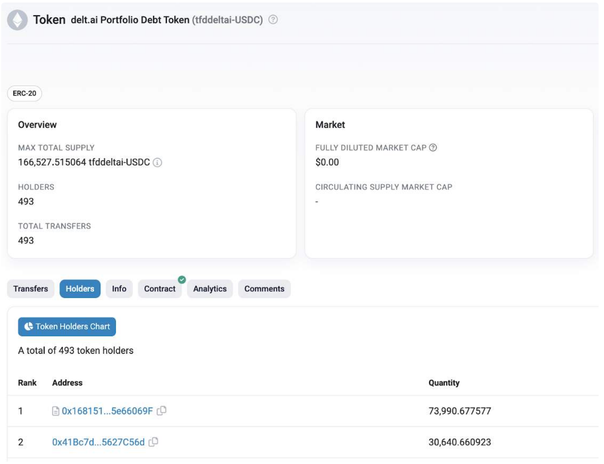
In order to prevent locked funds in the contract, you will have until a deadline of July 8, 2023, to redeem your shares of the interest airdrop.

After this date, TrueTrading will recover the remainder of funds.

NOTE: You may want to wait until gas prices are low before calling the `redeem()` function.

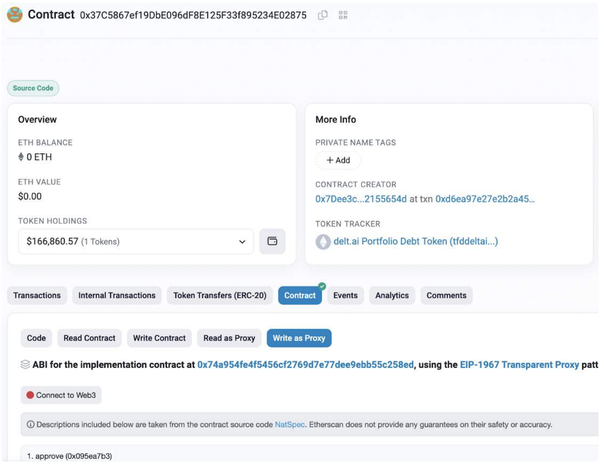
Step-by-step instructions

1. **Check your balance of shares here:**

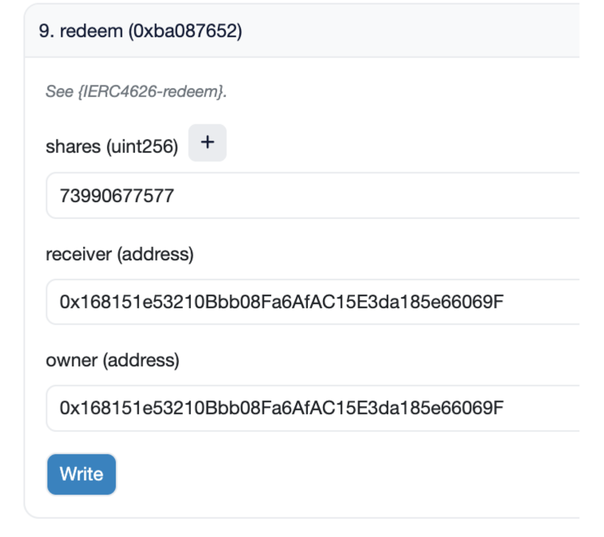


In this screenshot example above, the address `0x168151` has a balance of `73,990.677577`, which means it can claim 73,990.677577 USDC. There should be six digits after the decimal point.

2. **Connect your wallet to Etherscan on this page by clicking the "Connect to Web3" button as indicated below:**



3. **Click, "Contract" and "Write as Proxy", then call the redeem() function (#9, 0xba087652):**
   - For both receiver and owner, enter your wallet address from #1: `0x168151e53210Bbb08Fa6AfAC15E3da185e66069F`
   - For shares, enter your balance found in step #1, but using 6 decimals
     - Following the example from #1, the shares would be `73990677577`
     - Note: This number should contain no decimal point and no commas. There should be six digits after the decimal point from #1.



4. **Review this transaction carefully, click Write, and then confirm via your wallet.**

Support

Please reach out to support@archblock.com if you have a smart contract with claimable airdrop balance, but no ability to call redeem(). If you can prove ownership of the contract, then Archblock may be able to work out a way for you to receive your share.

If you have any questions or need assistance, please contact support@archblock.com.

delt.ai loan: July 2023 airdrop claiming instructions

Airdrop claiming instructions, provided by Archblock

Summary

The airdrop comes in the form of an ERC4626 tokenized vault (00xfBc33285d9f58d34fE239bFA3732036d5c53a1665), with mints and deposits disabled. This contract has undergone an internal security review and an external audit by ChainSecurity.

Shares of the airdrop are proportional to token holders of the tfUSDC lending pool as of January 9, 2023, the original end date of the delt.ai loan. This includes lenders who farmed their tfUSDC holdings, as well as claimable tfUSDC rewards from the stkTRU contract.

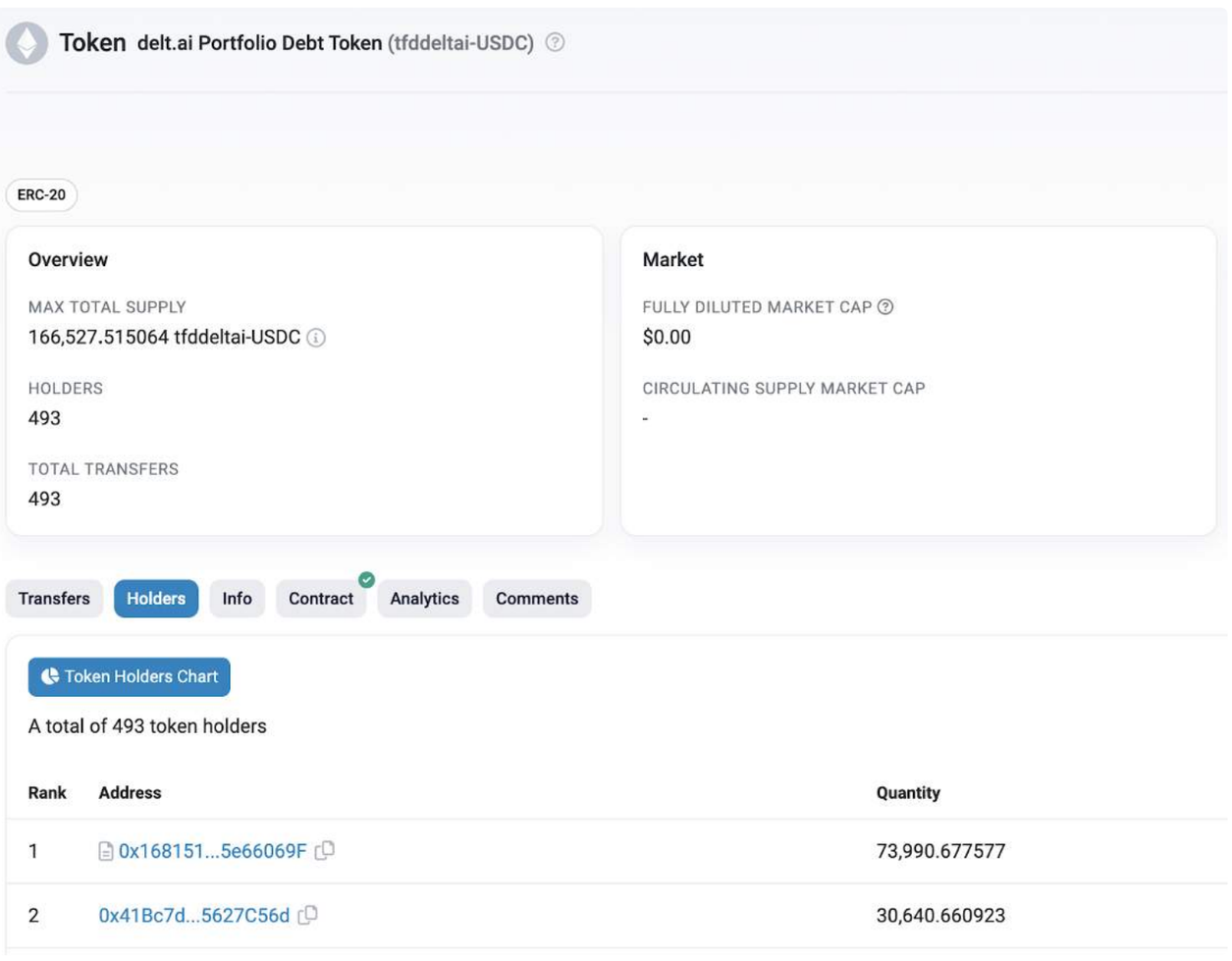
In order to prevent locked funds in the contract, users will have until a deadline of January 21, 2024, to redeem your shares of the interest airdrop.

After this date, TrueTrading will recover the remainder of funds.

> ⓘ You may want to wait until gas prices are low before calling the `redeem( )` function.

