

Euler

Welcome

Welcome to the Euler documentation pages!

Here you will find the white paper, guides on how to use the app, and all the developer materials needed to start building on top of Euler.

Don't be shy, if there is anything missing or unclear, please drop in to the community [Discord](#) server where you can have all of your questions answered.

Getting Started

Introduction

A brief introduction to Euler

Euler is a non-custodial permissionless lending protocol on Ethereum that helps users to earn interest on their crypto assets or hedge against volatile markets without the need for a trusted third-party.

Euler protocol features a number of innovations not seen before in DeFi, including permissionless lending markets, reactive interest rates, protected collateral, MEV-resistant liquidations, multi-collateral stability pools, and much more. For more information, read the [White Paper](#).

White Paper

Find out how Euler works and how it differs from other popular lending protocols

Abstract

Here, we present Euler: a permissionless lending protocol custom-built to help users lend and borrow digital assets. The purpose of this white paper is to describe how Euler works at a high level and highlight new features and innovations that help to set it apart from other popular lending protocols, like Compound and Aave.

Introduction

Euler comprises a set of smart contracts deployed on the Ethereum blockchain that can be openly accessed by anyone with an internet connection. Euler is managed by holders of a protocol native governance token called Euler Governance Token (EUL). Euler is entirely non-custodial; users are responsible for managing their own funds.

A convenient and user-friendly front-end to for the Euler smart contracts is hosted at <https://app.euler.finance>. However, users are free to access the protocol in whatever format they wish; a popular alternative can be found at <https://instadapp.io/>.

Permissionless Listing

Euler lets its users determine which assets are listed. To enable this functionality, Euler uses Uniswap v3 as a core dependency (4). Any asset that has a WETH pair on Uniswap v3 can be added as a lending market on Euler (5).

Asset Tiers

Permissionless listing is much riskier on decentralised lending protocols than on other DeFi protocols, like decentralised exchanges, because of the potential for risk to spill over from one pool to another in quick succession. For example, if a collateral asset suddenly decreases in price, and subsequent liquidations fail to repay borrowers' debts sufficiently, then the pools of multiple different types of assets can be left with bad debts.

To counter these challenges, Euler uses risk-based asset tiers to protect the protocol and its users:

Isolation-tier assets are available for ordinary lending and borrowing, but they cannot be used as collateral to borrow other assets, and they can only be borrowed in isolation. What this means is that they cannot be borrowed alongside other assets using the same pool of collateral. For example, if a user has USDC and DAI as collateral, and they want to borrow isolation-tier asset ABC, then they can *only* borrow ABC. If they later want to borrow another token, XYZ, then they can only do so using a separate account on Euler.

Cross-tier assets are available for ordinary lending and borrowing, and cannot be used as collateral to

borrow other assets, but they can be borrowed alongside other assets. For example, if a user has USDC and DAI as collateral, and they want to borrow cross-tier assets ABC and XYZ, then they can do so from a single account on Euler.

Collateral-tier assets are available for ordinary lending and borrowing, cross-borrowing, and they can be used as collateral. For example, a user can deposit collateral assets DAI and USDC, and use them to borrow collateral assets UNI and LINK, all from a single account.

EUL holders can vote to liberate assets from the isolation-tier and promote them to the cross-tier or collateral-tier through governance mechanisms. Promoting assets up the tiers increases capital efficiency on Euler because it allows lenders and borrowers to use capital more freely, but it may also expose protocol users to increased risk. It is therefore in EUL holders' interests to balance these concerns.

Lending and Borrowing

When lenders deposit into a liquidity pool on Euler, they receive interest-bearing ERC20 eTokens in return, which can be redeemed for their share of the underlying assets in the pool at any time, as long as there are unborrowed tokens in the pool (similar to Compound's [cTokens](#)). Borrowers take liquidity out of a pool and return it with interest. Thus, the total assets in the pool grows through time. In this way, lenders earn interest on the assets they supply, because their eTokens can be redeemed for an increasing amount of the underlying asset over time.

Tokenised Debts

Similarly to Aave's [debt tokens](#), Euler also tokenises debts on the protocol with ERC20-compliant interfaces known as dTokens. The dToken interface allows the construction of positions without needing to interact with underlying assets, and can be used to create derivative products that include debt obligations.

Rather than providing non-standard methods to transfer debts, Euler uses the regular transfer/approve ERC20 methods. However, the permissioning logic is reversed: rather than being able to send tokens to anyone, but requiring approval to take them, dTokens can be taken by anyone, but require approval to accept them. This also prevents users from "burning" their dTokens. For example, the zero address has no way of approving an in-bound transfer of dTokens.

Borrowers pay interest on their loans in terms of the underlying asset. The interest accrued depends on an algorithmically determined interest rate for each asset. A portion of the interest accrued is held in reserves to cover the accumulation of bad debts on the protocol.

Protected Collateral

On Compound and Aave, collateral deposited to the protocol is always made available for lending. Optionally, Euler allows collateral to be deposited, but not made available for lending. Such collateral is 'protected'. It earns a user no interest, but is free from the risks of borrowers defaulting, can always be withdrawn instantly, and helps protect against borrowers using tokens to influence governance decisions (see Maker governance issue [\(6\)](#)) or take short positions.

Defer Liquidity

Normally, an account's liquidity is checked immediately after performing an operation that could fail due to

insufficient collateral. For example, taking out a borrow, withdrawing collateral, or exiting a market could cause a transaction to be reverted due to a collateral violation.

However, Euler has a feature that allows users to defer their liquidity checks. Many operations can be performed, and the liquidity is checked only once at the very end. For example, without deferring liquidity checks, a user must first deposit collateral before issuing a borrow. However, if done in the same transaction, deferring the liquidity check will allow the user to do this in any order.

Feeless Flash Loans

Unlike Aave, Euler doesn't have a native concept of flash loans. Instead, users can defer their liquidity check, make an uncollateralised borrow, perform whatever operation they like, and then repay the borrow. This can be used to rebalance positions, build-up complex positions, take advantage of external arbitrage opportunities, and more.

Because Euler only charges fees according to the time value of money, and from the blockchain's perspective flash loans are held for a duration of 0 seconds, they are entirely free on Euler (ignoring gas costs). We believe that flash loan fees are ultimately in a race to the bottom that will be accelerated by advances like flash minting. The ecosystem benefits gained from simple and free flash loans outweigh the relatively modest benefit from flash loan fees.

Risk-adjusted Borrowing Capacity

Like other lending protocols, Euler requires users to ensure that the value of their collateral remains higher than the value of their liabilities (except during the intermediate period when liquidity checks have been deferred). Over-collateralisation is encouraged by limiting how much borrowers can take out as a loan in the first place.

Compound achieves this in a one-sided way by using collateral factors to adjust down the value of a borrower's collateral assets when deciding how much they can borrow. This gives rise to a 'risk-adjusted collateral value' that helps to create a buffer that can be drawn upon by liquidators in the event that the value of a borrower's assets and liabilities changes over time. One of the problems with this approach is that it only adjusts for the risks associated with a borrower's collateral assets decreasing in value. There may be an asymmetric risk, however, of the borrower's liabilities increasing in value. This risk is not factored into the collateral factors.

On Euler, we therefore use a two-sided approach where we also adjust up the market value of a borrower's liabilities to arrive at a 'risk-adjusted liability value'. This approach improves capital efficiency on the protocol because it allows Euler to factor in the asset-specific risks of both downside and upside price movements. These risks are encapsulated in asset-specific collateral factors (as on Compound) and borrow factors (new to Euler). Ultimately, this approach means that the liquidation threshold of every borrower is tailored to the specific risk profiles associated with the assets they are borrowing and using as collateral.

To give an example, suppose a user has \$1,000 worth of USDC, and wants to borrow UNI. How much can

they borrow? If USDC has a collateral factor of 0.9, and UNI has a borrow factor of 0.7, then a user can borrow up to $\$1,000 * 0.9 * 0.7 = \630 worth of UNI. At this level of borrowing, the risk-adjusted value of their collateral is $\$1,000 * 0.9 = \900 , and the risk-adjusted value of their liabilities is $\$630 / 0.7 = \900 . If UNI increases in price, then the risk-adjusted value of their liabilities will also increase to $>\$900$, and then they will be eligible for liquidation. The buffer allowing for liquidation is $\$1,000 - \$630 = \$370$.

Decentralised Price Oracles

To be able to calculate whether a loan is over-collateralised or not, Euler needs to monitor the value of users' assets. On Compound, Maker, and Aave, various systems are used to get prices from off-chain sources and put them on-chain so that they can be accessed by the relevant smart contracts.

This approach is unsuitable for Euler's purposes because it requires centralised intervention whenever a new lending market needs to be created. Euler therefore relies on Uniswap v3's decentralised time-weighted average price (TWAP) oracles to assess the solvency of users (4). The reference asset used to normalise prices on Euler is Wrapped Ether (WETH), which is the most common base pair on Uniswap (5).

TWAP

Uniswap TWAP is calculated using the geometric mean price of an asset over some interval of time. TWAP in general is both a smoothed and lagging indicator of the trade price: a TWAP over a short interval is a less smooth function, but more up-to-date, whilst a TWAP over a long interval is a smoother function, but less up-to-date. TWAP is ideal for Euler's purposes for several reasons.

First, TWAP is resistant to price manipulation attacks. It cannot be manipulated within a transaction or block (for example, with flash loans or flash bots) because it is calculated using historic data. It is also expensive to manipulate using large market orders because the manipulated price must be maintained for some period of time relative to the TWAP time interval. During this time, other users can take advantage of the manipulated price with arbitrage, which will cause it to revert back to the broader market price. Arbitrage is especially practical on the blockchain because arbitrageurs have access to large amounts of capital (including from flash loans) and the atomic nature of transactions means that arbitrage transactions have a low execution risk. For these reasons, manipulating the price on a single decentralised exchange usually requires more widespread manipulation of all on-chain exchanges simultaneously, although even this can't prevent the (less practical but still possible) arbitrage between on and off-chain exchanges.

Second, the smooth nature of TWAP helps to remove the impact of price shocks on borrowers. In the event of a large trade, the current price on Uniswap can be moved significantly. Usually, arbitrage bots will quickly converge this to the broader market value, so the maximum deviation of the TWAP will only be a fraction of the temporary price movement. This prevents some unnecessary liquidations and loans that may quickly become undercollateralised.

Third, instead of instantly jumping between two price levels, TWAPs change continuously, second-by-second. This property is used by Euler's liquidation process to implement Dutch auctions that reduce the value captured by miners and front-running bots.

Time Interval

One of the challenges in using TWAP is determining the right interval over which it should be calculated for

a given asset. The trade-offs involved with shorter (longer) intervals may sometimes need to be taken into consideration and altered for specific assets. Euler therefore allows the default time interval to be updated by governance if EUL holders deem it necessary.

Liquidations

A borrower is considered to be in violation on Euler when the value of their risk-adjusted liabilities exceeds the value of their risk-adjusted collateral. A borrower that has just become in violation still has enough collateral to repay their loan, but is adjudged to be at risk of defaulting on their loan. Consequently, they may be liquidated in order to limit the potential for them to default.

MEV-resistance

On Compound and Aave, liquidations are incentivised by offering up a borrower's collateral to liquidators at a fixed percentage discount, which typically ranges between 5-10%. One of the issues with this strategy is that would-be liquidators often have no choice but to engage in priority gas auctions (PGA) for profitable liquidations, which exposes the liquidation bonus as so-called miner extractable value (MEV) (7). Another issue with this approach is that a fixed discount can be punitive for large liquidations, and therefore discourage large borrowers, whilst being insufficient to cover costs and incentivise smaller liquidations.

To remedy these issues, Euler uses a different approach. Rather than a fixed discount percentage, we allow the discount to rise as a function of how under-water a position is. This turns a one-shot opportunity, where liquidators have no option but to engage in a PGA, into a type of Dutch auction. As the discount slowly increases, each would-be liquidator must decide whether or not to bid for a liquidation at the current discount on offer. Liquidator A might be profitable at 4%, but liquidator B might run a more efficient operation and be able to jump in sooner at 3.5%. The Dutch auction is aided by the TWAP oracles used on Euler because a shock to the price does not bring with it a singular point at which every liquidator becomes profitable all at once. Instead, the price moves more smoothly over time leading to a continuum of opportunities to liquidate, which further helps to limit PGAs. Overall, this process should help to drive the discount price towards the marginal operating cost of liquidating a borrower.

However, by itself, this process does not prevent MEV because miners and front-runners can still steal a liquidator's transaction. To limit this form of MEV, we allow liquidity providers on Euler to make themselves eligible for a "discount booster", which allows them to become profitable in the Dutch auction before miners and front-runners (who do not have the booster).

Soft Liquidations

The fraction of a borrower's debt that can be paid off by liquidators in one go is referred to by Compound as the 'close factor.' On both Compound and Aave, the close factor is currently fixed at 0.5, meaning liquidators can pay off up to half a borrower's loan in one go, regardless of how underwater their position is. This approach has a couple of potential drawbacks.

First, allowing liquidators to liquidate half a loan could be considered excessive if a smaller liquidation would have been sufficient to bring the borrower back to health. Larger borrowers are likely to be put off by such a process. Second, a large fixed discount can sometimes drive a borrower closer to insolvency and disincentivise them from repaying their loans (see (8)).

On Euler, we therefore use a dynamic close factor to try to 'soft liquidate' borrowers. Specifically, we allow

liquidators to repay up to the amount needed to bring a violator back out of violation (plus an additional safety factor). This means that borrowers who are only slightly in violation will often have much less than half their debts repaid during a liquidation, whilst borrowers who are heavily in violation will often have much more than half their debts repaid during a liquidation (their whole position might be closed in some circumstances).

Reserves

In rare circumstances, the value of a borrower's collateral might become less than the value of their liabilities. In this situation, the borrower is said to be 'insolvent.' Insolvent borrowers will typically be liquidated repeatedly until they have little to no collateral left. Any leftover liabilities after liquidations have stopped can be considered 'bad debt' that we can assume will never be repaid. If bad debt accumulates on the protocol, it increases the chance that lenders might all rush at once to withdraw their funds (to avoid becoming the bearer of the bad debt). This phenomenon is known as 'run on the bank.'

To reduce this risk, Euler follows Compound by allowing a portion of the interest paid by borrowers in each market to accumulate into a reserve. The idea behind this is to allow the reserves to act as a lender of last resort in the event of a run on the bank. Providing that reserves accumulate at a faster pace than bad debt, lenders do not need to worry about being able to withdraw their funds. Euler reserves operate similar to those on Compound, except that Euler reserves are tracked in eToken units, rather than underlying units, which means that Euler reserves earn interest automatically whereas Compound reserves do not.

The proportion of interest paid into the reserves is called the 'reserve factor' and it is a parameter specific to each lending market. There are trade-offs to consider when setting the reserve factor. A reserve factor of zero would mean no reserves accrue, which could stifle lending because of the bad debt issue. Nevertheless, a high reserve factor would mean a large portion of interest is diverted away from lenders, which could also stifle lending as lenders seek a better rate elsewhere. Thus, EUL holders may wish to use governance to select a reserve factor that balances these trade-offs for each type of asset.

Liquidation Surcharge

During a liquidation, the liquidator is required to provide a slightly larger amount of the borrowed asset than is being repaid on behalf of the violator. This extra amount is contributed to the reserves for the borrowed asset as a fee. The base liquidation discount starts at the level of this fee, so it is ultimately paid by the violator.

As a result, more volatile assets, which generally trigger more liquidations, will tend to accrue reserves at a faster pace than less volatile assets, helping to protect lenders of those assets. Additionally, this fee ensures that 'self-liquidating' is always net-negative, which adds a profitability threshold that some undesirable manipulation strategies are unlikely to meet.

Interest Rates

Both Compound and Aave use static linear (or piecewise linear) interest rate models to guide the cost of borrowing on their protocols. Broadly speaking, as demand for borrowing from the pool increases or supply decreases, interest rates go up, and when supply increases or the demand for borrowing decreases, interest rates go down.

Static models work well if they are appropriately parameterised ahead of time, but can be problematic when

parametrised incorrectly. For example, if the slope of the static linear function is too shallow, it can lead to the cost of borrowing being underpriced, with lenders unable to withdraw their assets because a pool has become over-utilised. On the other hand, if the slope of the static linear function is too steep, it can lead to the cost of borrowing being too expensive, which can stifle borrowing and lead to low capital efficiency.

Reactive Interest Rates

To avoid the problem of having to choose the right parameters for every lending market, Euler uses control theory to help autonomously guide the cost of borrowing towards a level that maximises capital efficiency on the protocol. Specifically, we use a PID controller to amplify (dampen) the rate of change in interest rates when utilisation is above (below) a target level of utilisation. This gives rise to reactive interest rates that adapt to market conditions for the underlying asset in real-time without the need for ongoing governance intervention. A similar approach has also recently been described by the Delphi Labs team [\(9\)](#).

Compound Interest

Compound interest is accrued on Euler on a per-second basis. This differs from other lending protocols, where interest is typically accrued on a per-block basis. A per-second basis is generally expected to perform more predictably in the long-run, even if upgrades to Ethereum lead to changes in the average time between blocks.

Gas Optimisations

Euler's smart contracts minimise the amount of storage used, implement a module system to reduce the amount of cross-contract calls, and have had a number of other gas usage optimisations applied. This makes the protocol cheaper on most operations than other lending protocols.

Transaction Builder

The user interface includes a convenient tool to help users batch up multiple transactions and reduce their gas costs, which we call a transaction builder. Advanced users can use this feature in conjunction with a 'defer liquidity checks' option provided on the protocol to rebalance loans or perform flash loans.

Sub-accounts

Asset tiers help to isolate risks on Euler, but they open up a new user-experience problem. Specifically, it would quickly become cumbersome for borrowers to use Euler if they had to send collateral to a new Ethereum account for each new isolation-tier loan they wanted to take out.

Euler therefore enables every Ethereum account using the protocol to access up to 256 sub-accounts (including the primary account), which can be used to cost-effectively manage multiple positions at the same time. A user only needs to approve Euler's access to a token once, and can then deposit into any sub-account. Additionally, no approvals are required to transfer assets and liabilities between sub-accounts, which allows users to isolate and segregate their collateral and debts as desired.

Governance

Euler will broadly follow the governance model pioneered by Compound [\(10\)](#). The protocol will be managed

by holders of a protocol native governance token called Euler Governance Token (EUL). EUL tokens will represent voting powers of the protocol software. Holders with enough EUL tokens will be able to make a formal proposal for change on the protocol. Token holders will then be able to vote on the proposal themselves or delegate their vote shares to a third party. Examples of the kinds of decisions token holders might vote on include proposals to alter include:

- The tier of an asset
- Collateral and borrow factors
- Price oracle parameters
- Reactive interest rate model parameters
- Reserve factors
- Governance mechanisms themselves

Acknowledgements

With special thanks to [Shaishav Todi](#), [Luke Youngblood](#), [Charlie Noyes](#), [samczsun](#), [Hasu](#), [Dave White](#), [Rick Pardoe](#), [Ayana Aspembitova](#) and the [Delphi Labs](#) team, [Mariano Conti](#), [Lev Livnev](#), and [Chainguys](#).

References

1. <https://docs.ethhub.io/built-on-ethereum/open-finance/what-is-open-finance/>
2. <https://compound.finance/documents/Compound.Whitepaper.pdf>
3. https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf
4. <https://uniswap.org/whitepaper-v3.pdf>
5. <https://weth.io/>
6. <https://www.theblockcrypto.com/post/82721/makerdao-issues-warning-after-a-flash-loan-is-used-to-pass-a-governance-vote>
7. <https://research.paradigm.xyz/MEV>
8. <https://docsend.com/view/bwiczmy>
9. <https://members.delphidigital.io/reports/dynamic-interest-rate-model-based-on-control-theory>
10. <https://medium.com/compound-finance/compound-governance-5531f524cf68>

Quick Links

Quick access to everything you need to know about Euler

Websites

[Website](#)

[DApp](#)

Governance

[***Forum](#)

[Off-chain \(Snapshot\)](#)

[On-chain \(WithTally\)](#)

EUL

[Etherscan](#)

[CoinMarketCap](#)

[CoinGecko](#)

[Messari](#)

DEX

[Balancer \(EUL/WETH\)](#)

[Uniswap v2 \(EUL/WETH\)](#)

[Uniswap v3 \(EUL/WETH\)](#)

[Uniswap v3 \(EUL/USDC\)](#)

Social

[Twitter](#)

[Discord](#)

[Telegram](#)

[Telegram Announcements](#)

Security

[Immunefi Bug Bounty](#)

[Report A Bug](#)

Content

[Blog](#)

[Newsletter](#)

Videos

[YouTube](#)

Dashboards

[Tokenterminal](#)

[DefiLlama](#)

[Zerion](#)

[DeBank](#)

[Ape Board](#)

[DeFiYield](#)

Dune Analytics

By [ShippoorDAO](#)

By [altoptimo.eth](#)

App

Getting Started

Common Errors

Learn about different errors users might encounter from time on the UI and what they mean

You have a collateral violation

Collateral violation means you're trying to borrow something without having enough collateral in your account to do so. Make sure you deposit a collateral asset first, like USDC. You can filter collateral assets on the markets page.

No match, code: NETWORK_ERROR

Please try refreshing the page and make sure you have a stable internet connection and your web3 wallet is on the correct RPC network (e.g., Goerli Test Network).

No match, code: UNPREDICTABLE_GAS_LIMIT

Try refreshing the page and trying the transaction again. This error also often occurs for other reasons. So if you keep having this issue, please check your browser's console log and please report the error that it shows.

Unknown error

Please try refreshing the page. If you receive the same error message, please check your browser's log for errors and create a support ticket with what you find.

Transfer amount exceeds balance, please check you have enough tokens in your wallet and make sure they are also not deposited into Euler

You don't have enough tokens in your wallet. If using the test, please make sure you have the correct testnet tokens from the official Euler testnet faucet.

execution reverted: e/too-many-entered-markets

For a given account, you can enter 10 markets max. You should try the transaction with another account or sub-account.

RPC error

Some RPC providers on the market (i.e. [Flashbots Protect RPC](#)) are not compatible with Euler simulation mode. If this error occurs, the user is advised to change the RPC (i.e. by changing the network in Metamask to default Ethereum network) and refresh the dapp. Having done that, the simulation feature should be functional again.

If, due to any reason, user wants to use their originally selected RPC to send the transaction, the following should be done (example described based on Metamask, actions might differ for other wallets):

- change the RPC to default Ethereum network
- add all the desired transactions to the batch
- assure that the simulation is passing without any error
- click 'Send txs' button
- when Metamask pops up
 - click on 'New address detected! Click here...!' at the top. Then click 'Add a nickname'. Input 'Euler Exec' as a nickname and click 'Save'
 - click on 'HEX', scroll down and click 'Copy raw transaction data'
 - click 'Reject'
 - disregard the error in the dapp
 - open Metamask and click 'Send'
 - select previously added 'Euler Exec' address
 - leave the asset set to ETH
 - leave the amount as 0 ETH
 - paste your clipboard contents into the 'Hex Data' field
 - change the network to desired
 - click 'Next' and sign the transaction

How To

Find out how to use the Euler Protocol through the interface at <https://app.euler.finance/>

Use the navigation bar on the left side to find guides for all the primary actions and functions on the Euler dApp.

Most of the functions and actions can be accessed through the **Quick Action** menu in the top navigation bar on the platform UI, while several actions will appear as buttons throughout the site for convenience.

If you come across an error while using the Euler dApp, please see the link below for the list of common errors. If the error persists or is not in the common errors list, please join the community Discord and open a ticket in the #support-ticket channel above the News and Announcement channels.

Errors



Common Errors

Connect a Wallet

Learn how to connect a wallet in Euler

About

Connecting a web3 wallet is the first step to utilise the Euler platform. Euler integrated Blocknative to enable users to connect with Formatic, MetaMask, Ledger, Trezor, WalletConnect, and Coinbase.

Step-by-step

1. Click the `Connect` button at the top right of the page.
2. Select your wallet provider in the Blocknative window.
3. Select your wallet from the list and unlock it.
4. Ensure you have selected the correct wallet, and your network is set to the correct one.

FAQ

What chains/networks does Euler support?

****Ethereum Mainnet and Goerli testnet (replacement for Ropsten).

Deposit

Learn how to deposit assets on Euler and begin earning interest

About

Depositing into Euler allows users to supply assets to borrowers and earn the Supply APY for an asset. Depositing collateral is also the first step in being able to borrow assets on Euler.

Step-by-step

1. Click the `Quick Action` button in the navigation bar.
2. From the menu, choose `Deposit`.
3. Select the sub-account you wish to deposit into and the asset you wish to deposit.
 - **Note:** if you have not yet approved Euler to be able to use this asset, you will need to `Enable` it first. In some cases you can achieve this with `Sign Permit` which is a type of gasless approval (free).
4. If you wish to use the asset as collateral to take out loans, check the `Enter market` box.
 - **Note:** you can only borrow against 'Collateral' tier assets. You will not be able to borrow against assets listed as 'Cross' or 'Isolated'. Entering the market for these type of assets will make them available to liquidators during liquidation events, but will not increase your borrowing power.
5. Enter the amount you wish to deposit (or hit `Max` to send the full available balance in your wallet).
6. Click `Deposit`. Your transaction will move to the `Transaction Builder`, which is a kind of shopping trolley for transactions. From here you have two options:
 1. Click `Send txs` to immediately submit the deposit.
 2. Add more transactions by revisiting the `Quick Action` button. You can then submit them all at once by clicking `Send txs`. This approach generally saves gas over submitting multiple individual transactions.

FAQ

I deposited, but the asset does not show up in my account.

****Make sure the deposit transactions did not fail, otherwise please create a support ticket in [Discord](#).

Withdraw

Learn how to withdraw assets from Euler

About

Users can withdraw assets from Euler at any time, directly to their wallet. Prior to withdraw, users should make sure debts are sufficiently repaid in order to withdraw their intended amount.

Step-by-step

1. Click on **Withdraw** in the **Quick Action** menu and select the Euler sub-account that you want to withdraw your tokens from and ensure that you have funds deposited.
2. Select the asset you are interested in.
3. Enter the amount you wish to withdraw:
 - Select **Max** to withdraw either your full balance or the most that the pool will allow if the pool has less available liquidity than you deposited.
 - Select **Safe Max** to withdraw enough such that your Euler sub-account will result in having a health score of 1.25.
 - Select **Liquidation** to withdraw enough such that your Euler sub-account will end up at health score 1 (right on the edge of a liquidation).

FAQ

I cannot withdraw my assets.

****Make sure you have enough assets to cover any loans, otherwise please create a support ticket in [Discord](#).

Borrow

Learn how to borrow assets on Euler

About

Borrowing assets on Euler creates a loan that users can repay at any time. Borrowers pay the Borrow APY rate to lenders. Note that users must first deposit collateral before they can borrow. All loans are over-collateralised, which means that users must deposit more value into the protocol than they can borrow.

Step-by-step

1. Select **Borrow** in the **Quick Action** menu.
2. Select the Euler sub-account that you want to borrow on.
 - Ensure that you have sufficient collateral deposited in the sub-account, and the asset is entered into the market.
3. Select the asset you are interested in.
4. Enter the amount you wish to borrow:
 - The **Max** button is representative of Liquidation x 1.5 to give a better UI experience, but we do not recommend you borrow more than **Safe Max** (unless it is a self-collateralized loan).
 - Select **Safe Max** to borrow enough such that your Euler sub-account will result in having a health score of 1.25 (not supported for self-collateralized loans).
 - Select **Liquidation** to borrow enough such that your Euler sub-account will end up at a health score of 1 (right on the edge of a liquidation).

FAQ

I've deposited an asset, but cannot borrow.

****Make your transactions were completed successfully. You can only borrow using approved collateral-tier assets, unless you're borrowing the same asset you've deposited. Make sure your collateral is sufficient for the amount you're trying to borrow. Otherwise, please create a support ticket in [Discord](#).

How come I can't borrow with an isolated asset?

****Note that isolated and Cross assets cannot be used as collateral, but Cross assets can be borrowed alongside other assets, while Isolated assets cannot. Euler aims to curb risk by limiting collateral tier assets to certain tokens with lower risk profiles.

Repay

Learn how to repay borrowed assets on Euler using funds from a wallet or deposits in Euler

About

Users can repay their borrowed assets using assets in their wallet or existing assets already deposited into Euler. Users are able to swap assets if they repay using a deposited asset that is different from the borrowed assets.

Step-by-step

1. Ensure that you have sufficient funds in your wallet or sufficient Euler deposits to repay a loan.
2. Select the Euler sub-account that you want to repay the loan on.
3. If repaying the loan using your Wallet Balance, select the liability asset (loaned asset) then the amount you want to repay.
4. If repaying the loan using Euler deposits, select the liability asset then the amount to repay.
5. Continue by selecting the swapped asset used to repay the loan and the amount.
 - If necessary, an appropriate swap will be made on an external exchange. Use the gear icon in the top-right to customise swap parameters.
6. Select **Max** to repay your full loan (or maximum amount possible based on your wallet balance/Euler deposits).
7. If a swap is necessary, wait for a quote and click the **Swap** and **Repay** button.

FAQ

Can I repay a loan with a different asset?

****Yes, either from the user's wallet or Euler deposits, which will be swapped for the loaned asset and repaid.

Mint

Learn how to mint assets on Euler for self-collateralised positions

About

Mint is a unique function on Euler that enables users to simulate a recursive borrowing strategy. Mint creates equal amounts of deposits and debts for the same asset. It is often the starting point for creating a multiplied long/short position or used for liquidity mining (when lending/borrowing on a particular market is incentivised).

Example

A user first does a **Deposit** of \$1000 of USDC. They then **Mint** \$5000 USDC. Their account now has a total of \$6000 USDC deposits, and \$5000 USDC debts. This gives them a multiplied position, since they hold more debt than their initial deposit would allow for.

The **Mint** function mimics what would happen if a user deposited \$1000 USDC, then borrowed \$900 USDC, then redeposited that \$900 USDC, to borrow \$810 more USDC, and so on.

Step-by-step

1. Ensure that you have sufficient collateral in the sub-account you are minting to.
2. Select the Euler sub-account that you want to mint from.
3. Select the asset you are interested in.
4. Enter the amount you wish to have upon completion.
 - **Max** here is representative of 19x multiplier (right on the edge of a liquidation), and hence we do not recommend you mint more than safe max.
 - Select **Safe Max** to mint enough deposits and debt such that your Euler sub-account will result in having a multiplier of 15x.
 - Select 0 or the Burn button to burn a previously minted position (burn removes an equal amount of deposits and debts from an account).
 - While entering the amount, observe the Multiplier and Time to Liquidation to make a decision.

FAQ

Can I be liquidated if I use mint?

***Mint creates a multiplied position which poses liquidation risk when Mint an asset different from the user's collateral. Additionally, the interest rates also present risk of loss as they're variable and dependent on the market.

Burn

Learn how to burn assets on Euler to remove Mint positions

About

Burn enables users to 'repay' and undo their multiplied positions created by the Mint function by removing an equal amount of deposits and debts from an account.

Step-by-step

1. Ensure that you have sufficient supply & debt tokens in the sub-account you are burning on.
2. Select the Euler sub-account that you want to burn on.
3. Select the asset you are interested in.
4. Enter the amount you wish to burn:
 - Select **Max** to burn all available of selected asset.
 - While entering the amount, observe the Multiplier to make a decision.

FAQ

Can I use Burn to repay my debts?

****Burn is only used to close leverage positions from **Mint** and **Long** functions. Use the **Repay** function to close borrow positions.

Sub-Accounts

Learn more about sub-accounts on Euler

About

Sub-accounts enable users to create new accounts without having to create or connect different wallets. Users can create sub-accounts to isolate positions, for example. Here are the steps to create sub-accounts.

Step-by-step

In the top right navigation, hover over the icon and click on the **Create New** button, which will automatically add a new sub-account and simultaneously switch to it.

- There is no cost to create sub-accounts.
- Part of opening up ourselves to lending on every market means that we have to protect the system against volatile and potentially malicious tokens. To do this we have categorised assets as isolated, cross and collateral.
- If the asset is isolated, then you can only borrow and lend on one single asset per sub-account.

FAQ

How many sub-accounts can I create?

****Users can create up to 255 sub-accounts (in addition to their Main account, which is sometimes called sub-account 0).

Can I rename sub-accounts?

****No, sub-accounts cannot be renamed. This is because all the information held about users' accounts is retrieved from the blockchain, and adding sub-account names to the chain would be expensive.

Transfer

Learn how to transfer assets between sub-accounts on Euler

About

Transfer allows users to move deposited and debt assets between sub-account in Euler. User should ensure any borrow positions will have enough collateral assets in the related sub-accounts prior to transferring any assets.

Step-by-step

Transfer ETokens

1. Ensure that you have sufficient deposits in the sub-account you are transferring from.
2. Select the Euler sub-account that you want to transfer from.
3. Select the Euler sub-account that you want to transfer to.
4. Select the asset you are interested in.
5. Enter the amount you wish to transfer:
 - Select **Max** to transfer all the selected asset deposits (or maximum amount possible based on your 'from' sub-account positions).
 - Select **Safe Max** to transfer enough such that your 'from' Euler sub-account will result in having a health score of 1.25.

Transfer DTokens

1. Ensure that you have sufficient debt tokens in the sub-account you are transferring from.
2. Select the Euler sub-account that you want to transfer from.
3. Select the Euler sub-account that you want to transfer to.
4. Select the asset you are interested in.
5. Enter the amount you wish to transfer:
 - Select **Max** to transfer all the selected asset debt (or maximum amount possible based on your 'to' sub-account positions).
 - Select **Safe Max** to transfer enough such that your 'to' Euler sub-account will result in having a health score of 1.25 (not supported for self-collateralized loans).

FAQ

Can I transfer multiple assets?

****Yes, you will need to make multiple actions sent to the transaction builder.

Swap

Learn how to swap assets in Euler

About

Swapping assets in Euler enables users to exchange one deposited asset for another using external exchanges that have been integrated into the platform (1inch or Uniswap).

Step-by-step

1. Ensure you have sufficient supply tokens in the sub-account you want to swap from.
2. Select the Euler sub-account that you want to swap from.
3. Select the Euler sub-account that you want to swap to.
4. Select the asset you wish to sell.
5. Select the asset you wish to buy.
6. Enter the amount you wish to sell.
 - Select **Safe Max** to sell the amount such that your Euler sub-account will result in having a health score of 1.25.
 - Select **Liquidation** to sell the amount such that your Euler sub-account will end up at health score 1 (right on the edge of a liquidation).
 - Select **Max** to sell all the selected asset.
 - Users can set the **slippage** to the desired value by clicking the **gear icon** in the top-right corner.
7. Wait for a quote and click the **Swap** button.

FAQ

Can I swap assets from my wallet?

****Currently, users can only swap assets they've deposited.

Short/Long

Learn how to create short or long positions in Euler

About

Short allows users to build a leveraged short position by borrowing and then immediately selling an asset using external exchange (1inch or Uniswap).

Step-by-step

1. Ensure that you have sufficient collateral in the sub-account you are shorting on.
2. Select the Euler sub-account that you want to short on.
3. Select the asset you wish to short on.
4. Select the asset you wish to use as the collateral.
5. Enter the amount you wish to short:
 - Select **Safe Max** to short the amount such that your Euler sub-account will result in having a health score of 1.25.
 - Select **Liquidation** to short the amount such that your Euler sub-account will end up at health score 1 (right on the edge of a liquidation).
 - Users can set the **slippage** to the desired value by clicking the **gear icon** in the top-right corner.
6. Wait for a quote and click the **Short** button.

FAQ

I'm creating a Short/Long position, so why do I see the asset in a Self Collateral and Self Liability row?

****The asset is sold and redeposited on Euler, which creates the self collateral/liability row, but your intended position is there.

Protected Collateral

Learn how to protect your collateral assets in Euler

About

Toggling collateral assets on Euler enables users to protect their tokens as collateral without permitting borrowing. Users may want to do this so that their collateral is purely being used as collateral for their own borrowed assets, while lowering other risks.

Step-by-step

1. At the desired sub-account page, find the asset under the supply column.
2. Click the `toggle` button in the `Entered` column in the Account view to enter/exit market.
 - Ensure that you have no outstanding borrows on the account that are collateralised by a given asset.

FAQ

The transaction builder won't let me toggle collateral, it presents a Collateral violation error.

****Toggling collateral will disallow using the asset as collateral, so you won't be able to borrow using the asset.

Transaction Builder

Learn how to use the transaction builder in Euler

About

The transaction builder lists all the functions and actions that the user generated via the Quick Action menu, such as deposit, borrow, mint, etc. The transaction builder also enables users to add, remove, rearrange and edit transactions before approving the final transaction.

Step-by-step

The transaction builder can be expanded out from the right side of the UI by clicking the `OPEN` button.

- Every time you alter your transaction builder, it automatically checks how much gas you will spend and tries to figure out if there will be any errors.
- Transactions are executed in the order you have specified, and the order of the transactions can be rearranged in a drag and drop fashion.
- You are able to send transactions from different sub-accounts at the same time.

FAQ

An error appeared as I added actions to the transaction builder.

****Check with the [Common Errors](#) and make sure the transaction builder is up-to-date with your actions (i.e. Swapping/transfer may become outdated if prices/interest rates significantly changed).

Wrap

Learn how to wrap or unwrap assets in Euler

About

Wrap is a versatile function that allows users to convert between wrapped and unwrapped assets. Two of the most common use cases are wrapping/unwrapping Ethereum (ETH) and wrapped Ethereum (WETH) as well as Lido Staked ETH (stETH) and wrapped Lido Staked ETH (wstETH). However, users can also wrap and unwrap their Protected Collateral tokens (pTokens).

Step-by-step

1. Ensure that you have sufficient amount of asset in your account.
2. Select the assets you want to wrap or unwrap to/from.
3. Enter the amount you wish to wrap or unwrap.
4. Click the [Wrap](#) or [Unwrap](#) button.

FAQ

Can I wrap/unwrap other assets?

***You can only wrap/unwrap the assets listed in the dropdown.

Activate

Learn how to activate a market in Euler

About

Users can activate new lending markets for assets on Euler in a permissionless way. Assets must have a Uniswap V3 WETH paired pool and should be listed on [CoinGecko](#) in order to appear in the asset's table. Since Euler uses Uniswap's V3 pools for the market's oracle, it's highly recommended to only activate pools with sufficient and wide range liquidity. Check out an asset's oracle risk [here](#).

Step-by-step

1. Search and select an unlisted asset.
2. Click `Activate` to create a market for the token.
 - Unlisted tokens come from community token lists, contact support if the desired token does not appear in the list.
 - Borrowing will not be available right away, somebody has to lend the asset first.

FAQ

Where can I Activate a market?

****Search for an asset in the search bar on the [Euler app](#).

I've activated a market, how come I cannot use it as collateral?

****Newly activated markets are automatically set as isolated tier assets. Create a proposal to make it a collateral tier asset if it has sufficient liquidity in its oracle.

Allowances

Learn how to change asset approvals through Allowances in Euler

About

Allowances enable users to add, update or revoke approval of the Euler protocol to access markets in the user's wallet. Users can change the amount of tokens in pre-approved markets.

Step-by-step

1. Select an asset from the list.
2. Enter the amount you wish to allow the protocol to approve for use and click `Update`.
 - `Max` will allow the protocol to use the total supply of that token, not just what the user owns.
 - Enter `0` (zero) to restrict the protocol from using any more than zero tokens.
3. If listed, change the pre-approved amounts by entering a new amount and clicking `Update` or by selecting `Max`.
4. To completely revoke allowance, select the `Revoke` button.

FAQ

I've changed my allowances, but now I cannot complete some transactions.

****Make sure the amount in your transaction is allowed in the amount you've permitted in the allowances.

Retrieve Browser Errors

Learn how to retrieve errors from the developer console in your browser

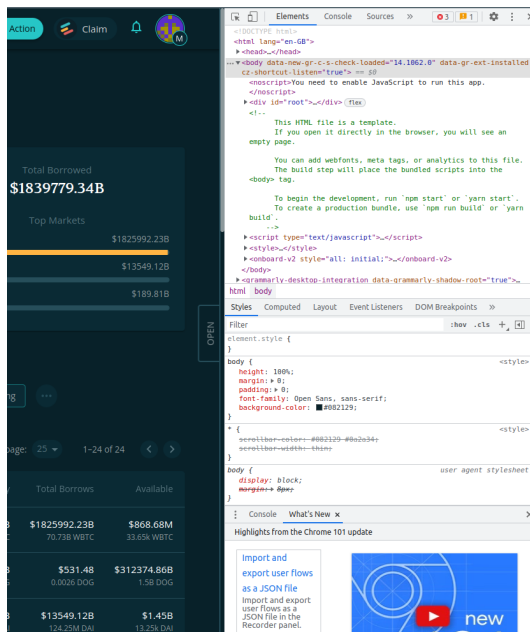
About

Sometimes an error will occur that needs further investigation by the Euler Labs team. If you have created a support ticket in [Discord](#), then you may be asked to report errors from the browser console to the team. You will find the instructions for doing this below.

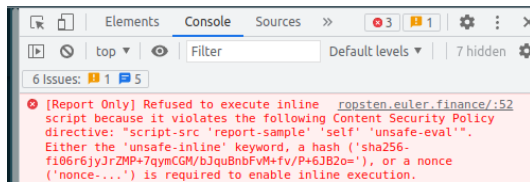
IMPORTANT: the Euler Labs team will never ask for your private keys or passphrase.

Step-by-step

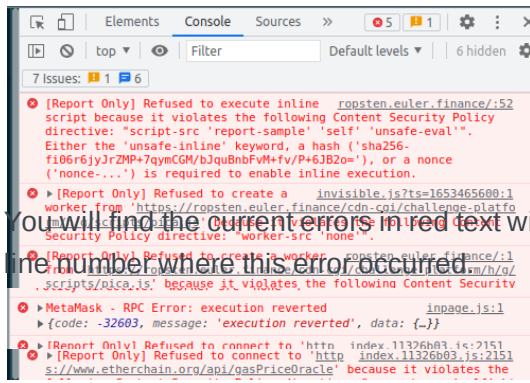
1. Open the developer console by pressing Option + ⌘ + J (on macOS) or Shift + CTRL + J (on Windows/Linux). You can open it by right-clicking anywhere on the webpage and selecting Inspect Element option.
2. On the top of the developer panel, there are multiple tabs like Elements, Console, Networks, and Sources.



3. Click on the Console tab



4. Console is a place where we have all errors, warnings and general logs that appeared during the run time of the website.



5. You will find the current errors in red text with a specific error code, the cause, and the file name and line number where this error occurred.

FAQ

Let us know if there's anything you think we should add here.

Lending and Borrowing

Below is a basic example of lending and borrowing on Euler protocol. This is not financial advice and only serves as an educational example on using the protocol.

Let's say you want to borrow XYZ token. XYZ token ranks high in the index list, and therefore has a borrow factor of 0.80.

Lending

Since the loan is collateralised, you first need to deposit (lend) one or more collateral assets. You deposit ABC, which is a stable coin with a collateral factor of 0.90.

You lend 1000 ABC (1000 USD worth), receive eABC (claim on deposited ABC and interest accrued from utilisation). Note that the more users borrow your ABC tokens, the more interest you accrue and hence the higher value of your collateral.

Borrowing

To calculate the maximum amount of XYZ you can borrow, multiply the borrow and collateral factors: $0.80 \times 0.90 = 0.72$. Consequently, you can borrow up to 720 USD worth of XYZ against 1000 ABC (0.72×1000 ABC).

If you decide to borrow 700 USD worth of XYZ, Euler will mint an equivalent amount of dXYZ (debt token equal to principal amount owed and accrued borrowing costs). Note that if borrowing rates rise, your debt levels will increase as well.

Liquidation Risks

If XYZ token rose in price and/or borrowing interest rose significantly so that the value of your dXYZ rose above 720 USD, you are subject to liquidation and your claim on the collateral (eABC) will be given to a liquidator at a discount as well as your debts (dXYZ) in exchange for repaying the debt.

Similarly, should the value of ABC plummet, your eABC won't be enough to maintain the 0.72 factor, and you will be subject to the same liquidation procedure.

Return Potential

Alternatively, if you borrowed XYZ, sold it and XYZ price plummeted, you'd be able to repay your debts at a lower level and make a profit. You could also utilize your position by selling XYZ, buying more collateral, borrowing more XYZ etc. until you reach the borrowing limit. This way, you'd have a multiplied short XYZ/ABC position. More on the mechanics of multiplied positions [here](#).

Note: we have implemented the swap module that allows users to utilize their positions easily without incurring additional gas fees.

FAQ

Get answers to frequently asked questions about Euler.

General

What is Euler?

Euler is a non-custodial protocol on Ethereum that allows users to lend and borrow almost any crypto asset. Learn more about how Euler works by reading the [white paper](#) here.

Who developed Euler?

Euler was initially created by a team of developers at a company called Euler Labs (see website [here](#)). Today it is progressively decentralising and receives contributions from the external developer community as well as ongoing contributions from Euler Labs.

What is EulerDAO?

EulerDAO is a decentralised autonomous organisation encapsulating all holders of a governance token called EUL. Holders of the token have voting powers to propose and make changes to the underlying code of the Euler Protocol.

What is the Euler Foundation?

Since DAOs not have a formal legal structure, the Euler Foundation was established as a non-profit Foundation Company designed to represent EulerDAO in the 'real world'. The Euler Foundation has no shareholders and cannot pay out dividends to its members. Its purpose is to provide a vehicle by which EulerDAO can sign a contract or engage a company for a service that the DAO requires.

Community Involvement

How can I get involved in EulerDAO?

Join the [Discord](#) and meet the community, make proposals and discussion on the [governance forum](#), or send a message if you have other ideas of contributing to the protocol.

Where is the developer documentation?

Please refer to the [Developers](#) section and check out the [Euler SDK](#).

Where are your branding guidelines/materials?

The branding materials are located at the following link: [euler.finance/branding](#). Copyright is owned by the Euler Foundation.

Euler dApp

How do I deposit and borrow?

Check out the How To guides [here](#). Or just go to the [Euler app](#), login via the Connect button on ETH mainnet. Click on the Quick Action button. You can deposit and borrow through the same named action buttons. Choose the asset and amounts, then approve of the transactions.

How do I activate a market?

Search for an asset in the search bar on the [Euler app](#). Unlisted assets can be activated by the Activate button, which will ask you to initiate the transaction. Once complete, the asset will be listed and activated.

How do I get testnet tokens?

The Goerli testnet has a test ERC20 token faucet smart contract deployed at [0x1215396CB53774dCE60978d7237F32042cF3a1db](#).

To get testnet tokens for Goerli, connect open the smart contract on Etherscan by clicking the link above. Click on the tab with the green tick, then click on `Write Contract` and connect your wallet. Once connected, expand the withdraw feature and paste the underlying ERC20 token smart contract address and click on the `Write` button. This will require you to confirm the transaction in your wallet which costs gas.

Once confirmed, the token faucet smart contract will send an amount of the specified testnet ERC20 token (up to a pre-defined threshold) to your connected wallet address.

The Goerli testnet token faucet supports the following ERC20 tokens: [WETH](#), [UNI](#), [USDC](#), [USDT](#), [DOGE](#), [WBTC](#), [COMP](#), [CRV](#).

Why can't I find a specific token to activate?

Some tokens might not be on the token list or might not have a pool on Uniswap v3. Please [send a message](#) if you have trouble finding an unlisted asset.

What are asset tiers?

Euler uses risk-based [asset tiers](#) to protect the protocol and its users. Isolated and Cross assets cannot be used as collateral, but Cross assets can be borrowed alongside other assets, while Isolated assets cannot.

What are sub-accounts?

[Sub-accounts](#) enable users to isolate positions into different accounts.

What oracle does Euler use?

By default Euler uses [Uniswap V3](#) Time Weighted Average Price (TWAP) as the pricing oracle. However, the oracle used can be changed in the governance process on one-to-one basis. Currently, Euler also supports Chainlink price feeds as a price source.

What is the oracle rating?

The [Oracle Rating system](#) attempts to rank Uniswap v3 price oracles for different markets high, medium, or low based on the ease with which they can be manipulated.

What are the reserves?

[Reserves](#) are protocol-owned liquidity deposited on Euler to provide a backstop against a 'run on the bank' scenario. Reserves build up over time as borrowers pay interest on their loans. The reserves are ultimately controlled by EulerDAO Governance.

How can an asset become collateral?

An asset must have substantial liquidity with wide distribution in its Uniswap V3 WETH paired pool. If the asset has a low-risk oracle, it is in a better position to be regarded as a safer asset to become collateral through a [governance proposal](#).

Is Euler on any other chains/L2s?

Euler is currently only on Ethereum Mainnet. Euler is open to and exploring other chains and layers, but nothing is currently imminent.

What is the difference between Sign Permit and Enable?

Enable is the standard approval transaction that allows Euler smart contracts access to that asset. You can edit the amount from unlimited in Metamask. Sign Permit is a gasless way of approving a contract to use your tokens as a one-time allowance ([EIP-2612](#)).

Features

What are Mint and Burn?

[Mint](#) enables users to more efficiently create leveraged positions of borrowers and deposits. For example, a user can deposit \$1,000 USDC, and mint \$2,000 USDC. Then you will have \$3,000 USDC deposits, and \$2,000 USDC liabilities. Burn closes Mint positions.

What does the Transfer action do?

Users can [transfer](#) deposits (eTokens) and debt (dTokens) to other accounts. Always check the accounts involved in your operation will have sufficient collateral to support it.

Can I swap tokens?

Yes, the swap feature enables users to exchange one deposited asset for another using Uniswap and 1inch DEXs.

How can I short a token?

The [Short](#) action allows users to build a leveraged short position by borrowing and then immediately selling an asset on an external exchange, including 1inch and Uniswap.

Should I mine at 19x?

Euler does not give financial advice, but users should note that mining at 19x is highly risky and can lead to liquidation if the user's collateral cannot cover the interest fees. Mining at 19x most often liquidates positions within the same day it is created.

What is Time To Liquidation (TTL)?

[TTL](#) allows users to see the amount of time they have until their position is liquidated based on the current interest rates and prices of those positions.

What is a soft liquidation?

Euler allows [liquidators](#) to repay up to the amount needed to bring a violator back out of violation (plus an additional safety factor). Other protocols are fixed so that liquidators can pay off up to half a borrower's loan in one go, regardless of how underwater their position is.

Gauges & EUL Distribution

How can I claim EUL tokens?

Users can go to the [Euler app](#), click on the Claim button in the navbar and claim any available tokens of previous epochs in the Distribution window once EUL tokens are available to claim.

What utility does EUL have?

EUL's main utility is as a [governance token](#) for the Euler protocol. Users with EUL can have a say in the future decisions and direction, as well as the EUL distribution in the Euler gauges.

What is the distribution of EUL tokens for borrowers?

The distribution is decided by the amount of EUL tokens staked in the [gauges](#).

How are markets decided for each epoch?

Users can add or remove EUL distribution eligible markets simply by staking EUL tokens in their preferred gauges. See the link in the above question for the results of these votes. Please see [eIP 51](#) for the latest iteration of EUL distribution.

In order to receive EUL emission, assets (as per [eIP 24](#)) need to have a Chainlink price feed as well as receive EUL votes in the gauge system.

EUL Token

Is there a token? How can I earn it?

The Euler token (EUL) is distributed to borrowers on select markets on the platform. Please see the [Gauges](#) section for more details.

What are the tokenomics?

Allocation, vesting and other information about the EUL token can be found in the About section under [EUL](#).

Is there a TGE/ICO/IDO?

There is no public sale.

Where can I trade EUL?

While EUL token has been released, Euler Labs does not give financial advice on trading the token.

How do I get an airdrop?

There is no airdrop. 1% of the supply was allocated to users who interacted with the dApp during [Epoch 0](#) as a one-off retroactive distribution.

Do testnet users receive rewards?

No, [testnet](#) is to preview upcoming features and for users to learn about the protocol without any mainnet gas fees.

Someone messaged me promising free tokens/ICO/NFTs/etc, is it real?

No, that is fake. No one related to Euler will ever message anyone directly, nor offer free tokens or investments of any kind.

Partnerships

Who can I contact about partnerships/integrations?

Feel free to reach out through Discord or other platforms in the Quick Links section. Also read the [integration guide](#) for more details.

Can we list EUL on our exchange?

Euler cannot advise on exchange listings, nor pay for any listings.

Can we offer our token in your liquidity mining programme?

Euler will be adding this feature in the near future. Please get in touch if you'd like to integrate with the distribution mechanisms.

Can you market our company/token if we pay you or activate our token?

Sorry, Euler does not accept payments or donations of any kind to promote activated markets.

Miscellaneous

How does the Euler dApp detect wallets associated with illicit activities?

In alignment with industry best-practices, Euler utilizes Chainalysis to identify and block wallets that are associated with certain illicit activities.

Chainalysis is a market leader in protecting against interactions with bad actors linked to sanctions, financial crime, child sexual abuse material, terrorist financing, scams, hacked or stolen funds, ransomware, and human trafficking.

It is Euler's aim to prevent those engaged or associated with illegal activity from using the protocol. Euler is committed to responsible development, innovation, and financial inclusion.

Euler Protocol

Getting Started

Risk Methodology

Learn more about how risk parameters on Euler are determined

Introduction

The Euler risk framework aims to do two things:

1. **Maximise** capital efficiency through borrowing and lending **activity**; and
2. **Minimise** risk and the probability of **bad debts**.

To achieve this, a methodology to stress test individual assets as well as simulate a portfolio of assets in tail risk scenarios.

Methodology

Ranking all available ERC20 tokens according to risk parameters:

1. Smart Contract Risk
2. Centralisation
3. Volatility
4. Liquidity

Additionally, assessing **Oracle Risk**

In order to **arrive** at:

1. **Collateral** Factor
2. **Borrow** Factor
3. **Cross Tier** Factor

Simulate risk scenarios to maximise borrowing and lending activity and minimise bad debts

Update factors and methodology through governance

Asset Tiers

Euler assets fall into three different tiers: isolated, cross and collateral tiers.

Isolated Tier

Isolated tier assets are available for ordinary lending and borrowing, but they cannot be used as collateral to borrow other assets, and they can only be borrowed in isolation. What this means is that they cannot be borrowed alongside other assets using the same pool of collateral. For example, if a user has USDC and DAI as collateral, and they want to borrow isolation-tier asset ABC, then they can only borrow ABC. If they later want to borrow another token, XYZ, then they can only do so using a separate account on Euler.

Governance allows to promote assets to cross and collateral tiers, but also to alter borrow factors. However, there will be a default borrow factor for all assets listed in the isolated tier.

Cross Tier

Cross tier assets are available for ordinary lending and borrowing, and cannot be used as collateral to borrow other assets, but they can be borrowed alongside other assets. For example, if a user has USDC and DAI as collateral, and they want to borrow cross-tier assets ABC and XYZ, then they can do so from a single account on Euler.

Collateral Tier

Collateral tier assets are available for ordinary lending and borrowing, cross-borrowing, and they can be used as collateral. For example, a user can deposit collateral assets DAI and USDC, and use them to borrow collateral assets UNI and LINK, all from a single account. \

Collateral and Borrow Factor

On Euler, we use a two-sided approach to estimate risks of borrowing any tokens versus any given collateral. These risks are encapsulated in asset-specific collateral factors (as on Compound) and borrow factors (an Euler innovation).

Consequently, collateral factor reflects the risk of the asset that's being used as collateral, whilst borrow factors reflect risks of the asset that's being borrowed.

Collateral Assets

When it comes to quality of collateral, it is of paramount importance that it ranks very high in our index list.

An illiquid collateral asset can be exploited by causing a price surge to allow a malicious user to borrow an inflated amount of tokens without an incentive to return them.

Alternatively, a collateral asset that's collapsing in price and is experiencing high slippage makes it uneconomical for liquidators to close positions, leading to bad debts. This scenario is more systemic, which is why only the highest quality assets can become eligible collaterals.

Borrowed Assets

While borrowed assets are less systemic than collateral assets, they also have specific risks.

For example, a borrowed token price that triples in a matter of seconds versus its collateral means there's no incentive for the borrower to return the token, which results in bad debt. This is why the borrow factor should reflect the volatility and liquidity of the asset.

Alternatively, crashing a borrowed asset's price allows a malicious actor to borrow more tokens than normally possible. This attack can happen if there's not enough liquidity overall, especially if liquidity is too concentrated around a tiny range.

Ranking Assets

Scale

Each risk parameter goes from 0 (highest risk) to 1 (lowest risk). The combination of risk parameters results in a comprehensive index for a given digital asset, which also goes from 0 to 1.

Risk Parameters

Smart Contract Risk

Smart contract risk parameter can either be 0, 0.5 or 1.

It is arguably the biggest tail risk, as a badly written smart contract can result in hacks and stolen money, leading to catastrophic collapse in price.

We assess this by checking **whether the protocol was audited**, the **number of days the protocol has been functioning without hacks** and deeper research if needed (for promoting up the tiers).

Centralisation

Centralisation parameter can go from 0 to 1.

It measures whether a small number of holders have undue influence over the token. For example, a founder with 70% ownership of the tokenomics pie and flexible vesting period can oversupply the market with tokens, leading to an abrupt sell-off. Alternatively, a whale holding a large chunk of tokens can easily pass deleterious changes through governance.

This risk can be measured by estimating the **median size of token amount per holder**. When the ownership structure isn't transparent, we will employ forensic due diligence.

Volatility

Volatility parameter can go from 0 to 1.

All other things equal, an asset with 100% realised volatility is more likely to cause a liquidation than an asset with a 10% realised volatility. Hence, less volatile assets should have more favourable borrowing and collateral factors.

This parameter can be measured via **realised** (and implied, if available) **volatility**, with emphasis on downside risk for collaterals and upside risk for borrowed assets.

Liquidity

Liquidity parameter can go from 0 to 1.

An asset with 100 mil USD daily turnover is easier to buy and sell versus an asset with only 1 mil USD turnover, all other things being equal.

This is particularly important for liquidators that receive collateral assets at discount for repaying borrowed tokens, as they typically immediately sell that collateral. If they're unable to sell that collateral and/or buy the borrowed asset at a decent price given a liquidator discount, then they do not have the incentive to liquidate. This leads to bad debts, which we need to minimise.

Measuring liquidity can be done by **estimating historical slippage** caused by a certain amount of volume.

It's important to note that we are estimating the liquidity **vs ETH on Uniswap v3**, as an existing ETH market on Uniswap v3 is a prerequisite to being activated on Euler and our oracles are based on Uniswap v3 TWAPs.

This means that even if a token has very high liquidity on Uniswap v2, it wouldn't necessarily have a high borrow factor if liquidity is low on Uniswap v3.

Oracle Rating

What is an oracle?

Within the context of pricing, an oracle is an on-chain API for price. Differently put, it simply tells you what the price of an asset is at a given time.

Oracle Risk

While we think Uniswap's oracles are best suited for our permissionless lending protocol, depositing into an Euler pool backed by illiquid liquidity pools on Uniswap can lead to devastating results.

For instance, inflating the value of a collateral allows the attacker to borrow an inflated amount of tokens, leading to bad debt. This is the most systemic and widespread attack on lending protocols.

Alternatively, if the Uniswap V3 oracle of the borrowed asset is manipulated to the upside, the attack could trigger liquidations and sweep borrowers' collateral.

Even more of concern is when the attacker can manipulate the asset pricing to the downside. Hypothetically, if the price drops to almost zero, the attacker only needs a small amount of collateral to borrow the entire pool and run away with a hefty profit.

Euler's Oracle Risk Grading System

In order to assess an oracle's safety, our team have developed a tool to calculate the cost of moving a given Uniswap v3 TWAP: oracle.euler.finance.

Using the tool, we can calculate the cost of moving the TWAP by 20.89% (minimum required to break even on highest-quality assets) up and down over 1 and 2 blocks:

UNI / WETH 0.3%

TWAP Window:	144
Collateral Factor:	0.88
Borrow Factor:	0.91
TWAP Pump Impact Target:	20.89%
TWAP Dump Impact Target:	20.89%

BLOCKS	↑ VALUE USD	↑ COST USD	↑ TOTAL COST USD	↓ VALUE USD	↓ COST USD	↓ TOTAL COST USD
1	52.75B	52.73B	52.73B	1330.9B	1330.9B	1330.9B
2	252M	234.82M	469.63M	341.5M	338.89M	677.78M
3	65.05M	47.87M	143.61M	44.2M	41.59M	124.77M
4	53.2M	36.02M	144.09M	13.7M	11.09M	44.36M
5	49.5M	32.33M	161.66M	7.75M	5.14M	25.69M
6	46.2M	29.05M	174.33M	5.76M	3.16M	18.94M
7	44.6M	27.48M	192.34M	4.84M	2.24M	15.7M
8	43.6M	26.5M	212.02M	4.36M	1.77M	14.14M
9	42.95M	25.88M	232.91M	4.07M	1.48M	13.3M
10	42.1M	25.08M	250.76M	3.87M	1.29M	12.88M

Then, we take the minimum of these 4 values: \$469.63 million and assign a rating to it according to this table:

Minimum Cost of Attack over 1-2 blocks	Rating	Meaning
Above \$50 million	High	Attack is extremely unlikely
Between \$50 million and \$5 million	Medium	Attack relatively unlikely
Below \$5 million	Low	Expect total loss of funds

Consequently, UNI/WETH pool safety is deemed **high** as the minimum cost of attack up and down over 1-2 blocks is > \$50 million.

This is displayed on the front-end page of the respective lending pool:

Euler MARKETS DASHBOARD ACCOUNT Quick Action Claim 🔔 🌍

Market

Market **USDC** Collateral USD Coin

Getting started Deposit Borrow ...

Mining APY **0%** 0 EUL Dist

Uniswap TWAP **\$1.00** 0.000372 ETH

Info <> Contracts 🔗 Uniswap Oracle 📅 30 mins TWAP

Wallet **\$0** 0 USDC

My supply **\$2.54** 2.55 USDC

My debt **\$0** 0 USDC

Liquidity mining (soon™) Mining Voting

Oracle rating ● ● ●

Total Supply \$65.83M 66.13M USDC	Supply APY 0.64%	Utilisation 41.06%	Borrow APY 2.03%	Total Borrowed \$27.03M 27.16M USDC
Reserves \$54.78k 55.02k USDC	Collateral factor 0.9	Reserve Factor 0.23	Borrow Factor 0.94	Available to Borrow \$38.8M 38.98M USDC

Historical interest rate graphs undergoing upgrade.

Keep in mind that this is merely an indicative tool and we bear no responsibility for loss of funds.

How to Improve the Oracle Rating?

If you are a project that wants to improve its token's oracle rating and be eligible for higher borrow and collateral factors, it's crucial to **provide full-range liquidity to the XYZWETH pair on Uniswap V3**.

By **full-range liquidity** we mean providing liquidity **from the lowest tick all the way to the highest tick** without any gaps in between.

A good example is [METIS/WETH](#):

Metis / ETH 0.3%

1 Metis = 0.0530 ETH | 1 ETH = 18.8551 Metis

Add Liquidity Trade

Total Tokens Locked

- Metis: 43.56k
- ETH: 2.05k

TVL **\$12.20m** ↑ 4.51%

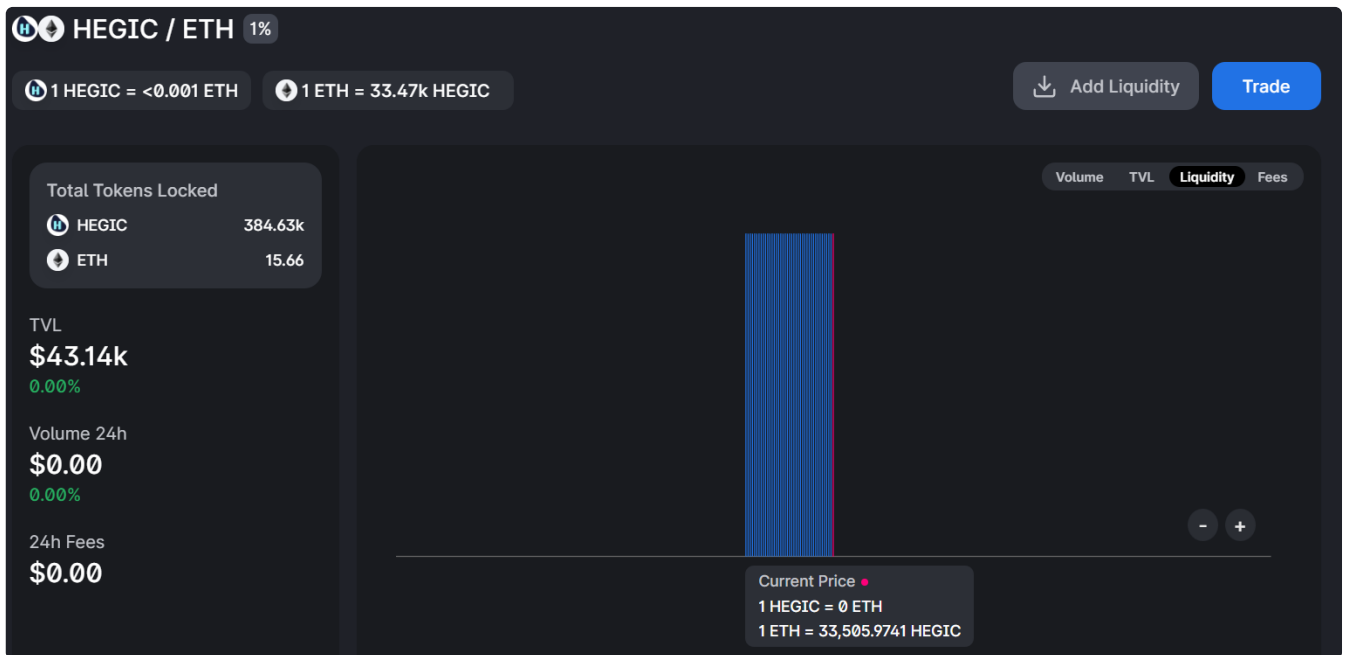
Volume 24h **\$4.11m** ↓ 26.10%

24h Fees **\$12.34k**

Volume TVL **Liquidity** Fees

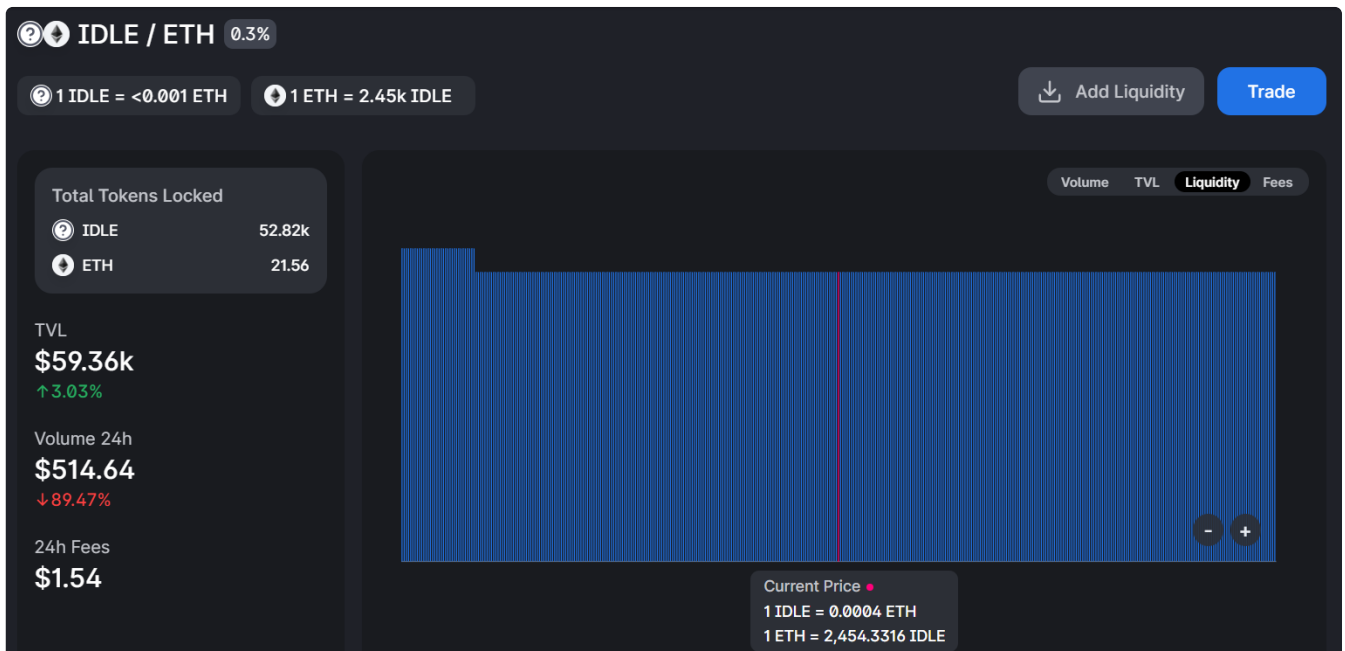
Current Price
1 Metis = 0.0529 ETH
1 ETH = 18.9131 Metis

A suboptimal scenario is [HEGIC/WETH](#), where liquidity is uber-concentrated:



Check out this [video](#) going through different manipulation scenarios for a more in-depth explanation:

It's important to note **that even a small amount of fully-spread liquidity can significantly increase the cost of attack**. For eg, the [IDLE/WETH](#) pool has a mere \$52k TVL, yet the minimum cost of attack is a whopping \$115 million:



T

References

Check out this blog post written by Darek explaining the oracle tool: <https://blog.euler.finance/uniswap-oracle-attack-simulator-42d18adf65af>

Check out Michael's paper on how even a small amount of full-range liquidity can make an attack incredibly costly: <https://github.com/euler-xyz/uni-v3-twap-manipulation/blob/master/cost-of-attack.pdf>

Simulation Environment

To make sure we minimise bad debts whilst maximising activity, we stress-test liquidation scenarios by simulating tail risk events on individual assets as well as on portfolios of assets to estimate bad debt risk.

Example: One asset

For instance, if our index list shows that a token jumped from 250th to 30th place and was able to maintain that position for a sufficient amount of time, we may simulate activity versus bad debts as a share of total loans given different borrow factors. As a result, we may improve the borrow factor from 0.28 to 0.35.

Example: Portfolio of assets

Alternatively, we can simulate a more complicated environment with 50 tokens from the lower quartile as borrowed assets backed uniformly by 4 collateral assets (3 of them time-tested and 1 proposed new collateral) in high volatility situations. By simulating tail risk scenarios, we can assess the worst-case scenario for the protocol and decide whether inclusion of the collateral asset is appropriate.\

Addresses

Smart contract addresses for Euler

Besides Euler, the smart contracts are upgradeable via Governance which will be controlled by EUL token holders (i.e., the implementation contracts will be upgraded periodically). Hence, for contract interaction, please use the proxy smart contract addresses.

For example, using [web3.js](#) to interact with the Markets contract, we use the Markets contract ABI but set the Markets contract proxy address as the target contract:

```
const markets = new Contract(MarketsABI, Markets_Proxy_Address); // proxy address
```

Networks

The Euler protocol is currently deployed to the following networks:

Mainnet

Contract	Proxy Address	Etherscan (Proxy)	Etherscan (Implementation)	Source Code
Euler	0x27182842E098f60e3D576794A5bFFb0777E025d3		Etherscan	GitHub
Markets	0x3520d5a913427E6F0D6A83E07ccD4A4da316e4d3	Etherscan	Etherscan	GitHub
Liquidation	0xf43ce1d09050BAfd6980dD43Cde2aB9F18C85b34	Etherscan	Etherscan	GitHub
Exec	0x59828Fdf7ee634AaaD3f58B19fDBa3b03E2D9d80	Etherscan	Etherscan	GitHub
Swap	0x7123C8cBBD76c5C7fCC9f7150f23179bec0bA341	Etherscan	Etherscan	GitHub
SwapHub	0x542ACC8E1db037d6008587aBfB1B7fB44014c629	Etherscan	Etherscan	GitHub
Governance	0xAF68CFba29D0e15490236A5631cA9497e035CD39	Etherscan	Etherscan	GitHub
EulerSimpleLens	0xAF68CFba29D0e15490236A5631cA9497e035CD39		Etherscan	GitHub

Goerli

Contract	Proxy Address	Etherscan (Proxy)	Etherscan (Implementation)	Source Code
Euler	0x931172BB95549d0f29e10ae2D079ABA3C63318B3		Etherscan	GitHub
Markets	0x3EbC39b84B1F856fAFE9803A9e1Eae7Da016Da36	Etherscan	Etherscan	GitHub
Liquidation	0x66326c072283feE63E1C3feF9BD024F8697EC1BB	Etherscan	Etherscan	GitHub
Exec	0x4b62EB6797526491eEf6eF36D3B9960E5d66C394	Etherscan	Etherscan	GitHub
Swap	0xA0AAb1Ddd165cE80AE2b9bC9bBE3b6EEFBB2300c	Etherscan	Etherscan	GitHub
Governance	0x496A8344497875D0D805167874f2f938aEa15600	Etherscan	Etherscan	GitHub
ERC20 Token Faucet	0x1215396CB53774dCE60978d7237F32042cF3a1db		Etherscan	GitHub
EulerSimpleLens	0x62626a0f051B547b3182e55D1Eba667138790D8D		Etherscan	GitHub

Note: the Governance smart contract is currently used for our risk-guarded launch (Phase 1 of the mainnet launch) while we build up to community-led governance in Phase 2.

Parameters

All of these parameters may be amended by governance

Borrow Factor

When someone activates a lending market on Euler for an asset XYZ, it is by default an isolated tier asset and its **borrow factor is 0.28**.

This means that if you lend USDC (Collateral Factor of 0.90) and borrow XYZ at Default Borrow Factor (Borrow Factor of 0.28), your effective factor is approximately 0.25 (0.90×0.28).

Consequently, when borrowing default assets, you are required to be at minimum 4x overcollateralised. The factor is even more conservative if you borrow against an asset with a lower collateral factor.

Conservative overcollateralisation is intended to prevent bad debts.

For example, suppose the borrowed asset XYZ sharply skyrockets in price, causing a user's health factor to dip below 1, which means he is subject to liquidation. For the liquidator to be incentivised to liquidate the user, he needs to receive an attractive liquidation bonus in terms of the violator's lent assets. If there are simply not enough lent assets to back the bonus, a liquidator will choose not to liquidate. This leads to bad debts.

Alternatively, someone could manipulate XYZ pricing lower to be able to borrow a lot more XYZ than normally possible. Eventually, the price gets arbitrated and normalises, which leads to the health factor dipping below 1. Since the position is heavily overcollateralised, there is still enough collateral backing the loan, and hence liquidators are incentivised to repay the debts.

For more info on collateral and borrow factors, check out our risk docs: <https://docs.euler.finance/risk-framework/collateral-and-borrow-factors>

Reserve Factor

The **default reserve factor is set at 23%**.

This means that for every \$1 of interest paid by borrowers on an XYZ asset, 23c is paid into the reserve pool of XYZ while the remaining 77c are paid to lenders of XYZ. These reserves may later be used to repay the bad debts that accrue in the pool.

While a higher reserve factor would in theory help accumulate more substantial reserves to backstop the Euler lending pools, at some point it would discourage lenders as they'll receive too little interest.

Alternatively, a reserve factor that's too low is a lost opportunity to build reserves and hence trust with lenders.

We think a reserve factor of 23% is a perfect balance between building reserves and encouraging lending, especially given our generous EUL distribution scheme that will run in the first few years of Euler protocol's existence.

Interest Rate Model

While we eventually plan to move to a reactive interest rate model that will optimise for utilisation, we start off with a standard kink model. The default kink model is the "small cap" model and its parameters are:

1. Base IR: **0%** (APY when utilisation is 0%)
2. Kink IR: **10%** (APY when utilisation is exactly Kink%)
3. Max IR: **300%** (APY when utilisation is 100%)
4. Kink%: **50%** (Percent utilisation where kink occurs)

Given the default assets and their respective utilisation can be extremely volatile, it's important that the lenders aren't constantly exposed to withdrawal risk.

To that end, should utilisation sharply rise beyond the Kink%, high Max IR makes borrowing too expensive to maintain and hence lowers the withdrawal risk

Pricing Parameters

Uniswap3 Pool Fee-Level

The default uniswap3 pool fee-level is the **first existing of 0.3%, 0.05%, 1%**.

In order to retrieve asset prices, Euler uses Uniswap3's Time-Weighted Average Price (TWAP) for the pair XYZ/WETH, where XYZ is the asset in question and WETH is the reference asset. Since Uniswap3 supports multiple pools for the same pair which differ by fee-level, which pool to actually query needs to be configured on a per-asset basis.

Although in theory pools should converge to similar prices due to arbitrage, this is not always the case. For example, when a pool has insufficient liquidity to be profitable for arbitrage bots. Additionally, the amount of liquidity has an impact on the cost of price manipulation.

In order to be promoted up a tier, a review of the configured pool fee must be conducted. In some cases, extra "full-range liquidity" must be added to increase the cost of price manipulation.

TWAP Length

The **default TWAP length is set at 30 minutes**.

Euler uses Uniswap v3 TWAPs as the pricing oracle for all assets and debts on Euler.

As explained in our [whitepaper](#), a TWAP is essentially a moving average of trades that occurred in a given Uniswap v3 pool. The intention for using TWAPs is that it makes price manipulations prohibitively expensive the longer the TWAP window is.