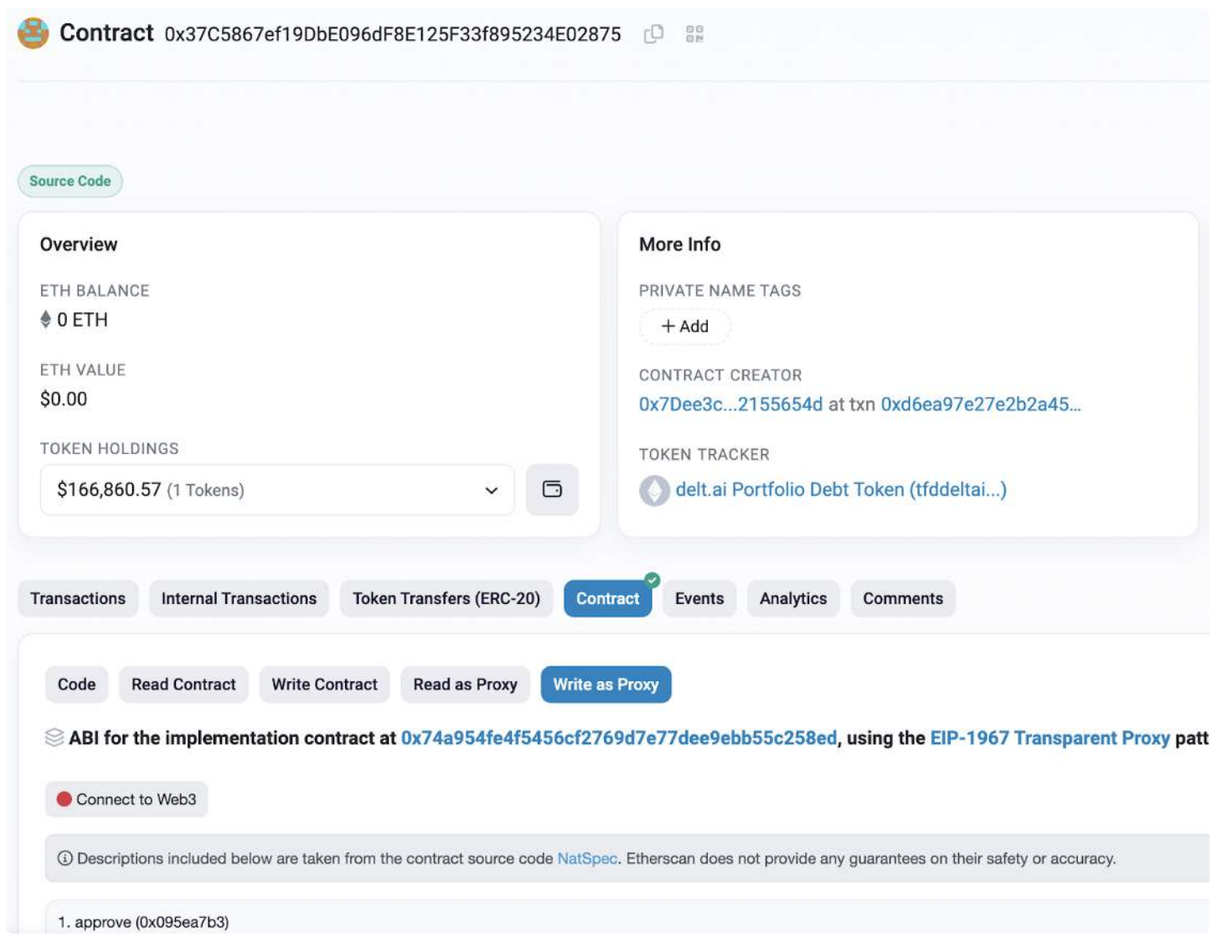
Step-by-step instructions

1. **Check your balance of shares here:**

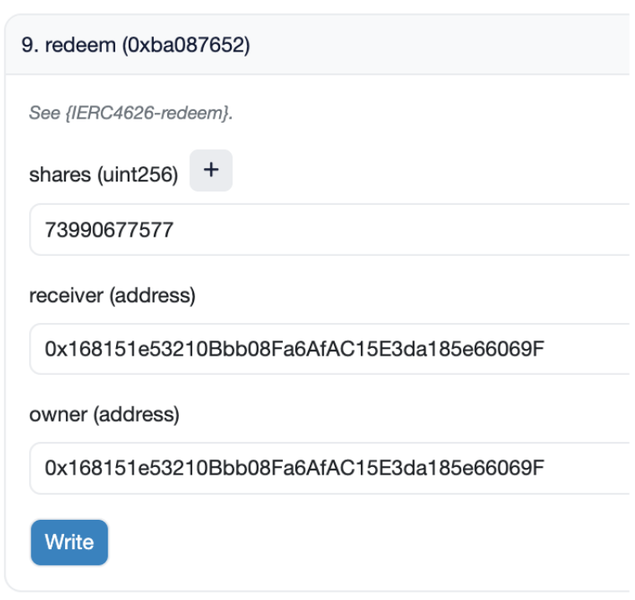


In this example, the address `0x168151e53210Bbb08Fa6AfAC15E3da185e66069F` has a balance of 73,990.677577, which means it can claim 73,990.677577 USDC. There should be six digits after the decimal point.

2. **Connect your wallet to Etherscan on this page by clicking the "Connect to Web3" button as indicated below**



3. **Click, "Contract" and "Write as Proxy", then call the redeem() function (#9, 0xba087652):**



For shares, enter your balance from #1 (but drop the decimal point and any commas): `73990677577` in the example.

For both receiver and owner, enter your wallet address from #1: `0x168151e53210Bbb08Fa6AfAC15E3da185e66069F` in the example

4. **Review this transaction carefully, click Write, and then confirm via your wallet.**

Support

If you have any questions or need assistance, please contact support@archblock.com.

Please reach out to support@archblock.com if you have a smart contract with claimable airdrop balance, but no ability to call `redeem()`. If you can prove ownership of the contract, they may be able to work out a way for you to receive your share.

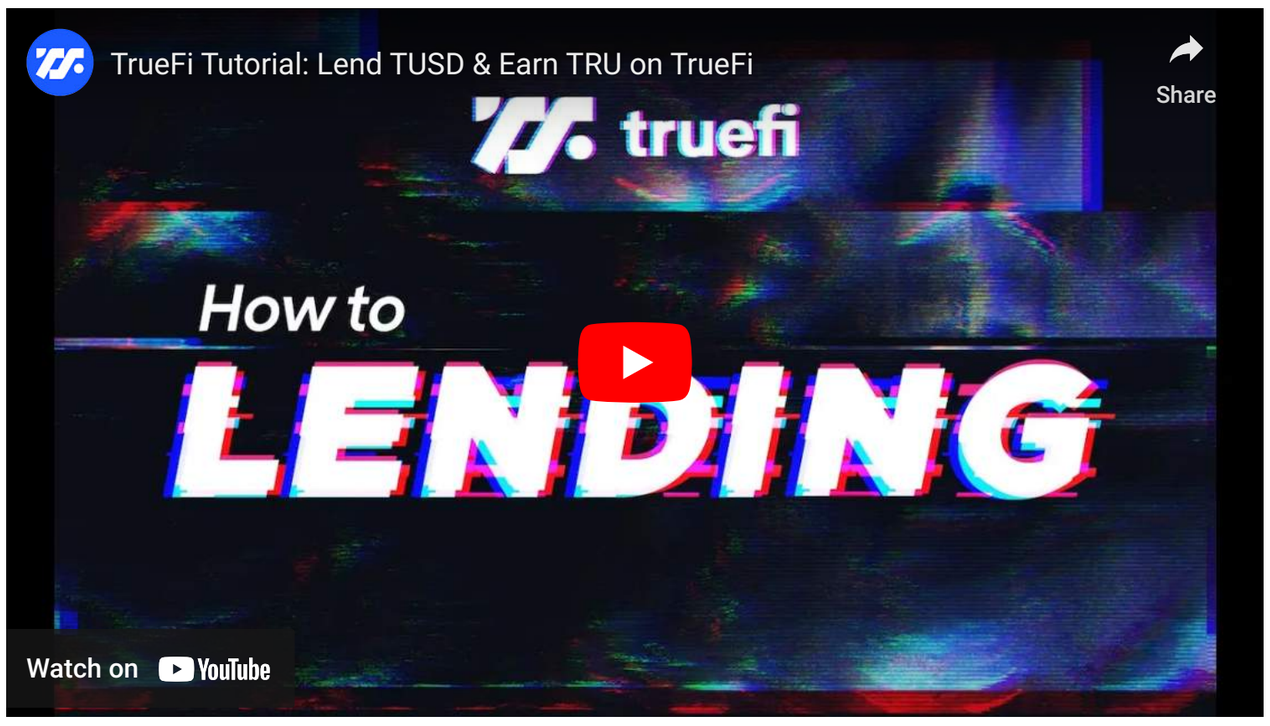
Lender FAQs

How does TrueFi generate yield?

TrueFi lending pools fund loans to borrowers who request loans from the protocol. Loans must be approved by the protocol and meet risk / return criteria set by the protocol. Uncollateralized lending has a higher risk profile than other markets and thus should provide higher returns.

How does lending on TrueFi work?

Lenders can lend stablecoins to TrueFi lending pools, which use predefined strategies to lend to creditworthy borrowers. See below for a brief demo:



TrueFi lending pools are controlled by TrueTrading, an affiliate company of TrustToken, Inc. The TrueFi lending pools only lend to a whitelist of trusted borrowers, and any excess capital that is not actively loaned out may be deployed in a DeFi protocol.

Lenders who lend to the TrueFi lending pool receive TrueFi lending pool tokens ("LP tokens"), which represent their proportion of lent capital in the TrueFi lending pool.

Is there a lockup period for lenders?

No. Lenders can redeem LP tokens any time idle funds are available in the pool. Lenders pay an *exit fee* in order to withdraw instant liquidity from the pool. This fee is calculated dynamically, depending on what proportion of the pool is idle vs. lent to borrowers.

Additionally, LP tokens can be traded on secondary markets, such as Uniswap.

Are there any fees for lending to the Lending Pools?