However, a TWAP that's too long causes a significant lag between last-traded price on Uniswap and the TWAP, leading to risks of bad debt.

For eg, a user deposits ETH as collateral and borrows XYZ token. Imagine that the user's XYZ debt swells to \$70 worth, and he's subject to liquidation. A liquidator would have to take on some of that XYZ debt and repay it.

However, recall that while the debt is priced in TWAP terms, a liquidator would need to buy XYZ on the market to repay that debt. If the market price of his debt is \$120, the liquidator would be buying high on the market to receive a smaller amount of TWAP-priced assets of the violator plus liquidation bonus.

Hence, when the TWAP - Market Price spread become significantly large due to the TWAP lag, a liquidation may be uneconomical.

Alternatively, a TWAP that's too short means it becomes a lot cheaper to manipulate asset prices. For example, one could artificially pump a collateral asset's price to be able to borrow a disproportionate amount of XYZ tokens and run away with them.

Check out this paper written by Michael Bentley on cost of attacking TWAP pricing: <https://github.com/euler-xyz/uni-v3-twap-manipulation/blob/master/cost-of-attack.pdf>

Additionally, have a look at this blog post by Seraphim on possible attacks involving oracle manipulation and how Euler is preventing them: <https://blog.euler.finance/risks-in-crypto-a-lending-protocol-perspective-376e19c1d01a>

Uniswap Observation Cardinality

The **minimum uniswap3 cardinality is 10**. When a market is activated, the cardinality of the uniswap3 pool that will be used for pricing is increased to this value, if it is currently below it.

In order to maintain the TWAP, each Uniswap3 pool needs to keep a historical record of accumulated prices at previous points in time. Each record is called an observation, and their number is called the observation cardinality. Unfortunately, reserving the storage for these records costs gas.

The larger the cardinality, the longer the possible TWAP window. If a swap is executed every block, the longest TWAP that is possible is the cardinality times the average block time over the last N blocks.

On Euler, if it is not possible to retrieve a TWAP of the configured length, then the oldest available price is used instead. This means that the protocol can always be interacted with, and prevents some types of attacks that aim to prevent liquidation. However, it also means that the cardinality is an important security parameter for a pool.

This low minimum value ensures that activating a market is not too expensive. However, since a larger cardinality is required to ensure a longer TWAP window, in order for an asset to be promoted to a higher tier, a larger value cardinality will be required, typically 144 or higher.

Liquidation Parameters

Target Health Factor

The **default target health factor is 1.25**.

When a user is in violation due to his risk-adjusted liabilities exceeding his risk-adjusted collateral, his health factor dips below 1. However, should a liquidator come in, he may only take enough debt and assets from the violator to shift his health factor to 1.25.

This feature is called soft liquidating [as described in the whitepaper](#), and it creates a much better borrowing experience than on other protocols where 50% of your debt is liquidated.

Nevertheless, had we set the target health factor to 1.00, we would have run into the risk of making liquidations uneconomic. Namely, the size of debt being repaid, and the consequent reward may be too small to incentivise a liquidator.

Similarly, in a volatile market, being restored to 1.00 means a user may quickly dip below 1.00 again and again, which implied higher gas fees for liquidators and ever-decreasing rewards.

We think 1.25 is a good trade-off between a good borrowing experience and well-incentivised liquidations.

Maximum Liquidation Discount

The **default maximum liquidation discount is set at 20%**.

When a user is subject to liquidation, some of his debt (dTokens) and assets (eTokens) are transferred to the liquidator, which leads to the health score shifting to 1.25.

However, to incentivise the liquidator to do this work, he receives the eTokens at a discount. Another way of thinking about the discount is receiving a bonus on top of the asset value.

Let's imagine a simple example without liquidation surcharges and boosters (explained in this page below): assume a user's health factor is 0.90. This implies a liquidation discount of 10% ($1 - 0.90$). Hence, for taking on \$100 worth of dTokens, a liquidator receives \$110 worth of eTokens.

If the discount is too low, a liquidator may be discouraged from conducting the liquidation. Alternatively, a discount that's too high means borrowers are giving away too much of their assets to liquidators. This also creates a risk of never going back to 1.25 health score, as every liquidation decreases the amount of assets a user has.

Hence, a liquidation discount has an upper ceiling of 20% to improve the borrowers' experience.

Liquidation Surcharge

The **default liquidation surcharge is set at 2%**.

Whenever a liquidator takes on someone's debt, they need to repay 2% more than originally taken from the violator. To compensate him for that, the violator pays the liquidator an additional 2% of his lent assets.

This is intended to do two things:

1. The surcharge accumulates in the reserve pool, and hence every liquidation makes the pool healthier. The more volatile an asset, the more often liquidations occur, and the bigger the pool is.
2. Discourage malicious self-liquidation strategies. Due to the surcharge, every self-liquidation ends up being net negative.

Liquidation Booster Ceiling and Slope

The **default liquidation booster ceiling and slope are set at 2.5% and 2x respectively.**

To incentivise liquidators to lend, they may be more competitive than their peers should they lend through Euler.

For example, if a user's health score is at 0.90 and value of risk-adjusted debt is \$1,000, the implied liquidation discount is 10% ($1 - \text{Health Score}$). However, due to slippage and gas fees, liquidators are only profitable at an 11% discount.

At the same time, a liquidator that supplied \$1,000 risk-adjusted collateral can be 2x more competitive than the rest of the market with a ceiling of 2.5%. Should he take on the user's debt at a 10% discount, he will actually receive a more generous 12.5% discount (2.5% booster + 10% liquidation discount).

How did we arrive at that number? The **total liquidation discount** is calculated the following way:

RASAL = Risk-adjusted supplied assets by the liquidator

RADV = Risk-adjusted debt by the violator

LiqSurcharge is explained above in this page and in this example is assumed to be 0 for simplicity's sake.

2x is the liquidation booster slope.

The Liquidation Booster is subject to a 2.5% ceiling.

The reasoning behind setting the ceiling at 2.5% is the same as the 20% maximum liquidation discount: preventing the borrowers from overpaying when being liquidated. The 2x slope, on the other hand, is intended to make the liquidators gradually more competitive the more assets they deposit, but not outrageously more competitive to avoid the creation of monopolies.

Interest Rates

Find information about interest rates and their parameters on Euler

Introduction

While we eventually plan to move to a reactive interest rate model that will optimise for utilisation, we start off with a standard kink model. The parameters of our kink model are:

1. Base IR (APY when utilisation is 0%)
2. Kink IR (APY when utilisation is exactly Kink%)
3. Max IR (APY when utilisation is 100%)
4. Kink% (Percent utilisation where kink occurs)

The main consideration is maintaining target utilisation (typically at Kink%).

Asset	Base IR %	Kink IR %	Max IR %	Kink %
1INCH	0.00	20.00	300.00	80.00
AAVE	0.00	20.00	300.00	80.00
AGEUR	0.00	4.00	100.00	80.00
ALCX	0.00	10.00	300.00	50.00
ANT	0.00	10.00	300.00	50.00
APE	0.00	10.00	300.00	50.00
AURA	0.00	10.00	300.00	50.00
AXS	0.00	20.00	300.00	80.00
BABL	0.00	10.00	300.00	50.00
BADGER	0.00	10.00	300.00	50.00
BAL	0.00	20.00	300.00	80.00
BANK	0.00	10.00	300.00	50.00
BaoUSD	0.00	10.00	300.00	50.00
BED	0.00	10.00	300.00	50.00
BRIGHT	0.00	10.00	300.00	50.00
BUSD	0.00	20.00	300.00	80.00
CADC	0.00	10.00	300.00	50.00
CARD	0.00	10.00	300.00	50.00
cbETH	0.00	8.00	200.00	80.00
cETH	0.00	10.00	300.00	50.00
CNV	0.00	10.00	300.00	50.00
COMP	0.00	20.00	300.00	80.00
CRV	0.00	20.00	300.00	80.00
CTX	0.00	10.00	300.00	50.00
CVX	0.00	20.00	300.00	80.00
CVXCRV	0.00	10.00	300.00	50.00
DAI	0.00	4.00	100.00	80.00

DFI	0.00	10.00	300.00	50.00
DFX	0.00	10.00	300.00	50.00
DPI	0.00	10.00	300.00	50.00
DPX	0.00	10.00	300.00	50.00
DYDX	0.00	10.00	300.00	50.00
ENJ	0.00	10.00	300.00	50.00
ENS	0.00	20.00	300.00	80.00
ETH2X-FLI	0.00	10.00	300.00	50.00
EUL	0.00	10.00	300.00	50.00
EXRD	0.00	10.00	300.00	50.00
FIAT	0.00	10.00	300.00	50.00
FLOAT	0.00	10.00	300.00	50.00
FLX	0.00	10.00	300.00	50.00
FNT	0.00	10.00	300.00	50.00
FPIS	0.00	10.00	300.00	50.00
FRAX	0.00	20.00	300.00	80.00
FTT	0.00	10.00	300.00	50.00
FXS	0.00	10.00	300.00	50.00
GAMMA	0.00	10.00	300.00	50.00
GFI	0.00	10.00	300.00	50.00
GOHM	0.00	10.00	300.00	50.00
GRT	0.00	10.00	300.00	50.00
GTC	0.00	10.00	300.00	50.00
GUSD	0.00	10.00	300.00	50.00
HMT	0.00	10.00	300.00	50.00
HOP	0.00	10.00	300.00	50.00
IDLE	0.00	10.00	300.00	50.00
ILV	0.00	10.00	300.00	50.00

IMX	0.00	10.00	300.00	50.00
INDEX	0.00	10.00	300.00	50.00
IPT	0.00	10.00	300.00	50.00
KP3R	0.00	10.00	300.00	50.00
LDO	0.00	20.00	300.00	80.00
LINK	0.00	20.00	300.00	80.00
LOOKS	0.00	10.00	300.00	50.00
LQTY	0.00	10.00	300.00	50.00
LRC	0.00	10.00	300.00	50.00
LUSD	0.00	4.00	100.00	80.00
LYXE	0.00	10.00	300.00	50.00
MATIC	0.00	20.00	300.00	80.00
MILADY	0.00	10.00	300.00	50.00
MIM	0.00	20.00	300.00	80.00
MKR	0.00	20.00	300.00	80.00
MPL	0.00	10.00	300.00	50.00
MTA	0.00	10.00	300.00	50.00
MVI	0.00	10.00	300.00	50.00
NEXO	0.00	10.00	300.00	50.00
NMR	0.00	10.00	300.00	50.00
OGN	0.00	10.00	300.00	50.00
OHM	5.00	20.00	300.00	80.00
OS	0.00	10.00	300.00	50.00
OSQTH	0.00	20.00	300.00	80.00
PERP	0.00	20.00	300.00	80.00
PUNK	0.00	10.00	300.00	50.00
QNT	0.00	10.00	300.00	50.00
QSP	0.00	10.00	300.00	50.00
RAD	0.00	10.00	300.00	50.00

RAI	0.00	20.00	300.00	80.00
RBN	0.00	20.00	300.00	80.00
RENDOGE	0.00	10.00	300.00	50.00
REQ	0.00	10.00	300.00	50.00
RETH	0.00	10.00	300.00	50.00
RLC	0.00	10.00	300.00	50.00
RNDR	0.00	10.00	300.00	50.00
RPL	0.00	10.00	300.00	50.00
SDL	0.00	10.00	300.00	50.00
SETH2	0.00	10.00	300.00	50.00
SHIB	0.00	20.00	300.00	80.00
SLP	0.00	10.00	300.00	50.00
SNX	0.00	20.00	300.00	80.00
SOCKS	0.00	10.00	300.00	50.00
SOS	0.00	10.00	300.00	50.00
SSV	0.00	10.00	300.00	50.00
stETH	*	4.00	100.00	80.00
STG	0.00	10.00	300.00	50.00
SUSD	0.00	10.00	300.00	50.00
SWISE	0.00	10.00	300.00	50.00
TCR	0.00	10.00	300.00	50.00
TON	0.00	10.00	300.00	50.00
TONCOIN	0.00	10.00	300.00	50.00
TORN	0.00	10.00	300.00	50.00
TRDL	0.00	10.00	300.00	50.00
TRYB	0.00	10.00	300.00	50.00
TSUKA	0.00	10.00	300.00	50.00
UBI	0.00	10.00	300.00	50.00

UNI	0.00	20.00	300.00	80.00
USDC	0.00	4.00	100.00	80.00
USDT	0.00	7.00	200.00	80.00
VEGA	0.00	10.00	300.00	50.00
WAMPL	0.00	10.00	300.00	50.00
WBTC	0.00	8.00	200.00	80.00
WETH	0.00	4.00	100.00	80.00
WILD	0.00	10.00	300.00	50.00
WNXM	0.00	10.00	300.00	50.00
WOO	0.00	10.00	300.00	50.00
WSTETH	0.00	8.00	200.00	80.00
XCN	0.00	10.00	300.00	50.00
XMON	0.00	10.00	300.00	50.00
YFI	0.00	20.00	300.00	80.00
YVBOOST	0.00	10.00	300.00	50.00

**stETH (current Lido staking APY)*

Risk Factors

Find information about the risk factors for each asset on Euler

Introduction

This page outlines the main risk parameters on Euler, as determined by [governance](#). All parameters are displayed in Table 1 below.

Table 1 | Collateral, borrow, and reserve factor parameter settings on Euler__

Token	collateral Factor	borrowFactor	reserveFactor	borrowIsolated	crossBorrow	InterestRateModel	Uniswap V3 fee tier (%)
RAD	0	0.28	0	true	false	Default	0.3
USDT	0.9	0.94	0.05	false	true	USDT	Chainlink
WSTETH	0.85	0.89	0.1	false	true	Mega	Chainlink
STETH	0.87	0.91	0.1	false	true	Lido	Chainlink
LUSD	0	0.7	0.15	false	true	Stable	Chainlink
FLX	0	2.5e-10	0.23	true	false	Default	0.3
ALCX	0	0.28	0.23	true	false	Default	Chainlink
ANT	0	0.28	0.23	true	false	Default	Chainlink
APE	0	0.28	0.23	true	false	Default	Chainlink
BABL	0	0.28	0.23	true	false	Default	0.3
BADGER	0	0.28	0.23	true	false	Default	Chainlink
BANK	0	0.28	0.23	true	false	Default	0.3
BED	0	0.28	0.23	true	false	Default	0.3
BRIGHT	0	0.28	0.23	true	false	Default	0.3
CARD	0	0.28	0.23	true	false	Default	0.3
CNV	0	0.28	0.23	true	false	Default	1
CTX	0	0.28	0.23	true	false	Default	1
CVXCRV	0	0.28	0.23	true	false	Default	0.3
DFI	0	0.28	0.23	true	false	Default	1
DPI	0	0.28	0.23	true	false	Default	Chainlink

DPX	0	0.28	0.23	true	false	Default	1
DYDX	0	0.28	0.23	true	false	Default	0.3
ENJ	0	0.28	0.23	true	false	Default	0.3
ETH2X- FLI	0	0.28	0.23	true	false	Default	0.3
EXRD	0	0.28	0.23	true	false	Default	1
FLOAT	0	0.28	0.23	true	false	Default	0.3
FNT	0	0.28	0.23	true	false	Default	1
FPIS	0	0.28	0.23	true	false	Default	1
FTT	0	0.28	0.23	true	false	Default	Chainli k
FXS	0	0.28	0.23	true	false	Default	Chainli k
GAMMA	0	0.28	0.23	true	false	Default	0.3
GFI	0	0.28	0.23	true	false	Default	1
GOHM	0	0.28	0.23	true	false	Default	0.3
GRT	0	0.28	0.23	true	false	Default	Chainli k
GTC	0	0.28	0.23	true	false	Default	Chainli k
GUSD	0	0.28	0.23	true	false	Default	Chainli k
IDLE	0	0.28	0.23	true	false	Default	0.3
ILV	0	0.28	0.23	true	false	Default	Chainli k
IMX	0	0.28	0.23	true	false	Default	0.3
INDEX	0	0.28	0.23	true	false	Default	1
KP3R	0	0.28	0.23	true	false	Default	1
LOOKS	0	0.28	0.23	true	false	Default	0.3
LQTY	0	0.28	0.23	true	false	Default	0.3
LRC	0	0.28	0.23	true	false	Default	Chainli k

LYXE	0	0.28	0.23	true	false	Default	0.3
MPL	0	0.28	0.23	true	false	Default	0.3
MTA	0	0.28	0.23	true	false	Default	0.3
MVI	0	0.28	0.23	true	false	Default	0.3
NEXO	0	0.28	0.23	true	false	Default	0.3
NMR	0	0.28	0.23	true	false	Default	Chainli k
OGN	0	0.28	0.23	true	false	Default	Chainli k
OS	0	0.28	0.23	true	false	Default	1
QNT	0	0.28	0.23	true	false	Default	0.3
QSP	0	0.28	0.23	true	false	Default	0.3
REND GE	0	0.28	0.23	true	false	Default	0.3
REQ	0	0.28	0.23	true	false	Default	Chainli k
RETH	0	0.28	0.23	true	false	Default	0.05
RLC	0	0.28	0.23	true	false	Default	Chainli k
RNDR	0	0.28	0.23	true	false	Default	1
RPL	0	0.28	0.23	true	false	Default	0.3
SETH2	0	0.28	0.23	true	false	Default	0.3
SLP	0	0.28	0.23	true	false	Default	0.3
SOCKS	0	0.28	0.23	true	false	Default	1
SOS	0	0.28	0.23	true	false	Default	1
SSV	0	0.28	0.23	true	false	Default	0.3
STG	0	0.28	0.23	true	false	Default	0.3
SUSD	0	0.28	0.23	true	false	Default	Chainli k
TCR	0	0.28	0.23	true	false	Default	0.3
TON	0	0.28	0.23	true	false	Default	1

TORN	0	0.28	0.23	true	false	Default	1
TRDL	0	0.28	0.23	true	false	Default	1
TRYB	0	0.28	0.23	true	false	Default	0.3
UBI	0	0.28	0.23	true	false	Default	1
VEGA	0	0.28	0.23	true	false	Default	0.3
WAMPL	0	0.28	0.23	true	false	Default	0.3
WILD	0	0.28	0.23	true	false	Default	0.3
WNXM	0	0.28	0.23	true	false	Default	Chainli k
WOO	0	0.28	0.23	true	false	Default	0.3
XCN	0	0.28	0.23	true	false	Default	Chainli k
XMON	0	0.28	0.23	true	false	Default	1
YVBOO ST	0	0.28	0.23	true	false	Default	1
AGEUR	0	0.5	0.23	false	true	Stable	0.05
CVX	0	0.5	0.23	false	true	Major	Chainli k
OHM	0	0.5	0.23	true	false	OHM	Chainli k
PERP	0	0.5	0.23	false	true	Major	Chainli k
RBN	0	0.5	0.23	false	true	Major	1
SHIB	0	0.5	0.23	false	true	Major	Chainli k
OSQTH	0	0.56	0.23	false	true	Major	0.3
AXS	0	0.66	0.23	false	true	Major	Chainli k
ENS	0	0.66	0.23	false	true	Major	Chainli k
MATIC	0	0.66	0.23	false	true	Major	Chainli k

MKR	0	0.66	0.23	false	true	Major	Chainli k
1INCH	0	0.7	0.23	true	false	Major	Chainli k
AAVE	0	0.7	0.23	true	false	Major	Chainli k
BAL	0	0.7	0.23	true	false	Major	Chainli k
BUSD	0	0.7	0.23	true	false	Major	Chainli k
COMP	0	0.7	0.23	true	false	Major	Chainli k
CRV	0	0.7	0.23	true	false	Major	Chainli k
FRAX	0	0.7	0.23	true	false	Major	Chainli k
LDO	0	0.7	0.23	true	false	Major	Chainli k
MIM	0	0.7	0.23	true	false	Major	Chainli k
RAI	0	0.7	0.23	true	false	Major	Chainli k
SNX	0	0.7	0.23	true	false	Major	Chainli k
YFI	0	0.7	0.23	true	false	Major	Chainli k
LINK	0	0.76	0.23	false	true	Major	Chainli k
UNI	0	0.76	0.23	false	true	Major	Chainli k
DAI	0.85	0.88	0.23	false	true	Stable	Chainli k
WBTC	0.88	0.91	0.23	false	true	Mega	Chainli k
WETH	0.88	0.91	0.23	false	true	Default	Peggec
USDC	0.9	0.94	0.23	false	true	Stable	Chainli

Note: the Collateral Factor of the lent asset(s) is multiplied by the Borrow Factor of the borrowed asset(s) to arrive at the final factor.

For example, if you lend 1,000 USD worth of USDC, you can borrow UNI in line with a final factor of 0.648 (0.90 x 0.72). Hence, 648 USD worth of UNI.

Alternatively, if you lend 500 USD worth of USDC and 500 USD worth of WETH, your risk-adjusted collateral value is $(500 \times 0.90) + (500 \times 0.88) = 890$ USD. If you were to borrow UNI, you could borrow $890 \times 0.72 = 640.8$ USD worth of UNI.

Note that if you borrowed less UNI, for example 500 USD worth, you could still borrow additional UNI or a cross tier asset like LINK against your risk-adjusted collateral before hitting the threshold.

Lastly, please note that the risk factors list will be periodically updated. If a token/market is activated on the DApp but not listed, please check back later for an updated list.

Euler Governance

Getting Started

Introduction

The code for Euler Protocol is controlled by a decentralised community through on-chain governance on the Ethereum network. The community are holders of a protocol-native governance token called [EUL](#) (pronounced 'oil'), which enables the community to effect change over the Euler Protocol code. Tokens can be used to propose upgrades to the protocol or vote on the proposals of others.

Protocol Code

Governance can vote to effect change over the Euler Protocol for parameters such as:

1. **Default isolated tier borrow factor**
2. **Collateral and borrow factors** of specific assets
3. Inclusion of an asset in the **cross** and **collateral tiers**
4. Change in choice of **risk parameters** and general **methodology**

On-Chain governance, allows unique features such as delegated voting and proposition powers, as well as protocol (and governance configuration) updates via a time lock executor. This ensures the protocol can adapt to evolving market conditions, as well as upgrade core parts of the protocol over time.

On the other hand, for off-chain governance, there is no code to review or implement as such. It is mainly a call for the Euler Foundation to carry out an action. Issue a grant, or pay a bill, for example. Thus mainly used to aid Euler in making difficult decisions in collaboration with the community.

Euler protocol uses the [Tally](#) governance interface for on-chain voting. [Snapshot](#) on the other hand is an interface use for off-chain or 'gasless' voting.

Process

The [General Governance Process](#) is documented on the Governance Forum.

The flow of the governance process is as follows:

1. Discuss the idea/draft proposal in the [Euler Discord #governance](#) channel
2. Draft & create a [RFC \(Request For Comment\) Proposal](#) on Governance forum for further feedbacks
3. Contact a forum moderator to create a [eIP \(Euler Improvement Proposal\)](#) on the Governance forum
4. eIP created on [Snapshot](#) (off-chain voting)
This is a necessary step for all types of proposals, and execution will be carried out by Euler

Foundation if the proposal is successful.

5. *Optional step — eIP created on [Tally](#) (on-chain voting)*
If and only if the proposal includes changes to the smart contract, the proposal will be voted on Tally after Snapshot voting. Execution will be targeting Euler protocol smart contract if the proposal is successful.

It is noteworthy that not all off-chain proposals that are either binding or non-binding on the protocols smart contracts will end up having an on-chain proposal depending on the outcome of the off-chain 'gas-less' voting and for gas cost savings. On the other hand, not all on-chain proposals will require an off-chain counterpart.

Depending on the outcome of an off-chain voting process, an on-chain proposal might be created which will be executed against a target protocol smart contract if successful.

If an off-chain proposal requires an on-chain proposal that will be executed against a protocol smart contract, then the general flow could be as follows:

Creation of a formal Idea/Proposal on Governance forum for discussion → Proposal created on Snapshot (off-chain proposal creation) → Off-Chain Voting → (if on-chain governance is required) eIP (Euler Improvement Proposal) creation on Governance forum by forum moderator → eIP created on Tally (on-chain proposal creation) → On-Chain Voting (and Execution on target Euler protocol smart contract if successful).

Idea

A great place to start a discussion on a potential governance proposal is the idea section on the forum website. If you feel confident that your idea is relevant to the community and is well-formulated, head over to the Governance Forum to begin a discussion with the community around your idea (following the process described on the [forum](#)).

Once a discussion / commenting begins around your idea, be proactive with the community and be open to suggestions. It typically takes a week for the request for comments to mature before it becomes an eIP.

Governance Proposal

If the discussion is well-formulated and the community has a clear understanding of the proposal and supports your idea, (for on-chain proposals) it will be moved by a moderator to the governance category as an eIP: Euler Improvement Proposal. Once the proposal has an eIP, an on or off-chain proposal can be created on the [Tally governance dashboard](#) or on the [Snapshot governance dashboard](#).

A Tally or Snapshot proposal does not always need to be created by the original eIP author / proposer, it can be posted by someone else or by one of the delegates in case the minimum threshold of EUL is not being met.

A good governance proposal example can be found here: [eIP: Promote WBTC to collateral tier 3](#).

Stay updated by subscribing to the [community newsletter](#) and follow the [Twitter Page](#)!

Phases

Learn more about the Euler Governance launch phases

Governance Launch Phases

Introduction

EulerDAO will kick off in three phases for a guarded launch towards full decentralisation of the Euler protocol. Each phase is described below.

The EulerDAO uses the [OpenZeppelin Governance smart contracts](#) (version 4.6.0) for governance (as well as the EUL token contract). It is a governance protocol — similar to the one Compound uses — where delegates vote on active proposals to make changes to the EulerDAO and Euler protocol.

Euler uses the Tally governance dashboard application to manage the governance process. Through Tally, you can set up your wallet to become a delegate, create on-chain proposals, vote on active proposals, discover delegates in the community, and delegate your voting power to a community member.

Phase 1

The first phase of the launch will be such that actions to be performed directly on the Euler protocol smart contracts will be executed by the Euler team on behalf of the community. In this case, all on-chain governance proposals will point to or target the `executeProposal(string description, bytes proposalData)` function in a [stub governance smart contract](#) (in place of the Euler protocol smart contracts).

Should the proposal become successful and executed, the target function will then be executed (via the `TimelockController` controller smart contract). Once executed, it will emit the proposal description string and proposal transaction data, which will then be validated by the Euler team and executed against the `Exec` module via the Euler multisig.

Hence, the `governorOnly()` modifier in the Euler Governance module smart contract will be checking that the caller of its functions is the Euler multisig and not the `TimelockController` smart contract.

To create the proposal transaction data, we have implemented a [tool](#) which will help the proposer to auto generate this depending on what actions should be executed.

The proposer can select a token from the dropdown token list (this will auto populate the fields with the current configuration for the token/market on Euler), the proposer can then make modifications and generate the proposal transaction hex to be executed via the Euler `Exec` module (`batchDispatch()` function) and use this hex data as the input to the target function in the [stub governance smart contract](#) when creating a proposal on Tally.

Examples of the kinds of decisions token holders might vote on include proposals to modify:

- The tier of an asset
- Collateral and borrow factors

- Price oracle parameters
- Reactive interest rate model parameters
- Reserve factors

Phase 2

In the second phase, the governor admin for the Euler Governance module will switch from the Euler multisig to the TimelockController smart contract. Hence, the `governorOnly()` modifier in the Euler Governance module smart contract will be checking that the caller of its functions is the TimelockController smart contract.

In this case, proposals created via the Tally governance dashboard will need to target the Euler [Governance module](#) directly. Successful proposals which receive a higher number of votes in support will then be executed without the control of the Euler team.

During execution, the TimelockController smart contract will call the target function in the Euler Governance module as specified within the on-chain proposal. Proposers will be able to add a batch of actions / functions to be executed within the Euler Governance module into a single on-chain proposal as well.

Phase 3

In this phase, the Installer Admin will also be switched to the TimelockController smart contract.

Not only giving the community control over the Governance module for asset configuration modifications but also full control over the protocol including the installer module. This module is used to bootstrap install the rest of the modules, and can later on be used to upgrade modules to add new features and/or fix bugs.

Euler's Governance Proposal Creation Tool for Phase 1

Introduction

The first phase of Euler's DAO / Governance launch will be where actions to be performed directly on the Euler protocol smart contracts will be performed or executed by the Euler team on behalf of the community. In this case, all on-chain governance proposals will point to or target a function in a [stub governance smart contract](#) (in place of the Euler protocol smart contracts).

Should the proposal become successful and executed, the target function will then be executed (via the TimelockController controller smart contract. It will emit the proposal description string and proposal transaction hex data, which will then be validated by the Euler team and executed against the `Exec` module via the Euler multisig (on OpenZeppelin Defender).

Hence, the `governorOnly()` modifier in the [Euler Governance module](#) will be checking that the caller of its functions is the Euler multisig and not the TimelockController smart contract of the DAO.

To create the proposal transaction data, we have implemented a [tool](#) which will help the proposal creator to auto-generate it depending on what actions should be executed. The proposal transaction data is simply an encoded version of the on-chain transaction to the Exec module's batchDispatch function.

The rest of the article will describe the process of creating the proposal transaction data on Euler's governance proposal creation tool and using the generated proposal transaction data to create an on-chain proposal on the [Tally](#) governance dashboard.

Section 1. Creating the proposal transaction data for Euler's Exec Module

Step 1

Navigate to the [proposal transaction data creation tool](#).

The web application should look like the following image:

The screenshot shows a web browser window with the URL `governance.euler.finance`. The interface is a form for creating proposal transaction data. On the left side, there are several input fields and dropdown menus:

- Proposal Description: (empty text box)
- Token: USDC (dropdown)
- Fee: 0.3% (dropdown)
- Pricing Type: UNISWAP_3_TWAP (dropdown)
- Borrow Isolated: false (dropdown)
- Collateral Factor: 0.9 (text box)
- Borrow Factor: 0.94 (text box)
- TWAP Window: 1800 (text box)
- Reserve Fee: 0.23 (text box)
- Interest Rate Model: STABLE (dropdown)
- Recipient: 0x0 (text box)
- Recipient Amount: 0 (text box)

At the top right, it says "Token Pool Connected Network: MAINNET". Below that, "Proposal Description:" is followed by "Batch Items: []". A table with columns "Contract", "Method", "Token", "Token Name", "Token Address", and "Updates" is present but empty. There are two blue buttons: "RESET/CLEAR" and "DEBUG TX HEX DATA". Below these are two more blue buttons: "CREATE PROPOSAL DATA" and "CREATE PROPOSAL ON DEFENDER". To the right of these buttons is a "Defender Key" text box.

Step 2

The tool requires MetaMask to be installed in your browser. Switch your MetaMask wallet to mainnet.

The tool currently supports the Ethereum Mainnet and our Goerli testnet Exec modules. It will create the appropriate transaction data to be executed depending on the selected network.

Step 3

At the top left of the window, we have a text field for proposal description and below that we have a dropdown menu representing a list of tokens, e.g., USDC, DAI, etc.

At the left of the window, under the token list, we have a set of dropdown menus and text fields which will be autopopulated with the current token configurations (if the token has an activated market on Euler).

To create the proposal transaction data, the proposer needs to enter a proposal description and then select a token from the token list. Once selected, the rest of the fields will be automatically populated with the current configuration of the token or market on Euler if it has an activated market.

The image below shows the proposal description and the fields on the left populated with the current market configuration for USDC on Euler:

Proposal Description: A proposal to change th

Token: USDC

Fee: 0.3%

Pricing Type: UNISWAP_3_TWAP

Borrow Isolated: false

Collateral Factor: 0.9

Borrow Factor: 0.94

TWAP Window: 1800

Reserve Fee: 0.23

Interest Rate Model: STABLE

Recipient: 0x0

Recipient Amount: 0

Token Pool Connected Network: MAINNET

Proposal Description: A proposal to change the collateral factor of USDC to 0.5

Batch Items: []

Contract	Method	Token	Token Name	Token Address	Updates
----------	--------	-------	------------	---------------	---------

Buttons: RESET/CLEAR, DEBUG TX HEX DATA, CREATE PROPOSAL DATA, CREATE PROPOSAL ON DEFENDER

Defender Key: [Empty field]

Step 4

As shown in the image above, the hex data is decoded to show the updates we selected to be applied to DAI and USDC. There is a close button at the bottom right of the window to close the debug modal and return to the main page.

Once you get to this point, you now have your proposal transaction data which you can use to create your on-chain proposal on Tally. After the voting period, if successfully executed, the decoded hex data will be submitted to OpenZeppelin Defender for the Euler team to execute on behalf of the community.

Please let us know if you have any questions or feedback while using the tool in our Discord Governance channel.

Section 2. Using the Auto-Generated transaction data for Euler's Exec Module to Create an on-chain proposal for the DAO on Tally

At this point, we assume you now have the proposal transaction hex data needed for the Euler Exec module's batchDispatch function. And you want to create the on-chain proposal for members of the community (and delegates) to vote on. If so, please read on!

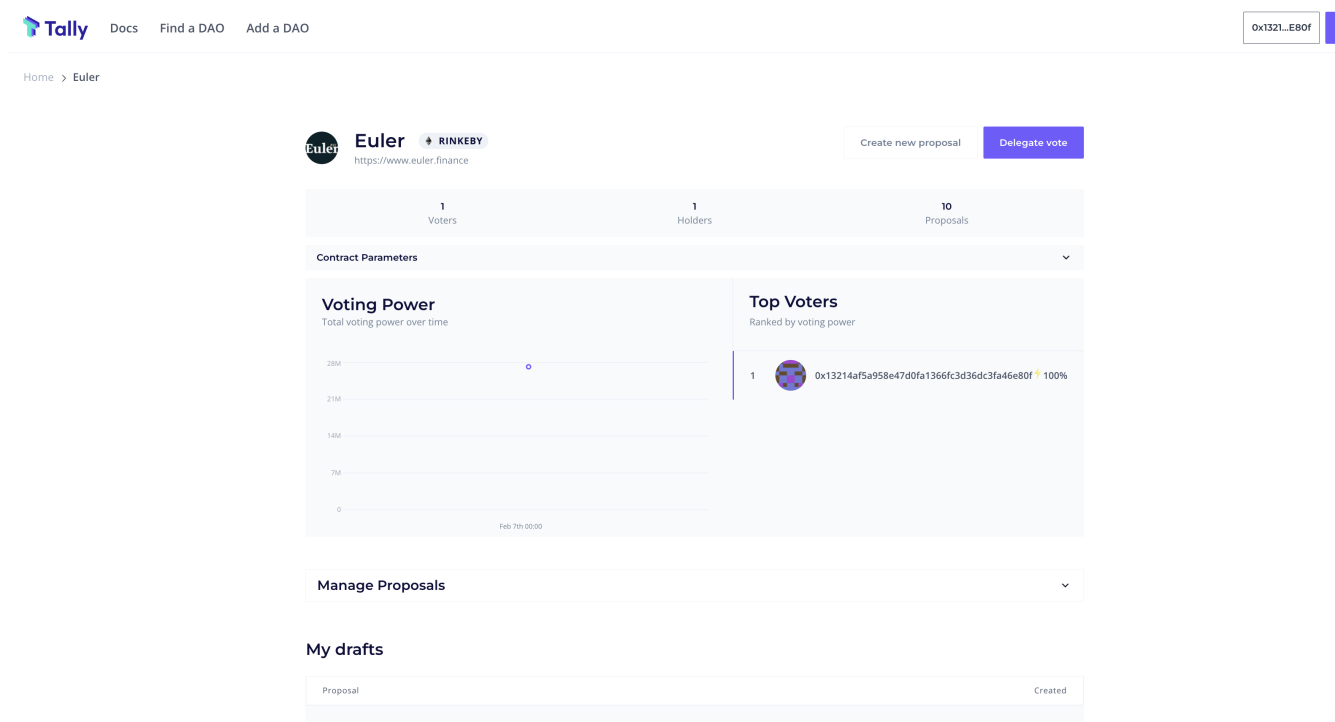
Below, we will describe the steps required to accomplish this goal.

Step 1

Head over to the EulerDAO dashboard on Tally and connect your MetaMask wallet.

Step 2

Click on **Create new proposal** at the right corner of the window.



The screenshot shows the EulerDAO dashboard on Tally. At the top left, there is a navigation bar with the Tally logo and links for Docs, Find a DAO, and Add a DAO. On the right, there is a user profile for 0x1321...E80f. The main content area displays the EulerDAO dashboard for the Euler project (https://www.euler.finance). It features a header with 'Euler' and 'RINKEBY' network information, and buttons for 'Create new proposal' and 'Delegate vote'. Below this, there are statistics: 1 Voter, 1 Holder, and 10 Proposals. The dashboard is divided into several sections: 'Contract Parameters', 'Voting Power' (Total voting power over time, with a line graph showing a single data point at 2884 on Feb 7th, 00:00), 'Top Voters' (Ranked by voting power, showing one voter with 100% power), 'Manage Proposals', and 'My drafts' (showing a table with columns for Proposal and Created).

It should then take you to the proposal creation window below.

Create a new proposal



✓ Connect your wallet

- ✓ Wallet connected
- ✓ Correct chain selected
- ✓ You have 27.18M voting power. You've reached the proposal threshold! 🎉

Switch wallet

Continue

2 Name your proposal

3 Add actions

4 Preview your proposal



Click on `Continue` to move onto the next step (`Name your proposal`).

Step 3

Enter a proposal title and description and click `Continue` .

✓ Name your proposal

Give your proposal a title and a description. They will be public when your proposal goes live!

Title

A proposal to update the collateral factor of USDC to 0.4

Description

[Write](#) [Preview](#)

B

I

H1

H2

H3

↔

☰

☰

🖼️

</>

“

A proposal to update the collateral factor of USDC to 0.4

Preview image (optional)

Drag and drop your preview image PNG file

Or click to browse your files

Back

Continue

Step 4

In the next section, you will be required to specify the governance proposal actions, i.e., target smart contract, target function and parameters. The Tally dashboard allows you to specify multiple actions in a single proposal which will be called/executed if the proposal is successful and executed.

✓ Connect your wallet

✓ Name your proposal

3 Add actions

Add up to 10 actions to be executed if the proposal passes.

Action #1

Transfer tokens

Add custom action

Add action

Remove action

Back

Continue

4 Preview your proposal

To add the proposal transaction hex from Section 1 and set the governance [stub governance smart contract](#) as the target smart contract, we will click on `Add custom action` => enter the [stub governance smart contract](#) address as the target smart contract. Then select the `executeProposal` function from the dropdown menu under `contract method` as the target function in the target smart contract. Here is the interesting part: enter the required parameters, i.e., proposal description string and the `proposalData` which is your proposal transaction hex data from Section 1.



Preview your proposal

Hey there! 🙌

You have completed all steps successfully. Now is the time to review your proposal and submit it.