There are no fees for lending funds into lending pools. To withdraw funds, lenders may pay an *exit fee*.

What are the risks involved in lending to the TrueFi lending pool?

While borrowers are usually willing to pay higher rates for uncollateralized loans, these higher yields do not come without risks. Compared with collateralized lending, uncollateralized lending has two major risks:

- Potentially increased risk of loss: Protocols that require collateral are protected by that collateral in case of default. While this allows such platforms to be less selective in approving loans, uncollateralized loans come with a much higher standard of trust that must be met by a borrower. In case of default on an uncollateralized loan, a delinquent borrower will have been assessed for creditworthiness before the loan was made and will face both reputational damage and legal action.
- Potentially lower liquidity: While instant withdrawals are becoming a norm for new protocols, uncollateralized lending may not offer the same flexibility. Most borrowers for uncollateralized loans are interested in fixed-rate, fixed-term loans for predictable repayment. This means lenders who fund such loans need to be comfortable locking up their assets for the duration of the loan. TrueFi offers an alternative: the ability to withdraw their proportion of the pool tokens which would consist of stablecoins and loan tokens that you hold to maturity. You can redeem the loan tokens for the stablecoin at the end of loan terms.

You can learn more about how TrueFi mitigates risk here.

Who is responsible for taking legal actions against delinquent borrowers?

TrueTrading is currently responsible for pursuing legal recourse if a borrower defaults and may be replaced by another non-profit entity in the future as TrueFi progressively decentralizes.

How can lending pool token holders farm TRU?

Lending pool token holders can farm TRU by staking their lending pool tokens (tfTUSD, tfUSDC, tfUSDT, or tfBUSD) on the Farm page.

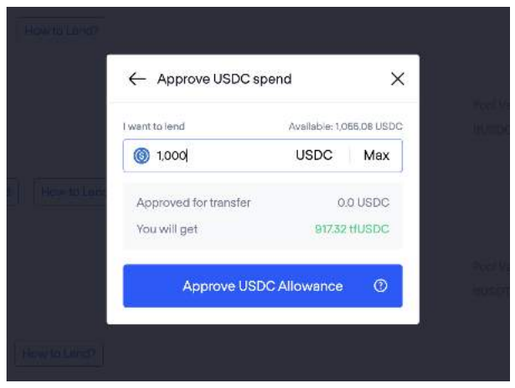
Lending to DAO pools

How do I lend?

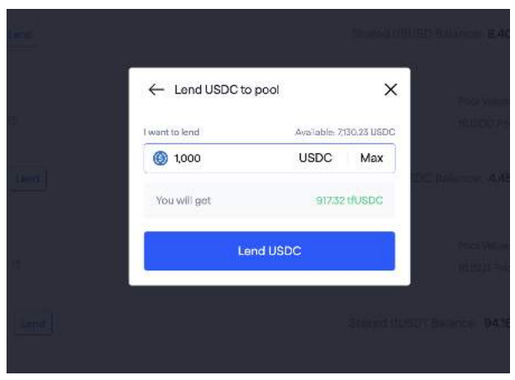
Users can lend to TrueFi DAO pools at https://app.truefi.io/lend.

Once a user is ready to lend, the user will need to complete two transactions:

1) `approve()`: User must approve the lending pool smart contract to transfer up to a certain allowance of the asset. Read more here.



2) `lend()`: User lends funds to the lending pool. In return, the lender receives lending pool tokens ("LP tokens").



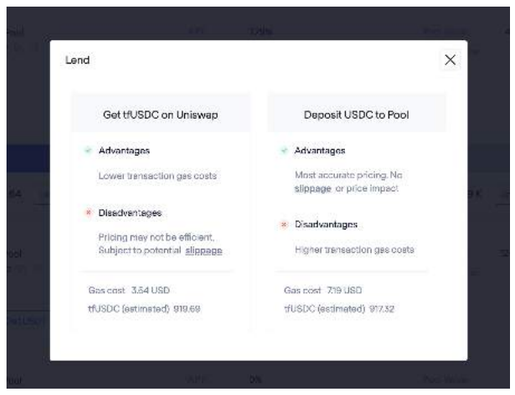
The lender can optionally stake their LP tokens to farm TRU rewards, which generate additional yield on top of underlying returns in the pool.

Why do I see two options after I click 'Lend'?

The TrueFi app helps to find lenders the best price when they lend to the pool.

Lenders can choose between (i) lending funds directly to the pool and minting LP tokens, or (ii) buying LP tokens on a secondary market (Uniswap).

In the example shown in the screenshot below, the UI shows that the user may be able to get a better price and incur lower gas costs by going directly to Uniswap to get tfUSDC.



TrueFi Lending Pool smart contracts

Pool	Address
tfTUSD	0x97cE06c3e3D027715b2d6C22e67D5096000072E5
tfUSDC	0xA991356d261fbaF194463aF6DF8f0464F8f1c742
tfUSDT	0x6002b1dcB26E7B1AA797A17551C6F487923299d7
tfBUSD	0x1Ed460D149D48FA7d91703bf4890F97220C09437
Legacy tfTUSD	0xa1e72267084192db7387c8cc1328fade470e4149

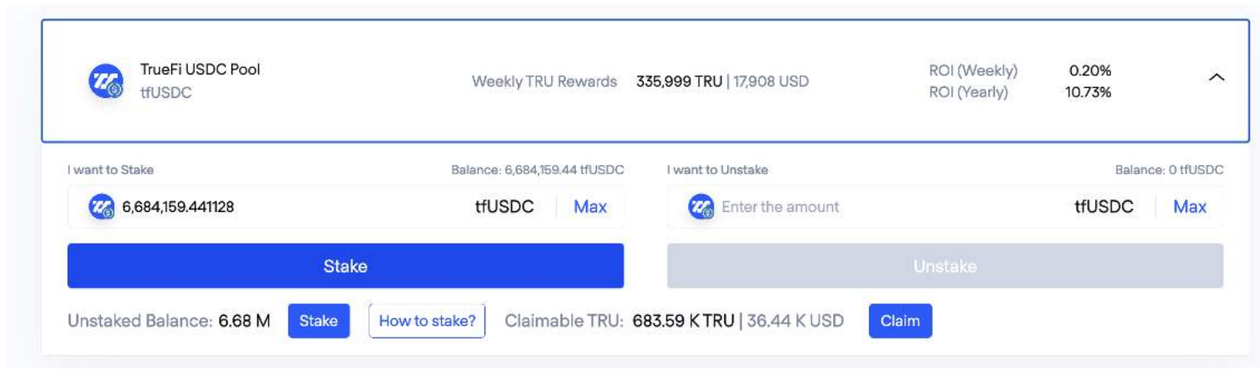
Farming TRU rewards

In addition to earning yield from loan activity in DAO pools, **lenders can also earn additional yield in the form of TRU rewards** ("yield farming").

How do I farm TRU rewards?

TrueFi DAO Pool lenders begin accruing TRU rewards as soon as their LP tokens have been staked.

Lenders can stake lending pool tokens in the liquidity gauge via the Farm page, or via the `stake()` function on the `TrueMultiFarm` contract.



How to stake LP tokens via https://app.truefi.io/farm

How do I check and claim my farm rewards?

At any given time, a lender can check the # of accrued rewards and claim rewards by visiting https://app.truefi.io/farm.

Additionally, lenders can find the # of TRU rewards by checking `claimable()` on the `TrueMultiFarm` contract below:

Name	Contract
TrueMultiFarm	0xec6c3FD795D6e6f202825Ddb56E01b3c128b0b10

Lenders can claim TRU rewards by calling `claim()` on this contract.

Additionally, lenders can call `exit()` to unstake LP tokens and claim rewards within the same transaction.

How are TRU rewards distributed?

To find the current emissions rate for a farm, users can divide the totalAmount by duration. Visit the etherscan link to the smart contract, click on Contract, then click on Read as Proxy. You will find the two parameters totalAmount and duration. The duration is in seconds.

```
TRU distribution per day = (totalAmount/10^8) / (duration/(24*3600))
```

Liquidity Gauge TRU Distributor Smart contract

Name	Contract
LinearTrueDistributor	0xc7AB606e551bebD69f7611CdA1Fc473f8E5b8f70

Are there additional economic risks involved in farming?

Farming on TrueFi lending pools involves no additional economic risk beyond risks involved as a lender. Users are inherently exposed to borrower default risk as a lender.

What is TRU?

Read more about TRU, TrueFi's governance token here.

Withdrawing funds

Liquid Exit and other FAQs

How can I exit the lending pools?

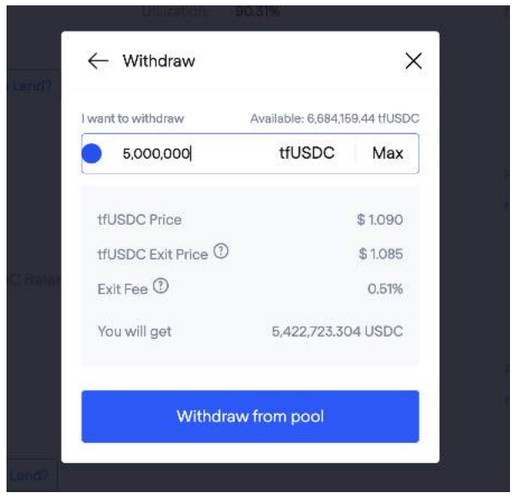
You can exit the pool by selling your lending pool tokens to the TrueFi lending pools for the stablecoin if there is enough liquid asset in the pool to support the transaction. This feature is also called liquid exit.

What is Liquid Exit?