Edit

Update collateral factor of USDC to 0.4 and DAI borrow factor to 0.4

Proposed by: 0x1321...E80f

Description

A proposal to update collateral factor of USDC to 0.4 and DAI borrow factor to 0.4.

Edit

Actions

Function 1:

```
Signature:
executeProposal(string,bytes)

Calldata:
string A proposal to update USDC collateral factor to 0.5 and DAI borrow factor to
: 0.4
0xeb937aeb0000000000000000000000000000000000000000000000000000000000000000004000000000
00000000000000000000000000000000000000000000000000000000000000000001e000000000000000000000
000000000000000000000000000000000000000000000000010000000000000000000000000000000000000
0000000000000000000000002000000000000000000000000000000000000000000000000000000000000
t 00000000000000000000000000000078ee171d6c8d3808b72dab8ce647719db3bb4cc9000000000000
} 0000000000000000000000000000000000000000000000000000006000000000000000000000000000000
1 00000000000000000000000000000000000000000000c4b533461c000000000000000000000095689faeed669
€ 1757df1ad48b7bea1b8acf2dabe0000000000000000000000000000638ecfe7857f83160615d50a905bfc
: b40662efc10000000000000000000000000000000000000000000000000000000000000000000000000000000
: 00000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000d693a4000000000000000000000000000000000000000000000
0000000000000000007080000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000

Target:
0xbEcee0E5989e5B7af3F42C5380f32AD7061cfa9D

Value:
0
```


How To

Use the navigation bar on the left side to find guides for all the primary actions and functions required for governance.

Delegate Voting Power

If you wish to have a say in governance, you need to delegate your voting power to yourself or someone in the community. Without performing this action, you will not have any voting power which means being unable to create proposals or vote on Snapshot (off-chain) and Tally (on-chain).

About

In summary, delegates are token holders that have completed a one-time setup process (executing the delegate function of the token to delegate another user or the token holder themselves to enable the governor contract to determine their voting power). Once you become a delegate, you can vote on active proposals, and create proposals if you have enough voting power. If you choose not to directly vote on proposals, you can pass your voting power on to a delegate as well.

The delegate sections below describe the delegation using the EUL token smart contract and via the Tally Governance Dashboard.

Delegate votes from the sender to a delegatee. Users can delegate to 1 address at a time, and the number of votes added to the delegatee's vote count is equivalent to the balance of EUL in the user's account. Votes are delegated from the current block and onward, until the sender delegates again, or transfers their EUL. Delegation can be carried out via the smart contract function described below or via the Tally user interface.

Step-by-step

Voting power delegation can be done via Etherscan, Tally (On-Chain) Governance dashboard or programmatically.

Etherscan

1. Visit the [EUL token page](#) on Etherscan (shown below).

Contract 0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b

Buy Exchange Earn Gaming

Contract Overview

Euler: EUL Token

Balance: 0 Ether
 Ether Value: \$0.00

More Info

More

My Name Tag: Not Available, login to update
 Contract Creator: Euler: Deployer at txn 0x6c1246bb6e35b165a8...
 Token Tracker: Euler (EUL) (@\$7.5069)

Transactions Internal Txns Erc20 Token Txns Contract Events Analytics Info Comments

Latest 25 from a total of 7,571 transactions (+1 Pending)

Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0x12eeaccb27af072616...	Transfer	(pending)	23 hrs 44 mins ago	0x448fda59cda7c93d8e...	Euler: EUL Token	0 Ether	(Pending)
0x1c1b74067efdd1ee3f7...	Transfer	15452228	2 hrs 6 mins ago	0x3bf5dc82ab5cd08aa8...	Euler: EUL Token	0 Ether	0.00066007
0x0714d8ad23eb1b0945...	Transfer	15452214	2 hrs 10 mins ago	0x30741289523c2e4d2a...	Euler: EUL Token	0 Ether	0.00171312
0x16a8ebd07d59f028aa...	Transfer	15451388	5 hrs 22 mins ago	0xf6890e3114ebc79f56f...	Euler: EUL Token	0 Ether	0.00043857
0x34d4d5b8380e9b419...	Transfer	15451362	5 hrs 27 mins ago	qq55386322.eth	Euler: EUL Token	0 Ether	0.00038417
0x8379560cfe9c89eb82...	Transfer	15451362	5 hrs 27 mins ago	0x59141af2478701c6e3...	Euler: EUL Token	0 Ether	0.00043581

2. Click on **Contract** (shown below) to view the EUL token smart contract code and interact with the contract via etherscan.

Contract 0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b

Buy Exchange Earn Gaming

Contract Overview

Euler: EUL Token

Balance: 0 Ether
 Ether Value: \$0.00

More Info

More

My Name Tag: Not Available, login to update
 Contract Creator: Euler: Deployer at txn 0x6c1246bb6e35b165a8...
 Token Tracker: Euler (EUL) (@\$7.5069)

Transactions Internal Txns Erc20 Token Txns **Contract** Events Analytics Info Comments

Code Read Contract Write Contract

Search Source Code

3. Click on **Write Contract** (shown below).

Contract 0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b

Buy Exchange Earn Gaming

Contract Overview

Euler: EUL Token

Balance: 0 Ether
Ether Value: \$0.00

More Info

More

My Name Tag: Not Available, login to update
Contract Creator: Euler: Deployer at txn 0x6c1246bb6e35b165a8...
Token Tracker: Euler (EUL) (@\$7.5069)

Transactions Internal Txns Erc20 Token Txns Contract Events Analytics Info Comments

Code Read Contract Write Contract

Connect to Web3

[Expand all] [Reset]

Descriptions included below are taken from the contract source code NatSpec. Etherscan does not provide any guarantees on their safety or accuracy.

- 1. approve
- 2. decreaseAllowance
- 3. delegate

4. Click **Connect to Web3** to connect your Metamask wallet which will be used to confirm the delegate transaction. Once connected, you should see your wallet connect on etherscan as shown below.

Contract 0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b

Buy Exchange Earn Gaming

Contract Overview

Euler: EUL Token

Balance: 0 Ether
Ether Value: \$0.00

More Info

More

My Name Tag: Not Available, login to update
Contract Creator: Euler: Deployer at txn 0x6c1246bb6e35b165a8...
Token Tracker: Euler (EUL) (@\$7.5069)

Transactions Internal Txns Erc20 Token Txns Contract Events Analytics Info Comments

Code Read Contract Write Contract

Connected - Web3 [0x7039...7007]

[Expand all] [Reset]

5. Click on **3. delegate** to expand the delegatee address input text field.

Contract 0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b

Token Contract

Buy

Exchange

Earn

Gaming

Contract Overview

Euler: EUL Token

Balance: 0 Ether

Ether Value: \$0.00

More Info

My Name Tag: Not Available, [login to update](#)

Contract Creator: Euler: Deployer at txn 0x6c1246bb6e35b165a8...

Token Tracker:  Euler (EUL) (@\$7.5069)Transactions Internal Txns Erc20 Token Txns **Contract** Events Analytics Info Comments

Code Read Contract Write Contract

Connected - Web3 [0x7039...7007]

[\[Expand all\]](#) [\[Reset\]](#)Descriptions included below are taken from the contract source code [NatSpec](#). Etherscan does not provide any guarantees on their safety or accuracy.

1. approve

2. decreaseAllowance

3. delegate

Delegate votes from the sender to 'delegatee'.

delegatee (address)

delegatee (address)

Write

6. Enter the address you wish to delegate your voting power to. This can be your wallet address if you are self-delegating or another wallet address (i.e., a community member or one of the active delegates on the [delegates list](#)).

7. Click on the blue `Write` button directly under the delegatee address text field.

8. Finally, regardless of whether you are delegating to yourself or delegating to a delegate, you will be required to confirm the transaction in your Metamask wallet and this transaction will cost gas.

The screenshot displays the Etherscan.io interface for the Euler: EUL Token contract. The top navigation bar includes the Etherscan logo, search filters, and account information. The main content area is divided into several sections:

- Contract Overview:** Shows the contract address (0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b) and its balance (0 Ether) with an ether value of \$0.00.
- More Info:** Provides details about the contract creator (Euler: Deployer at 0xb6c1248bbe35b165a8...) and the token tracker (Euler (EUL) (@\$7.5069)).
- Contract Tab:** Features tabs for Code, Read Contract, and Write Contract. The Read Contract tab is active, showing a list of functions: approve, decreaseAllowance, and delegate. The delegate function is selected, and the delegatee address is 0x914E50278B8613D243b848CCb47e611F1b1C3C.

A MetaMask notification is overlaid on the right side of the screen, indicating a new address detected and providing options to add it to the address book or confirm a transaction.

To check the address you are currently delegated to, you can click on the **Read Contract** tab next to the **Write Contract** tab and you will be presented with the window below.












The screenshot shows the Etherscan interface for the Euler (EUL) token contract. The contract address is 0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b. The page includes a 'Contract Overview' section with a balance of 0 Ether and an ether value of \$0.00. A 'More Info' section shows the contract creator as Euler: Deployer and the token tracker as Euler (EUL) with a price of \$7.3482. Below this is an advertisement for MAX offering a 750 USDT reward. The main content area shows a list of contract variables, with '10. delegates' selected. Below this variable, there is a text input field for an account address and a 'Query' button.

Click on `10. delegates` (shown below) and paste your address in the text field and click on `Query` and it should show the address you have set as a delegate. If it shows the zero address, (i.e., `0x00`) then it implies you have not delegated to your wallet or to another address.

You can also check your current voting power.







To do this, click on `14. getVotes` (shown below) and paste your address in the text field and click on `Query`. If you have self-delegated, your voting power should be equal to the number of tokens you hold. If you have not self-delegated, or if you have delegated to another address, the query will return zero voting power.

Code **Read Contract** Write Contract

- ADMIN_ROLE  
- DEFAULT_ADMIN_ROLE  
- DOMAIN_SEPARATOR  
- MINT_MAX_PERCENT  
- MINT_MIN_INTERVAL  
- allowance  
- balanceOf  
- checkpoints  
- decimals  
- delegates  

account (address)

Query

↳ address
- getPastTotalSupply  
- getPastVotes  
- getRoleAdmin  
- getVotes  

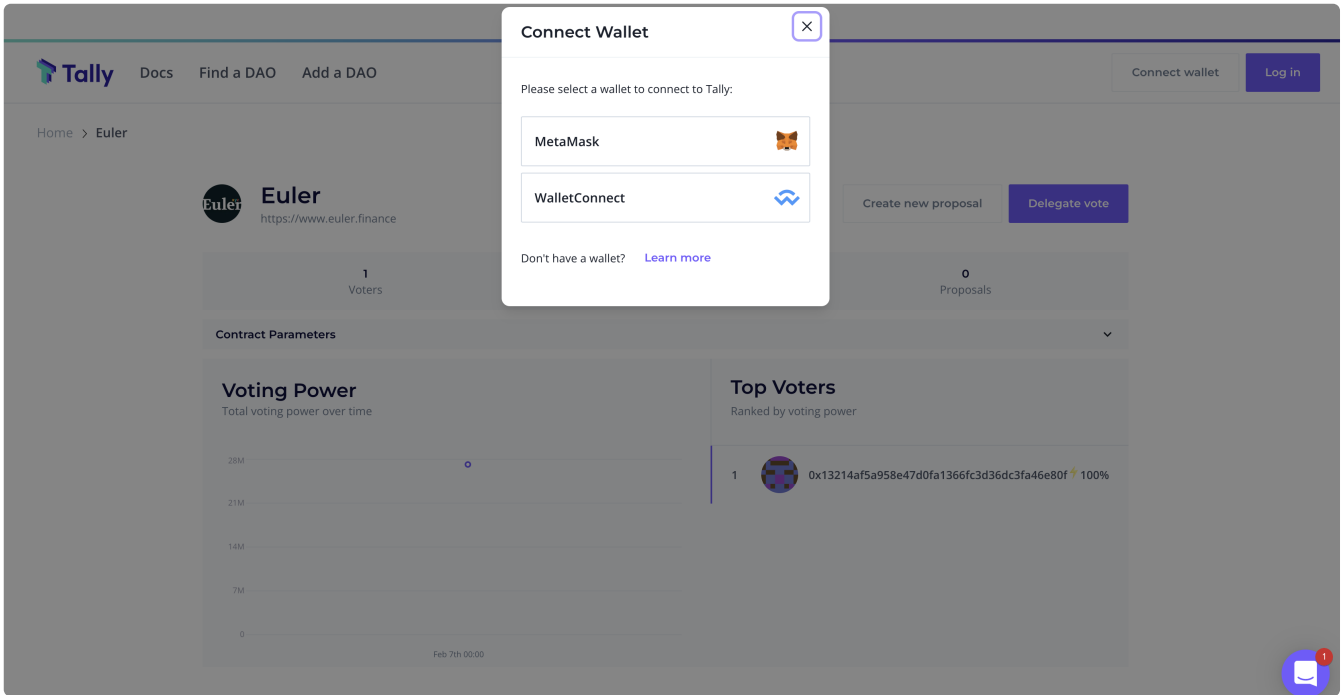
account (address)

Query

↳ uint256

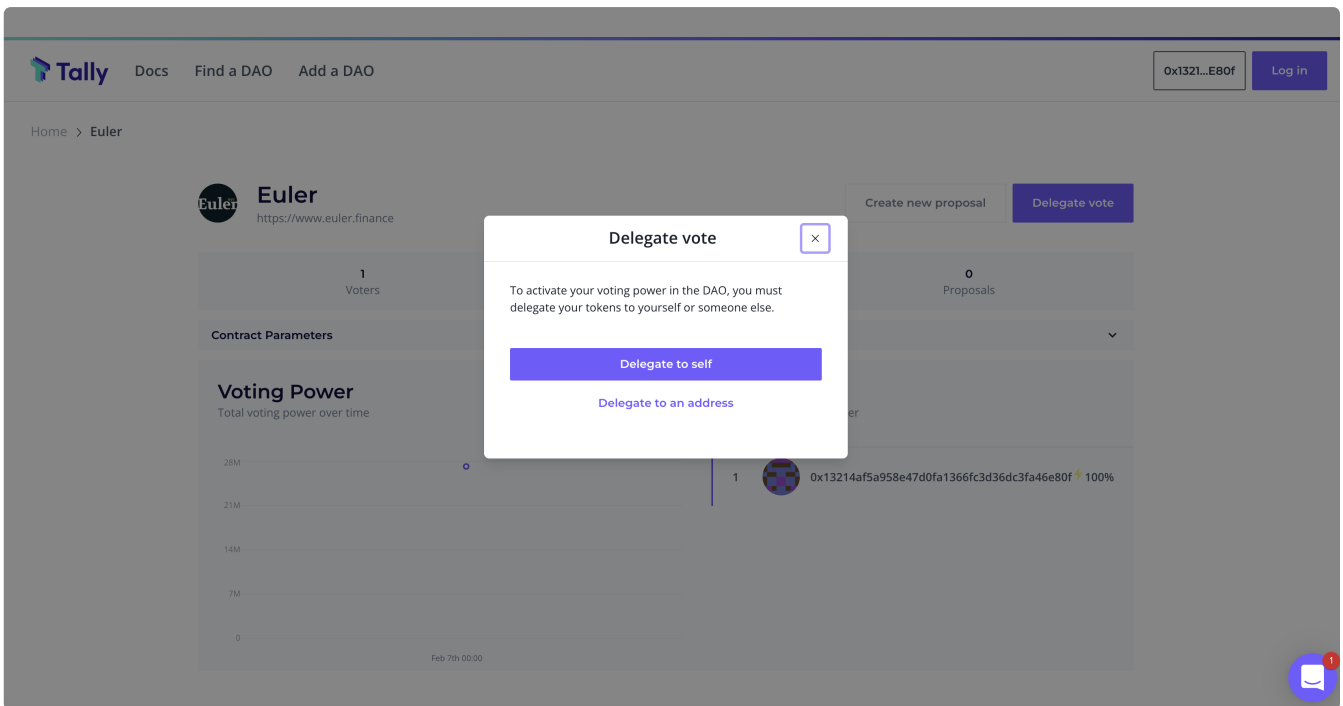
Tally (On-Chain) Governance Dashboard

1. Visit the [Euler on-chain governance dashboard](#) on Tally and connect your wallet where you hold EUL tokens.

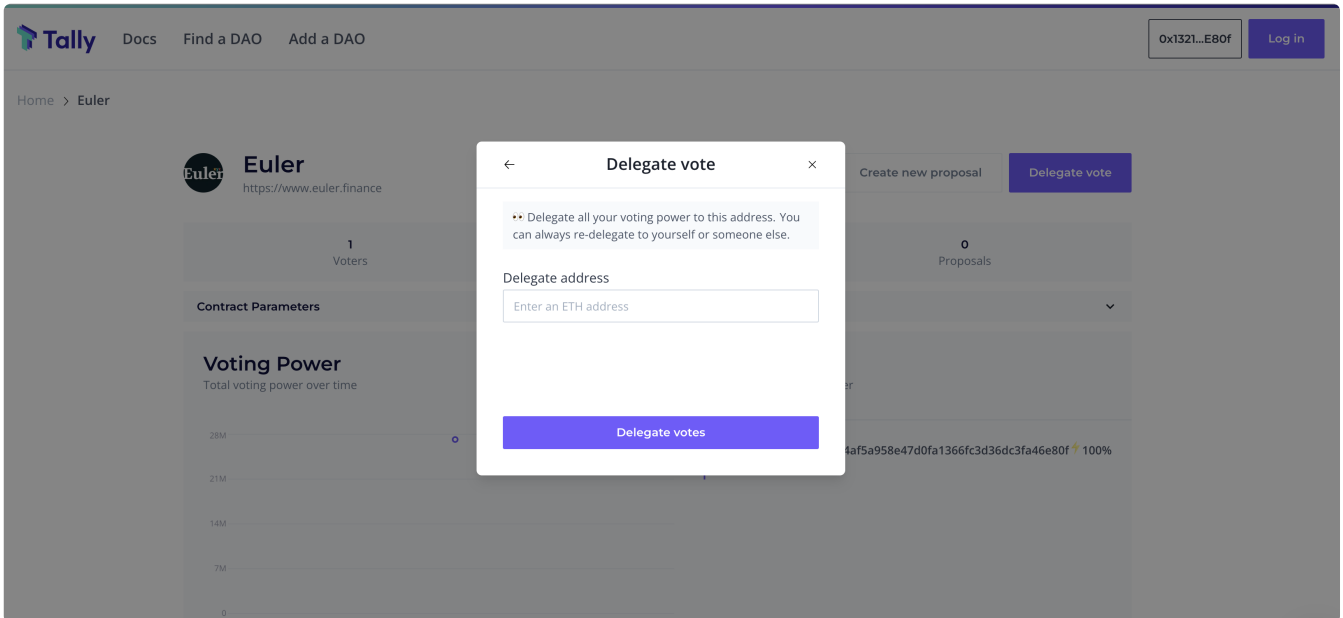


2. Click on **Delegate vote** at the top right corner of the screen.

3. A pop-up window will appear with two options as shown below, i.e., Delegate to self or Delegate to an address. You can then choose on of these options from the pop up window, either to delegate to yourself or to another wallet address (i.e., a community member or one of the active delegates on the [delegates list](#)). By delegating to self, you retain your voting power. Next time there is an active proposal, you can choose to vote in any way you choose.



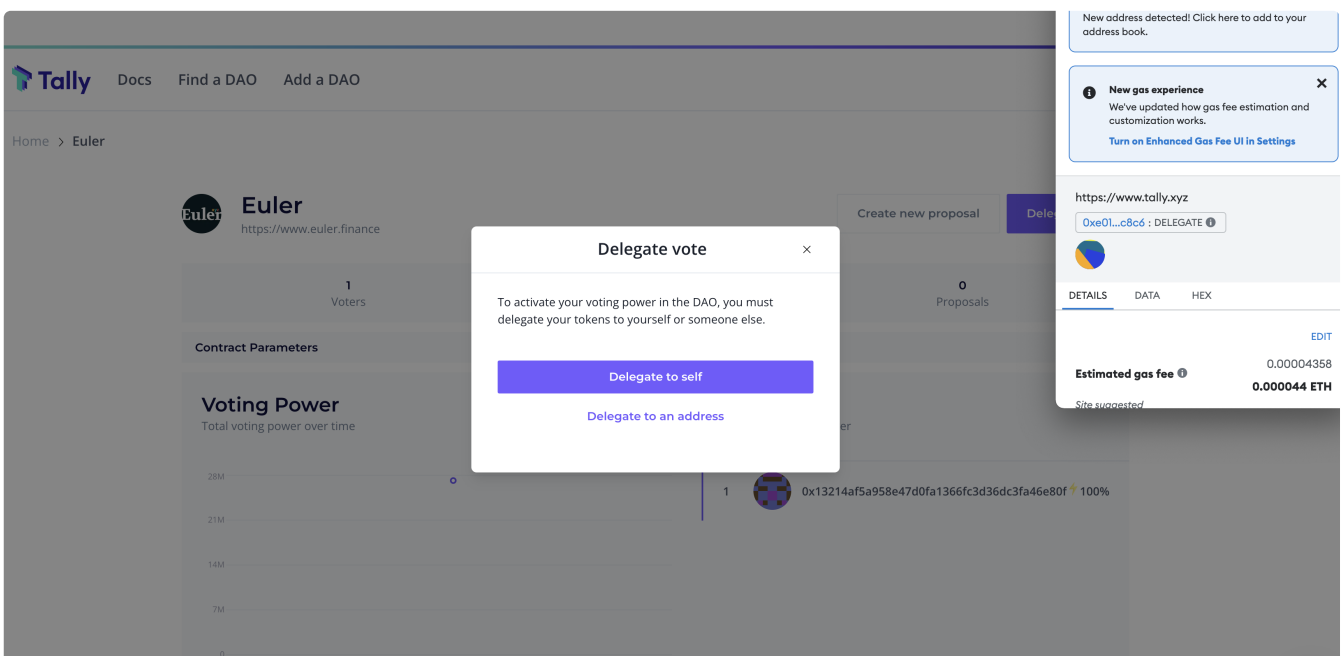
If you choose to delegate to an address or delegate, the following screen will be shown instead where you can enter the address you wish to delegate your voting power to.



This will not transfer any of your EUL tokens to the delegate, but rather only delegate all your voting power, i.e., you will be voting via a delegate or proxy who will be voting on your behalf or representing you at the polls!

You can always change the delegate later on or delegate to yourself again. This helps to ensure that there is a good degree of participation from the community on on-chain governance proposals voting.

4. Finally, regardless of whether you are delegating to yourself or delegating to a delegate, you will be required to confirm the transaction in your Metamask wallet and this transaction will cost gas.



To recap, delegates are token holders that have completed a one-time setup process. Once you become a delegate, you can then vote on active proposals, and create proposals if you have enough voting power. If you choose not to directly vote on proposals, you can pass your voting power on to a delegate as we have seen.

Programmatically

For developers who wish to interact with the EUL token smart contract directly, the EUL contract has a `delegate` function defined with examples on how to interact with it shown below.

```
function delegate(address delegatee)
```

- `delegatee` : The address the sender wishes to delegate their votes to.
- `msg.sender` : The address of the EUL token holder that is attempting to delegate their votes.
- `RETURN` : No return.

Solidity

```
EUL eul = EUL(0x123...); // contract address  
eul.delegate(delegateeAddress);
```

Web3.js

```
const tx = await eul.methods.delegate(delegateeAddress).send({ from: sender });
```

Write a Proposal

About

This guide describes how to get started with writing a governance proposal, firstly on the forum followed by a proposal on Snapshot (off-chain) or Tally (on-chain).

Step-by-step

The Request for Comment (RFC) section on the forum is the first step in creating a governance proposal. Head over to the [RFC section](#) on the [Euler Governance Forum](#) to create a new proposal. Make sure it's well-formulated, be proactive with the community, engage with their comments, and be open to suggestions.

For consistency, the following proposal structure is advised:

Proposal Structure

- **Title:** [Enter Proposal Title]
- **Author(s):** [enter name(s) of associated authors]
- **Related Discussions:** [paste link here]
- **Submission Date:** [Enter Date]

Body Paragraphs/Sections:

- **Simple Summary:** Give the community a TL;DR on your proposal; no more than 2-3 sentences.
- **Abstract:** Introduce and expand on the proposal. Highlight key points on how the proposal will improve stakeholder/token holder experience, protocol performance, and the overall implementation process.
- **Motivation:** What problems will this proposal address/solve? What's the value-add?
- **Specification:** Answer key relevant question to the protocol.

1. What is the link between the eIP author and the asset?
2. Provide a brief description of the asset
3. How is the asset primarily used?
4. Explain why the eIP would benefit Euler's ecosystem?
5. Where does the asset trade?
6. What are the volumes and market capitalisation?
7. What is the liquidity like in the Uniswap V3 liquidity pool versus ETH?
8. What security/auditing reports have been done?

- **Implementation:** Present the implementation of the proposal using [proposal transaction generation tool](#).
- **Risk Assessment:** Give evaluation of the risk parameters involved with the proposal

1. Oracle grading
2. Decentralisation
3. Volatility
4. Liquidity
5. Smart Contract Risk

- **Voting:** Define what a “yes” and “no” vote entails. If there are any Snapshot votes or forum polls associated with this proposal, please attach them.
- **Relevant Links:** If you used Euler’s oracle grading tool or other tools/references please add it here, eg. ‘Oracle grading tool: <https://oracle.euler.finance/>’

A good governance proposal example can be found here: [eIP: Promote WBTC to collateral tier](#)

2. Governance Proposal (on-chain or off-chain)

If the RFC is well-formulated and the community has a clear understanding of the proposal and supports your RFC, it will be moved by a mod to the governance category as an **eIP: Euler Improvement Proposal**. Please note that it usually takes a week (7 days) for a proposal to be moved from RFC to the eIP stage. Once an eIP has been assigned, the proposal can then be created on Snapshot or Tally using this eIP.

A Snapshot or Tally proposal does not always need to be posted by the original eIP author, it can be posted by someone else or one of the delegates in case the minimum threshold of EUL is not being met. The proposal on Snapshot or Tally should always have the link to the eIP attached. The parameters of the poll (Yes, No, different options for values) need to mirror the options discussed in the eIP.

Proposals that are rejected due to invalidity or insufficient support can be resubmitted. Approved proposals with sufficient support via governance/voting will be implemented by the Euler Foundation.

Create a Tally (On-Chain) Proposal

About

On-Chain governance actions (proposal, voting, etc.) for the Euler protocol can be done via the [Tally](#) governance dashboard (described below). It is only required for proposals proposing changes to the smart contracts.

Tally is a web-based governance application focused on enabling on-chain governance. The Tally governance web application [provides transparency around the governance of various DeFi protocols, e.g., Compound, Uniswap, etc.](#) bringing all of the proposals and voting for these protocols under a shared user interface.

Tally empowers user owned governance through a voting dashboard, governance tooling, and real time research and analysis. Users can use the app to review data on governance systems, active and prior proposals, and individual delegates or token holders. The platform also enabled direct on-chain voting and vote delegation, helping users put their governance insights into action. Through integration with the Euler governance smart contract, Euler token holders can connect their wallets and create proposals, vote, delegate voting power to a community member, discover other delegates in the community, and more.

The [Euler Governance Dashboard](#) can be accessed on Tally. The guide below demonstrates how to create an on-chain governance proposal on this dashboard.

Step-by-step

Now that you have created a proposal using [proposal.euler.finance](#) and it has passed off-chain voting on Snapshot, you are ready to make a proposal on Tally for on-chain voting.

1. Visit the [Euler on-chain governance dashboard](#) on Tally and connect your wallet where you have EUL voting power.

⚠ You CANNOT vote if you have not delegated your token., You have to either delegate your token power to yourself or a delegate in order to vote in governance. Self-delegate or delegate to others [here](#).

2. To create a new proposal, click on `Create New Proposal` from the DAO home page on Tally as shown in the top right corner in the image below.

The screenshot shows the Euler DAO interface. At the top, there are navigation links for 'Docs', 'Find a DAO', and 'Add a DAO'. The main header includes the Euler logo, the name 'Euler', and the network 'RINKEBY'. There are two buttons: 'Create new proposal' and 'Delegate vote'. Below the header, there are three statistics: '1 Voters', '1 Holders', and '10 Proposals'. A 'Contract Parameters' dropdown menu is visible. The 'Voting Power' section shows a line graph with a single data point at approximately 27.18M. The 'Top Voters' section lists one voter with 100% of the voting power. A 'Manage Proposals' dropdown menu is also present. At the bottom, there is a 'My drafts' section with a table containing one proposal entry.

This will then open up the proposal creation dialog taking users through the required steps to create an on-chain proposal. In the initial step / screen, it will check that the user has enough voting power to meet the proposal threshold specified within the governance smart contract.

Create a new proposal

The screenshot shows the 'Create a new proposal' dialog. It features the Euler logo at the top. Below the logo, there are four status items, each with a checkmark: 'Connect your wallet', 'Wallet connected', 'Correct chain selected', and 'You have 27.18M voting power. You've reached the proposal threshold!'. A 'Switch wallet' button is located below these items. A prominent blue 'Continue' button is positioned at the bottom of the dialog. Below the dialog, a second step is visible: '2 Name your proposal'.

3. The `Continue` button shown above will become active if the connected wallet has the required voting power that meets the proposal threshold. Upon clicking continue, you will be presented with a form to input the proposal name and add a short description as shown below.

2 Name your proposal

Give your proposal a title and a description. They will be public when your proposal goes live!

Title

Description

[Write](#) [Preview](#)

B *I* **H1** **H2** **H3**

Preview image (optional)

4. Users will need to add the actions to be executed should the proposal become successful or receive majority of vote in support. In this step, users can specify the target smart contract address (this should be the [stub governance smart contract address -] (<https://etherscan.io/address/0x8233f21dda26229c8b0586c3c2521be5da0e63280x8233f21dda26229c8b0586c3c2521be5da0e6328>) for phase one of the governance launch), smart contract function and required function parameters (the hex data for your proposal created on proposal.euler.finance). Up to a maximum of 10 actions can be added in a single proposal.

You do not need to upload smart contract ABI file as it will be automatically imported from the verified contract on etherscan.

✓ Add actions

Add up to 10 actions to be executed if the proposal passes.

Action #1

Target contract address

0x8f61dc4ec11f91dd3522f9f29784e332d30c4176

✓ Verified Contract found on Etherscan. ABI automatically imported.

Drag and drop your ABI file

Or click to browse your files

✓ ABI file uploaded

Contract method

mockFunctionNonPayable

Also send ETH to the target address? (this is not common)

Add action

Remove action

Back

Continue

5. The following page will then be the review page allowing the user to review and confirm that the specified actions are correct:

✓ Preview your proposal

Hey there! 🙌

You have completed all steps successfully. Now is the time to review your proposal and submit it.

 Edit

Test mockFunctionNonPayable() function governance modifier

Proposed by:  0x1321...E80f

Description

A call to the mockFunctionNonPayable() with the onlyGovernorAdmin modifier to verify that the set admin is valid (Timelock contract) and is the target contract caller from Governance.

 Edit

Actions

Function 1:

Calldata:
0x0f63e42c

Target:
0x8f61dc4ec11f91dd3522f9f29784e332d30c4176

Value:
0

Back

Save Draft

Submit on-chain

Once confirmed, the proposal will then be created on-chain and if successful, Tally will display the proposal page with the description and status as it progresses (e.g., pending, active, succeeded, queued, executed).

Submitting on-chain, this may take a few minutes ↔ Copy link Submitting...

SUBMITTED

Test mockFunctionNonPayable() function governance modifier

ID 505b73...e60b • Created by: 0x13214Af5a958E47D0FA1366fC3D36dC3Fa46E80f

Details

Description Executable code

Test mockFunctionNonPayable() function governance modifier

A call to the mockFunctionNonPayable() with the onlyGovernorAdmin modifier to verify that the set admin is valid (Timelock contract) and is the target contract caller from Governance.

Status history

- Draft
Created: May 19th - 10:57 am
- Submitted

Full Tally documentation can be accessed online at: [Tally](#). The documentation describes how to navigate the web app, voting and delegation and creating a Tally account.

Create a Snapshot (Off-Chain) Proposal

About

Off-chain governance actions (proposal, voting, etc.) for the Euler protocol can be done via the [Snapshot](#) governance dashboard (described below). For off-chain governance, there is no code to review or implement as such. It is mainly a call for the Euler Foundation to carry out an action. Issue a grant, or pay a bill, for example. Thus mainly used to aid Euler in making difficult decisions in collaboration with the community.

[Snapshot](#) is an off-chain, 'gasless', multi-governance community polling dashboard used by a number of decentralised finance projects including the likes of Aave and Balancer.

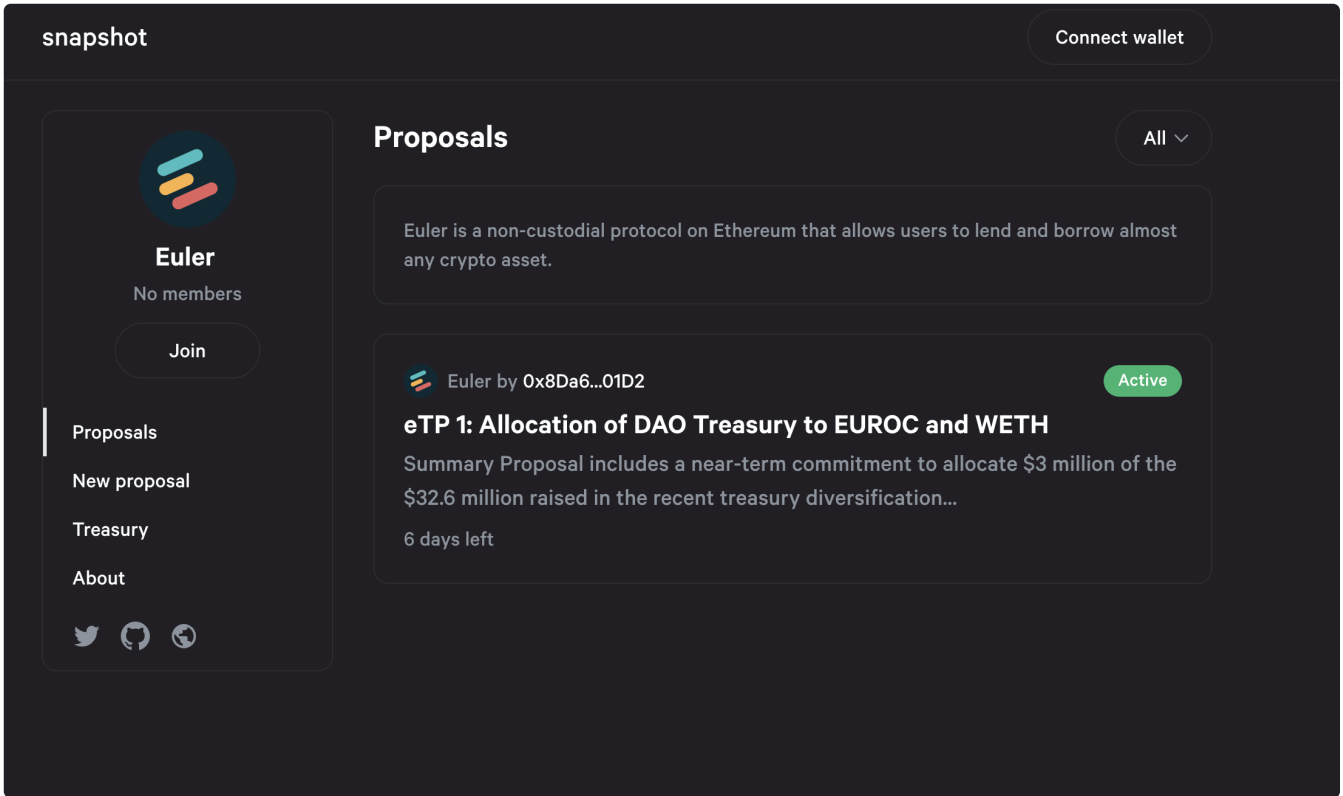
It provides a simple interface to create governance proposals and lets users vote on them by connecting their wallets and the governance tokens contained within. However, the actual voting process is conducted off-chain to save on gas costs and complexity to enable community members 'signal' their preferences on proposals before any on-chain actions or governance process.

The [Euler Governance Dashboard](#) can be accessed on Snapshot.

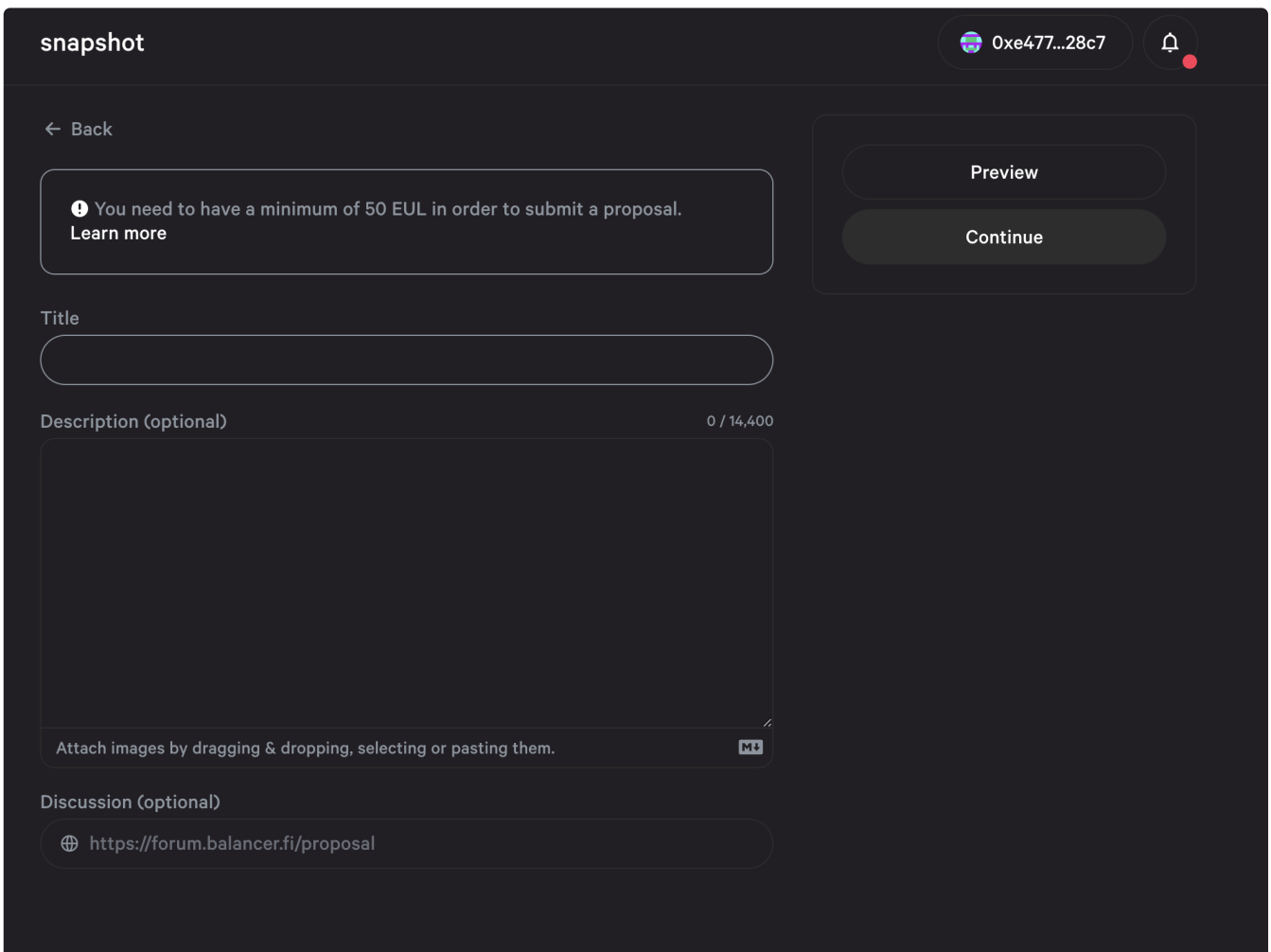
Step-by-step

1. Navigate to the [Euler](#) home page on Snapshot and connect your wallet where you have EUL voting power. You should see the Euler space home page as shown below.