Liquid exit addresses a key community request which is the ability to exit the TrueFi Lending Pool directly into the underlying stablecoin. Lending pool token holders can redeem their LP tokens for the stablecoin for an exit fee.

The exit fee is inversely proportional to the amount of available idle liquidity in the pool. For example, when there is a large amount of liquid assets in the pool, the fee is low. When there is a small amount of liquid asset in the pool, the fee is high. This fee is earned by the pool for the existing lending pool token holders.

The exit fee charged to you for an exit would be made available to you in the UI. If you feel that the fee charged is too high then you can wait till the pool is more liquid and try again later.



1) If there is no liquid asset in the lending pool and no liquid exit is deployed in Curve.

2) If the pool needs to liquidate its position in Curve and will incur a loss of more than 10 basis points.

Click here to view the relationship between Pool utilization and exit fees.

How lending pool (LP) tokens work

After lending to the pool, a lender receives lending pool tokens ("LP tokens"). Read further to find how users can track the value of their LP tokens and trade LP tokens.

What are Lending Pool tokens ("LP tokens")?

Lending Pool tokens (or "LP tokens") are tradable ERC-20 tokens that represent a lender's proportional representation in the pool.

In the beginning, when no loans have been disbursed by the TrueFi lending pool, lenders will receive one TrueFi Lending Pool Token (tfTUSD, tfUSDC, tfUSDT, or tfBUSD) for every stablecoin lent to the pool.

As the pool starts earning yields and disbursing loans, the value of the pool tokens may increase or decrease depending on returns within the pool. The value of the pool represents the present value of all its underlying tokens (stablecoins, loan tokens, and other tokens earned).

How do I calculate LP token prices?

We can calculate LP token price by checking the `poolValue()` and `totalSupply()` read functions on the lending pool smart contract:

```
LP token price = poolValue() / totalSupply()
```

We can use this LP token price to find how many LP tokens a lender will receive in return for lending tokens to the pool.

> **Example**:
>
> Bob lends 2,000,000 USDC to the tfUSDC pool.
>
> Given that tfUSDC `poolValue()` = 46226887530770 and `totalSupply()` 42405680290948 at the time of lending, we calculate *tfUSDC LP price = 1.0901*.
>
> Bob will thus receive 2,000,000 / 1.0901 = **1,834,675.99 tfUSDC LP tokens**

Additionally, we can calculate the value of a lender's position at any point in time:

```
Lender's position value = (# of LP tokens held) * (LP token price)
```

Are LP tokens tradable on secondary markets?

Yes, LP tokens can be traded on secondary markets. There are existing markets on Uniswap v3 today.

- tfUSDC/USDC: https://info.uniswap.org/#/pools/0xd7c13ee6699833b6641d3c5a4d842a4548030a82
- tfUSDT/USDT: https://info.uniswap.org/#/pools/0x5cc644472ed7d3198eeb23353bd9236ca578a895

Why is there a difference between calculated LP token price and its price on Uniswap?

LP token prices calculated by the lending pool assume that loans will be repaid successfully within the term, among other assumptions.

Other market participants may have use different assumptions in their calculation of LP token prices. There are several market factors that may govern the price of lending pool tokens and the TrueFi platform does not have any control over them.

How loan tokens work

What are loan tokens?

Loan tokens are non-tradable ERC-20 tokens which represent the lender's proportional representation in a loan that is issued from the Lending Pool.

Each loan issued from the Lending Pool will create a unique loan token used to track the present value of each loan. Loan tokens form an important building block of TrueFi, as they can operate independently from the TrueFi lending pools. Tokenized loans open up opportunities for calculating and tracking the value to the individual loans within the Lending Pool. TrueFi does not allow for the transfer of Loan tokens and will not create a secondary market for loan tokens. It is important to note that all Loan tokens are unique and can be tracked on the Loans page of the TrueFi website.

Creating a Loan token requires a borrower's wallet address, principal amount, term, and interest rate or APR associated with a loan. When a loan is approved by the pool, loan tokens are minted by funding the loan token contract with the principal amount. The minted loan tokens represent a share in the total amount payable at the end of the term which is the sum of principal and interest.

Loan tokens are minted at a discounted rate, meaning that loan tokens paid back in full will converge to a price of 1.000:

```
# of loan tokens minted = principal + interest owed at loan maturity
```

Once a loan is funded, the borrower can call a function which allows them to borrow the funds from the smart contract. At the end of the term, once the borrower pays back the loan, loan token holders can exchange their loan tokens for an equivalent number of stablecoins.

> **Example**:
>
> When a loan with *principal = 1,000,000 USDC, term = 30 days, APR = 12%* is approved, 1,009,863.013 (`= 1,000,000 + 1,000,000 x 12% x 30/365`) loan tokens are minted.
>
> At maturity, if the borrower repays the entire loan amount along with interest, the lending pool can exchange 1 loan token for 1 USDC.

What is the value of a loan token at any point of time?

The theoretical present value of a loan token is calculated by assuming the loan is repaid in full by the end of the loan term.

The following formulas walk through how lending pools value loan tokens (*where loan token `0xabc` represents a single loan, and `t` represents time since loan origination):*

`Value of all 0xabc loan tokens minted` `= principal + (t / term ) x interest`

`Value of a single 0xabc loan token` `= (Value of all 0xabc loan tokens) / (supply of 0xabc loan tokens)`

`Value of a single 0xabc loan token` `= (principal + (t / term ) x interest)/(principal + interest)`

> **Example**:
>
> When a loan with *principal = 1,000,000 USDC, term = 30 days, APR = 12%* is approved, 1,009,863.013 (`= 1,000,000 + 1,000,000 x 12% x 30/365`) loan tokens are minted.
>
> The value of a single loan token at `n` days (where `n` is less than or equal to 30) is **1,009,863.013 USDC** `(=1,000,000 + (n/30) x 9,863.013)`).

SAFU (Secure Asset Fund for Users)

What is the SAFU?

The SAFU is an overhaul of how TrueFi handles borrower defaults. The SAFU smart contract is responsible for all bad debt accrued by the protocol. The SAFU has been initially funded by TrustToken and the funds will help cover defaults.

In case of a loan default, TrueFi lending pools will transfer all bad debt assets to the SAFU in exchange for the full expected value of those assets. Then, the SAFU will slash staked TRU tokens, up to 10% of the defaulted amount. If the value of these slashed tokens is not enough to cover the default, the SAFU will use its funds to help repay the affected lending pool for lost funds.

SAFU default handling

In the event of a default, the following occurs:

1. Up to 10% of TRU is slashed from the staking pool and transferred to the SAFU to cover the defaulted amount, equal to the principal amount plus the full amount of expected interest (“Defaulted Amount”)
2. All the defaulted LoanTokens will be transferred from the lending pool to the SAFU
3. If the current SAFU funds are insufficient to cover the defaulted loan; the SAFU can sell TRU for the respective borrowed asset at its manager’s discretion
4. If the value of the SAFU funds can not satisfy the defaulted loan:
    1. The difference between the defaulted loan and the SAFU is calculated (“Uncovered Amount”).
    2. The SAFU will issue ERC-20 tokens representing a claim for the Uncovered Amount (“Deficiency Claim”).
    3. Then, the affected lending pool will receive a Deficiency Claim for the Uncovered Amount, assuming its successful recovery.
    4. The affected lending pool will have a first-priority claim on the funds recouped through arbitration for the Deficiency Claim amount.
5. If a debt is repaid:
    1. The recouped funds will be used to purchase the asset that the Loan Token was originally denominated in, which will be transferred to the LoanToken contract.
    2. The SAFU will burn the Loan Tokens for the underlying value of those tokens (“Recovered Amount”)
    3. The SAFU is going to repurchase the issued Deficiency Claim tokens from the lending pool up to the Recovered Amount.
    4. If there is a remainder of the recovered funds after repurchasing the lending pool’s Deficiency Claim, the SAFU keeps those funds.
6. If any portion of the original loan amount is not repaid after the completion of the legal recovery process; the lending pool’s remaining Deficiency Claim tokens are going to be burned thus reducing the LP token price.

Smart Contract Architecture

The SAFU replaces what was called “Liquidator” in previous TrueFi versions. Therefore, it will have permission to slash TRU from the staked TRU pool.

The funds in the SAFU will be managed by an approved address, automating as much capital management as possible through DeFi. In the initial version, the funds' management will be somewhat centralized to maximize the capital efficiency when making exchanges between tokens. For example, the price impact of exchanging TRU on decentralized exchanges is much higher than the impact of OTC or centralized exchange opportunities.

Nevertheless, following the ethos of progressive decentralization, future unlocks will include updates to the SAFU which will further decentralize the management of the SAFU funds.

Risk Mitigation

Has TrueFi been audited?

Yes, please see TrueFi's technical audits here.

How does TrueFi mitigate risk?

TrueFi takes multiple measures to help protect lenders:

- Staked TRU provides default protection for lenders and governs the loan approval process
- Borrowers on TrueFi follow a thorough Know Your Business (“KYB”) workflow and credit review which incorporates both on-chain and off-chain data, such as company background, repayment history, operating & trading history, assets under management, and credit metrics.
- TrueFi handles bad debt via a Secure Asset Fund for Users (“SAFU”) smart contract.
- TrueFi lenders can also purchase smart contract cover through Nexus Mutual to hedge risks when lending on TrueFi. Coverage is paid out at the discretion of mutual members but has covered technical exploits in the past.

This is not investment advice. Please Do Your Own Research.

Developer docs

[Legacy] TrueFi DAO pools contracts

Below is a brief guide to TrueFi lending pool contracts.

For detailed questions, please reach out to the Discord #dev channel.

Lending pool addresses

Contract Name	Address
tfUSDC	0xA991356d261fbaF194463aF6DF8f0464F8f1c742
tfUSDT	0x6002b1dcB26E7B1AA797A17551C6F487923299d7
tfTUSD	0x97cE06c3e3D027715b2d6C22e67D5096000072E5
tfBUSD	0x1Edd60D149D48FA7d91703bf4890F97220C09437

TrueFi DAO Pool contracts

- **TrueFiPool2**: lenders provide and withdraw liquidity to the pool, and can get lending pool details, such as pool value, pool token price, etc. from this contract
- **TrueLender2**: implements the lending strategy for the TrueFi pool, i.e. how loans are approved and funded
- **LoanFactory2**: deploys LoanTokens, which represent details of each loan on-chain
- **TrueRatingAgencyV2**: loan applications are rated and approved by TRU stakers
- **TrueMultiFarm**: lenders can stake lending pool tokens to earn TRU rewards (read more here)
- **SAFU**: handles bad debt in TrueFi lending pools (read more here)

TrueFiPool2 contract

The `TrueFiPool2` contract is used by TrueFi DAO lending pools – tfUSDC / tfUSDT / tfBUSD / tfTUSD. This contract enables accounts to pool tokens with the goal of earning yields on underlying tokens.

Contract Name	Address
TrueFiPool2	0x8d35372ea3e85c49a6Of0a72ede8ff8629da3999a

Lending & Withdrawing methods

Method	Notes
`join(uint256 amount)`	Lends a certain amount of underlying tokens to the portfolio, and mints and transfers corresponding amount of ERC-20 lending pool ("LP") tokens to the lender.
`liquidExit(uint256 amount)`	Withdraws liquidity from the lending pool. Lender transfers pool tokens to pool and receives underlying token, but with a small penalty for liquidity as calculated by `liquidExitPenalty()` described below.

Determining pool values

Calculating LP token price

To calculate the price of the lending pool token ("LP token"), take `poolValue()` divided by `totalSupply()`.

Calculating pool APY

To calculate yield for a pool, one must calculate the weighted average of rates across each active loan in the pool and idle funds held in the pool:

```
pool_apy = SUM(loan_1_amount * loan_1_apy + ... + loan_n_amount*loan_n_value) / poolValue()
```

For an example of how this can be done via SQL, see this Dune query.

Method	Notes
`poolValue()`	Returns pool value in underlying token. This is the virtual price of the entire pool, including values for loan tokens and liquid underlying tokens held by the pool. *Note: this assumes defaulted loans are worth their full value.*
`liquidValue()`	Returns virtual value of liquid assets in the pool.
`totalSupply()`	Returns total number of pool LP tokens.
`liquidRatio()`	Ratio of liquid assets in the pool to the pool value, in basis points. For example, 4683 => 46.83%.
`utilization()`	Returns utilization in basis points. For example, 5316 => 53.16%. Calculated as `1 - liquidRatio()`.
`liquidExitPenalty(uint256 amount)`	Calculates lender will pay to withdraw liquid funds from the pool. This returns a proportion to be applied if `liquidExit()` is performed with the given amount of LP tokens. For example, when a pool is at ~75% utilization a small withdrawal would return `liquidExitPenalty` = ~0.9980, meaning that the lender would pay an exit penalty of ~0.20% (20bps) to withdraw from the pool. For more detail on liquid exit and how the penalty is calculated, see here.
`averageExitPenalty(uint256 from, uint256 to)`	Internal function for calculating exit penalty.

Default handling

Read more about default handling in 🎶 **SAFU default handling**

Method	Notes
`liquidate(ILoanToken2 loan)`	Calls SAFU function to liquidate bad debt. Liquidates a defaulted Loan, withdraws a portion of TRU from staking pool then tries to cover the loan with own funds, to compensate lending pool. Loan must be in `defaulted` state. If SAFU does not have enough funds, deficit is saved to be redeemed later.
`reclaimDeficit(ILoanToken2 loan, uint256 amount)`	After a defaulted loan's debt has been redeemed by the SAFU, the lending pool can reclaim call this function to redeem "deficiency claim" tokens for underlying tokens held in the SAFU.

TrueLender2 contract

Implements the lending strategy for the TrueFi pool, i.e. how loans are approved and funded.

Contract Name	Address
TrueLender2	0xa606dd423dF7dFb65Efe14ab66f5fDEBf62FF583

Method	Notes
`fund()`	When called, sends funds from the pool to the loan token contract. Pool receives and holds loan tokens. Must be called by the loan borrower.
`loanIsCredible(uint256 yesVotes, uint256 noVotes)`	Checks whether loan receives sufficient votes via stkTRU rating to be funded. Checks whether loan receives minimum ratio of yes to no votes and hits threshold of minimum votes set by owner.
`reclaim(ILoanToken2 loanToken, bytes calldata data)`	Redeems LoanTokens held by the pool for underlying funds held in the loan token. Loan token must be settled before calling this function.
`reclaimDeficit(ILoanToken2 loan, uint256 amount)`	After a defaulted loan's debt has been redeemed by the SAFU, the lending pool can reclaim call this function to redeem "deficiency claim" tokens for underlying tokens held in the SAFU.

Loan Token contracts

Loan tokens are created by LoanFactory2.

Contract Name	Address
LoanFactory2	0x69d844f85928d0e78c530c06325A88e53d6685BC

Each LoanToken has the following parameters:

- `borrower()`: address of borrower
- `amount()`: principal amount of fixed term loan
- `term()`: loan term in seconds
- `apy()`: loan APR in basis points (i.e. 971 = 9.71%)
- `status()`: returns loan's status, which progresses through the following states:
  - `0` -> `Awaiting`: Waiting for funding to meet capital requirements
  - `1` -> `Funded`: Capital requirements met, borrower can withdraw
  - `2` -> `Withdrawn`: Borrower withdraws money, loan waiting to be repaid
  - `3` -> `Settled`: Loan has been paid back in full with interest
  - `4` -> `Defaulted`: Loan has not been paid back in full
  - `5` -> `Liquidated`: Loan has Defaulted and stakers have been Liquidated
- `canTransfer()`: returns LoanTokens are non-transferable except for whitelisted addresses
- `debt()`: returns `amount + interest`. Note that this does not take into account whether the loan has been repaid.
- `isRepaid()`: returns boolean value, indicating whether a loan has been repaid in full.
- `profit()`: returns difference between `debt()` and `amount()`, the anticipated interest to be paid on the loan

Method	Notes
`fund()`	Transfers tokens to loan token contract from pool. Can be called only by lender contract. Sets status to Funded, start time, lender.
`withdraw(address _beneficiary)`	Borrower calls this function to withdraw funds to address provided. This sets the status of the loan to `Withdrawn`.
`repayInFull(address _sender)`	Function for borrower to repay all of the remaining loan balance. Borrower should use this to ensure full repayment.
`_repay(address _sender, uint256 _amount)`	Internal function for loan repayment. If `_amount` is sufficient, then this also settles the loan.
`settle()`	Moves loan to status = `settled` if loan is fully repaid.
`reclaim()`	Function for borrower to reclaim any underlying currency stuck in the loan token contract. Only the borrower can call this function after the loan has been settled, defaulted, or liquidated.

Repaying loans to lending pools has two steps:

1. Borrower repays funds to loan token
2. `reclaim()` function is called on loan token, which burns the loan token and pays out fees to stkTRU.

TrueRatingAgencyV2

As described here, the `TrueRatingAgencyV2` contract determines whether a loan request should be funded by `TrueLender2`.

Contract Name	Address
TrueRatingAgencyV2	0x054613343405680759bE35438b2214Ja00261Fb6b

stkTRU holders can vote `yes()` or `no()` on each loan. The `TrueLender2` contract then checks to see if each loan satisfies threshold conditions in order to be funded.

stkTRU voters receive rewards in the form of TRU tokens for voting on loan requests. `claimable()` reward tokens are calculated as follows:

`claimable() = (# TRU voted by user / # total TRU votes) * (Total TRU Reward)`, where

- `Total TRU Reward = (Loan interest * TRU distribution factor * rewardMultiplier)`
  - where `Loan interest = (loan APR * term in days * principal) /365`. Loan APR, term, and principal can be obtained from the respective loan token contract
  - `rewardMultiplier` can be found from the TrueRatingAgencyV2 contract
  - `TRU distribution factor` is calculated as `remaining` divided by `amount` from the RatingAgencyV2Distributor contract

TrueMultiFarm

As described here, lenders can stake LP tokens into the TrueMultiFarm contract to get TRU rewards.