⚠ You CANNOT vote if you have not delegated your token., You have to either delegate your token power to yourself or a delegate in order to vote in governance. Self-delegate or delegate to others [here](#).



2. Click on **New proposal** on the left hand side of the window of the Euler space home page (shown above). It should open up the new proposal form for you to complete which looks like this:



It will also check your connected wallet for voting power and let you know the current proposal threshold.

3. Enter the proposal title.

4. Enter the proposal description (it can be formatted using markdown) and you can also enter a link at the bottom pointing to your proposal on the Euler Governance Forum on Discourse.

5. Click on `Continue` after previewing your proposal and go to the `Actions` box and select the voting type and start date of your proposal (end date will be shown based on the current voting period settings). The proposal creation block number is the snapshot where the voting power of voters will be counted.

6. Click on `Publish` and your proposal will be created. You will be prompted by Metamask to sign a transaction which is free and the proposal will then become active on Snapshot.

Full Snapshot documentation can be accessed via the [Snapshot documentation site](#).

Vote on Tally (On-Chain)

About

This guide describes how to cast a vote on a proposal (on-chain) created on Tally.

Step-by-step

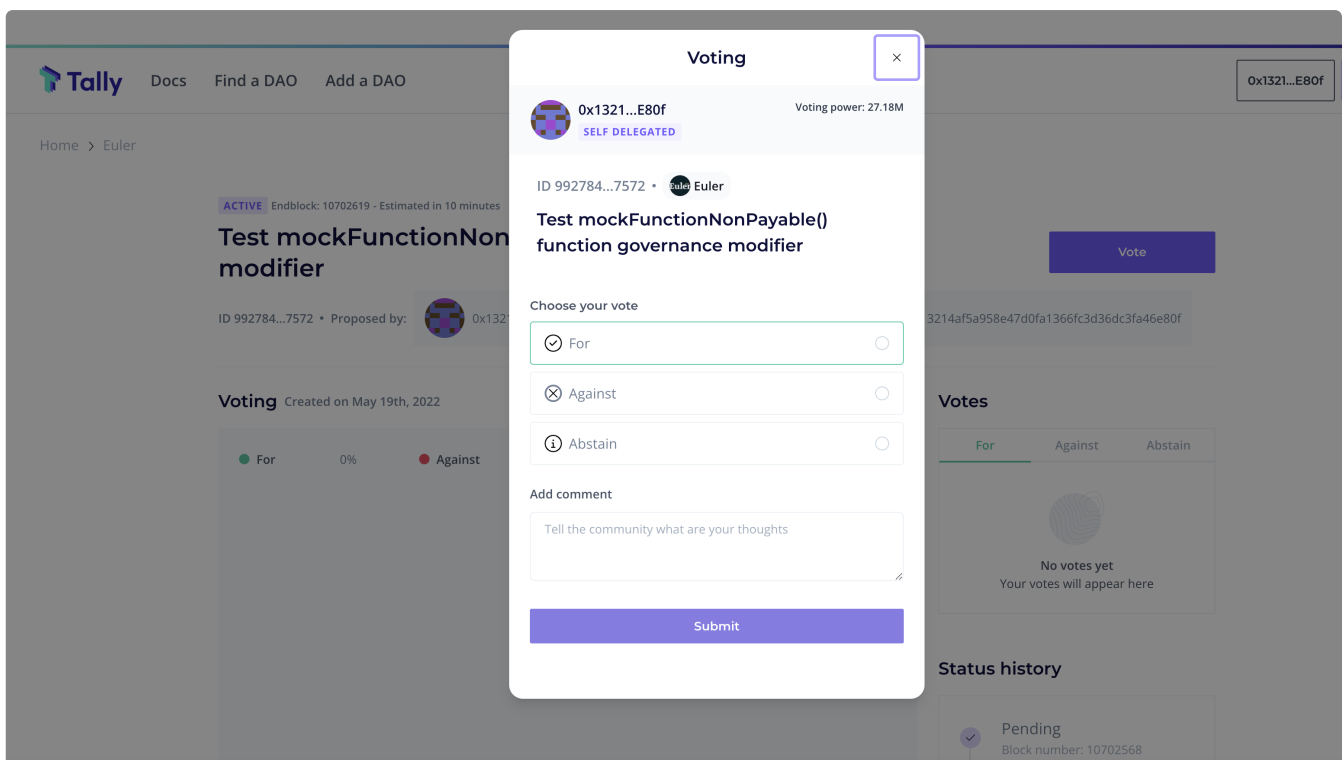
1. Visit the [Euler on-chain governance dashboard](#) on Tally and connect your wallet where you have EUL voting power.

⚠️ *You CANNOT vote if you have not delegated your token., You have to either delegate your token power to yourself or a delegate in order to vote in governance. Self-delegate or delegate to others [here](#).*

2. Click on an active proposal of your choice.

3. Click on **Vote**

4. When voting, voters have the option to vote for or against a proposal or an abstain vote as shown below. Voters also have the option of casting a vote with or without a comment for the community.



5. You will be asked to confirm the vote transaction in your wallet after clicking **Submit**.

Full Tally documentation can be accessed online at [Tally](#). The documentation describes how to navigate the web app, voting and delegation and creating a Tally account.

Vote on Snapshot (Off-Chain)

About

This guide describes how to cast a vote on a proposal (off-chain) created on Snapshot.

Step-by-step

1. Navigate to the [Euler](#) home page on Snapshot and connect your wallet where you have EUL voting power.

⚠️ *You CANNOT vote if you have not delegated your token., You have to either delegate your token power to yourself or a delegate in order to vote in governance. Self-delegate or delegate to others [here](#).*

2. Click on an active proposal of your choice to view more details and see the voting options (as well as existing votes).

The screenshot shows a web browser displaying a Snapshot proposal page. The URL is snapshot.org/#/eulerdao.eth/proposal/0x3b4b7e79c40df6860e7d612bdccc4969753e283dfd84673dc5fc4d201abcb317. The page title is "eTP 1: Allocation of DAO Treasury to EUROC and WETH". The proposal is marked as "Active" and was created by Euler by 0x8Da6...01D2. The summary states: "Proposal includes a near-term commitment to allocate \$3 million of the \$32.6 million raised in the recent treasury diversification event from USDC to EUROC and \$1 million into WETH. These funds will be intended for securing Uniswap v3 oracles and laying the foundation for promoting EUROC to the collateral tier. This proposal paves the way to enabling on-chain EURUSD FX trading via Euler." The motivation is described as three-fold. The current results show 321 EUL (65.56%) for "Yes" and 168 EUL (34.44%) for "No". The quorum is 489 / 1K. The voting system is "Single choice voting" and the start date is Jun 29, 2022, 9:24 PM, with an end date of Jul 5, 2022, 9:24 PM. The snapshot ID is 15,046,757.

Option	Count	Percentage
Yes	321 EUL	65.56%
No	168 EUL	34.44%

Quorum: 489 / 1K

Scrolling down on the page, you will see the current votes, voters (votes and voting power) and the options available for voting.

3. Select an option to vote on, and your voting power will be displayed and you will be prompted by Metamask to sign a transaction which is free.

snapshot.org/#eulerdao.eth/proposal/0x3b4b7e79c40df6860e7d612bdccc4969753e283dfd84673dc5fc4d201abcb317

snapshot Oxe477...28c7

Show more

Discussion

eTP 1: Allocation of DAO Treasury to EUROC and WETH
Title: Allocation of DAO Treasury to EUROC and WETH Author(s): Seraphim C...

Cast your vote

Yes

No

Vote

Votes 4

0x32a.eth	Yes	260 EUL <i>m</i>
jib0xd.eth	No	168 EUL <i>m</i>
0xa854...8591	Yes	54 EUL <i>m</i>
0xc029...B1F2	Yes	7 EUL <i>m</i>

Full Snapshot documentation can be accessed via the [Snapshot documentation site](#).

Join the Forum

About

This [Euler Forum](#) is dedicated to discussions on Euler governance. Relevant topics include:

- Governance proposals
- Proposal discussions
- Site feedback
- Risk assessments

Joining the forum helps members of the community to keep up to date with the latest discussions within the community around governance proposals as well as engage in the comments section or create your own proposals.

To access the forum, simply navigate to forum.euler.finance. You will need to register an account in order to have full access to the features, e.g., replying to posts and proposals or creating a new proposal for others to comment on.

If you need technical help, or want a place for more general discussion, visit the official [Euler Discord](#).

Step-by-step

The steps below describe how to join the forum.

1. Click on the `Sign Up` button at top right corner of the form home page.
2. Complete the account creation form which will pop up upon clicking `Sign Up`.
3. Once completed, click on `Create your account`

Treasury

Learn more about the Euler Treasury

Introduction

All newly created EUL tokens enter circulation initially via a smart contract called the Euler Treasury. The treasury is managed by EUL token holders through on-chain and off-chain governance procedures and overseen by the Euler Foundation.

Address

The address for the treasury is: `0xcAD001c30E96765aC90307669d578219D4fb1DCe` .

It can be viewed on Etherscan [here](#).

Multisig

The wallet holding treasury assets is a [Gnosis Safe](#) MultiSig smart contract wallet. A MultiSig wallet requires multiple private key signatures to authorise transactions. In the case of the Euler Treasury, 4 out of 9 signatures are required for every transaction.

The identity of the signers of the MultiSig cannot be revealed, for obvious security reasons. However, signers come from a pool of 6 different organisations and, through their contract with the Euler Foundation, are obliged to carry out the wishes of the Euler DAO.

The MultiSig, along with its signers, are can be viewed publicly here:

[\https://gnosis-safe.io/app/eth:0xcAD001c30E96765aC90307669d578219D4fb1DCe/home.\](https://gnosis-safe.io/app/eth:0xcAD001c30E96765aC90307669d578219D4fb1DCe/home)

Grants

Learn about how to receive a grant for contributing to EulerDAO

Introduction

To encourage developers to build on top of Euler protocol and help integrate it into the wider DeFi ecosystem, a portion of the Euler [Treasury](#) will be allocated to a Grants programme. The purpose of the grants is to foster the growth of Euler protocol by establishing a culture of community-driven development, where individuals making improvements to the Euler Protocol get a say in its future.

Addresses

Smart contract addresses for Euler Governance (On-Chain)

Networks

The EulerDAO is currently deployed to the following networks:

Mainnet

Contract	Address	Etherscan	Source Code
EUL	<code>0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b</code>	Etherscan	GitHub
Governance	<code>0xd8E2114f6bCbaee83CDEB1bD6650a28BBcF144D5</code>	Etherscan	GitHub
Timelock Controller	<code>0xd4Ee8939a537D943a4E46E7Ae04069C9451d724F</code>	Etherscan	GitHub
Stub Target Contract	<code>0x8233f21dda26229c8b0586c3c2521be5da0e6328</code>	Etherscan	GitHub

Rinkeby

Contract	Address	Etherscan	Source Code
EUL	<code>0xe013C993A77Cdd 1aC0d8c1B15a6eFf 95EB36c8c6</code>	Etherscan	GitHub
Governance	<code>0x681E9cf95e26c6 C2cEF09fdc476C7f 8De6AFf2D5</code>	Etherscan	GitHub
Timelock Controller	<code>0x16fBC769237cE1 7830799e6faD9d53 536c3B8389</code>	Etherscan	GitHub
Stub Target Contract	<code>0x57848100bc0771 61805fdDcF6D9bA1 5D4aab06d8</code>	Etherscan	GitHub

Parameters

Learn more about the Euler Governance smart contract parameters

Introduction

This page outlines the governance parameters for both on-chain and off-chain governance.

Tally (On-Chain) Governance Parameters

This section outlines the governance parameters for the Euler Governance smart contracts (managed via [Tally](#)). All parameters are displayed in Table 1 below.

Execution Delay, Voting Delay and Voting Period are based on the assumption of a 15 seconds block creation time on the Ethereum Mainnet.

The governance smart contract inherits functionality from the OpenZeppelin [GovernorSettings.sol module](#) allowing Voting Delay, Voting Period and Proposal Threshold to be updated through an on-chain governance proposal and voting process.

Table 1 Euler On-Chain Governance Parameters

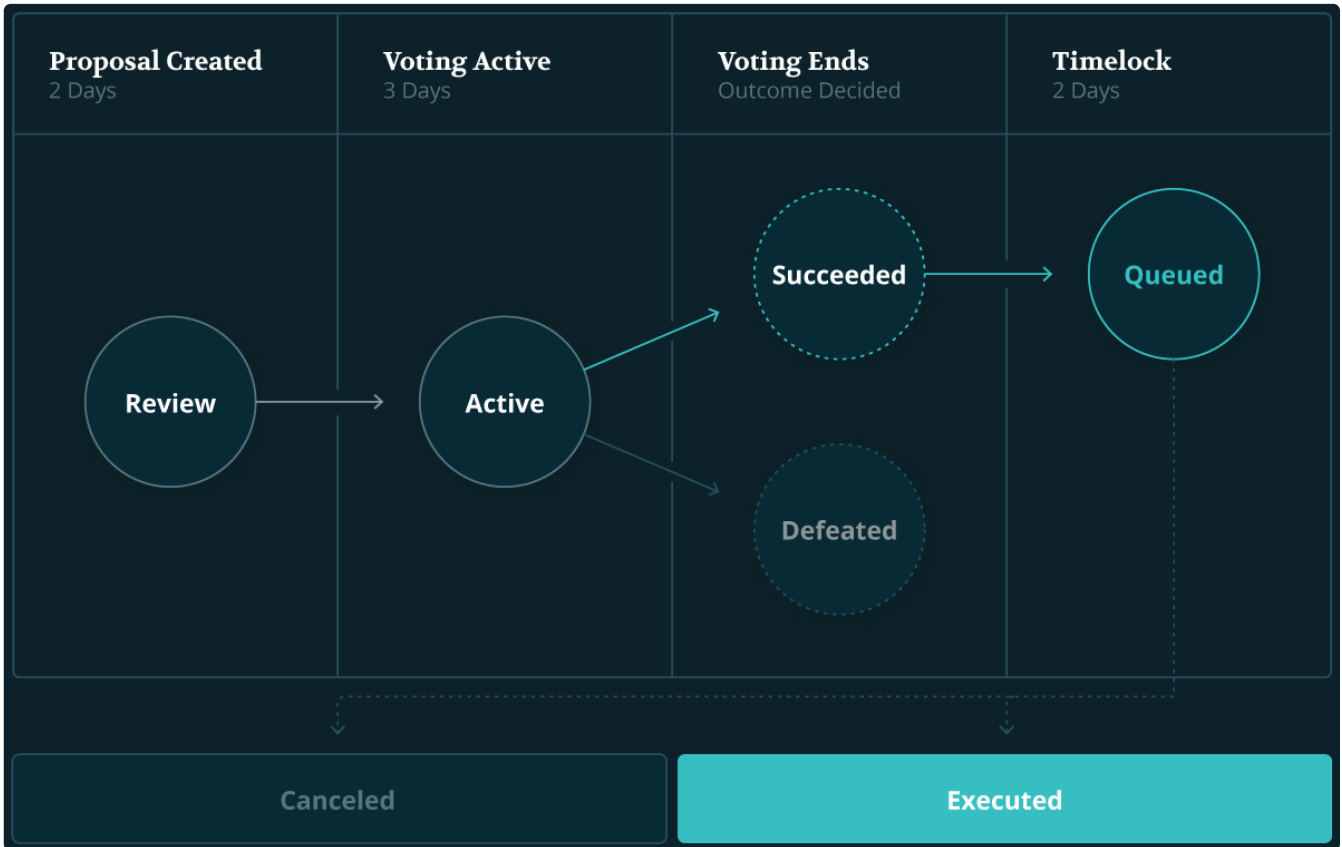
Parameter	Value
Voting Delay	11520 blocks (2 days)
Voting Period	17280 blocks (3 days)
Execution Delay	172800 seconds (2 days)
Quorum Numerator	3% of EUL Supply
Proposal Threshold	75,000 EUL

When a governance proposal is created, it enters a 2-day review period (i.e., Voting Delay), after which voting weights are recorded and voting begins.

Voting lasts for 3 days (i.e., Voting Period); once the voting period is over, if quorum was reached (enough voting power participated) and the majority voted in favour, the proposal is considered successful and can proceed to be executed 2 days (48 hours) later (i.e., Execution Delay).

Addresses delegated at least 75,000 EUL can create governance proposals having met the Proposal Threshold.

The image below depicts the on-chain governance phases and durations for each phase:



Snapshot (Off-Chain) Governance Parameters

This section outlines the governance parameters for off-chain governance (managed via [Snapshot](#)). All parameters are displayed in Table 2 below.

Table 2 Euler Off-Chain Governance Parameters

Parameter	Value
Voting Period	6 days
Quorum	1,000 EUL
Proposal Threshold	50 EUL

There is no voting delay or execution delay for the off-chain governance process, given there is no direct effect on the protocol's smart contracts.

Addresses holding or delegated at least 50 EUL can create governance proposals having met the Proposal Threshold. With regard to voting power, the delegated voting power or EUL balance at the proposal creation block number is counted towards voting power. The [Snapshot voting strategies](#) enabled are `erc20-balance-of` and `erc20-votes`.

EUL

About

Learn more about the protocol-native governance token of Euler

Introduction

EUL tokens represent voting powers to effect change over the protocol code. EUL is an ERC20 token that acts as the native governance token of the Euler Protocol. The EUL token address is:

`0xd9Fcd98c322942075A5C3860693e9f4f03AAE07b` .

More information about EUL can be found on [Etherscan](#), [CoinMarketCap](#) or [CoinGecko](#).

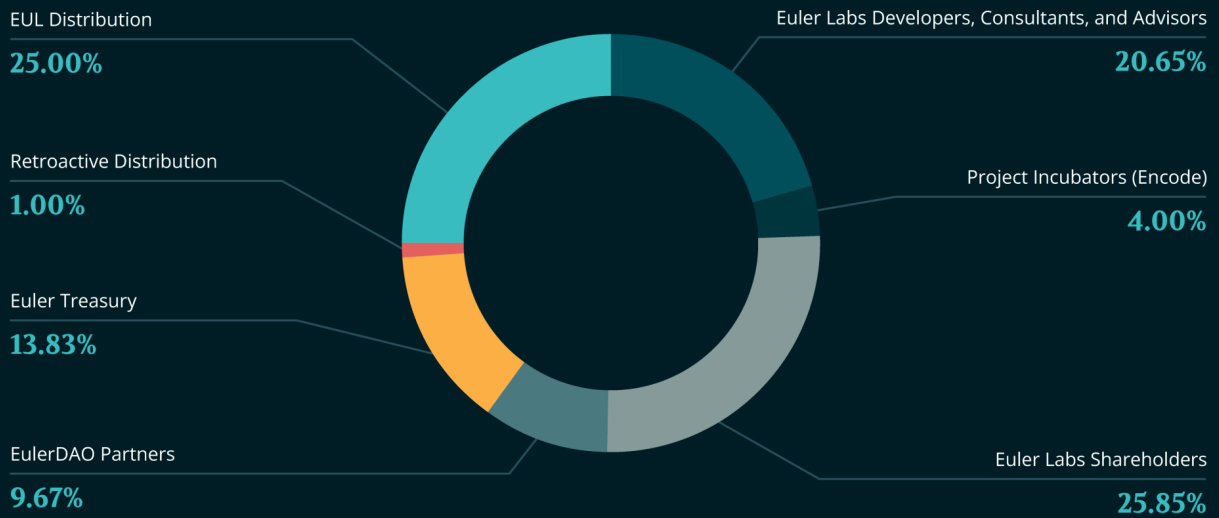
Breakdown

The total supply of EUL is 27,182,818 (in homage to Euler's number, *e*). The initial four-year breakdown of the EUL total supply is as follows:

- **25%** (6,795,705 EUL) to users of the Euler protocol over 4 years (see [Distribution](#)).
- **1%** (271,828 EUL) to all users who deposited or borrowed assets on Euler during its soft launch (see [Epoch0](#)).
- **13.83%** (3,759,791 EUL) to the Euler Treasury, unlocked (see [Treasury](#)).
- **25.85%** (7,026,759 EUL) to Euler Labs shareholders, with an 18 month linear unlock schedule starting on 01/01/2022.
- **9.67%** (2,628,170 EUL) to partners of EulerDAO, with an 18 month linear unlock schedule starting on 01/01/2022.
- **4%** (1,087,313 EUL) to Encode, an early project incubator, with a linear 30 month unlock schedule starting on 01/01/2022.
- **20.65%** (5,613,252 EUL) to employees, advisors and consultants of Euler Labs. Co-founders with a 48 month linear unlock schedule starting on 01/01/2022. All others with individual agreements.

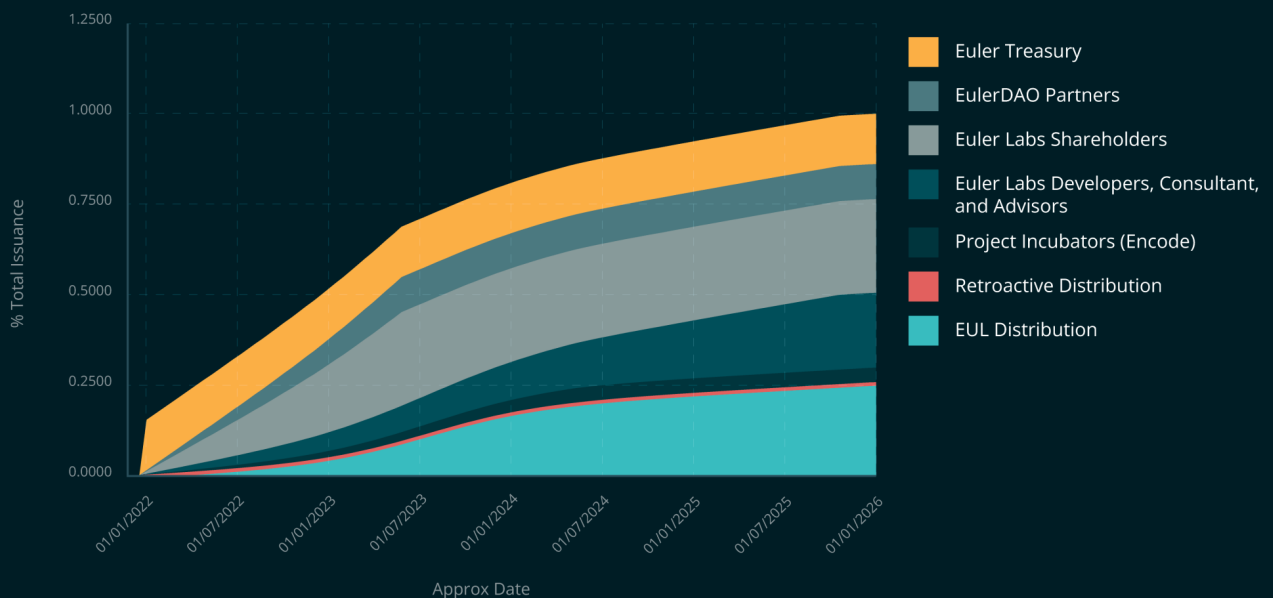
Here is the breakdown:

Breakdown



The unlock schedule for different groups is as follows:

Supply Unlock

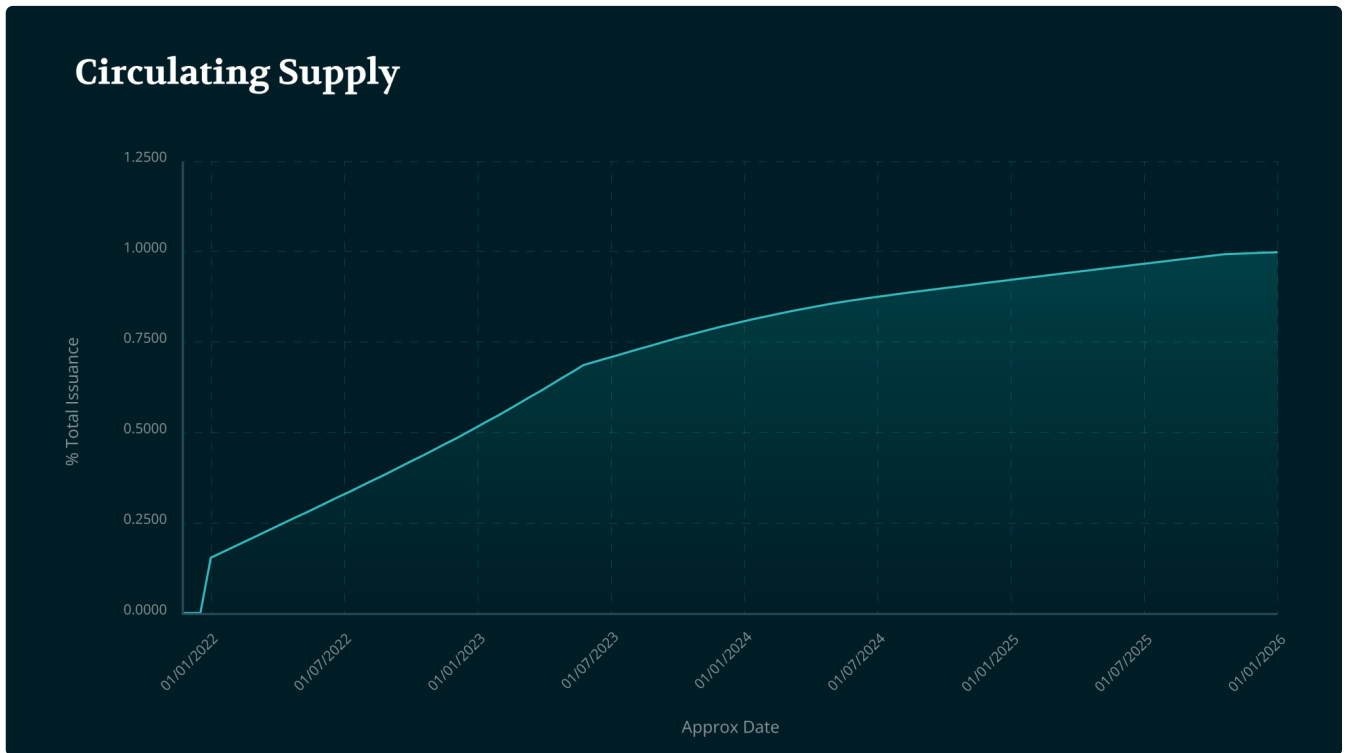


Note that the initial allocations may be subject to change as the ecosystem evolves. As EUL is distributed to users of the protocol they may see fit to vote to alter the EUL Distribution, for example.

The total supply of EUL is fixed for the first 4 years, after which EUL token holders may enact a governance proposal to inflate the supply by a maximum 2.718% per year. In that scenario, newly minted EUL will enter circulation via the [Treasury](#).

Circulating Supply

The approximate schedule for the circulating supply of EUL is shown below.



Distribution

Learn more about how EUL is distributed to protocol users to decentralise governance

Introduction

In order to progressively decentralise governance of the Euler Protocol, EUL will be distributed to protocol users over a period of (approx) 4 years. See how much EUL is being distributed today [here](#).

The distribution programme is broken down into cycles called [epochs](#). On Euler, you can be eligible to receive EUL by either staking your lending positions ([eTokens](#)) to Euler staking contract or by borrowing one of the incentivized assets, which have been determined either by governance or by the staking [gauge](#) system.

How it Works

The amount of EUL each borrower receives is proportional to the time-weighted amount of debt they held of an asset within the epoch. For example, if 50 EUL are to be distributed to the DAI market in epoch 3, and if Alice borrows 10 DAI for 1 day, and Bob borrows 5 DAI for 2 days, then at the end of the epoch, and after the merkle-root update (typically takes a few hours), they will both be able to claim an equal share of 25 EUL.

The same logic applies to the lenders that stake their eTokens to one of the Euler staking contracts. For example, if 50 EUL are to be distributed to the USDC staking pool in epoch 3, and if Alice stakes 10 eUSDC for 1 day, and Bob stakes 5 eUSDC for 2 days, they will both be able to claim an equal share of 25 EUL at any time.

How to Claim

Users with an EUL distribution allocation can navigate to the top right of the UI and click the 'Claim' button. That will open a dialogue box, showing a user's projected EUL distribution after the current epoch has completed. This will tick up second-by-second as a user accrues more time-weighted borrowing. The pending balance below that shows EUL tokens already distributed to the user but which remain in the distribution smart contract unclaimed. Users can click the Claim button in the bottom right of the dialogue to transfer those tokens to their wallet.

Markets

Total Supply
\$296.56M

Top Markets

USDC	\$112.94M
WETH	\$57.19M
WSTETH	\$55.22M

Search

Search token name, symbol or address

Assets

7 Collateral 11 Cross 42 Isolated

Market Oracle Rating

USDC Collateral

Distribution

EUL
Euler Finance

Add to MetaMask

About

Borrowers on Euler are allocated voting powers allowing them to participate in governance of the protocol. Voting powers are dispensed as an ERC20 governance token called EUL. Read more about governance [here](#).

Your Wallet

Address:
Balance: 0.0
Value: -

Distribution

Projected tokens: 5.35457763299099284
Projected value: -
Epoch 4 Progress

Claim

Pending balance: 26.622483790336279161
Pending value: -

Claim

Total Borrowed
\$200.51M

Top Markets

USDC	\$82.18M
WETH	\$41.62M
WSTETH	\$33.16M

Short/Long

Rows per page: 25 1-25 of 60

Total Supply Total Borrows Available

\$112.94M \$82.18M \$30.76M

Epochs

Learn more about the phases of the EUL Distribution Programme

Introduction

The EUL Distribution Programme has two phases. Epoch 0, in which governance tokens were distributed to early protocol users retroactively; and Epoch 1-96, in which governance tokens are distributed on an ongoing basis to active users of the protocol.

Epoch 0

Epoch 0 covers the 3 month period from 26/11/2021 to 21/03/2022 during which Euler protocol was in a soft-launch mode. Users were experimenting with the protocol in a risk-minimised manner.

Implementation

Lenders and **borrowers** using the protocol during this period were allocated a share of 1% of the total supply of EUL as a one-off retroactive distribution.

The distribution took place as follows:

1. A snapshot of all users on the protocol was taken at block 14,430,000.
2. A total of 271,828 EUL (1% of the total supply) was distributed to anyone interacting with the protocol upto that point.
3. The distribution amount per address was calculated as follows:
 - 2/3 of the issuance was distributed proportionally based on the time-weighted* average USD value of the deposits and borrows in the mentioned time frame.
 - 1/3 of the total issuance is distributed evenly amongst all the unique addresses that interacted with the protocol in the mentioned time frame.

**For simplicity, the accrued interest on deposits and borrows was neglected.*

In conclusion, 3407 unique addresses received at least 26.59 EUL tokens plus an individual amount proportional to the time-weighted average USD value of their deposits and borrows. See [here](#) for the final allocations.

Epochs 1-96

Epochs 1-96 cover the period from 21/03/2022 to 17/12/2025 during which Euler will progressively decentralise.

Initial Implementation

Borrowers using the protocol during this period will be allocated EUL via a rolling merkle distribution. The amount distributed each epoch will follow a non-linear schedule (see below).

Within each market, borrowers will receive an EUL distribution proportional to their time-weighted borrowing on that market. The amount of EUL allocated to each market every epoch will be determined by EUL token holders (see [Gauges](#)).

Users will be able to claim their EUL governance tokens after an epoch has completed by using the 'Claim' button at <https://app.euler.finance/>.

Updates

[eIP 24](#)

The DAO voted to alter the initial EUL Distribution Programme. Beginning epoch 18, the DAO began allocating a fixed amount of 40,000 EUL each epoch to the [Gauges](#) to be voted on each epoch by existing EUL token holders, with an additional 15,000 EUL allocated evenly to lenders of USDC, USDT, and WETH for a trial period of 6 epochs.

[eIP29](#)

Creation of an Euler boosted USDC/DAI/USDT pool that is allocated 5,000 EUL as voting incentives every two weeks. This would run for a trial period of three months.

[eIP51](#)

Proposal to change distribution as follows:

- 9,000 EUL per epoch to stakers of WETH market.
- 5,000 EUL per epoch to stakers of USDC market.
- 1,000 EUL per epoch to stakers of USDT market.
- 8,000 EUL per epoch via gauges to borrowers on each of USDC, WETH, and WStETH
- 8,000 EUL per epoch shared proportionally among assets with Chainlink oracle

Schedule

The following table outlines the block numbers for previous and forthcoming epochs.

Epoch	Block Number	EUL Distribution
0	14,430,000	271,828.18
1	14,530,000	36,915.69
2	14,630,000	37,673.39
3	14,730,000	38,531.30
4	14,830,000	39,501.78
5	14,930,000	40,598.43
6	15,030,000	41,836.14
7	15,130,000	43,231.14
8	15,230,000	44,800.97
9	15,330,000	46,564.39
10	15,430,000	48,541.27
11	15,530,000	50,752.38
12	15,630,000	53,219.03
13	15,730,000	55,962.62
14	15,830,000	59,004.03
15	15,930,000	62,362.78
16	16,030,000	66,056.03
17	16,130,000	70,097.30
18	16,230,000	eIP24
19	16,330,000	55000
20	16,430,000	55000
21	16,530,000	55000
22	16,630,000	55000
23	16,730,000	55000
24	16,830,000	eIP29 & eIP51
25	16,930,000	52000
26	17,030,000	52000

27	17,130,000	52000
28	17,230,000	52000
29	17,330,000	47000
30	17,430,000	47000
31	17,530,000	47000
32	17,630,000	47000
33	17,730,000	47000
34	17,830,000	47000
35	17,930,000	47000
36	18,030,000	47000
37	18,130,000	47000
38	18,230,000	47000
39	18,330,000	47000
40	18,430,000	47000
41	18,530,000	47000
42	18,630,000	47000
43	18,730,000	47000
44	18,830,000	47000
45	18,930,000	47000
46	19,030,000	47000
47	19,130,000	47000
48	19,230,000	47000
49	19,330,000	47000
50	19,430,000	47000
51	19,530,000	47000
52	19,630,000	47000
53	19,730,000	47000
54	19,830,000	47000

55	19,930,000	47000
56	20,030,000	47000
57	20,130,000	47000
58	20,230,000	47000
59	20,330,000	47000
60	20,430,000	47000
61	20,530,000	47000
62	20,630,000	47000
63	20,730,000	47000
64	20,830,000	47000
65	20,930,000	47000
66	21,030,000	47000
67	21,130,000	47000
68	21,230,000	47000
69	21,330,000	47000
70	21,430,000	47000
71	21,530,000	47000
72	21,630,000	47000
73	21,730,000	47000
74	21,830,000	47000
75	21,930,000	47000
76	22,030,000	47000
77	22,130,000	47000
78	22,230,000	47000
79	22,330,000	47000
80	22,430,000	47000
81	22,530,000	47000
82	22,630,000	47000
83	22,730,000	47000

84	22,830,000	47000
85	22,930,000	47000
86	23,030,000	47000
87	23,130,000	47000
88	23,230,000	47000
89	23,330,000	47000
90	23,430,000	47000
91	23,530,000	47000
92	23,630,000	47000
93	23,730,000	47000
94	23,830,000	47000
95	23,930,000	47000
96	24,030,000	47000

Gauges

Learn about how Euler enables community-selected markets to receive a governance token distribution

Introduction

The Euler community helps to determine which markets receive an EUL distribution through the use of staking gauges. EUL token holders can visit the [Gauge](#) page on the app UI and stake their tokens against a particular market to indicate their preference for that market receiving an EUL distribution in future epochs.

Good to know

As it stands, users cannot vote on DAO proposals if they participate in gauge voting. Only users with EUL can direct further EUL emissions. You can remove your EUL at any point from the gauge to distribute further EUL emissions.

For an emissions schedule of EUL, please see the [Epochs page](#).

Staking

Learn about how Euler rewards different assets with lending incentives

Staking on Euler is based on Synthetix's staking contracts. This is an overhaul to Euler's gauge system, which thanks to [eIP51](#) is modified from its previous iteration coming into effect with the arrival of [Epoch 24](#).

To stake an asset and receive some of the EUL being distributed, users should stake their [eTokens](#) into the staking contract.

Should you please, you can immediately unstake your tokens at any time and the accrued EUL earnings will be instantly claimable. There is no lockup period for this staking process.

According to [eIP51](#), the DAO has made the decision to keep the staking rewards program running indefinitely, unless another vote is held to terminate the program. The staking contracts will receive EUL tokens distributed in the following manner:

- 9,000 EUL per epoch to stakers of WETH market.
- 5,000 EUL per epoch to stakers of USDC market.
- 1,000 EUL per epoch to stakers of USDT market.

Considerations

While these eTokens are held in the staking contract, users should be aware that they cannot collateralise loans. You cannot borrow against tokens that are earning these EUL in the staking contract.

Make sure when depositing assets into the staking contract that if you have any outstanding liabilities, they are adequately collateralised AFTER you have deposited your USDC/USDT/WETH. Your account will be flagged for [liquidation](#) otherwise.

Developers

Getting Started

Contract Integration Guide

Find out how to start working with the Euler smart contracts

Modules

The Euler protocol is a collection of smart contracts connected together with a module system. Each module handles specific areas of the protocol, so depending on what you want to do, you will interact with several different contract addresses.