Contract Name	Address
TrueMultiFarm	0xec6c3FD795D6e6f202825Ddb56E01b3c128b0b10
LinearTrueDistributor	0xc7A8fc6De551beb069f76111C6A1Fc473F8E5b8f70

How to calculate lender emissions per day

- Total TRU emissions per day can be calculated using `totalAmount()` and `duration()` found on the distributor contract:
  - `TRU distribution per day = (totalAmount/10^8) / (duration/(24*3600))`
- To calculate farm rates for each token, use the formula below:
  - `TRU rewards per farm per day = TRU distribution per day * getShare(IERC20 token) / shares()` as described below

Method	Notes
`stake(IERC20 token, uint256 amount)`	Stake tokens to the farm. Upon staking, this will claim any claimable rewards.
`unstake(IERC20 token, uint256 amount)`	Remove staked tokens
`claim(IERC20[] calldata tokens)`	Claim TRU rewards
`exit(IERC20[] calldata tokens)`	Unstake amount and claim rewards
`claimable(IERC20 token, address account)`	Returns the claimable TRU reward for an account that is staking tokens.
`shares()`	Returns denominator for total farm rewards rate
`getShare(IERC20 token)`	Returns numerator for LP token's share of farm rewards

Managed Portfolio [legacy]

> ⓘ `ManagedPortfolio` is in the process of being sunset.
>
> New TrueFi Capital Markets portfolios deployed after June 2022 use contract `FlexiblePortfolio`

`ManagedPortfolio` represents a portfolio of BulletLoans. Lenders put funds into the portfolio, the manager uses funds to issue loans to borrowers, and borrowers repay principal and interest back into the portfolio. `ManagedPortfolio` issues ERC-20 Liquidity Provider tokens to lenders in proportion to the amount they lend. These tokens represent a lender's share of the funds in the pool, including the interest accumulated from loans repaid by borrowers.

contracts-ragnarok/ManagedPortfolio.sol at main · trusttoken/contracts-ragnarok
GitHub

All of the portfolio operations are up to the manager's discretion. The Portfolio Manager makes decisions about issuing new loans, marking them as defaulted, altering portfolio params and so on. Manager also specifies an `ILenderVerifier` contract that is responsible for handling the permissions to join the portfolio (portfolios might be permissionless, but typically are not and only offer entrance to a defined group of lenders).

Portfolio tokens represent lenders' share in the pooled funds. Lenders put funds into the portfolio if they trust the manager is going to abide by a reasonable policy. Funds from the portfolio are only available for withdrawal after the portfolio's close date.

Methods

Method	Notes
`deposit(uint256 depositAmount, bytes memory metadata)`	Lends a certain amount of underlying tokens to the portfolio. If the portfolio has lender restrictions enabled, this function requires metadata to validate the lender's address is allowed to lend.
`withdraw(uint256 sharesAmount, bytes memory)`	After the portfolio's close date, lender can withdraw funds, i.e. redeeming the portfolio token for underlying pool tokens.
`createBulletLoan( uint256 loanDuration, address borrower, uint256 principalAmount, uint256 repaymentAmount )`	Creates bullet loan token using BulletLoans contract.
`markLoanAsDefaulted(uint256 instrumentId)`	Only the portfolio's manager can mark a loan as defaulted.
`markLoanAsResolved(uint256 instrumentId)`	Only the portfolio's manager can mark a loan as resolved. Intended to be used for situations after partial repayment where a loan workout has been agreed to.
`setEndDate(uint256 newEndDate)`	Manager can change portfolio's close date, or `EndDate`. Note that the portfolio's close date can only be decreased. Close date can NOT be moved further into the future, nor can it be decreased to a date earlier than any active loan's maturity date.
`setLenderVerifier(ILenderVerifier _lenderVerifier)`	Manager can set the lender verifier contract to enforce lender restrictions. Manager can leverage three different types of restrictions: • *Whitelist:* contract that implements simple, universal whitelist. • *WhitelistLenderVerifier:* contract that implements a unique whitelist for each of the portfolios, which are using this verifier. Managers of particular portfolios have the authority to manage respective whitelists. • *SignatureOnlyLenderVerifier:* contract that requires the lender to provide a signature of a predefined message.
`setManagerFee(uint256 _managerFee)`	Manager can set the portfolio fee, or managerFee, in basis points. Portfolio fee is charged on deployed capital
`setMaxSize(uint256 _maxSize)`	Manager sets a cap, or `maxSize`, for a portfolio. Lenders can not put more funds into the portfolio if it would cause the total principal amount lent into the portfolio to exceed the `maxSize`.
`updateLoanParameters( uint256 instrumentId, uint256 newTotalDebt, uint256 newRepaymentDate )`	Manager can modify loan terms, changing maturity date or total debt to be repaid. In order to change the maturity date to an earlier date or increase the repayment value, the borrower must consent and provide a signature.

View Methods

Method	Notes
`getStatus()`	Returns status of the portfolio. If current date is past the portfolio's close date, will return 'Closed'. If portfolio holds defaulted loans, will return 'Frozen'.
`getAmountToMint(uint256 amount)`	Returns the amount of portfolio tokens that would be minted for a given amount of tokens to be lent (e.g. if 1000 USDC are lent, 995 tfExamplePortfolio tokens would be minted).
`getOpenLoanIds()`	Returns list of open loan tokens.
`illiquidValue()`	Returns estimated value of active, illiquid loans. Calculated on straight-line basis.
`liquidValue()`	Returns balance of idle underlying tokens held by portfolio.
`value()`	Returns total estimated value of portfolio (`liquidValue` + `illiquidValue`).

Typical Flows 🌊

Below are examples of typical flows in TrueFi Capital Markets pools:

- `Manager` creates `ManagedPortfolio` using `ManagedPortfolioFactory`.
- `Lender` lends funds into the `ManagedPortfolio`.
- `Manager` issues a `BulletLoans` token to the `ManagedPortfolio` (portfolio sends funds to the `Borrower`).
- `Borrower` repays `BulletLoans` an owed amount (`BulletLoans` will send funds to the debt owner - in this case the `ManagedPortfolio`).
- `Lender` withdraws funds from the `ManagedPortfolio`.

Additional actions that might happen:

- `Manager` can change loan parameters (with a `Borrower`'s approval if necessary).
- `Manager` can set loan status to `Defaulted` if `Borrower` does not return funds on time (and eventually set loan status to `Resolved` once the debt is settled).
- `Manager` can change various `ManagedPortfolio` properties.

BulletLoans

BulletLoans is an ERC-721 contract. Each of the tokens represents a single loan. All the loan parameters can be read from LoanMetadata struct. BulletLoans contract enables loan creation, facilitates loan repayment and allows managing the loan's state and parameters.

ManagedPortfolio

ManagedPortfolio is an ERC-20 token facilitates funds management and allows loan issuance. Portfolio tokens represent lenders' share in the pooled funds. All of the portfolio operations are up to the managers discretion. Manager makes the decisions about issuing new loans, marking them as defaulted, altering portfolios params and so on. Lenders only lend funds into the portfolio if they trust the manager is going to abide by a reasonable policy. Manager also specifies an ILenderVerifier contract that is responsible for handling the permissions to join the portfolio (portfolios might be permissionless, but typically are not and only offer entrance to a defined group of lenders). Funds from the portfolio are only available for the withdrawal after the final closing date.

ManagedPortfolioFactory

Contract that allows easy portfolio configuration and creation. A particular instance of the factory can only be accessed by whitelisted addresses.

ProtocolConfig

Contract holding key system params.

BorrowerSignatureVerifier

A contract that verifies borrower's consent to change the loan parameters. Manager can freely change the loan parameters in borrower's favour (reduce owned amount, increase time), but needs an explicit, borrower's approval to do the opposite.

ILenderVerifier

Whitelist

A contract that implements simple, universal whitelist.

WhitelistLenderVerifier

A contract that implements a unique whitelist for each of the portfolios, which are using this verifier. Managers of particular portfolios have the authority to manage respective whitelists.

SignatureOnlyLenderVerifier

A contract that requires lender to provide a signature of a predefined message.

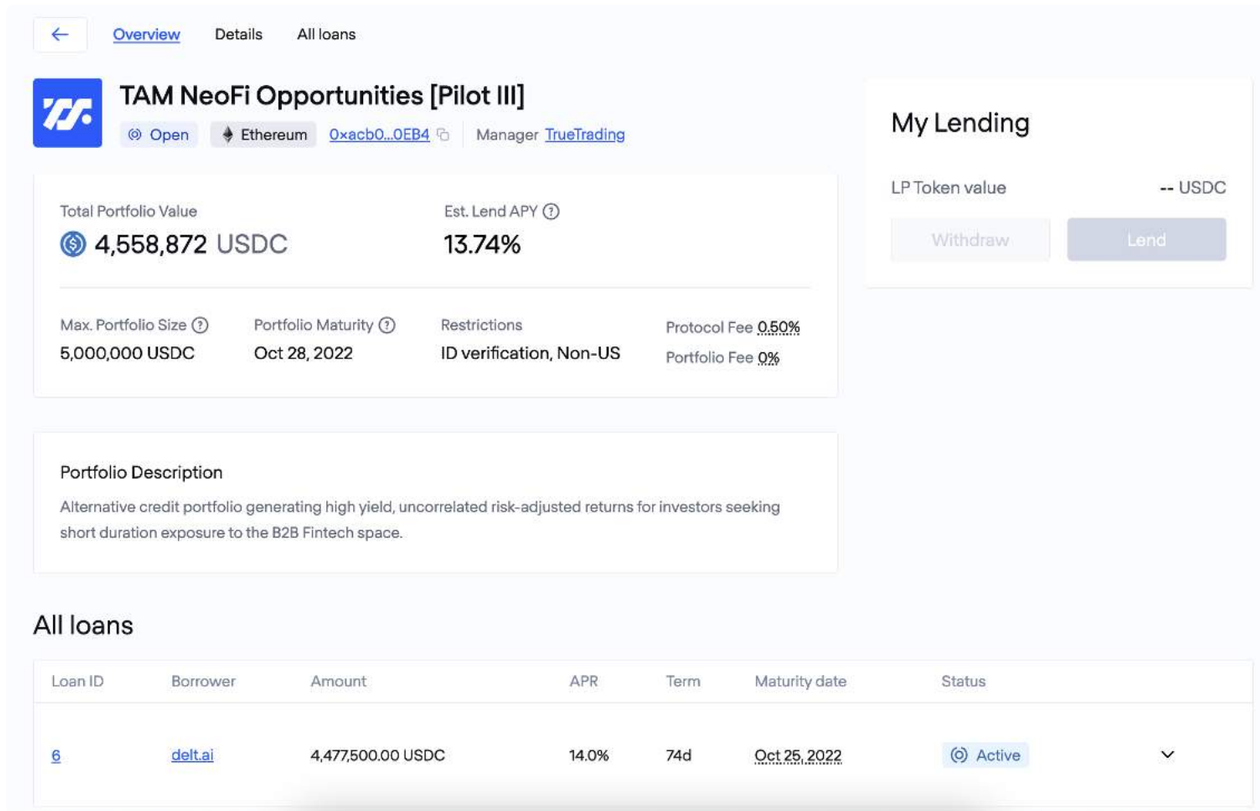
Flexible Portfolios [legacy]

Configurable lending pools governed by 3rd party PMs

> For technical docs, see **Flexible Portfolio contracts**

Flexible Portfolios are configurable lending pools run by independent managers on TrueFi infrastructure. Portfolio Managers ("PMs") have discretion over loan terms, as well as other items such as the pool's maximum size, maturity date, and lender access/restrictions.

Portfolios can serve real world financing borrowers and use cases, as covered by PYMNTS here, as well as crypto-focused borrowers (institutions, DAOs, etc., as covered by Bloomberg here).



Portfolio on TrueFi

Who can lend to portfolios on TrueFi?

Portfolio managers define who can lend into each portfolio (i.e. whether portfolios are permissioned or permissionless).

For permissioned portfolios, lenders can gain access to the portfolio by completing onboarding per the manager's instruction (ex. completing KYC process directed by the portfolio manager).

How are portfolios managed?

Portfolio Managers (PMs) make decisions on underwriting loans, managing relationships with borrowers, and configuring portfolios. Lenders are responsible for conducting diligence on PMs and portfolios before lending.

When can lenders withdraw?

Lenders can withdraw funds only after the portfolio's maturity date. Funds are locked up in the portfolio until the portfolio's maturity is reached.

Can lenders transfer portfolio tokens?

No, portfolio tokens are non-transferrable by default.

Managers can enable transfers if desired.

What are the fees on portfolios?

Portfolios pay a protocol fee per annum to the TrueFi DAO treasury. Fees accrue block-by-block and are paid upon each smart contract interaction (lend/withdraw/disburse loan/repay loan).

The example below illustrates how the protocol fee works:

> **Protocol Fee example**
>
> Take an example portfolio *Verum Fund,* which holds 1,000,000 USDC worth of loans and assume protocol fee = 50 bps per annum (0.50%).
>
> Assuming the value of Verum Fund grows linearly from 1,000,000 USDC to 1,100,000 USDC over the course of 30 days (avg. value of 1,050,000 USDC), the portfolio would pay a protocol fee of 431.51 USDC for this time period:
>
> ```
> Protocol fee = 1,050,000 USDC * 0.50% * (30/365) = 431.51 USDC
> ```

Additionally, PMs can set an optional Portfolio Fee. Portfolio Fees are paid to the PM, and can be configured such that they are accrued linearly over time, or paid as a flat fee at time of deposit and/or withdrawal.

Flexible Portfolio contracts

`FlexiblePortfolio` is a contract that serves as a portfolio of funds managed by the portfolio manager.

`FlexiblePortfolio` allows the portfolio manager to define allowed debt instruments that the portfolio will handle. This is the latest iteration of the previous portfolio contract version (called `ManagedPortfolio`).

Contract Name	Address
FlexiblePortfolio	0x5c6b0d8070ddc0a6a85c460819c3a91c628070ec
FlexiblePortfolioFactory	0x122d3ba54975bdeb7579938fb0ffc54f8baefd2a

GitHub - trusttoken/contracts-helium: TrueFi smart contracts second release 2022.
GitHub

`FlexiblePortfolio` satisfies ERC-4626 requirements, meaning it meets all features described here.

`FlexiblePortfolio` supports loans that yield both principal and interest amounts on repayments. That is, the principal amount goes straight to portfolio liquid funds and can be withdrawn/lent at any moment. Interest is split between all lenders proportionally to the amount of portfolio tokens they hold. This implies that if the tokens are held in a contract (e.g. Uniswap pool), the interest will be frozen in the portfolio.

Valuation

`FlexiblePortfolio` valuation is handled by the `ValuationStrategy`, which is a contract that holds information on all the loans the portfolio holds and can evaluate them properly. This requires the portfolio to call `onInstrumentFunded` when a loan is funded and `onInstrumentUpdated` when a loan might change its status (e.g. is repaid or gets defaulted). Valuation strategies cannot be switched as they hold the state of the portfolio loans.

Fees

Whenever an action that changes `FlexiblePortfolio` value is performed (`fundInstrument` / `repay` / `deposit` / `mint` / `withdraw` / `redeem` / `updateAndPayFee`), a fee for the TrueFi DAO is calculated and immediately transferred to the TrueFi DAO Treasury address. The fee is deducted from the `FlexiblePortfolio` value, so actions like borrow/withdraw/redeem cannot move the funds that are designated as fees. If the accrued fee cannot be repaid at the moment because of the lack of liquidity, additional information about the fee amount is stored in the contract and will be used to make the overdue payment the moment it will be possible. The same behavior is followed by the fee of the manager.

The `protocolAccruedFee` value is equal to:

`(current_timestamp - last_update_timestamp) / YEAR * protocol_fee_rate * portfolio_value`

The `managerAccruedFee` value is equal to:

`(current_timestamp - last_update_timestamp) / YEAR * manager_fee_rate * portfolio_value`

** `protocol_fee_rate` is the rate taken from the Protocol Config contract the last time an update was made. The same goes for `manager_fee_rate`, which is taken from Fee Strategy

Deposit / Withdrawal Strategies

Functions enabling Lenders to deposit/withdraw funds to/from the contract can additionally be limited by Deposit/Withdraw Strategy. These strategies (if set), are called with a hook every time a specific action is performed on the contract. The same calldata as for the initial call is passed to them, and then the strategy independently decides if this action can or cannot be performed. The strategies might also write some state to themselves on such hooks, but this is not a mandatory behavior. Additionally, these hooks can return a fee that would be applied for the action that is being performed. The fees are calculated independently by the strategy and have to be returned as absolute values and always in assets(not shares).

Here we present a few examples to better understand this process:

- `deposit(assets: 100)` is being called, the strategy `onDeposit()` returns `(true, fee: 10).` This means that 10 would be transferred to the manager, and the shares would be minted to the Lender based on the remaining 90.
- `mint(shares: 100)` is being called, the strategy `onMint()` returns `(true, fee: 10)`

Let's assume shares:assets are 1:1. 100 has to be transferred to the portfolio and 10 to the manager to satisfy the fee and the Lender's desire to have 100 shares minted. We calculate the fees for withdraw/redeem the same way.

FlexiblePortfolioFactory

This contract serves for deploying a new Flexible Portfolio. All of the `FlexiblePortfolio` parameters are chosen by the portfolio manager, except for `ProtocolConfig`.

TRU token

The native token of the TrueFi protocol is TRU.

Staked (stkTRU) is used in on-chain voting and governs decisions such as onboarding new portfolio managers to the protocol and managing key protocol parameters.

Token	Contract
TRU	0x4c19596f5aaff459fa38b0f7ed92f11ae6543784
stkTRU	0x23696914ca9737466d8553a2d619948f548ee424

Staked TRU (stkTRU)

For more information, please see **Staked TRU**.

TRU Token Supply & Distribution

For more information about TRU supply and distribution, please see the following link.

Staked TRU

What is stkTRU?

TRU holders can stake TRU to participate in the **Loan approval process** for TrueFi DAO pools. In return the user receives stkTRU tokens.

Stakers vote to approve/reject loan applications, and in return can potentially earn additional TRU and protocol fees. Stakers risk losing a portion of their TRU staked in the event of default on loans in TrueFi DAO pools.

Contract	Address
stkTRU	0x23696914Ca9737466D8553a2d619948f548Ee424

How do I stake?

On the Stake page, click on "Stake TRU", input the amount of TRU you wish to stake and complete the transaction on the connected wallet. Users staking for the first time are required to approve a token allowance for the stkTRU contract first.

What is the incentive for staking?

Staking TRU on the protocol enables stakers to earn a portion of the protocol fees generated by the TrueFi protocol. The proportion of fees paid to stakers can be changed through protocol governance.

Protocol fees have historically consisted of 10% of all interest from successfully repaid loans made by TrueFi DAO-managed pools, but were set to 0% as of May 2022 (read more here).

Stakers may receive TRU rewards as incentive for staking, but as of May 2023 stkTRU rewards have also been turned off. For the latest TRU distribution per day, one can check the parameters `totalAmount` and `duration` in the stkTRU distributor smart contract.

Visit the etherscan link below, click on Contract, then click on Read as Proxy. You will find these two parameters among the list of other parameters. The duration is in seconds.

```
TRU distribution per day = (totalAmount/10^8) / (duration/(24*3600))
```

Contract	Address
stkTRU distributor	0xecfD4F2C07EABdb7b592308732B59713728A957F

What is the risk of staking?

TRU stakers are acting as a risk management system for the TrueFi Lending Pools. In the event of a default in a TrueFi Lending Pool, a certain percentage of the staked TRU will be liquidated and transferred to the TrueFi Lending Pool. The maximum percentage of staked TRU that can be liquidated and transferred to the affected TrueFi Lending Pool per default is the *Maximum Liquidation Rate*. Therefore, in the event a loan provided by a TrueFi lending pool defaults, Stakers can lose up to the Maximum Liquidation Rate of their TRU staked.