Some modules are global, for example:

- [markets](#): Activating markets, enter/exiting markets, and querying various market-related information.
- [exec](#): Batch requests, liquidity deferrals (ie, flash loans)
- [liquidation](#): Seizure of assets for users in violation

Other modules are asset-specific:

- [eTokens](#): ERC20-compatible tokens that represent assets
- [dTokens](#): ERC20-compatible tokens that represent liabilities

Deposit and withdraw

In order to invest an asset to earn interest, you need to `deposit` into an eToken.

```
// Approve the main euler contract to pull your tokens:
IERC20(underlying).approve(EULER_MAINNET, type(uint).max);

// Use the markets module:
IEulerMarkets markets = IEulerMarkets(EULER_MAINNET_MARKETS);

// Get the eToken address using the markets module:
IEulerEToken eToken = IEulerEToken(markets.underlyingToEToken(underlying));

// Deposit 5.25 underlying tokens (assuming 18 decimal places)
// The "0" argument refers to the sub-account you are depositing to.
eToken.deposit(0, 5.25e18);

eToken.balanceOf(address(this));
// -> internal book-keeping value that doesn't increase over time

eToken.balanceOfUnderlying(address(this));
// -> 5.25e18
// ... but check back next block to see it go up (assuming there are borrowers)

// Later on, withdraw your initial deposit and all earned interest:
eToken.withdraw(0, type(uint).max);
```

Borrow and repay

If you would like to borrow an asset, you must have sufficient collateral, and be "entered" into the collateral's market.

```

// Use the markets module:
IEulerMarkets markets = IEulerMarkets(EULER_MAINNET_MARKETS);

// Approve, get eToken addr, and deposit:
IERC20(collateral).approve(EULER_MAINNET, type(uint).max);
IEulerEToken collateralEToken = IEulerEToken(markets.underlyingToEToken(collateral));
collateralEToken.deposit(0, 100e18);

// Enter the collateral market (collateral's address, *not* the eToken address):
markets.enterMarket(0, collateral);

// Get the dToken address of the borrowed asset:
IEulerDToken borrowedDToken = IEulerDToken(markets.underlyingToDToken(borrowed));

// Borrow 2 tokens (assuming 18 decimal places).
// The 2 tokens will be sent to your wallet (ie, address(this)).
// This automatically enters you into the borrowed market.
borrowedDToken.borrow(0, 2e18);

borrowedDToken.balanceOf(address(this));
// -> 2e18
// ... but check back next block to see it go up

// Later on, to repay the 2 tokens plus interest:
IERC20(borrowed).approve(EULER_MAINNET, type(uint).max);
borrowedDToken.repay(0, type(uint).max);

```

Flash loans

Euler has flash loans built-in as an integral component of the protocol. There are three ways to take a flash loan, a low-level Euler-specific way, a way that uses an [EIP-3156](#) compatible flash-loan adaptor, and a gas-efficient direct interface.

Low-level Flash Loans

The low-level way to take a flash loan is to defer the liquidity check for your account. The Euler contract will call back into your contract, where you can perform operations like `borrow()` without worrying about liquidity violations. As long as your callback leaves the account in a non-violating state, the transaction will complete successfully.

Since Euler only charges interest for a loan when it is held for a non-zero amount of time, this results in fee-less flash loans.

Here is an example contract that demonstrates this:

```

contract MyFlashLoanContract {
    struct MyCallbackData {
        uint whatever;
    }

    function somethingThatNeedsFlashLoan() {
        // Setup whatever data you need
        MyCallbackData memory data;
        data.whatever = 1234;

        // Disable the liquidity check for "this" and call-back into onDeferredLiquidityCheck
        IExec(exec).deferLiquidityCheck(address(this), abi.encode(data));
    }

    function onDeferredLiquidityCheck(bytes memory encodedData) external override {
        MyCallbackData memory data = abi.decode(encodedData, (MyCallbackData));

        // Borrow 10 tokens (assuming 18 decimals):

        IEulerDToken(borrowedDToken).borrow(0, 10e18);

        // ... do whatever you need with the borrowed tokens ...

        // Repay the 10 tokens:

        IERC20(borrowed).approve(EULER_MAINNET, type(uint).max);
        IEulerDToken(borrowedDToken).repay(0, 10e18);
    }
}

```

`encodedData` is a pass-through parameter that lets you transfer data to your callback without requiring storage writes.

EIP-3156 Flash Loans

There is also an adaptor smart contract that exposes Euler's flash loan functionality as an [EIP-3156](#) compatible API.

The smart contract addresses are: [mainnet](#), [goerli](#).

Examples of how to use the adaptor can be found in the EIP documentation, as well as the [Euler test suite](#). The fee value is always 0.

Gas-Efficient Direct Flash Loans

As of [eIP-14](#), DTokens also support a `flashLoan` method. In most cases, this is now the recommended way to perform a pure flash loan. It is simpler and consumes less gas than either of the above methods.

To use this, your contract should implement the `IFlashLoan` interface:

```
interface IFlashLoan {
    function onFlashLoan(bytes memory data) external;
}
```

When you wish to perform a flash loan, your contract should invoke the `flashLoan` function on the `DToken` that corresponds to the asset you wish to borrow:

```
function flashLoan(uint amount, bytes calldata data) external;
```

The `DToken` contract will transfer the requested `amount` of tokens to your contract address (decimals are the same as in the external token contract -- no normalisation needed), and then invoke your contract's `onFlashLoan` function. The `data` parameter you specify is passed to the callback unchanged, which allows you to pass extra data to your contract without requiring expensive storage writes.

Note that any address could call `onFlashLoan` on your contract at any time. You may want to ensure that `msg.sender` is the Euler contract's address, or use some other kind of authentication scheme.

Your contract is expected to repay `amount` back to the Euler contract (which will be `msg.sender`) within the `onFlashLoan` function.

Here is an example:

```
import "IEuler.sol";

contract MyContract {
    function myFunction() external {
        require(msg.sender == myAdminAddress, "not allowed");
        IEulerDToken dToken = IEulerDToken(markets.underlyingToDToken(underlying));
        dToken.flashLoan(amount, abi.encode(underlying, amount));
    }

    function onFlashLoan(bytes memory data) external {
        require(msg.sender == EulerAddrMainnet.euler, "not allowed");
        (address underlying, uint amount) = abi.decode(data, (address, uint));

        // ...

        IERC20(underlying).transfer(msg.sender, amount); // repay
    }
}
```

Contract Reference

IEuler

Main storage contract for the Euler system

moduleIdToImplementation

Lookup the current implementation contract for a module

```
function moduleIdToImplementation(uint moduleId) external view returns (address);
```

Parameters:

- **moduleId**: Fixed constant that refers to a module type (ie MODULEID__ETOKEN)

Returns:

- An internal address specifies the module's implementation code

moduleIdToProxy

Lookup a proxy that can be used to interact with a module (only valid for single-proxy modules)

```
function moduleIdToProxy(uint moduleId) external view returns (address);
```

Parameters:

- **moduleId**: Fixed constant that refers to a module type (ie MODULEID__MARKETS)

Returns:

- An address that should be cast to the appropriate module interface, ie `IEulerMarkets(moduleIdToProxy(2))`

AssetConfig

Euler-related configuration for an asset

```
struct AssetConfig {
    address eTokenAddress;
    bool borrowIsolated;
    uint32 collateralFactor;
    uint32 borrowFactor;

    uint24 twapWindow;
}
```

IEulerMarkets

Activating and querying markets, and maintaining entered markets lists

activateMarket

Create an Euler pool and associated EToken and DToken addresses.

```
function activateMarket(address underlying) external returns (address);
```

Parameters:

- **underlying**: The address of an ERC20-compliant token. There must be an initialised uniswap3 pool for the underlying/reference asset pair.

Returns:

- The created EToken, or the existing EToken if already activated.

activatePToken

Create a pToken and activate it on Euler. pTokens are protected wrappers around assets that prevent borrowing.

```
function activatePToken(address underlying) external returns (address);
```

Parameters:

- **underlying**: The address of an ERC20-compliant token. There must already be an activated market on Euler for this underlying, and it must have a non-zero collateral factor.

Returns:

- The created pToken, or an existing one if already activated.

underlyingToEToken

Given an underlying, lookup the associated EToken

```
function underlyingToEToken(address underlying) external view returns (address);
```

Parameters:

- **underlying**: Token address

Returns:

- EToken address, or address(0) if not activated

underlyingToDToken

Given an underlying, lookup the associated DToken

```
function underlyingToDToken(address underlying) external view returns (address);
```

Parameters:

- **underlying**: Token address

Returns:

- DToken address, or address(0) if not activated

underlyingToPToken

Given an underlying, lookup the associated PToken

```
function underlyingToPToken(address underlying) external view returns (address);
```

Parameters:

- **underlying**: Token address

Returns:

- PToken address, or address(0) if it doesn't exist

underlyingToAssetConfig

Looks up the Euler-related configuration for a token, and resolves all default-value placeholders to their currently configured values.


```
function underlyingToAssetConfig(address underlying) external view returns (IEuler.AssetConfig)
```

Parameters:

- **underlying**: Token address

Returns:

- Configuration struct

underlyingToAssetConfigUnresolved

Looks up the Euler-related configuration for a token, and returns it unresolved (with default-value placeholders)

```
function underlyingToAssetConfigUnresolved(address underlying) external view returns (IEuler.AssetConfig)
```

Parameters:

- **underlying**: Token address

Returns:

- **config**: Configuration struct

eTokenToUnderlying

Given an EToken address, looks up the associated underlying

```
function eTokenToUnderlying(address eToken) external view returns (address underlying);
```

Parameters:

- **eToken**: EToken address

Returns:

- **underlying**: Token address

dTokenToUnderlying

Given a DToken address, looks up the associated underlying

```
function dTokenToUnderlying(address dToken) external view returns (address underlying);
```

Parameters:

- **dToken**: DToken address

Returns:

- **underlying**: Token address

eTokenToDToken

Given an EToken address, looks up the associated DToken

```
function eTokenToDToken(address eToken) external view returns (address dTokenAddr);
```

Parameters:

- **eToken**: EToken address

Returns:

- **dTokenAddr**: DToken address

interestRateModel

Looks up an asset's currently configured interest rate model

```
function interestRateModel(address underlying) external view returns (uint);
```

Parameters:

- **underlying**: Token address

Returns:

- Module ID that represents the interest rate model (IRM)

interestRate

Retrieves the current interest rate for an asset

```
function interestRate(address underlying) external view returns (int96);
```

Parameters:

- **underlying**: Token address

Returns:

- The interest rate in yield-per-second, scaled by 10^{27}

interestAccumulator

Retrieves the current interest rate accumulator for an asset

```
function interestAccumulator(address underlying) external view returns (uint);
```

Parameters:

- **underlying**: Token address

Returns:

- An opaque accumulator that increases as interest is accrued

reserveFee

Retrieves the reserve fee in effect for an asset

```
function reserveFee(address underlying) external view returns (uint32);
```

Parameters:

- **underlying**: Token address

Returns:

- Amount of interest that is redirected to the reserves, as a fraction scaled by `RESERVE_FEE_SCALE` (4e9)

getPricingConfig

Retrieves the pricing config for an asset

```
function getPricingConfig(address underlying) external view returns (uint16 pricingType, uint);
```

Parameters:

- **underlying**: Token address

Returns:

- **pricingType**: (1=pegged, 2=uniswap3, 3=forwarded, 4=chainlink)
- **pricingParameters**: If uniswap3 pricingType then this represents the uniswap pool fee used, if chainlink pricing type this represents the fallback uniswap pool fee or 0 if none
- **pricingForwarded**: If forwarded pricingType then this is the address prices are forwarded to, otherwise address(0)

getChainlinkPriceFeedConfig

Retrieves the Chainlink price feed config for an asset

```
function getChainlinkPriceFeedConfig(address underlying) external view returns (address chainlink);
```

Parameters:

- **underlying**: Token address

Returns:

- **chainlinkAggregator**: Chainlink aggregator proxy address

getEnteredMarkets

Retrieves the list of entered markets for an account (assets enabled for collateral or borrowing)

```
function getEnteredMarkets(address account) external view returns (address[] memory);
```

Parameters:

- **account**: User account

Returns:

- List of underlying token addresses

enterMarket

Add an asset to the entered market list, or do nothing if already entered

```
function enterMarket(uint subAccountId, address newMarket) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **newMarket**: Underlying token address

exitMarket

Remove an asset from the entered market list, or do nothing if not already present

```
function exitMarket(uint subAccountId, address oldMarket) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **oldMarket**: Underlying token address

IEulerExec

Batch executions, liquidity check deferrals, and interfaces to fetch prices and account liquidity

LiquidityStatus

Liquidity status for an account, either in aggregate or for a particular asset

```
struct LiquidityStatus {
    uint collateralValue;
    uint liabilityValue;
    uint numBorrows;
    bool borrowIsolated;
}
```

AssetLiquidity

Aggregate struct for reporting detailed (per-asset) liquidity for an account

```
struct AssetLiquidity {
    address underlying;
    LiquidityStatus status;
}
```

EulerBatchItem

Single item in a batch request

```
struct EulerBatchItem {  
  
    bool allowError;  
    address proxyAddr;  
    bytes data;  
}
```

EulerBatchItemResponse

Single item in a batch response

```
struct EulerBatchItemResponse {  
    bool success;  
    bytes result;  
}
```

BatchDispatchSimulation

Error containing results of a simulated batch dispatch

```
error BatchDispatchSimulation(EulerBatchItemResponse[] simulation);
```

liquidity

Compute aggregate liquidity for an account

```
function liquidity(address account) external view returns (LiquidityStatus memory status);
```

Parameters:

- **account**: User address

Returns:

- **status**: Aggregate liquidity (sum of all entered assets)

detailedLiquidity

Compute detailed liquidity for an account, broken down by asset

```
function detailedLiquidity(address account) external view returns (AssetLiquidity[] memory as);
```

Parameters:

- **account**: User address

Returns:

- **assets**: List of user's entered assets and each asset's corresponding liquidity

getPrice

Retrieve Euler's view of an asset's price

```
function getPrice(address underlying) external view returns (uint twap, uint twapPeriod);
```

Parameters:

- **underlying**: Token address

Returns:

- **twap**: Time-weighted average price
- **twapPeriod**: TWAP duration, either the twapWindow value in AssetConfig, or less if that duration not available

getPriceFull

Retrieve Euler's view of an asset's price, as well as the current marginal price on uniswap

```
function getPriceFull(address underlying) external view returns (uint twap, uint twapPeriod, uint currPrice);
```

Parameters:

- **underlying**: Token address

Returns:

- **twap**: Time-weighted average price
- **twapPeriod**: TWAP duration, either the twapWindow value in AssetConfig, or less if that duration not available
- **currPrice**: The current marginal price on uniswap3 (informational: not used anywhere in the Euler protocol)

deferLiquidityCheck

Defer liquidity checking for an account, to perform rebalancing, flash loans, etc. msg.sender must implement IDeferredLiquidityCheck

```
function deferLiquidityCheck(address account, bytes memory data) external;
```

Parameters:

- **account**: The account to defer liquidity for. Usually `address(this)`, although not always
- **data**: Passed through to the `onDeferredLiquidityCheck()` callback, so contracts don't need to store transient data in storage

batchDispatch

Execute several operations in a single transaction

```
function batchDispatch(EulerBatchItem[] calldata items, address[] calldata deferLiquidityCheck
```

Parameters:

- **items**: List of operations to execute
- **deferLiquidityChecks**: List of user accounts to defer liquidity checks for

batchDispatchSimulate

Call batch dispatch, but instruct it to revert with the responses, before the liquidity checks.

```
function batchDispatchSimulate(EulerBatchItem[] calldata items, address[] calldata deferLiqui
```

Parameters:

- **items**: List of operations to execute
- **deferLiquidityChecks**: List of user accounts to defer liquidity checks for

trackAverageLiquidity

Enable average liquidity tracking for your account. Operations will cost more gas, but you may get additional benefits when performing liquidations

```
function trackAverageLiquidity(uint subAccountId, address delegate, bool onlyDelegate) extern
```

Parameters:

- **subAccountId**: subAccountId 0 for primary, 1-255 for a sub-account.
- **delegate**: An address of another account that you would allow to use the benefits of your account's average liquidity (use the null address if you don't care about this). The other address must also reciprocally delegate to your account.
- **onlyDelegate**: Set this flag to skip tracking average liquidity and only set the delegate.

unTrackAverageLiquidity

Disable average liquidity tracking for your account and remove delegate

```
function unTrackAverageLiquidity(uint subAccountId) external;
```

Parameters:

- **subAccountId**: subAccountId 0 for primary, 1-255 for a sub-account

getAverageLiquidity

Retrieve the average liquidity for an account

```
function getAverageLiquidity(address account) external returns (uint);
```

Parameters:

- **account**: User account (xor in subAccountId, if applicable)

Returns:

- The average liquidity, in terms of the reference asset, and post risk-adjustment

getAverageLiquidityWithDelegate

Retrieve the average liquidity for an account or a delegate account, if set

```
function getAverageLiquidityWithDelegate(address account) external returns (uint);
```

Parameters:

- **account**: User account (xor in subAccountId, if applicable)

Returns:

- The average liquidity, in terms of the reference asset, and post risk-adjustment

getAverageLiquidityDelegateAccount

Retrieve the account which delegates average liquidity for an account, if set

```
function getAverageLiquidityDelegateAccount(address account) external view returns (address);
```

Parameters:

- **account:** User account (xor in subAccountId, if applicable)

Returns:

- The average liquidity delegate account

pTokenWrap

Transfer underlying tokens from sender's wallet into the pToken wrapper. Allowance should be set for the euler address.

```
function pTokenWrap(address underlying, uint amount) external;
```

Parameters:

- **underlying:** Token address
- **amount:** The amount to wrap in underlying units

pTokenUnWrap

Transfer underlying tokens from the pToken wrapper to the sender's wallet.

```
function pTokenUnWrap(address underlying, uint amount) external;
```

Parameters:

- **underlying:** Token address
- **amount:** The amount to unwrap in underlying units

usePermit

Apply EIP2612 signed permit on a target token from sender to euler contract

```
function usePermit(address token, uint256 value, uint256 deadline, uint8 v, bytes32 r, bytes32 s);
```

Parameters:

- **token:** Token address
- **value:** Allowance value
- **deadline:** Permit expiry timestamp
- **v:** secp256k1 signature v
- **r:** secp256k1 signature r
- **s:** secp256k1 signature s

usePermitAllowed

Apply DAI like (allowed) signed permit on a target token from sender to euler contract

```
function usePermitAllowed(address token, uint256 nonce, uint256 expiry, bool allowed, uint8 v,
```

Parameters:

- **token:** Token address
- **nonce:** Sender nonce
- **expiry:** Permit expiry timestamp
- **allowed:** If true, set unlimited allowance, otherwise set zero allowance
- **v:** secp256k1 signature v
- **r:** secp256k1 signature r
- **s:** secp256k1 signature s

usePermitPacked

Apply allowance to tokens expecting the signature packed in a single bytes param

```
function usePermitPacked(address token, uint256 value, uint256 deadline, bytes calldata signat
```

Parameters:

- **token:** Token address
- **value:** Allowance value
- **deadline:** Permit expiry timestamp
- **signature:** secp256k1 signature encoded as rsv

doStaticCall

Execute a staticcall to an arbitrary address with an arbitrary payload.

```
function doStaticCall(address contractAddress, bytes memory payload) external view returns (b
```

Parameters:

- **contractAddress**: Address of the contract to call
- **payload**: Encoded call payload

Returns:

- **result**: Encoded return data

IEulerEToken

Tokenised representation of assets

name

Pool name, ie "Euler Pool: DAI"

```
function name() external view returns (string memory);
```

symbol

Pool symbol, ie "eDAI"

```
function symbol() external view returns (string memory);
```

decimals

Decimals, always normalised to 18.

```
function decimals() external pure returns (uint8);
```

underlyingAsset

Address of underlying asset

```
function underlyingAsset() external view returns (address);
```

totalSupply

Sum of all balances, in internal book-keeping units (non-increasing)

```
function totalSupply() external view returns (uint);
```

totalSupplyUnderlying

Sum of all balances, in underlying units (increases as interest is earned)

```
function totalSupplyUnderlying() external view returns (uint);
```

balanceOf

Balance of a particular account, in internal book-keeping units (non-increasing)

```
function balanceOf(address account) external view returns (uint);
```

balanceOfUnderlying

Balance of a particular account, in underlying units (increases as interest is earned)

```
function balanceOfUnderlying(address account) external view returns (uint);
```

reserveBalance

Balance of the reserves, in internal book-keeping units (non-increasing)

```
function reserveBalance() external view returns (uint);
```

reserveBalanceUnderlying

Balance of the reserves, in underlying units (increases as interest is earned)

```
function reserveBalanceUnderlying() external view returns (uint);
```

convertBalanceToUnderlying

Convert an eToken balance to an underlying amount, taking into account current exchange rate

```
function convertBalanceToUnderlying(uint balance) external view returns (uint);
```

Parameters:

- **balance:** eToken balance, in internal book-keeping units (18 decimals)

Returns:

- Amount in underlying units, (same decimals as underlying token)

convertUnderlyingToBalance

Convert an underlying amount to an eToken balance, taking into account current exchange rate

```
function convertUnderlyingToBalance(uint underlyingAmount) external view returns (uint);
```

Parameters:

- **underlyingAmount:** Amount in underlying units (same decimals as underlying token)

Returns:

- **eToken:** balance, in internal book-keeping units (18 decimals)

touch

Updates interest accumulator and totalBorrows, credits reserves, re-targets interest rate, and logs asset status

```
function touch() external;
```

deposit

Transfer underlying tokens from sender to the Euler pool, and increase account's eTokens

```
function deposit(uint subAccountId, uint amount) external;
```

Parameters:

- **subAccountId:** 0 for primary, 1-255 for a sub-account
- **amount:** In underlying units (use max uint256 for full underlying token balance)

withdraw

Transfer underlying tokens from Euler pool to sender, and decrease account's eTokens

```
function withdraw(uint subAccountId, uint amount) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **amount**: In underlying units (use max uint256 for full pool balance)

mint

Mint eTokens and a corresponding amount of dTokens ("self-borrow")

```
function mint(uint subAccountId, uint amount) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **amount**: In underlying units

burn

Pay off dToken liability with eTokens ("self-repay")

```
function burn(uint subAccountId, uint amount) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **amount**: In underlying units (use max uint256 to repay the debt in full or up to the available underlying balance)

approve

Allow spender to access an amount of your eTokens in sub-account 0

```
function approve(address spender, uint amount) external returns (bool);
```

Parameters:

- **spender**: Trusted address
- **amount**: Use max uint256 for "infinite" allowance

approveSubAccount

Allow spender to access an amount of your eTokens in a particular sub-account

```
function approveSubAccount(uint subAccountId, address spender, uint amount) external returns
```

Parameters:

- **subAccountId:** 0 for primary, 1-255 for a sub-account
- **spender:** Trusted address
- **amount:** Use max uint256 for "infinite" allowance

allowance

Retrieve the current allowance

```
function allowance(address holder, address spender) external view returns (uint);
```

Parameters:

- **holder:** Xor with the desired sub-account ID (if applicable)
- **spender:** Trusted address

transfer

Transfer eTokens to another address (from sub-account 0)

```
function transfer(address to, uint amount) external returns (bool);
```

Parameters:

- **to:** Xor with the desired sub-account ID (if applicable)
- **amount:** In internal book-keeping units (as returned from balanceOf).

transferFromMax

Transfer the full eToken balance of an address to another

```
function transferFromMax(address from, address to) external returns (bool);
```

Parameters:

- **from:** This address must've approved the to address, or be a sub-account of msg.sender
- **to:** Xor with the desired sub-account ID (if applicable)

transferFrom

Transfer eTokens from one address to another

```
function transferFrom(address from, address to, uint amount) external returns (bool);
```

Parameters:

- **from**: This address must've approved the to address, or be a sub-account of msg.sender
- **to**: Xor with the desired sub-account ID (if applicable)
- **amount**: In internal book-keeping units (as returned from balanceOf).

donateToReserves

Donate eTokens to the reserves

```
function donateToReserves(uint subAccountId, uint amount) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **amount**: In internal book-keeping units (as returned from balanceOf).

IEulerDToken

Tokenised representation of debts

name

Debt token name, ie "Euler Debt: DAI"

```
function name() external view returns (string memory);
```

symbol

Debt token symbol, ie "dDAI"

```
function symbol() external view returns (string memory);
```

decimals

Decimals of underlying

```
function decimals() external view returns (uint8);
```

underlyingAsset

Address of underlying asset

```
function underlyingAsset() external view returns (address);
```

totalSupply

Sum of all outstanding debts, in underlying units (increases as interest is accrued)

```
function totalSupply() external view returns (uint);
```

totalSupplyExact

Sum of all outstanding debts, in underlying units normalized to 27 decimals (increases as interest is accrued)

```
function totalSupplyExact() external view returns (uint);
```

balanceOf

Debt owed by a particular account, in underlying units

```
function balanceOf(address account) external view returns (uint);
```

balanceOfExact

Debt owed by a particular account, in underlying units normalized to 27 decimals

```
function balanceOfExact(address account) external view returns (uint);
```

borrow

Transfer underlying tokens from the Euler pool to the sender, and increase sender's dTokens

```
function borrow(uint subAccountId, uint amount) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **amount**: In underlying units (use max uint256 for all available tokens)

repay

Transfer underlying tokens from the sender to the Euler pool, and decrease sender's dTokens

```
function repay(uint subAccountId, uint amount) external;
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **amount**: In underlying units (use max uint256 for full debt owed)

flashLoan

Request a flash-loan. A `onFlashLoan()` callback in `msg.sender` will be invoked, which must repay the loan to the main Euler address prior to returning.

```
function flashLoan(uint amount, bytes calldata data) external;
```

Parameters:

- **amount**: In underlying units
- **data**: Passed through to the `onFlashLoan()` callback, so contracts don't need to store transient data in storage

approveDebt

Allow spender to send an amount of dTokens to a particular sub-account

```
function approveDebt(uint subAccountId, address spender, uint amount) external returns (bool)
```

Parameters:

- **subAccountId**: 0 for primary, 1-255 for a sub-account
- **spender**: Trusted address
- **amount**: In underlying units (use max uint256 for "infinite" allowance)

debtAllowance

Retrieve the current debt allowance

```
function debtAllowance(address holder, address spender) external view returns (uint);
```

Parameters:

- **holder**: Xor with the desired sub-account ID (if applicable)
- **spender**: Trusted address

transfer

Transfer dTokens to another address (from sub-account 0)

```
function transfer(address to, uint amount) external returns (bool);
```

Parameters:

- **to**: Xor with the desired sub-account ID (if applicable)
- **amount**: In underlying units. Use max uint256 for full balance.

transferFrom

Transfer dTokens from one address to another

```
function transferFrom(address from, address to, uint amount) external returns (bool);
```

Parameters:

- **from**: Xor with the desired sub-account ID (if applicable)
- **to**: This address must've approved the from address, or be a sub-account of msg.sender
- **amount**: In underlying units. Use max uint256 for full balance.

IEulerLiquidation

Liquidate users who are in collateral violation to protect lenders

LiquidationOpportunity

Information about a prospective liquidation opportunity

```

struct LiquidationOpportunity {
    uint repay;
    uint yield;
    uint healthScore;

    // Only populated if repay > 0:

    uint baseDiscount;
    uint discount;
    uint conversionRate;
}

```

checkLiquidation

Checks to see if a liquidation would be profitable, without actually doing anything

```

function checkLiquidation(address liquidator, address violator, address underlying, address collateral)

```

Parameters:

- **liquidator**: Address that will initiate the liquidation
- **violator**: Address that may be in collateral violation
- **underlying**: Token that is to be repayed
- **collateral**: Token that is to be seized

Returns:

- **liqOpp**: The details about the liquidation opportunity

liquidate

Attempts to perform a liquidation

```

function liquidate(address violator, address underlying, address collateral, uint repay, uint minYield)

```

Parameters:

- **violator**: Address that may be in collateral violation
- **underlying**: Token that is to be repayed
- **collateral**: Token that is to be seized
- **repay**: The amount of underlying DTokens to be transferred from violator to sender, in units of underlying
- **minYield**: The minimum acceptable amount of collateral ETokens to be transferred from violator to sender, in units of collateral

IEulerSwap

Trading assets on Uniswap V3 and 1Inch V4 DEXs

SwapUniExactInputSingleParams

Params for Uniswap V3 exact input trade on a single pool

```
struct SwapUniExactInputSingleParams {
    uint subAccountIdIn;
    uint subAccountIdOut;
    address underlyingIn;
    address underlyingOut;
    uint amountIn;
    uint amountOutMinimum;
    uint deadline;
    uint24 fee;
    uint160 sqrtPriceLimitX96;
}
```

Parameters:

- **subAccountIdIn**: subaccount id to trade from
- **subAccountIdOut**: subaccount id to trade to
- **underlyingIn**: sold token address
- **underlyingOut**: bought token address
- **amountIn**: amount of token to sell
- **amountOutMinimum**: minimum amount of bought token
- **deadline**: trade must complete before this timestamp
- **fee**: uniswap pool fee to use
- **sqrtPriceLimitX96**: maximum acceptable price

SwapUniExactInputParams

Params for Uniswap V3 exact input trade routed through multiple pools

```
struct SwapUniExactInputParams {
    uint subAccountIdIn;
    uint subAccountIdOut;
    uint amountIn;
    uint amountOutMinimum;
    uint deadline;
    bytes path; // list of pools to hop - constructed with uni SDK
}
```

Parameters:

- **subAccountIdIn**: subaccount id to trade from
- **subAccountIdOut**: subaccount id to trade to
- **underlyingIn**: sold token address
- **underlyingOut**: bought token address
- **amountIn**: amount of token to sell
- **amountOutMinimum**: minimum amount of bought token
- **deadline**: trade must complete before this timestamp
- **path**: list of pools to use for the trade

SwapUniExactOutputSingleParams

Params for Uniswap V3 exact output trade on a single pool

```
struct SwapUniExactOutputSingleParams {
    uint subAccountIdIn;
    uint subAccountIdOut;
    address underlyingIn;
    address underlyingOut;
    uint amountOut;
    uint amountInMaximum;
    uint deadline;
    uint24 fee;
    uint160 sqrtPriceLimitX96;
}
```

Parameters:

- **subAccountIdIn**: subaccount id to trade from
- **subAccountIdOut**: subaccount id to trade to
- **underlyingIn**: sold token address
- **underlyingOut**: bought token address
- **amountOut**: amount of token to buy
- **amountInMaximum**: maximum amount of sold token
- **deadline**: trade must complete before this timestamp
- **fee**: uniswap pool fee to use
- **sqrtPriceLimitX96**: maximum acceptable price

SwapUniExactOutputParams

Params for Uniswap V3 exact output trade routed through multiple pools

```
struct SwapUniExactOutputParams {
    uint subAccountIdIn;
    uint subAccountIdOut;
    uint amountOut;
    uint amountInMaximum;
    uint deadline;
    bytes path;
}
```

Parameters:

- **subAccountIdIn**: subaccount id to trade from
- **subAccountIdOut**: subaccount id to trade to
- **underlyingIn**: sold token address
- **underlyingOut**: bought token address
- **amountOut**: amount of token to buy
- **amountInMaximum**: maximum amount of sold token
- **deadline**: trade must complete before this timestamp
- **path**: list of pools to use for the trade

Swap1InchParams

Params for 1Inch trade

```
struct Swap1InchParams {
    uint subAccountIdIn;
    uint subAccountIdOut;
    address underlyingIn;
    address underlyingOut;
    uint amount;
    uint amountOutMinimum;
    bytes payload;
}
```

Parameters:

- **subAccountIdIn**: subaccount id to trade from
- **subAccountIdOut**: subaccount id to trade to
- **underlyingIn**: sold token address
- **underlyingOut**: bought token address
- **amount**: amount of token to sell
- **amountOutMinimum**: minimum amount of bought token
- **payload**: call data passed to 1Inch contract

swapUniExactInputSingle

Execute Uniswap V3 exact input trade on a single pool

```
function swapUniExactInputSingle(SwapUniExactInputSingleParams memory params) external;
```

Parameters:

- **params**: struct defining trade parameters

swapUniExactInput

Execute Uniswap V3 exact input trade routed through multiple pools

```
function swapUniExactInput(SwapUniExactInputParams memory params) external;
```

Parameters:

- **params**: struct defining trade parameters

swapUniExactOutputSingle

Execute Uniswap V3 exact output trade on a single pool

```
function swapUniExactOutputSingle(SwapUniExactOutputSingleParams memory params) external;
```

Parameters:

- **params**: struct defining trade parameters

swapUniExactOutput

Execute Uniswap V3 exact output trade routed through multiple pools

```
function swapUniExactOutput(SwapUniExactOutputParams memory params) external;
```

Parameters:

- **params**: struct defining trade parameters

swapAndRepayUniSingle

Trade on Uniswap V3 single pool and repay debt with bought asset

```
function swapAndRepayUniSingle(SwapUniExactOutputSingleParams memory params, uint targetDebt)
```

Parameters:

- **params**: struct defining trade parameters (amountOut is ignored)
- **targetDebt**: amount of debt that is expected to remain after trade and repay (0 to repay full debt)

swapAndRepayUni

Trade on Uniswap V3 through multiple pools pool and repay debt with bought asset

```
function swapAndRepayUni(SwapUniExactOutputParams memory params, uint targetDebt) external;
```

Parameters:

- **params**: struct defining trade parameters (amountOut is ignored)
- **targetDebt**: amount of debt that is expected to remain after trade and repay (0 to repay full debt)

swap1Inch

Execute 1Inch V4 trade

```
function swap1Inch(Swap1InchParams memory params) external;
```

Parameters:

- **params**: struct defining trade parameters

IEulerSwapHub

Common logic for executing and processing trades through external swap handler contracts

SwapParams

Params defining a swap request

```
struct SwapParams {
    address underlyingIn;
    address underlyingOut;
    uint mode;
    uint amountIn;
    uint amountOut;
    uint exactOutTolerance;
    bytes payload;
}
```

swap

Execute a trade using the requested swap handler

```
function swap(uint subAccountIdIn, uint subAccountIdOut, address swapHandler, SwapParams memo)
```

Parameters:

- **subAccountIdIn**: sub-account holding the sold token. 0 for primary, 1-255 for a sub-account
- **subAccountIdOut**: sub-account to receive the bought token. 0 for primary, 1-255 for a sub-account
- **swapHandler**: address of a swap handler to use
- **params**: struct defining the requested trade

swapAndRepay

Repay debt by selling another deposited token

```
function swapAndRepay(uint subAccountIdIn, uint subAccountIdOut, address swapHandler, SwapParams memo, uint targetDebt)
```

Parameters:

- **subAccountIdIn**: sub-account holding the sold token. 0 for primary, 1-255 for a sub-account
- **subAccountIdOut**: sub-account to receive the bought token. 0 for primary, 1-255 for a sub-account
- **swapHandler**: address of a swap handler to use
- **params**: struct defining the requested trade
- **targetDebt**: how much debt should remain after calling the function

IEulerPToken

Protected Tokens are simple wrappers for tokens, allowing you to use tokens as collateral without permitting borrowing

name

PToken name, ie "Euler Protected DAI"

```
function name() external view returns (string memory);
```

symbol

PToken symbol, ie "pDAI"

```
function symbol() external view returns (string memory);
```

decimals

Number of decimals, which is same as the underlying's

```
function decimals() external view returns (uint8);
```

underlying

Address of the underlying asset

```
function underlying() external view returns (address);
```

balanceOf

Balance of an account's wrapped tokens

```
function balanceOf(address who) external view returns (uint);
```

totalSupply

Sum of all wrapped token balances

```
function totalSupply() external view returns (uint);
```

allowance

Retrieve the current allowance

```
function allowance(address holder, address spender) external view returns (uint);
```

Parameters:

- **holder:** Address giving permission to access tokens
- **spender:** Trusted address

transfer

Transfer your own pTokens to another address

```
function transfer(address recipient, uint amount) external returns (bool);
```

Parameters:

- **recipient:** Recipient address
- **amount:** Amount of wrapped token to transfer

transferFrom

Transfer pTokens from one address to another. The euler address is automatically granted approval.

```
function transferFrom(address from, address recipient, uint amount) external returns (bool);
```

Parameters:

- **from:** This address must've approved the to address
- **recipient:** Recipient address
- **amount:** Amount to transfer

approve

Allow spender to access an amount of your pTokens. It is not necessary to approve the euler address.

```
function approve(address spender, uint amount) external returns (bool);
```

Parameters:

- **spender:** Trusted address
- **amount:** Use max uint256 for "infinite" allowance

wrap

Convert underlying tokens to pTokens

```
function wrap(uint amount) external;
```

Parameters:

- **amount:** In underlying units (which are equivalent to pToken units)

unwrap

Convert pTokens to underlying tokens

```
function unwrap(uint amount) external;
```

Parameters:

- **amount:** In pToken units (which are equivalent to underlying units)

claimSurplus

Claim any surplus tokens held by the PToken contract. This should only be used by contracts.

```
function claimSurplus(address who) external;
```

Parameters:

- **who:** Beneficiary to be credited for the surplus token amount

IEulerEulDistributor

claim

Claim distributed tokens

```
function claim(address account, address token, uint claimable, bytes32[] calldata proof, address distributor) external;
```

Parameters:

- **account**: Address that should receive tokens
- **token**: Address of token being claimed (ie EUL)
- **proof**: Merkle proof that validates this claim
- **stake**: If non-zero, then the address of a token to auto-stake to, instead of claiming

IEulerEulStakes

staked

Retrieve current amount staked

```
function staked(address account, address underlying) external view returns (uint);
```

Parameters:

- **account**: User address
- **underlying**: Token staked upon

Returns:

- Amount of EUL token staked

StakeOp

Staking operation item. Positive amount means to increase stake on this underlying, negative to decrease.

```
struct StakeOp {  
    address underlying;  
    int amount;  
}
```

stake

Modify stake of a series of underlyings. If the sum of all amounts is positive, then this amount of EUL will be transferred in from the sender's wallet. If negative, EUL will be transferred out to the sender's wallet.

```
function stake(StakeOp[] memory ops) external;
```

Parameters:

- **ops**: Array of operations to perform

stakeGift

Increase stake on an underlying, and transfer this stake to a beneficiary

```
function stakeGift(address beneficiary, address underlying, uint amount) external;
```

Parameters:

- **beneficiary**: Who is given credit for this staked EUL
- **underlying**: The underlying token to be staked upon
- **amount**: How much EUL to stake

stakePermit

Applies a permit() signature to EUL and then applies a sequence of staking operations

```
function stakePermit(StakeOp[] memory ops, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
```

Parameters:

- **ops**: Array of operations to perform
- **value**: The value field of the permit message
- **deadline**: The deadline field of the permit message
- **v**: Signature field
- **r**: Signature field
- **s**: Signature field

Architecture

Contract architecture

Module System

Except for a small amount of dispatching logic (see `Euler.sol`), the contracts are organised into modules, which live in `contracts/modules/`.