For example, let's consider a scenario when a loan of principal amount 1,000,000 TUSD defaults where the stkTRU contract has 500,000 TRU staked and a *Maximum Liquidation Rate* of 10%. The maximum amount of TRU that would be liquidated is (500,000 x 10%) 50,000 TRU.

The value of the loan defaulting may be less than the value of the maximum TRU that can be liquidated. In such cases the market value of TRU that would be liquidated will be equal to the loan amount and the rate of TRU liquidation would be less than the maximum liquidation rate.

For the latest *Maximum Liquidation Rate*, you can check the parameter `fetchMaxShare` in the Liquidator smart contract below:

Contract	Address
Liquidator (proxy)	0x7aC899754Dd042024bb168fd5c9a07420F444Bdf

Visit the etherscan link to the smart contract, click on Contract, then click on Read as Proxy. You will find the parameter `fetchMaxShare` among the list of other parameters.

```
Max liquidation rate = (fetchMaxShare / 10^2) %
```

How can I unstake or withdraw my stake?

Once TRU is staked on the protocol, it is locked into the protocol. You will need to initiate cooldown to unstake your staked TRU. During the cooldown period you cannot unstake or withdraw your staked tokens. Once the cooldown period is over you will have a window of time, called the unstakePeriodDuration in the stkTRU smart contract to unstake or withdraw the staked tokens. If you do not unstake during this window your staked tokens will be staked back into the protocol and you will be required to re-initiate a new cooldown period. If you have recently acquired stkTRU and you wish to withdraw your stake, you will need to initiate cooldown.

During the unstaking window you can visit the Stake page and click on the "Unstake" button present over your Staked TRU amount.

What is the cooldown period and window to unstake?

The cooldown period has been added to prevent any kind of gamification which might be possible by staking for a very short period of time. For example, without a cooldown period Stakers could stake on the protocol to earn the protocol fee and then immediately unstake. This might result in some Stakers earning a return which is disproportionate to the amount of risk they are helping mitigate. Staked TRU cannot be unstaked from the protocol during the cooldown period.

For the latest cooldown period and the unstaking window, you can check cooldownTime and `unstakePeriodDuration` in the stkTRU smart contract. Visit the etherscan link to the smart contract, click on Contract, then click on Read as Proxy. You will find both these parameters cooldownTime and `unstakePeriodDuration` mentioned in seconds. To change this to days you can use the following formula.

```
Cooldown time in days = cooldownTime / (24 x 60 x 60)
```

Can I stake more TRU during the cooldown period?

Yes, you can stake more TRU during the cooldown period. Staking more TRU, with the same wallet, during the cooldown period will increase the cooldown period. The cooldown period timer will be reset to the maximum cooldown period if you stake any additional TRU.

How do I claim my staking rewards?

On the Stake page, you can regularly check your claimable rewards. Click on the 'Claim' button and complete the transaction in your wallet to claim them.

How are loan requests approved?

Information on approving loans can be found in the Loan approval process section of the FAQs.

Can I transfer stkTRU?

There are no restrictions on when stkTRU holders can transfer the token to other wallet addresses. Users can transfer stkTRU from their wallet when they are in a cooldown period and even if there are unclaimed rewards associated with their wallet.

Transferring stkTRU does not change the cooldown status of the associated wallet. The cooldown process is independent of the token and it only depends on the wallet address.

If a user is currently in the unstake window after initiating cooldown, they can only unstake their original cooldown amount and not any additional amount of stkTRU they received after initiating the cooldown. The user would have to initiate a new cooldown on the additional stkTRU amount.

Accruing stkTRU Rewards

Rewards for staking are linked to the wallet address. If a user has claimable rewards and decides to transfer their stkTRU without claiming the rewards, they can always claim the rewards later. Rewards (TRU or TrueFi LP tokens) associated with staking are not transferred when stkTRU is transferred and does not depend on whether the staker is in a cooldown period or unstake window. The receiver of stkTRU starts accruing rewards based on when they received the tokens.

stkTRU Delegation and Governance

Transferring stkTRU does change the voting power for snapshot voting and, in the future, on-chain governance by the exact amount of stkTRU which has been transferred. Receivers of stkTRU will be required to delegate the balance to themselves upon receiving the tokens.

Loan approval process

Voting on loan applications

To vote on loan applications, you need stkTRU. To learn more about how to acquire stkTRU please view the section on Staking. Once you have stkTRU you can visit the Stake page and vote on the loan applications listed on the page.

The stkTRU balance with which you can vote on loan applications is equal to the stkTRU balance held by your wallet when the loan application was created. You will not be able to vote on loan applications with any stkTRU balance acquired after a loan application was created.

Stakers can either vote YES or NO with stkTRU on a loan application. Voting YES means you are predicting that the loan is not likely to default, and voting NO means you are predicting that the loan is likely to default. Stakers can vote with the entire stkTRU balance of their wallet, including the stkTRU balance delegated to their wallet address.

Voting on loan applications does not lock stkTRU, stakers can use their stkTRU balance to vote on multiple loan applications.

What's the voting period for voting on loan applications?

There is no specific time period for stkTRU holders to vote on loan applications. After a loan application is live on the TrueFi platform, stkTRU holders can start voting on the loan until the loan application is funded by the lending pool or cancelled by the borrower.

However, there is a minimum time period that must pass before a loan can be funded by the lending pool. This time period is called the minimum voting period which corresponds to the votingPeriod parameter set in the TrueLender smart contract.

Visit the etherscan link to the TrueLender (proxy) smart contract, click on Contract, then click on Read as Proxy. You will find the parameter votingPeriod in seconds.

stkTRU holders can modify or cancel their votes any number of times before the loan has been funded by the lending pool or cancelled by the borrower.

How are loan applications approved?

stkTRU holders vote YES or NO on whether to approve a loan request. A loan is approved if it satisfies two conditions.

1. Minimum # of votes (15 million as of Sept 2022)
2. Minimum ratio of YES-to-NO votes (at least 80% of votes YES, as of Sept 2022)

Only whitelisted borrowers can submit their loan applications to the `TrueRatingAgencyV2` contract. Whitelisted addresses will return true when queried against `allowedSubmitters` in the `TrueRatingAgencyV2` contract.

Loans are approved or rejected based on conditions set in three smart contracts: TrueFi lending pool contract (`tfUSDC` et al), `TrueRatingAgencyV2` , and `TrueLender2` .

Contract	Address
TrueLender2	0xa606dd423dF7dFb65Efe14ab66f5fDEBf62FF583
TrueRatingAgencyV2	0x05461334340568075bE35438b221A3a0D261Fb6b

Incentive for voting on loan applications

stkTRU holders receive rewards in the form of TRU tokens for voting on loan requests. The rewards become claimable as soon as the loan is withdrawn by the borrower and the status of the loan becomes active. Rewards are distributed to voters based on their share of the total votes received:

`TRU Reward for voting = (# TRU voted by user / # total TRU votes) * (Total TRU Reward)` , where

- `Total TRU Reward = (Loan interest * TRU distribution factor * rewardMultiplier)`
  - where `Loan interest = (loan APR * term in days * principal) /365` . Loan APR, term, and principal can be obtained from the respective loan token contract
  - `rewardMultiplier` can be found from the TrueRatingAgencyV2 contract
  - `TRU distribution factor` is calculated as `remaining` divided by `amount` from the RatingAgencyV2Distributor contract

How to Get TRU

> *The information below is for informational purposes only, it is not investment advice. Please do your own research (DYOR) before using any of the third-party services discussed or linked to below. For more information regarding these third-party services, please review the disclaimer below.*

TRU is available for trading on both centralized exchanges (CEXs) and decentralized exchanges (DEXs).

Centralized exchanges (CEXs)

TrueFi is listed on centralized exchanges, including Coinbase, Binance, Gemini, Crypto.com, MEXC, Kucoin, Bitrue, and Poloniex.

For a full up-to-date list of TRU listings, see CoinGecko.

TrueFi Price: TRU Live Price Chart & News | CoinGecko
CoinGecko

Decentralized exchanges (DEXs)

The token contract address of TRU is: `0x4C19596f5aAfF459fA38B0f7eD92F11AE6543784`

> Please make sure to double check the token contract address of TRU before making any transactions. Be aware of price slippage and transaction costs when trading on DEXs.

TRU trading pairs are available on Uniswap and Sushiswap, as well as other DEXs.

Aggregators such as 1inch, Matcha, DefiLlama, and Paraswap can help users find the best price for TRU swaps.

Disclaimer

> *The links provided within the “Centralized exchanges (CEXs)” and “Decentralized Exchanges (DEXs)” sections will bring you to third-party websites, owned and operated by independent parties over which we have no control (a "Third-Party Website"). Any link you make to or from a Third-Party Website will be at your own risk.*
>
> *Any use of a Third-Party Website will be subject to and any information you provide will be governed by the terms of the Third-Party Website, including those relating to confidentiality, data privacy and security.*
>
> *Unless otherwise expressly agreed to in writing, TrustToken, Inc. and all of its affiliates (“we”) are not in any way associated with the owner or operator of a Third-Party Website or responsible or liable for the goods and services offered by them or for anything in connection with such Third-Party Website. We do not endorse or approve and make no warranties, representations or undertakings relating to the content of a Third-Party Website.*
>
> *We specifically disclaim liability for any loss, damage and any other consequence resulting directly or indirectly from or relating to your access to a Third-Party Website or any information that you may provide or any transaction conducted on or via a Third-Party Web site or the failure of any information, goods or services posted or offered at a Third-Party Website or any error, omission or misrepresentation on a Third-Party Website or any computer virus arising from or system failure associated with a Third-Party Website.*