There are several reasons why modules are used:

- A proxy indirection layer which is used for dispatching calls from sub-contracts like ETokens and DTokens (see below)
 - Each token must have its own address to conform to ERC-20, even though all storage lives inside the Euler contract
- Contract upgrades
 - Modules can be upgraded, which can immediately upgrade all ETokens (for example)
- Avoid hitting the max contract size limitation of ~24kb

See the file `contracts/Constants.sol` for the registry of module IDs. There are 3 categories of modules:

- **Single-proxy modules:** These are modules that are only accessible by a single address. For example, market activation is done by invoking a function on the single proxy for the Markets module.
- **Multi-proxy modules:** These are modules that have many addresses. For example, each EToken gets an address, but any calls to them are dispatched to the single EToken module instance.
- **Internal modules:** These are modules that are called internally by the Euler system and don't have any public proxies. These are only useful for their upgrade functionality, and the ability to stub in non-production code during testing/development. Examples are the RiskManager and interest rate model (IRM) modules.

Since modules are invoked by `delegatecall`, they should not have any storage-related initialisation in their constructors. The only thing that should be done in their constructors is to initialise immutable variables, since these are embedded into the contract's bytecode, not storage. Modules also should not define any storage variables. In the rare cases they need private storage (ie interest rate model state), they should use unstructured storage.

Proxies

Modules cannot be called directly. Instead, they must be invoked through a proxy. All proxies are implemented by the same code: `contracts/Proxy.sol`. This is a very simple contract that forwards its requests to the main Euler contract address, along with the original `msg.sender`. The call is done with a normal `call()`, so the execution takes place within the Euler contract's storage context, not the proxy's.

Proxies contain the bare minimum amount of logic required for forwarding. This is because they are not upgradeable. They should ideally be small so as to minimise gas costs since many of them will be deployed (at least 2 per market activated).

The Euler contract ensures that all requests to it are from a known trusted proxy address. The only way that addresses can become known trusted is when the Euler contract itself creates them. In this way, the original `msg.sender` sent by the proxy can be trusted.

The only other thing that proxies do is to accept messages from the Euler contract that instruct them to issue log messages. For example, if an EToken proxy's `transfer` method is invoked, a `Transfer` event must be logged from the EToken proxy's address, not the main Euler address.

One important feature provided by the proxy/module system is that a single storage context (ie the main Euler contract) can have multiple possibly-colliding function ABI namespaces, which is not possible with systems like a conventional upgradeable proxy, or the Diamond standard. For example, Euler provides multiple ERC-20 interfaces but there is no worry that the `balanceOf()` methods of the ETokens and DTokens (which necessarily have the same selector) will collide.

For more details on the proxy protocol see `docs/proxy-protocol.md`.

Dispatching

Other than the proxies, `contracts/Euler.sol` is the only other code that cannot be upgraded. It is the implementation of the main Euler contract. Essentially its only job is to be a placeholder address for the Euler storage, and to `delegatecall()` to the appropriate modules.

When it invokes a module, `contracts/Euler.sol:dispatch()` appends some extra data onto the end of the `msg.data` it receives from the proxy:

- Its own view of `msg.sender`, which corresponds to the address of the proxy.
- The `msgSender` passed in from the (trusted) proxy, which corresponds to the original `msg.sender` that invoked the proxy.

The reason it appends onto the end of the data is so that this extra information does not interfere with the ABI decoding that is done by the module: Solidity's ABI decoder is tolerant of extra trailing data and will ignore it. This allows us to use the module interfaces output from `solc` directly when communicating with the proxies, while still allowing the functions to extract the proxy addresses and original `msg.sender`s as seen by the proxies.

Since the modules are invoked by `delegatecall()`, the proxy address is typically available to module code as `msg.sender`, so why is it necessary to pass this in to the modules? It's because batch requests allow users to invoke methods without going through the proxies (see below).

Modules

Installer

The first module used is the installer module. This module is used to bootstrap install the rest of the modules, and can later on be used to upgrade modules to add new features and/or fix bugs.

Currently the functions are gated so that the `upgradeAdmin` address is the only address that can upgrade modules. However, the installer module itself is also upgradeable, so this logic can be restricted as we move towards greater levels of decentralisation.

EToken

Every market has an EToken. This is the primary interface for the tokenisation of **assets** in the Euler protocol:

- **deposit:** Transfer tokens from your wallet into Euler, and receive interest earning tokens in return.
- **withdraw:** Redeem your ETokens for the underlying tokens, which are transferred from Euler to your wallet, along with any interest accrued.

Additionally, ETokens provide an ERC-20 compliant interface which allows you to transfer and approve transfers of your ETokens, as is typical.

Like Compound, but unlike AAVE, these tokens have static balances. That is, accrued interest will not cause the value returned from `balanceOf` to increase. Rather, that fixed balance entitles you to reclaim more and more of the underlying asset as time progresses. Although the AAVE model is conceptually nicer, experience has shown that increasing balance tokens causes a lot of pain to integrators. In particular, if you transfer X ETokens into a pool contract and later withdraw that same X, you have not earned any interest and the pool has some left over dust ETokens that typically aren't allocated to anyone.

A downside of the Compound model is that the values returned from `balanceOf` are in internal bookkeeping units and don't really have any meaning to external users. There is of course a `balanceOfUnderlying` method (named the same as Compound's method, which may become a defacto standard) that returns the amount in terms of the underlying and *does* increase block to block.

DToken

Every market also has a DToken. This is the primary interface for the tokenisation of **debts** in the Euler protocol:

- **borrow:** If you have sufficient collateral, Euler sends you the underlying tokens and issues you a corresponding amount of debt tokens.
- **repay:** Transfer tokens from your wallet in order to burn the DTokens, which reduces your debt obligation.

DTokens also implement a partially ERC-20 compliant interface. Unlike AAVE, where these are non-transferrable, DTokens *can* be transferred. The permissioning logic is the opposite of ETokens: While you can send your ETokens to anyone without their permission, with DTokens you can "take" anybody else's DTokens without their permission (assuming you have sufficient collateral). Similarly, just as you can approve another address to take some amount of your ETokens, you can use `approveDebt()` to grant another account permission to send you some amount of DTokens.

The `approveDebt()` name was used instead of the ERC-20 `approve()` due to concerns that some contracts might unintentionally allow themselves to receive "negative value" tokens.

As well as providing a flexible platform for debt trading and assignment, this system also permits easy transferring of debt positions between sub-accounts (see below).

Unlike ETokens, DToken balances *do* increase block-to-block as interest is accrued. This means that in order to pay off a loan in full, you should specify `MAX_UINT256` as the amount to pay off, so that all interest accrued at the point the repay transaction is mined gets repaid. Note that most Euler methods accept this `MAX_UINT256` value to indicate that the contract should determine the maximum amount you can deposit/withdraw/borrow/repay at the time the transaction is mined.

In the code you will also see `INTERNAL_DEBT_PRECISION`. This is because DTokens are tracked at a greater precision versus ETokens (27 decimals versus 18) so that interest compounding is more accurate. However, to present a common external decimals amount, this internal precision is hidden from external users of the contract. Note that these decimal place amounts remain the same even if the underlying token uses fewer decimal places than 18 (see the Decimals Normalisation section below).

Markets

This module allows you to activate new markets on the Euler protocol. Any token can be activated, as long as there exists a Uniswap 3 pair between it and the reference asset (WETH version 9 in the standard deployment, although any 18-decimal token could be used).

It also allows you to enter/exit markets, which controls which of your ETokens are used as collateral for your debts. This terminology was chosen deliberately to match Compound's, since many of our users will be familiar with Compound already.

Unlike Compound which keeps both an array and a mapping for each user, we only keep an array. Upon analysis we realised that almost every access to the mapping will be done inside a transaction that *also* scans through the array (usually as a liquidity check) so the mapping was (nearly) redundant and we thus could eliminate an SSTORE when entering a market. Furthermore, instead of a normal length-prefixed storage array, we store the length in a packed slot that is loaded for other reasons. This saves an additional SSTORE since we don't need to update the array length, and saves an SLOAD on every liquidity check (more important post Berlin fork). Taking it one step further, there is also an optimisation where the first entered market address is stored in a special variable that is packed together with this length.

Finally, the markets module allows external users to query for market configuration parameters (ie collateral factors) and current states (ie interest rates).

RiskManager

This is an internal module, meaning it is only called by other modules, and does not have a proxy entry point.

RiskManager is called when a market is activated, to get the default risk parameters for a newly created market. Also, it is called after every operation that could affect a user's liquidity (withdrawal, borrow, transferring E/DTokens, exiting a market, etc) to ensure that no liquidity violations have occurred. This logic could be implemented in BaseLogic, and would be slightly more efficient if so, but then upgrading the risk parameters would require upgrading nearly every other module.

In order to check liquidity, this module must retrieve prices of assets (see the Pricing section below).

Governance

This module lets a particular privileged address update market configuration parameters, such as the TWAP intervals, borrow and collateral factors, and interest rate models.

Eventually this logic will be enhanced to support EUL-token driven governance.

Liquidation

This module implements the liquidations system (see below).

Exec

This module implements some of the more advanced ways of invoking the Euler contract (described further below):

- Batch requests
- Deferred liquidity checks

It also has an entry point for querying detailed information about an account's liquidity status.

Swap

This module allows users to swap their deposited underlying tokens on Uniswap V3 and 1inch DEXes. Under the hood, the tokens are swapped directly from the pool, thus saving gas, which would normally be spent to withdraw and deposit back the traded assets. From the user's perspective the swap will change the balances of their eTokens.

Paired with deferred liquidity check (see below), the swap module allows users to put on one-click leveraged long and short positions on any collateral vs collateral asset pairs and one-click leveraged short positions on any collateral vs non-collateral pairs.

Available swap methods:

- all four methods of [UniswapV3 SwapRouter](#)
- full 1inch aggregator functionality, integrated through [1Inch API](#)

Note: The Swap module may become deprecated in favor of the new SwapHub module.

SwapHub

This module is a redesigned version of the `Swap` module. The improvements include:

- modular architecture, easily extendible to support additional DEXs
- support for rebasing and fee-on-transfer tokens, like stETH

`SwapHub` doesn't execute trades on its own. It relies on external swap handlers, which can be created by anyone and are not a part of the platform. The swap handlers are required to share a common interface, namely an `executeSwap` function which takes a `SwapParams` struct with the requested trade options. It is up to the user to select a swap handler to use by passing its address to the module's function calls. The swap handlers receive a transfer of the sold token before being invoked, and are expected to return both the bought tokens as well as any unused input. The module's only responsibility is to process the trade and verify its results (tokens sold and received) fall within user specified bounds in terms of amounts requested and slippage settings. To support exact output swaps for rebasing and fee-on-transfer tokens, it is possible to set a maximum difference of tokens requested vs received: `exactOutTolerance`.

Swap handlers

Currently there are 3 swap handlers available in the Euler repository, executing trades on:

- Uniswap V3 through [SwapRouter](#)
- 1Inch
- Uniswap V2 and V3 using Uniswap's [smart order router](#)

See [Swap Handlers](#) for more details.

Storage and Inheritance

Most of the modules inherit from `BaseLogic` which provides common lending logic related functionality. This contract inherits from `BaseModule`, which inherits from `Base`, which inherits from `Storage`.

Almost all the functions in the Base modules are declared as private or internal. This is necessary so that modules don't export unexpected functions, and also so that the solidity compiler can optimise away unneeded functions (not all modules use all functions).

`contracts/Storage.sol` contains the storage layout that is used by all modules. It is important that this match, since all modules are called with `delegatecall()` from the Euler contract context. Furthermore, it is important that upgrades preserve the storage ordering and offsets. The test `test/storage.js` has the beginning of an implementation to take the Solidity compiler's storage layout output and verify that it is consistent across upgrades. After we deploy our first version, we will "freeze" the storage layout and encode this in the `test/storage.js` test.

Pricing

Euler uses Uniswap 3 as its default pricing oracle. In order to ensure that prices are not vulnerable to snapshot manipulation, this requires using the time-weighted average price (TWAP) of a recent time period.

When a market is activated, the RiskManager calls `increaseObservationCardinalityNext()` on the uniswap pool to increase the size of the uniswap oracle's ring buffer to a minimum size. By default this size is 144, because this is on-average sufficient to satisfy a TWAP window of 30 minutes, assuming 12.5 second block times.

The Euler contracts will try to retrieve prices averaged over the per-instrument `twapWindow` parameter. If it cannot be serviced because the oldest value in the ring buffer is too recent, it will use the oldest price available (which we have ensured is at least 144 blocks old).

Our blog series describes our pricing system in more detail: <https://medium.com/euler-xyz/prices-and-oracles-2da0126a138>

Chainlink prices

To support the assets that do not have a WETH pair on Uniswap 3 or the pair has insufficient liquidity to provide secure TWAP oracle, Euler extended its pricing types to include Chainlink price feeds as the pricing source.

Chainlink is the most used data provider in the industry. It has a very good reputation and provides secure pricing feeds that are used by lending protocol industry leaders like Aave, Compound and others. Integration with Chainlink on Euler brings a reduction of the protocol's dependency on Uniswap. It lowers the oracle manipulation risks for those assets that have very little liquidity in WETH pair on Uniswap 3. Also, for all the assets that have the Chainlink oracle set as a price source, it reduces the gas usage for all the operations that require price fetching.

Learn more about Chainlink Price Feeds: <https://docs.chain.link/docs/using-chainlink-reference-contracts/>

Pegged prices

An exception to the Uniswap 3 and Chainlink pricing above is for assets that are equivalent to the reference asset. These assets can have a pricing type of "pegged" which indicates their price is always 1:1 with the reference asset. Currently the only asset that is pegged is the reference asset itself, which is WETH.

Price forwarding

Another exception is for assets that are equivalent to another asset, in which case the pricing can be "forwarded". This is currently only used for [pTokens](#).

Liquidity Deferrals

Normally, upon the completion of an operation that could fail due to a collateral violation (ie taking out a loan, withdrawing ETokens, exiting a market), the user's liquidity must be checked. This is done immediately after each operation by calling `contracts/BaseLogic.sol:checkLiquidity()`, which calls the internal RiskManager module's `requireLiquidity()` which will revert the transaction if the account is insufficiently collateralised.

However, this pattern causes some sequences of operations to fail unnecessarily. For example, a user must deposit ETokens and enter the market first, before taking out a loan, even if this is done in the same atomic transaction.

Furthermore, this can result in needless gas consumption. Consider a user taking out two loans in the same transaction: If the liquidity is checked each time, that means two separate liquidity checks are done, each of which requires accessing prices, looping over the entered markets list, and computing the liquidity (net of assets and liabilities, converted to the reference asset, and scaled by corresponding collateral and borrow factors).

Liquidity deferral is a general purpose solution to this. Users (which must be smart contracts, but see Batch Requests below) can call the `deferLiquidityCheck()` function in the Exec module. This function disables all liquidity checking for a specified account, and then re-enters the caller by calling the `onDeferredLiquidityCheck()` function on `msg.sender`. While this callback is executing, `checkLiquidity()` will not bother checking the liquidity for the specified account. After the function returns, the liquidity will then be checked.

As well as gas optimisation, and normal use-cases like refinancing loans, this also allows users to take out [flash loans](#).

For flash loans in particular, the protocol provides an adaptor contract `FlashLoan`, which complies with the [ERC-3156](#) standard. The adaptor internally uses liquidity deferral to borrow tokens and additionally requires that the loan is paid back in full within the transaction.

eToken <> dToken Symmetry

The primary operations on eTokens and dTokens are deposit/withdraw and borrow/repay, respectively. However, there is another interface that in some ways is more fundamental: mint/burn. These operations work on both eTokens and dTokens simultaneously. A mint operation creates both eTokens and dTokens in equivalent amounts, and assigns both to the user. A burn operation destroys eTokens and dTokens in equivalent amounts. These operations can be thought of as borrowing from yourself and repaying yourself. Alternatively, eTokens and dTokens can be thought of as a sort of matter and anti-matter, appearing from "nowhere" when minted (no underlying tokens required) and cancelling one another out of existence when burned.

All of the primary operations can be re-conceptualised as variants of mint and burn. For example, if there were no borrow function, it could be implemented in terms of a mint and a withdraw: the mint would create both eTokens and dTokens, and then the withdraw would destroy the eTokens leaving just dTokens.

- **deposit:** mint, repay
- **withdraw:** borrow, burn
- **borrow:** mint, withdraw
- **repay:** deposit, burn

There are some practical advantages with the mint and burn operations. One of which is that it becomes possible to repay a loan with eTokens instead of the underlying by burning a corresponding amount of eTokens together with the dTokens from the loan. This may be useful when the underlying token is illiquid -- perhaps because it has been paused -- but there is still a market for eTokens (incidentally, the stability pools described in the liquidation section are examples of eToken to eToken markets).

With the Swap module, Euler users can swap one eToken for another by performing an external swap on Uniswap. This saves users gas by avoiding deposit/withdraw overhead. When combined with mint, allows the construction of leveraged positions without any underlying token ever transiting user wallets.

Another area where the eToken/dToken symmetry is exposed is liquidations. Instead of the liquidator sending borrowed tokens and receiving collateral, Euler's liquidation flow simply transfers borrowed dTokens and collateral eTokens from the violator to the liquidator. The liquidator will typically withdraw the collateral, exchange it, and then repay to destroy the dTokens, but this is not strictly necessary. The liquidator could choose to retain the debt if, for example, there is insufficient available collateral tokens in the pool, or the swapping conditions are temporarily sub-optimal.

Sub Accounts

In order to prevent a problem inherent with borrowing multiple assets using the same backing collateral, it is sometimes necessary to "isolate" borrows. This is especially important for volatile and/or untrusted tokens that shouldn't have the capability to affect more stable tokens.

Euler implements this borrow isolation to protect lenders. However, this can lead to a suboptimal user experience. In the event a user wants to borrow multiple assets (and one or more are isolated), a separate wallet must be created and funded. Although there is nothing wrong with having many metamask accounts, this can be a bad experience, especially when they are using hardware wallets.

In order to improve on this, Euler supports the concept of sub-accounts. Every ethereum address has 256 sub-accounts on Euler (including the primary account). Each sub-account has a sub-account ID from 0-255, where 0 is the primary account's ID. In order to compute the sub-account addresses, the sub-account ID is treated as a `uint8` and XORed (exclusive ORed) with the ethereum address.

Yes, this reduces the security of addresses by 8 bits, but creating multiple addresses in metamask also reduces security: if somebody is trying to brute-force one of your $N > 1$ private keys, they have N times as many chances of succeeding per guess. Although it has to be admitted that the subaccount model is weaker because finding a private key for a subaccount gives access to *all* subaccounts, but there is still a very comfortable security margin.

You only need to approve Euler once per token, and then you can then deposit/repay into any of your sub-accounts. No approvals are necessary to transfer assets or liabilities between sub-accounts. Operations can also be done to multiple sub-accounts within a single transaction by using batch requests (see below).

The Euler UI will make it convenient to view at a glance the composition of your sub-accounts, and to rebalance collateral as needed to maintain your debt positions.

Batch Requests

Sometimes it is useful to be able to do multiple operations within a single transaction. This can be useful to reduce gas overhead by amortising the fixed transaction costs, especially if the operations involve multiple writes to the same storage slots (post Istanbul fork) and/or multiple reads from the same storage slots (post Berlin fork). It can also be useful to add atomicity to a sequence of operations (either they all succeed or they all fail).

In Ethereum these benefits are available to smart contracts, but not EOAs (normal private/public keypair accounts). This is unfortunate because many users can't/won't deploy smart contract wallets.

As a partial solution to this, the `contracts/modules/Exec.sol:batchDispatch()` function allows a group of Euler interactions to be executed within a single blockchain transaction. This is a "partial" solution since users cannot execute arbitrary logic in between the interactions, but is nonetheless sufficient for a wide variety of use-cases.

For example, in order to provide collateral to Euler, two separate steps must occur: Depositing into the EToken, and entering the market for that EToken. Rather than requiring users to make two separate transactions, or implementing a hypothetical `depositAndEnter()` function (which would imply a combinatorial explosion of method combinations), batch transactions can be employed.

Additionally, liquidity checks can be deferred on one or more accounts in a batch transaction. This can provide significant gas cost savings and can allow flash-loan-like rebalancing without the need for a smart contract. We are planning on implementing a built-in swap functionality that will convert one EToken to another by performing a swap on Uniswap. This would be very gas-efficient since tokens do not need to be moved from external wallets to and from Euler's wallet, and would also allow an easy way to create leveraged positions, even for EOA users.

Reserves

Similar to Compound and AAVE, a proportion of the interest earned in a pool is collected by the protocol as a fee. Euler again uses the same terminology as Compound, calling the aggregate amount of collected fees the "reserve". These fees are controlled by governance, and may be paid out to EUL token holders, used to compensate lenders should pools become insolvent, or applied to other uses that benefit the protocol.

Reserves provide a buffer of funds that can cover losses due to positions that are too small to liquidate, and can also be a source of funds for governance to implement insurance, distribute to EUL stakers, or apply to some other purpose that benefits the protocol.

Unlike Compound where the reserves are denominated in the underlying, Euler's reserves are stored in the internal bookkeeping units that represent EToken balances. This means that they accrue interest over time, as with any other EToken deposit. Of course, Compound governance could periodically choose to withdraw their reserves and re-deposit them in the pool to earn this interest, but in Euler it happens automatically and continuously. Similar to Euler, AAVE deposits earned reserve interest into a special treasury account that owns the aTokens, however this is much less efficient than the special-cased reserves model of Compound/Euler, involving several cross-contract calls. In Euler, the reserves overhead is primarily two SSTORE operations, to slots that would be written to anyway.

When we issue "eTokens" to the reserve, it inflates the eToken supply (making them less valuable). However, we only do this after we increase `totalBorrows`, ensuring that the inflation is less than what was earned as interest, proportional to the reserve fee configured for that asset.

Derivation of Reserves Formulas

Compound

In Compound, the assets owned by CToken holders are the total "cash" (unallocated underlying units in the pool) plus the total outstanding borrows (which increase as interest is accrued), minus the total reserves (which are owned by Compound governance):

$$\text{assetsCompound} = \text{totalCash} + \text{totalBorrows} - \text{totalReserves}$$

Prior to most operations, the `accruedInterest` since the last operation is computed. In Compound, this is added to `totalBorrows`, and `accruedInterest * reserveFactor` is added to `totalReserves`, resulting in new value for the assets:

$$\text{newAssetsCompound} = \text{assets} + \text{accruedInterest} - (\text{accruedInterest} * \text{reserveFactor})$$

Compound's `totalReserves` is in units of the underlying, so it does not accrue interest, however governance could vote to withdraw these reserves and re-deposit them in exchange for CTokens, which would.

If `totalSupply` is the sum of the balances of all CToken holders, the exchange rate between these CToken balances and the underlying token is:

$$\text{newExchangeRateCompound} = \text{newAssetsCompound} / \text{totalSupply}$$

Euler

After applying interest, the new exchange rate is the same for both Compound and Euler, although how it is computed differs. Rather than deducting the reserve fees from the `newAssets`, Euler increases `totalSupply`. So the new value for assets is computed as though no reserve fees were being deducted:

$$\text{newAssetsEuler} = \text{assets} + \text{accruedInterest}$$

In Euler, reserves are tracked in EToken units which means they earn interest automatically. When interest is accrued it is added to `totalBorrows` in the same way as Compound. But then, instead of adding the collected fee to `totalReserves` (causing it to be deducted from `newAssetsCompound`), a special number of new ETokens are minted and credited to the reserves, which increases `totalSupply`. This number of newly minted ETokens is selected so as to inflate the supply just enough to divert a `reserveFactor` proportion of the interest away from EToken holders to the reserves.

In order to show that this results in the same exchange rate as Compound's method, we can derive the algorithm that Euler uses to compute `newTotalSupply` using Compound's value:

```
newExchangeRate = newAssetsEuler / newTotalSupply
```

```
newTotalSupply = newAssetsEuler / newExchangeRate
```

Substituting in Compound's value for `newExchangeRate` :

```
newTotalSupply = newAssetsEuler / (newAssetsCompound / totalSupply)
```

```
newTotalSupply = newAssetsEuler / ((assets + accruedInterest - (accruedInterest * reserveFactor)) / totalSupply)
```

Simplifying:

```
newTotalSupply = totalSupply * newAssetsEuler / (newAssetsEuler - (accruedInterest * reserveFactor))
```

Finally, the reserve balance (denominated in ETokens) is increased by `newTotalSupply - totalSupply` .

This is the algorithm used in the code, except for operation re-ordering done to avoid rounding truncation.

PTokens

"Protected" tokens exist to provide users the option to deposit tokens and use them as collateral, while not permitting them to be loaned out. PTokens provide users with additional safety, at the expense of not earning any interest on the deposited asset. PToken depositors don't need to worry about a pool becoming insolvent, or that their assets will be loaned out when they wish to retrieve them.

Rather than applying this as universal setting on an asset, it is up to the user to decide whether they want to protect their collateral. Since Euler only supports one eToken per underlying asset, a token wrapper contract is used. Users first wrap their underlying tokens into pTokens, and then deposit these pTokens into Euler, receiving "epTokens" which can then be used for collateral.

Another use-case of pTokens is to prevent tokens from being borrowed to perform governance-related attacks. Because the borrowing-prevention check happens inside `increaseBorrow()` (and not in `checkLiquidity()`), pTokens cannot even be flash borrowed.

Interest rate models

FIXME: describe

Liquidations

Borrowers must maintain sufficient collateral in order to support their borrows. In particular, each account must maintain a "health score" above 1. The health score is computed by dividing the account's [risk-adjusted](#) collateral value by its risk-adjusted liability value. Since the collateral factor decreases the effective value of the collateral, and the borrow factor increases the effective value of the liability, when the health score is 1, then the account is still technically solvent (assets are worth more than liability), but the account is said to be in "violation".

When an account is in violation, the `liquidate()` method of the Liquidation module can be invoked by anyone (except by the violating account itself, to avoid aliasing bugs). The account invoking this method is called the "liquidator". This method does two things:

1. Transfers some DTokens from the violator to the liquidator. This represents debt that is being taken over by the liquidator.
2. Transfers some ETokens from the violator to the liquidator. This represents the collateral being seized by the liquidator in exchange for taking the debt.

Because of the collateral and borrow factors, reducing the assets and liabilities in equal values (relative to the reference asset ETH) will result in a user's health score increasing (except in certain pathological circumstances). The amount of DTokens/ETokens is selected to be just enough to return a user to a higher health score, by default 1.25. This is what is referred to as a [soft liquidation](#), in contrast with the simpler method of liquidating a fixed proportion of the loan.

Since the liquidator is taking on debt, the liquidating account's liquidity must be checked after a liquidation. Typically a liquidator will be a smart contract so it can atomically perform other operations in addition to the liquidation, and in particular can [defer the liquidity check](#) to later in the same transaction, allowing "flash liquidations".

Liquidation bots that wish to operate without capital requirements may follow the following pattern for liquidations:

- Defer liquidity check
- Liquidate an account, receiving underlying DTokens and collateral ETokens
- Withdraw enough of the collateral to repay the debt
- Convert this collateral on a decentralised exchange such as Uniswap
- Repay the debt, zeroing-out the DToken balance

At this point, there is no outstanding debt so the deferred liquidity check will succeed. Any left over ETokens are profit, and can be held by the liquidation smart contract or transferred to another account.

Dynamic discounts

If the seized debt and collateral were each worth the same in terms of a reference asset (for example ETH), then there would be no point in performing liquidations. In order to incentivise liquidators, the amount of collateral seized is increased by a certain factor. Since the violator is effectively receiving a lower price for purchasing collateral, this factor is known as a "discount".

Euler uses a dynamic value for this discount rather than a fixed value. The discount increases by how far the violator's health score has decreased below 1. For example, if an account's health score has fallen to 0.98, then the discount received is $1 - 0.98 = 0.02$, or 2%.

Euler uses Uniswap3 TWAPs for the price feeds of all assets which has the property in which the prices on the protocol change smoothly over time. This is central to how the dynamic discount is designed to work, and creates a Dutch auction-like mechanism that finds the lowest possible market-clearing discount level.

Dynamic discount example

Let's suppose a borrower has a health score of 1.1 and then a large swap is performed on a borrowed asset which increases its current price on Uniswap significantly. Immediately after this swap (ie, throughout the rest of the block the swap was included in) then the TWAP of the asset is unchanged (since no time has passed). This means that the account's health score is also unchanged.

However, as time goes on and the weight of the new price increases, then the TWAP will increase which means that the health score will decrease (the liability is becoming more valuable). Note that this in fact happens at second-granularity. If the swap was large enough, then at some point in the future the averaged price will be such that the health score is exactly equal to 1. Assuming that the TWAP hasn't yet caught up with the current price, then in any subsequent block the health score will be below 1 and there will therefore be a liquidation opportunity.

Now, at this point, the discount will be extremely small. If the health score is 0.999 then the discount would be a mere 0.1%. This level of discount is most likely not enough to make a liquidation worth-while. First of all, because the prices used to calculate the equivalent values of assets are TWAPs, they don't yet take into account the current (non-averaged) price of the underlying asset. Secondly, the discount must compensate the liquidator for any execution slippage, gas costs, and other operational overhead.

All of this is to say that it is unlikely that anybody will perform the liquidation at this point. But as time goes on and the TWAP increases, the health score decreases and the discount improves. At some instant in time a bot will determine that the current discount will result in a profitable liquidation. At this point it has two options: it can either execute the liquidation and take the small profit, or wait until the discount increases further. If the bot waits then it risks losing the liquidation opportunity to another liquidator.

Reserves

When a liquidation happens, a small amount of additional borrowed asset beyond the soft-liquidation amount must be repaid by the liquidator (which is compensated by a corresponding extra discount amount). This additional amount is credited to the borrowed asset's reserves.

This is done to pad the reserves for assets that are frequently liquidated, since this may be indicative of volatile asset which may have a higher risk of accruing bad debt.

Another option could have been to pad the reserves of the collateral asset, however on Euler not all assets can be used as collateral so many pools would have no opportunity for their reserves to be increased in this manner.

Front-running protection

The Dutch auction-like mechanism described above can provide a discount to anyone who calls `Liquidate()`. This means that liquidations are permission-less which is desirable for various reasons, not least of which because liquidations cannot be censored.

However, permissionless liquidations are often affected by so-called "front-running". This is when a bot sees a profitable new transaction and submits it for themselves with a higher gas price. While front-running isn't directly a problem for the protocol, it can be detrimental for the ecosystem:

- The capture of value by miners means that it is less profitable to operate liquidation bots, so there may be fewer people doing so, and those who do may be less aggressive.
- A lot of resources are wasted on failed transactions and bidding up the gas prices of liquidations.

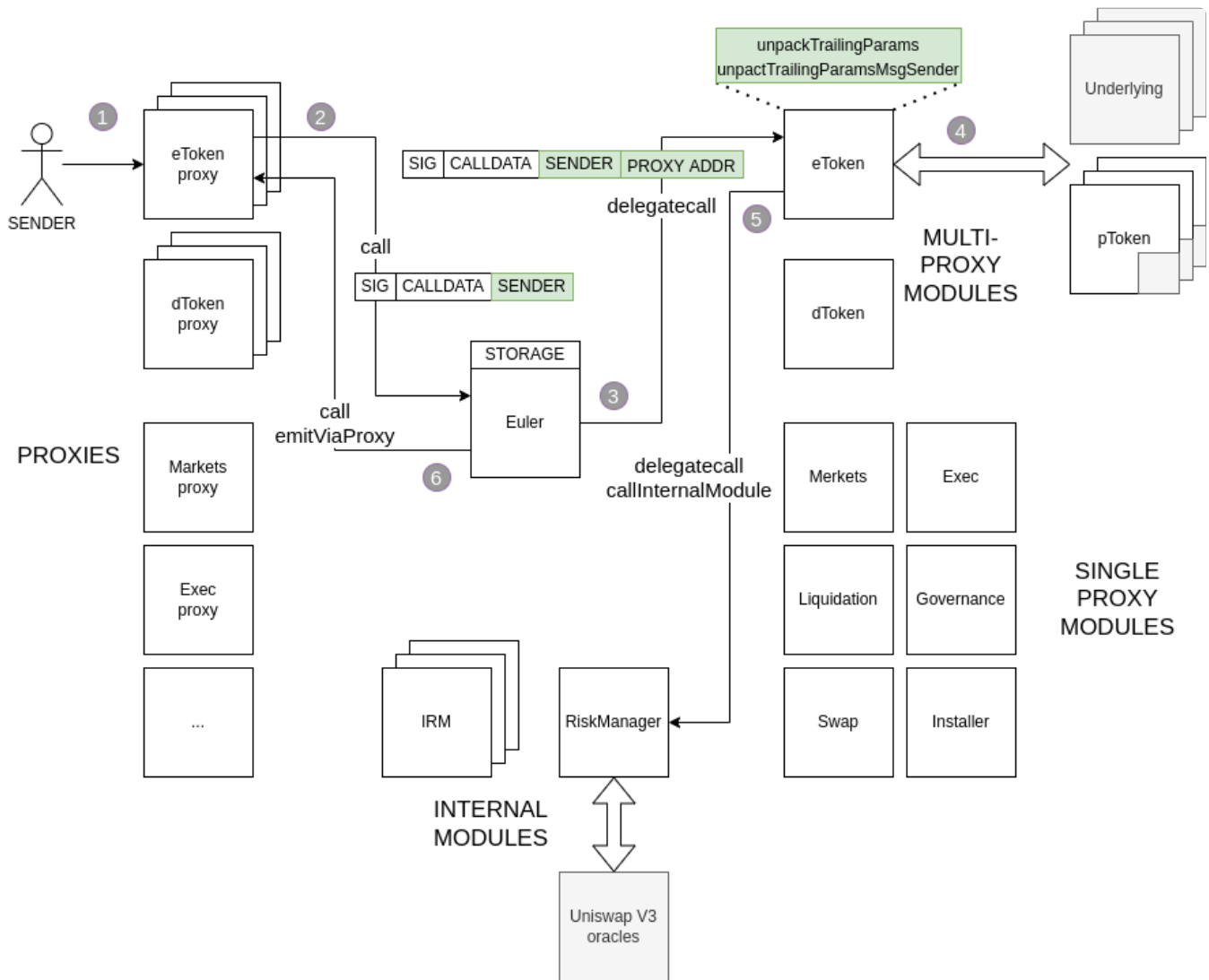
In Euler we would like to reward the operators of liquidation bots instead of miners, and reduce their level of reward to the minimal level that a competitive market will bear. In order to do this, an extra "bonus" is applied to the discount for users who have assets deposited into the Euler protocol. This bonus works by increasing the slope of the discount. For instance, if a user has enough assets deposited to provide a 2x bonus, then instead of getting a 1% discount, they would get a 2% discount.

If you are operating a liquidation bot, you can become profitable before front-running bots by keeping a balance of non-zero collateral factor assets. In order to receive a full bonus, your risk-adjusted collateral should be at least equal to the risk-adjusted value of the liquidation you are processing (anything less will result in a smaller bonus).

With bonuses and the Dutch action mechanism, our hope is that gas auctions will be rare, and the majority of the value of liquidations will accrue to users who benefit the protocol by supplying assets.

Note that the liquidity used to claim a bonus must be held in the Euler contracts for a period of time. The full averaged liquidity will be achieved after a day (see [Average Liquidity Tracking](#)). This means that no bonus will be applied if someone atomically supply liquidity, liquidates, and then withdraws.

Functional Diagram



The functional diagram depicts the smart contract architecture and how proxies, the Euler contract and modules relate to each other.

Let's follow an execution of the `deposit` function on an `eToken`.

1. The user calls `deposit` on an `eToken proxy` of the underlying she wants to deposit to Euler.
2. The proxy attaches `msg.sender` to the call data and calls `dispatch` on `Euler` contract in the `fallback` function.
3. Through a lookup of the proxy address `Euler` finds the currently installed module implementation for an `eToken` and delegate-calls it, attaching the proxy address to the call data.
4. The `deposit` function in `EToken` module contract unpacks the trailing params from call data to determine the original sender's address. The proxy address determines the underlying of the `eToken`, which the `deposit` function pulls from the user's wallet. An underlying of an `eToken` can be a `pToken`, which wraps a collateral asset.
5. Internal modules `IRM` and `RiskManager` are delegate-called to compute the new interest rate and check the account health.
6. Finally `emitViaProxy` function is called to emit standard `Transfer` event from the proxy address in compliance with ERC20. The proxy only allows this if the `msg.sender` is the `Euler` contract.

Misc Details

Average Liquidity Tracking

In order to provide a liquidation discount that privileges investors in the system, Euler can optionally track the liquidity of an account. In order to opt-in to this, an account should call the `trackAverageLiquidity()` function in the `exec` module. This will cause most operations such as depositing and withdrawing to consume more gas, but will make the account eligible for extra discount privileges, if it participates in liquidations.

The actual value that is tracked is the risk adjusted liquidity, that is, after applying collateral factors and borrow factors. So, only assets with non-zero collateral factors will contribute to the liquidity. Similarly, outstanding borrows will reduce the average liquidity.

In order to prevent a user (or front-running bot) from simply depositing a large amount prior to a liquidation (perhaps using a flash loan), the average value of the liquidity over a period of time is tracked. For example, immediately after depositing for the first time, your average liquidity will be 0. Only after `AVERAGE_LIQUIDITY_PERIOD` seconds have elapsed will your full liquidity value be reflected.

The averaging is implemented with an exponential moving average, that is updated with the current liquidity value before any operation (such as deposit) is performed. This is not perfect, since the average liquidity will not reflect price movements between updates. Also, a user could opportunistically cause an update when prices are exceptionally high or low, although the prices are of course TWAPs so are more difficult to manipulate.

We don't believe these limitations will be significant with respect to the use case described above. That said, if enabled, the average liquidity for an account is available with `exec.getUpdatedAverageLiquidity()`, so long as your application can accept the limitations described above.

Decimals

The ERC-20 specification allows contracts to choose the number of decimal places that a token supports. This is now widely regarded as problematic, since it causes a lot of annoying integration work (see ERC-777 for an interesting alternative).

Rather than exposing this to our system, we have decided to normalise the decimals up to 18 for all tokens (Euler for now does not support tokens with > 18 decimals). As well as simplifying our contract and off-chain logic, this allows more precise interest accrual.

Rounding

Debts are always rounded *up* to the smallest possible external unit (1 "wei" on 18 decimal tokens). This means that after any interest at all is accrued (ie, after 1 second), borrowers already owe at least 1 unit. When you repay, this extra fraction is added to the EToken pool. This works because debt amounts are tracked at a higher precision (27 decimals) than the external units (0-18 decimals).

Compounding behaviour

Unlike Compound, where the compounding occurs whenever a user interacts with a token, Euler compounds deterministically every second. The amounts owed/earned are independent of how often the contract is interacted with, except of course for interactions that result in interest rate changes. As mentioned, the compounding precision is done to 27 decimal places, according to the current interest rate in effect (determined by the interest rate model).

In the Compound system, interest accumulators are opportunistically updated for all operations that affect assets, which is necessary because this is how compounding is achieved (simple interest being charged between updates). Because Euler precisely tracks the per-second compounded balance, there is no advantage to updating the accumulator frequently, and therefore Euler does it lazily only when it actually needs to (before operations that affect debt obligation balances). So as well as being more accurate, this means that fewer storage writes are needed.

External access to interest accumulators

There is a method in the markets module, `interestAccumulator()` that retrieves the current interest accumulator for an asset. Because the accumulators are updated lazily as described above, rather than just returning the stored value, this method computes the updated accumulator given the most recent block's timestamp (sometimes called a "counterfactual" value).

Although the returned values are in opaque internal units, they can be used to determine the accumulated interest over time by comparing snapshots. This is sort of like using Uniswap-style "TWAPs": By dividing a recent snapshot by an older snapshot, the actual amount of interest collected between those two time periods is computed.

Unusual/Malicious tokens

We try to work as well as possible with "deflationary" tokens. These are tokens where when you request a transfer for X, fewer than X tokens are actually transferred. For these, we check the Euler contract's balance in the token before and after to determine how much was transferred.

Relatedly, on some tokens the `balanceOf` method can return different results with no intervening operations. In this case, the total pool available to owners of ETokens of these underlyings will be affected, but the protocol itself will not be (assuming such tokens have collateral factors of 0, which is the default).

Since we allow arbitrary tokens to be activated, our threat model is larger than that of Compound/AAVE. We need to worry about misbehaving tokens, even one-off tokens written specifically to attempt theft from the protocol. See the file `docs/attacks.md` for some more notes on the threat modelling.

Tokens may return extremely large values in an attempt to cause math overflows. This could be disastrous, especially if a user could cause their own liquidity checks to fail. In this case, a user could create an unliquidatable position. To prevent this, when we receive a very large result from `balanceOf`, we treat that result as though it were 0 (which a malicious token could also do of course). This way liquidity checks will at least succeed, allowing non-malicious collaterals to be liquidated.

Proxy Protocol

Proxies are non-upgradeable stub contracts that have two jobs:

- Forward method calls from external users to the main Euler contract
- Receive method calls from the main Euler contract and log events as instructed

Although proxies themselves are non-upgradeable, they integrate with Euler's module system, which does allow for upgrades.

The following protocols all use custom assembly routines instead of the solidity ABI encoder/decoder. While we don't take this lightly, the measured overhead of keeping this in pure solidity was too high. In order to make up for this otherwise regrettable use of assembly, this document explains the protocols in detail.

Proxy -> Euler

To the calldata received in a fallback, the proxy prepends the 4-byte selector for `dispatch()` (`0xe9c4a3ac`), and appends its view of `msg.sender`:

```
[dispatch() selector (4 bytes)][calldata (N bytes)][msg.sender (20 bytes)]
```

This data is then passed to the Euler contract with a `CALL` (not `DELEGATECALL`).

Euler -> module

In the `dispatch()` method, the Euler contract looks up *its* view of `msg.sender`, which corresponds to the proxy address.

The presumed proxy address is then looked up in the `trustedSenders` mapping, which must exist otherwise the call is reverted. It is determined to exist by having a non-zero entry in the `moduleId` field (modules must have non-zero IDs).

The only way a proxy address can be added to `trustedSenders` is if the Euler contract itself creates it (using the `_createProxy` function in `contracts/Base.sol`).

In the case of a single-proxy module, the same storage slot in `trustedSenders` will also contain an address for the module's implementation. If not (ie multi-proxy modules), then the module implementation must be looked up with an additional lookup in the `moduleLookup` mapping. This is because during an upgrade, single-proxy modules just have to update this one spot, whereas multi-proxy modules would otherwise need to update every corresponding entry in `trustedSenders`.

At this point we know the message is originating from a legitimate proxy, so the last 20 bytes can be assumed to correspond to an actual `msg.sender` who invoked a proxy. The length of the calldata is checked. It should be at least `4 + 4 + 20` bytes long, which corresponds to:

- 4 bytes for the `dispatch()` selector.
- 4 bytes for selector used to call the proxy (non-standard ABI invocations and fallback methods are not supported in modules).
- 20 bytes for the trailing `msg.sender`.

The Euler contract then takes the received calldata and strips off the `dispatch()` selector, and then appends *its* view of `msg.sender` (`caller()` in assembly), which corresponds to the proxy's address. This results in the following:

```
[original calldata (N bytes)][original msg.sender (20 bytes)][proxy addr (20 bytes)]
```

This data is then sent to the module implementation with `DELEGATECALL`, so the module implementation code is executing within the storage context of the main Euler contract.

The module implementation will unpack the original calldata using the solidity ABI decoder, ignoring the trailing 40 bytes.

Modules are not allowed to access `msg.sender`. Instead, they should use the `unpackTrailingParamMsgSender()` helper in `contracts/BaseModule.sol` which will retrieve the message sender from the trailing calldata.

When modules need to access the proxy address, there is a composite helper `unpackTrailingParams()` that returns both trailing params. `msg.sender` is still not allowed to be used for this, since modules can be invoked via a batch dispatch, instead of via the proxy.

module -> Proxy

When a module directly emits a log (or "event" at the solidity level) it will happen from the main Euler contract's address. This is fine for many logs, but not in certain cases like when a module is implementing the ERC-20 standard. In these cases it is necessary to emit the log from the address of the proxy itself.

In order to do this, the Euler contract (specifically one of the modules) does a `CALL` to the proxy address.

When the proxy sees a call to its fallback from the Euler contract (its creator), it knows not to re-enter the Euler contract. Instead, it interprets this call as an instruction to issue a log message. This is the format of the calldata:

```
[number of topics as uint8 (1 byte)][topic #i (32 bytes)]{0,4}[extra log data (N bytes)]
```

The proxy unpacks this message and executes the appropriate log instruction, `log0`, `log1`, etc, depending on the number of topics.

Numeric Limits

amounts

uint112

- Maximum sane amount (result of balanceOf) for external tokens
- Uniswap2 limits token amounts to this
- Spec: For an 18 decimal token, more than a million billion tokens (1e15)

small amounts

uint96

- For holding amounts that we don't expect to get quite as large, in particular reserve balances
- Can pack together with an address in a single slot
- Spec: For an 18 decimal token, more than a billion tokens (1e9)

debt amounts

uint144

- Maximum sane amount for debts
- Packs together with an amount in a single storage slot
- Spec: Should hold the maximum possible amount (uint112) but scaled by another 9 decimal places (for the internal debt precision)
 - Actual: 2e16

prices

- Minimum supported price:
 - Fraction: $1e3 / 1e18 = 1e-15$
 - Tick: -345405
 - sqrtPriceX96: 2505418623681149822473
- Maximum supported price:
 - Fraction: $1e33 / 1e18 = 1e15$
 - Tick: 345405
 - sqrtPriceX96: 2505410343826649584586222772852783278

The supported price range was chosen for the following reason:

- The maximum price squared fits in a uint256: $6e73 < 1e77$
 - Not necessary to use FullMath library
- The maximum supported price times the maximum supported amount fits within a uint256: $5e66 < 1e77$
 - Also holds with debt and its extra 9 digits of precision: $5e75 < 1e77$

interestRate

int96

- "Second Percent Yield"
- Fraction scaled by $1e27$
 - Example: $10\% \text{ APR} = 1e27 * 0.1 / (86400 * 365) = 1e27 * 0.000000003170979198376458650 = 3170979198376458650$
- Spec: 1 billion % APR, positive or negative

interestAccumulator

uint256

- Starts at $1e27$, multiplied by $(1e27 + \text{interestRate})$ every second
- Spec: 100% APR for 100 years

```
-> 2^256
~= 1.1579208923e+77
-> 10^27 * (1 + (100/100 / (86400*365)))^(86400*365*100)
~= 2.6881128798e+70
```

moduleId

uint32

- One per module, so this is way more than needed
- Divided into 3 sections
 - $<500_000$: Public single-proxy
 - $=500_000$ and $<1_000_000$: Public multi-proxy
 - $=1_000_000$: Internal
- Spec: A dozen or so modules, with room to grow in all sections

collateralFactor/borrowFactor

`uint32`

- Fraction between 0 and 1, scaled by $2^{32} - 1$
- Spec: At least 3 decimal places (overkill)

SDK

[Euler-sdk](#) is a JavaScript SDK for the Euler platform. It is used in production in the Euler Dapp and a few other applications, although it's currently considered an alpha software.

See the [Github repo](#) for docs and examples.

Please feel free to open a pull request, create an issue in the repo, or reach out to us with feature requests and suggestions in the `#mainnet-development` [Discord](#) channel.

Subgraph

Subgraph

Links

- Querying the subgraph: <https://thegraph.com/docs/en/developer/query-the-graph/>

Mainnet

- The Graph: <https://thegraph.com/hosted-service/subgraph/euler-xyz/euler-mainnet>
- API: <https://api.thegraph.com/subgraphs/name/euler-xyz/euler-mainnet>

About

- All amounts have a fixed decimal precision of 1e18.
- Are timestamps are represented as [unix timestamps](#).
- USD and ETH exchange rates of all assets are pulled through Euler directly
- All ratios have a fixed decimal precision of 1e27.

Entities

Asset

Contains information about all Euler markets, pulled from EulerGeneralView contract.

```
type Asset @entity {
  "asset_address"
  id: ID!
  "Block hash at which asset was created"
  blockHash: Bytes!
  "Block number at which asset was created"
  blockNumber: Int!
  "Timestamp at which asset was created"
  timestamp: Int!
  "Block hash at which asset was last updated"
  updatedBlockHash: Bytes!
  "Block number at which asset was last updated"
  updatedBlockNumber: Int!
  "Timestamp at which asset was last updated"
  updatedTimestamp: Int!
  "Tx hash at which asset was created"
  transactionHash: Bytes!
  "Tx origin at which asset was created"

  transactionOrigin: Bytes!
  dTokenAddress: Bytes!
  eTokenAddress: Bytes!
  pTokenAddress: Bytes!
  symbol: String!
  name: String!
  decimals: BigInt!
  totalSupply: BigInt!
  totalBalances: BigInt!
  totalBalancesUsd: BigInt!
  totalBalancesEth: BigInt!
  totalBorrows: BigInt!
  totalBorrowsUsd: BigInt!
  totalBorrowsEth: BigInt!
  reserveBalance: BigInt!
  reserveBalanceEth: BigInt!
  reserveBalanceUsd: BigInt!
  reserveFee: BigInt!
  borrowAPY: BigInt!
  supplyAPY: BigInt!
  twap: BigInt!
  twapUsd: BigDecimal!
  "Deprecated in favor of twapUsd. Do not use."
  twapPrice: BigDecimal!
  twapPeriod: BigInt!
  currPrice: BigInt!
  currPriceUsd: BigInt!
  pricingType: Int!
  pricingParameters: BigInt!
  pricingForwarded: Bytes!
  collateralValue: BigInt!
  liabilityValue: BigInt!
  numBorrows: BigInt!
```

```

borrowIsolated: Boolean!
poolSize: BigInt!
interestRate: BigInt!
interestAccumulator: BigInt!
config: AssetConfig
}

```

Sometimes, governance will manually set asset configuration, it is displayed in the AssetConfig entity. If an asset has a null AssetConfig, it is safe to assume that it was never explicitly set and is isolated.

For further tier information, refer to [Tier Methodology](#).

`borrowFactor` and `collateralFactor` can be transformed in a decimal fraction by dividing by `4e9`.

```

type AssetConfig @entity {
  "asset_address"
  id: ID!
  twapWindowInSeconds: Int!
  borrowFactor: BigInt!
  borrowIsolated: Boolean!
  collateralFactor: BigInt!
  tier: String!
}

```

Account

Account that transacts on Euler. Could be a sub-account or a main account.

```

type Account @entity {
  "account_address"
  id: ID!
  createdTimestamp: Int!
  topLevelAccount: TopLevelAccount!
  balances: [Balance!]
  balanceChanges: [BalanceChange!]
  balanceChangesCount: Int!
}

```

A Balance represents the current amount of each assets held by an account.

```

type Balance @entity {
  "account_address:underlying"
  id: ID!
  account: Account!
  amount: BigInt!
  asset: Asset!
}

```

A BalanceChange is a transaction within the platform. `type` can be one of the following values:

- borrow
- deposit
- withdraw
- repay

```
type BalanceChange @entity {
  "transaction_hash:event_log_index"
  id: ID!
  transactionHash: Bytes!
  type: String!
  account: Account!
  topLevelAccount: TopLevelAccount!
  amount: BigInt!
  amountUsd: BigDecimal!
  timestamp: Int!
  asset: Asset!
```

TopLevelAccount

Aggregate of the sub-accounts associated with a wallet. TopLevelAccount id corresponds to main sub-account id.

```
type TopLevelAccount @entity {
  "account_address"
  id: ID!
  createdTimestamp: Int!
  accounts: [Account!]
  balances: [TopLevelAccountBalance!]
  balanceChanges: [BalanceChange!]
  balanceChangesCount: Int!
}
```

```
type TopLevelAccountBalance @entity {
  "top_level_account_address:underlying"
  id: ID!
  topLevelAccount: TopLevelAccount!
  amount: BigInt!
  asset: Asset!
}
```

EulerMarketStore

Contains list of all active markets on Euler.

```
type EulerMarketStore @entity {
  id: ID!
  markets: [Asset!]
}
```

EulerOverview

Aggregated metrics for all markets as a whole.

```
type EulerOverview @entity {
  id: ID!
  reserveBalanceUsd: BigInt!
  reserveBalanceEth: BigInt!
  totalBalancesUsd: BigInt!
  totalBalancesEth: BigInt!
  totalBorrowsUsd: BigInt!
  totalBorrowsEth: BigInt!
}
```

Liquidation

Contains all liquidation transactions.

`liquidator` and `violator` fields correspond to account addresses.

```
type Liquidation @entity {
  "transaction_hash:event_log_index"
  id: ID!
  timestamp: Int!
  transactionHash: Bytes!
  liquidator: Bytes!
  violator: Bytes!
  asset: Asset!
  collateralAsset: Asset!
  repay: BigInt!
  repayUsd: BigDecimal!
  harvest: BigInt!
  yieldUsd: BigDecimal!
  healthScore: BigInt!
  discount: BigInt!
  baseDiscount: BigInt!
}
```

Governance

`GovConvertReserve` and `GovSetReserveFee` contain all on-chain information about governance.

```
type GovConvertReserve @entity {
  "transaction_hash:event_log_index"
  id: ID!
  timestamp: Int!
  blockNumber: Int!
  transactionHash: Bytes!
  amount: BigInt!
  amountUsd: BigDecimal!
  asset: Asset!
  recipient: Bytes!
}
```

```
type GovSetReserveFee @entity {
  "transaction_hash:event_log_index"
  id: ID!
  timestamp: Int!
  blockNumber: Int!
  transactionHash: Bytes!
  reserveFee: Int!
  asset: Asset!
}
```

Hourly/Daily/MonthlyAssetSnapshot

Snapshot of Asset entity at the end of every hour, day and month.

Useful for querying historical data on markets.

```

type HourlyAssetSnapshot @entity {
  "start_of_hour_timestamp:asset_address"
  id: ID!
  asset: Asset!
  "Block number at which snapshot was created"
  blockHash: Bytes!
  "Block hash at which snapshot was created"
  blockNumber: Int!
  "Timestamp at which snapshot was created"
  timestamp: Int!
  "Tx hash at which asset was created"
  transactionHash: Bytes!
  "Tx origin at which asset was created"
  transactionOrigin: Bytes!
  totalSupply: BigInt!
  totalBalances: BigInt!
  totalBalancesUsd: BigInt!
  totalBalancesEth: BigInt!
  totalBorrows: BigInt!
  totalBorrowsUsd: BigInt!
  totalBorrowsEth: BigInt!
  reserveBalance: BigInt!
  reserveBalanceEth: BigInt!
  reserveBalanceUsd: BigInt!
  reserveFee: BigInt!
  borrowAPY: BigInt!
  supplyAPY: BigInt!
  twap: BigInt!
  twapUsd: BigDecimal!
  twapPeriod: BigInt!
  currPrice: BigInt!
  currPriceUsd: BigInt!
  pricingType: Int!
  pricingParameters: BigInt!
  pricingForwarded: Bytes!
  collateralValue: BigInt!
  liabilityValue: BigInt!
  numBorrows: BigInt!
  borrowIsolated: Boolean!
  poolSize: BigInt!
  interestAccumulator: BigInt!
  interestRate: BigInt!
}

```

Hourly/Daily/MonthlyAssetStatus

This entity has been deprecated in favor of Hourly/Daily/MonthlyAssetSnapshot

Aggregated metrics of a specific market over hourly, daily and monthly time period.

```
type HourlyAssetStatus @entity {
  "start_of_hour_timestamp:asset_address"
  id: ID!
  timestamp: Int!
  totalBalances: BigInt!
  totalBorrows: BigInt!
  reserveBalance: BigInt!
  poolSize: BigInt!
  interestAccumulator: BigInt!
  interestRate: BigInt!
  twapPrice: BigDecimal!
  twapUsd: BigDecimal!
  asset: Asset!
}
```

Hourly/Daily/MonthlyRepay

Aggregated metrics for repay transactions over hourly, daily and monthly time period.

```
type HourlyRepay @entity {
  "start_of_hour_timestamp"
  id: ID!
  timestamp: Int!
  count: Int!
  totalAmount: BigInt!
  totalUsdAmount: BigDecimal!
}
```

Hourly/Daily/MonthlyDeposit

Aggregated metrics for deposits over hourly, daily and monthly time period.

```
type HourlyDeposit @entity {
  "start_of_hour_timestamp"
  id: ID!
  timestamp: Int!
  count: Int!
  totalAmount: BigInt!
  totalUsdAmount: BigDecimal!
}
```

Hourly/Daily/MonthlyWithdraw

Aggregated metrics for withdrawals over hourly, daily and monthly time period.


```

type HourlyWithdraw @entity {
  "start_of_hour_timestamp"
  id: ID!

  timestamp: Int!
  count: Int!
  totalAmount: BigInt!
  totalUsdAmount: BigDecimal!
}

```

Hourly/Daily/MonthlyBorrow

Aggregated for borrow transactions over hourly, daily and monthly time period.

```

type HourlyBorrow @entity {
  "start_of_hour_timestamp"
  id: ID!

  timestamp: Int!
  count: Int!
  totalAmount: BigInt!
  totalUsdAmount: BigDecimal!
}

```

Querying time based aggregates

Every entity that has `hourly`, `daily` or `monthly` in its name can be queried by its ID. The documentation for those is located within each entity. Below lies the rules used to create the required timestamps.

Parameter	Value
<code>start_of_hour_timestamp</code>	unix timestamp at minute 0
<code>start_of_day_timestamp</code>	unix timestamp at hour 0, minute 0
<code>start_of_month_timestamp</code>	unix timestamp at first day of the month, hour 0, minute 0

Examples

Fetch the 5 biggest markets by total borrowed in USD

```
{
  assets(first: 5, orderBy: totalBorrowsUsd, orderDirection: desc) {
    symbol

    totalBorrows
    totalBorrowsUsd
    currPriceUsd
  }
}
```

Fetch historical interest rate, supply and borrow balances of a given market

```
{
  hourlyAssetSnapshots(where: {asset: "0x03ab458634910aad20ef5f1c8ee96f1d6ac54919"}, orderBy:
    id
    supplyAPY
    borrowAPY
    totalBorrowsUsd
    totalBalancesUsd
  }
}
```

Get the current balances of an account

```
{
  account(id: "0x0000000002732779240fe05873611dc4203dfb71") {
    balances {
      amount
      asset {
        symbol
      }
    }
  }
}
```

Get the transaction history of an account

```
{
  account(id: "0x0000000002732779240fe05873611dc4203dfb71") {
    balanceChanges {
      type
      timestamp
      amount
      amountUsd
      asset {
        symbol
      }
    }
  }
}
```

Get USD amount borrowed on February 10th 2022

First we need to create our ID using the parameters define in the [Querying time based aggregates](#) section. In this case, February 10th 2020 = 1644451200.

```
{
  dailyBorrow(id: "1644451200") {
    count
    totalUsdAmount
  }
}
```

Last 30 days of deposit amounts

```
{
  dailyDeposits(first: 30, orderBy: timestamp, orderDirection: desc) {
    id
    timestamp
    totalUsdAmount
  }
}
```

All transactions between February 1st 2022 and February 3rd 2022

```

{
  balanceChanges(orderBy: timestamp, orderDirection: asc, where: {timestamp_gte: 1643673600,
    timestamp
    type
    amount
    amountUsd
    account {
      id
    }
    asset {
      symbol
    }
  }
}

```

Querying time based aggregates

Every entity that has `hourly`, `daily` or `monthly` in its name can be queried by its ID. The documentation for those is located within each entity. Below lies the rules used to create the required timestamps.

Parameter	Value
<code>start_of_hour_timestamp</code>	unix timestamp at minute 0
<code>start_of_day_timestamp</code>	unix timestamp at hour 0, minute 0
<code>start_of_month_timestamp</code>	unix timestamp at first day of the month, hour 0, minute 0

Examples

Fetch the 5 biggest markets by total borrowed in USD

```

{
  assets(first: 5, orderBy: totalBorrowsUsd, orderDirection: desc) {
    symbol
    totalBorrows
    totalBorrowsUsd
    currPriceUsd
  }
}

```

Get the current balances of an account

```
{
  account(id: "0x0000000002732779240fe05873611dc4203dfb71") {
    balances {
      amount
      asset {
        symbol
      }
    }
  }
}
```

Get the transaction history of an account

```
{
  account(id: "0x0000000002732779240fe05873611dc4203dfb71") {
    balanceChanges {
      type
      timestamp
      amount
      amountUsd
      asset {
        symbol
      }
    }
  }
}
```

Get USD amount borrowed on February 10th 2022

First we need to create our ID using the parameters define in the [Querying time based aggregates](#) section. In this case, February 10th 2020 = 1644451200.

```
{
  dailyBorrow(id: "1644451200") {
    count
    totalUsdAmount
  }
}
```

Last 30 days of deposit amounts

```
{
  dailyDeposits(first: 30, orderBy: timestamp, orderDirection: desc) {
    id
    timestamp
    totalUsdAmount
  }
}
```

All transactions between February 1st 2022 and February 3rd 2022

```
{
  balanceChanges(orderBy: timestamp, orderDirection: asc, where: {timestamp_gte: 1643673600, timestamp_lte: 1643846400}) {
    timestamp
    type
    amount
    amountUsd
    account {
      id
    }
    asset {
      symbol
    }
  }
}
```

Security

Audits

View smart contract and UI audits from a number of security partners

Smart Contract Audits

The Euler protocol has been reviewed and audited by top security firms including: Halborn, Solidified and ZK Labs, Certora and Sherlock.

Omniscia - September 2022

[Audit Report](#)

Sherlock - July 2022

[Audit Report](#)

Omniscia - June 2022

[Audit Report](#)

Omniscia - March 2022

[Audit Report](#)

Sherlock - December 2021

Note: the EulDistributor contracts audited by Omniscia in the link above were also audited by Trail of Bits in Jun 2022 (see [Audit Report](#)) as part of an audit on the Morpho smart contracts.

Sherlock - December 2021

[Audit Report](#)

Certora - September 2021

[Audit Report](#)

Halborn - May 2021

[Audit Report](#)

Solidified and ZK Labs - May 2021

[Audit Report](#)

UI Audits

PenTestPartners.com - June 2022

[Audit Report](#)

Bug Bounty

Discover how to participate in the Euler Bug Bounty programme

The Euler protocol smart contracts have gone through several audits, but may still contain implicit or hidden vulnerabilities.

We strongly encourage the community to undertake their own audits, and will reward individuals who responsibly disclose any vulnerabilities they find. Up to \$1m is up for grabs. Start hunting today!

Bugs can be reported directly [here](#) or through DeFi's leading bug bounty platform Immunefi, [here](#).

Insurance

Find information about how users can insure their positions on Euler

Euler protocol has access to \$10M in smart contract coverage provided by [Sherlock](#) as part of a collaborative [partnership](#). This also includes an audit report and a \$1M ImmuneFi bug bounty provided by Sherlock.

Sherlock's coverage is managed by the protocols/DAO instead of users, while its claims decisions are made by an unbiased 3rd party.

Read more about Sherlock's insurance program at their [website](#) or in their [documentation](#) for further details.

Languages

White Paper (ENG-CHN)

Find out how Euler works and how it differs from other popular lending protocols

White Paper

Euler 白皮书-了解 Euler 的特别之处，与其他借贷协议又有何不同 (ENG-CHN)

作者 - Authors

Michael Bentley & Doug Hoyte

<https://www.euler.finance>

本文翻译自@Euler 团队的关于 Euler 项目的白皮书。已得到作者的授权。译者为@chainguys。转载请注明作者和译者。（Copyright©2021 by @Euler, translated by @chainguys）

概览 - Abstract

Here, we present Euler: a permissionless lending protocol custom-built to help users lend and borrow more Ethereum-based tokens than ever before. The purpose of this white paper is to describe how Euler works at a high level and highlight new features and innovations that help to set it apart from other popular lending protocols, like Compound and Aave.

我们在此向您介绍 Euler:为帮助用户借贷更多基于以太坊的代币而生，去（无）审批化的借贷协议。我们在此向您介绍 Euler:一个为了帮助更多用户更方便地借贷各类以太坊代币而出生，去（无）审批化的借贷协议。本白皮书的目的是描述 Euler 如何在高层次上（宏观上）工作，并强调（那些）有助于将 Euler 与其他流行的借贷协议（如 Compound 和 Aave）区分开来的新功能和创新。

导览 - Introduction

The ability to lend and borrow assets efficiently is a crucial feature of any financial system. In the world of traditional finance, this process is typically facilitated by trusted and permissioned third-parties such as banks, who connect people with a surplus of money to those who need access to it in the short-term. In the world of decentralised finance (DeFi), trusted and permissioned third-parties are no longer needed; banks have been replaced by trustless and permissionless lending protocols running on the blockchain (1).

对任何金融系统来说，高效借贷能力都是至关重要的特征。在传统金融中，这个过程通常由可信和经审批（监管）的第三方来完成，比如银行，由它们来连接那些资本充裕和短时间需要资金的人。在 DeFi 的世界中，可信和经受审批（监管）的第三方将不再需要：银行已经被链上运行的去信任化和去审批化借贷协议所取代(1)。

Among the first-generation of DeFi lending protocols are Compound (2) and Aave (3). These protocols

provide users with access to lending and borrowing capabilities for a handful of the most liquid ERC20 tokens. However, these protocols were not designed to handle the risks associated with lending and borrowing illiquid or volatile assets and have therefore relied on a permissioned listing system to protect their users from the risks associated with such assets.

第一代 DeFi 借贷协议主要有 Compound (2) and Aave (3)。这些协议为用户提供了借贷某些流动性最好的 ERC20 代币的入口。然而，这些协议在设计中却并未考虑处理由（借贷）流动性不佳，价格有较大波动的资产所带来的风险，所以这些协议要依赖一个需要审批的上市系统来帮用户规避这些风险。

Consequently, there remains significant unmet demand for lending and borrowing the long tail of crypto assets. On the lending side, users want to deposit tokens to earn yield and take leveraged long positions. On the borrowing side, users want to reduce their exposure to volatility and take leveraged short positions. Here, we present Euler: a permissionless lending protocol custom-built with an array of new features to help users lend and borrow more types of tokens than ever before.

从结果上来说，市场上存在着借贷长尾资产的巨大需求。在出借方的角度看，用户希望存入代币来获得收益并且建立带杠杆的多仓。在借款方来看的角度看，用户需要减少波动性（风险）并且建立带杠杆的空仓。在此，我们向各位介绍 Euler: 一个去审批化的借贷协议——它拥有多项量身定做的新特性，可以帮助用户借贷更多种类的代币。

准备开始 - Getting Started

Euler comprises a set of smart contracts deployed on the Ethereum blockchain that can be openly accessed by anyone with an internet connection. Euler is managed by holders of a protocol native governance token called Euler Governance Token (EUL). Euler is entirely non-custodial; users are responsible for managing their own funds.

As creators of the protocol, the Euler development team have produced a convenient and user-friendly front-end to the Euler smart contracts which is hosted at <https://app.euler.finance>. However, users are free to access the protocol in whatever format they wish, and we encourage developers to create their own front-end access points to the protocol to help decentralise access and increase censorship resistance.

Euler 包含了一些部署在以太坊上，任何人都可以上网公开进入的智能合约。Euler 由治理代币 Euler Governance Token (EUL) 的持有人管理。Euler 没有托管方，用户需要自己管理自己的资产。作为该协议的创造者，Euler 研发团队已经研发出了一个对用户友好的智能合约管理界面 (<https://app.euler.finance>)。当然，用户也可以用自己喜欢的方式进入协议，而且我们鼓励开发者创造他们自己的前端界面来促进去中心化和增加抗审查性。

去审批化上市 - Permissionless Listing

Euler lets its users determine which assets are listed. To enable this functionality, Euler uses Uniswap v3 as a core dependency (4). Any asset that has a WETH pair on Uniswap v3 can be added as a lending market on Euler by anyone straight away (5).

Euler 让用户自行决定哪些资产可以上市。Euler 主要使用了 Uniswap V3 来实现这个功能(4)。任何在 Uniswap V3 有 WETH 交易对的资产都可以被 Euler 直接添加(5)。

资产梯队/分层 - Asset Tiers

Permissionless listing is much riskier on decentralised lending protocols than on other DeFi protocols, like decentralised exchanges, because of the potential for risk to spill over from one pool to another in quick succession. For example, if a collateral asset suddenly decreases in price, and subsequent liquidations fail to repay borrowers' debts sufficiently, then the pools of multiple different types of assets can be left with bad debts.

To counter these challenges, Euler uses risk-based asset tiers to protect the protocol and its users:

在去中心化借贷协议上进行去审批化上市，要比在其他 Defi 项目（如去中心化交易所）更具风险，因为风险可能会迅速从一个流动性池传导到其他流动性池。举例来说，如果一个抵押资产的价格突然快速下降，并且在清算中无法充分偿还借款人的债务，那么多个流动性池子都会陷入债务危机。

Isolation-tier assets are available for ordinary lending and borrowing, but they cannot be used as collateral to borrow other assets, and they can only be borrowed in isolation. What this means is that they they cannot be borrowed alongside other assets using the same pool of collateral. For example, if a user has USDC and DAI as collateral, and they want to borrow isolation-tier asset ABC, then they can *only* borrow ABC. If they later want to borrow another token, XYZ, then they can only do so using a separate account on Euler.

隔离层资产可以用来常规的借贷，但是它们不能作为抵押物使用来借出别的资产，它们也只能隔离使用。这意味着它们不能在相同抵押物的流动性池中使用。举例来说，如果一个用户用 USDC 和 DAI 来做抵押物(这两者都不是隔离层资产)，当用户想借一种隔离资产 ABC 时，他就只能借到 ABC。如果这时要借出另一种代币 XYZ，那么就需要在 Euler 上创建另外单独的账户（才能借到 XYZ）。

Cross-tier assets are available for ordinary lending and borrowing, and cannot be used as collateral to borrow other assets, but they can be borrowed alongside other assets. For example, if a user has USDC and DAI as collateral, and they want to borrow cross-tier assets ABC and XYZ, then they can do so from a single account on Euler.

跨层资产可以用于普通借贷，可以与其他资产一起使用，但不能作为抵押物使用。举例来说，如果一个用户用 USDC 和 DAI 做抵押物，那他可以在同一个账户内借到跨层资产 ABC 和 XYZ。

Collateral-tier assets are available for ordinary lending and borrowing, cross-borrowing, and they can be used as collateral. For example, a user can deposit collateral assets DAI and USDC, and use them to borrow collateral assets UNI and LINK, all from a single account.

EUL holders can vote to liberate assets from the isolation-tier and promote them to the cross-tier or collateral-tier through governance mechanisms. Promoting assets up the tiers increases capital efficiency on Euler, because it allows lenders and borrowers to use capital more freely, but it may also expose protocol users to increased risk. It is therefore in EUL holders' interests to balance these concerns.

抵押层资产可以用来进行普通借贷，交叉借款，也可以作为抵押物。举例来说，一个用户可以存入 DAI 和 USDC 作为抵押物在一个账号中借到抵押层资产 UNI 和 LINK。EUL 持有者可以通过治理机制，投票决定这三类资产如何互相转化。将资产“上升”梯度会增加资产在 Euler 上的资金效率，因为借贷双方交易更加自由了，但是也可能会增加协议用户暴露在风险中的几率。综合来看，EUL 持有者对这些担忧进行平衡是符合自身利益的。

PS:关于这三种资产，虽然白皮书原文描述比较晦涩，但是核心就可以用一句话阐述：能不能用作抵押物且被

借后是否需要隔离管理。抵押层资产最方便，既可以做抵押物，但被借到之后也不需要隔离管理。跨层资产不能作为抵押物，但是被借到之后不需要隔离管理。隔离层资产不能作为抵押物使用，被借到之后必须隔离管理（一个子账号只能有一种隔离资产）。

借贷 - Lending and Borrowing

When lenders deposit into a liquidity pool on Euler, they receive interest-bearing ERC20 eTokens in return, which can be redeemed for their share of the underlying assets in the pool at any time, as long as there are unborrowed tokens in the pool (similar to Compound's cTokens). Borrowers take liquidity out of a pool and return it with interest. Thus, the total assets in the pool grows through time. In this way, lenders earn interest on the assets they supply, because their eTokens can be redeemed for an increasing amount of the underlying asset over time.

当出借人向流动性池中添加流动性时，他们会收到会产生利息的 ERC20 代币（eTokens），这些代币可以随时按比例赎回他们在流动性池中的资产，只要池中还有未借出的代币（与 Compound 的 cToken 类似）。借款方从池中借出流动性，返还时支付利息。于是，池中的资产就会随着时间而成长。如此，出借人就可以获得利息，因为它们的 eTokens 可以溢价退出。

代币化债务 - Tokenised Debts

Similarly to Aave's debt tokens, Euler also tokenises debts on the protocol with ERC20-compliant interfaces known as dTokens. The dToken interface allows the construction of positions without needing to interact with underlying assets and can be used to create derivative products that include debt obligations.

与 AAVE 的债务代币类似，Euler 也通过名为 dTokens 的，兼容 ERC20 代币的界面/接口，将债务代币化。dToken 接口/界面使得无需和标的资产交互就可以构筑仓位，也可以用来制作包含债务凭证的衍生品。

Rather than providing non-standard methods to transfer debts, Euler uses the regular transfer/approve ERC20 methods. However, the permissioning logic is reversed: rather than being able to send tokens to anyone, but requiring approval to take them, dTokens can be taken by anyone, but require approval to accept them. This also prevents users from "burning" their dTokens. For example, the zero address has no way of approving an in-bound transfer of dTokens.

Borrowers pay interest on their loans in terms of the underlying asset. The interest accrued depends on an algorithmically determined interest rate for each asset. A portion of the interest accrued is held in reserves to cover the accumulation of bad debts on the protocol.

没有选择提供非标准化措施来转移债务，Euler 使用常规的 ERC20 方法。但是，这个去审批化的逻辑就反过来了：不是等（当前）持有方同意后再发送给任何人，而是必须接收方同意后，dTokens 才可以被拿走。这也可以防止用户燃烧 dTokens。举例来说，零余额的地址就无法允许 dTokens 转入。借款方付出利息。而利息的归集则取决于一个决定每类资产利息水平的算法。一部分利息会作为“风险金/储备金”而被归集到一起用来对冲坏账。

受保护的抵押物 - Protected Collateral

On Compound and Aave, collateral deposited to the protocol is always made available for lending.

Optionally, Euler allows collateral to be deposited, but not made available for lending. Such collateral is 'protected'. It earns a user no interest, but is free from the risks of borrowers defaulting, can always be withdrawn instantly, and helps protect against borrowers using tokens to influence governance decisions (see Maker governance issue (6)) or take short positions.

在 Compound 和 Aave 上，存在协议上的抵押物是随时可以出借的。但 Euler 提供了另一个选项：抵押物可以先存入，但并不（马上）对外出借。这样的抵押物就是“受保护”的。虽然这样用户无法收取利息，但是也免除了默认出借所带来的风险。受保护的抵押物可以随时提现，也帮助避免有人利用代币恶意影响治理决策（参见 Maker governance issue (6)）

展期/延期流动性 - Defer Liquidity

Normally, an account's liquidity is checked immediately after performing an operation that could fail due to insufficient collateral. For example, taking out a borrow, withdrawing collateral, or exiting a market could cause a transaction be reverted due to a collateral violation.

However, Euler has a feature that allows users to defer their liquidity checks. Many operations can be performed and the liquidity is checked only once at the very end. For example, without deferring liquidity checks, a user must first deposit collateral before issuing a borrow. However, if done in the same transaction, deferring the liquidity check will allow the user to do this in any order.

正常情况下，一个账户在每完成一次可能因抵押物不足而失败的操作后，都会立刻被检测流动性。举例来说，借一笔钱，提出抵押物，或者是退出市场都可能因为抵押物（价值）波动而导致交易失败。但是，Euler 有一个特性使得用户可以让流动性检验延期。许多操作可以被执行，流动性检测也只会最后进行。举例来说，如果流动性检测不延期，一个用户在发起借款之前必须先存入抵押物。但是，展期流动性检测，用户就可以任意顺序完成上述交易。

无费用闪贷 - Feeless Flash Loans

Unlike Aave, Euler doesn't have a native concept of flash loans. Instead, users can defer their liquidity check, make an uncollateralised borrow, perform whatever operation they like, and then repay the borrow. This can be used to rebalance positions, build-up leveraged positions, take advantage of external arbitrage opportunities, and more.

Because Euler only charges fees according to the time value of money, and from the blockchain's perspective flash loans are held for a duration of 0 seconds, they are entirely free on Euler (ignoring gas costs). We believe that flash loan fees are ultimately in a race to the bottom that will be accelerated by advances like flash minting. The ecosystem benefits gained from simple and free flash loans outweigh the relatively modest benefit from flash loan fees.

与 Aave 不同，Euler 并没有原生概念的闪贷。不过，用户可以延期流动性检查，使用无抵押借款，进行他们喜欢的任何操作，最后偿还或借款。这可以被用来做仓位的再平衡，建立杠杆仓位，进行外部套利等。因为 Euler 只根据资金的时间价值来收费，且从区块链的角度来看闪贷的持续时间为 0，所以它是没有费用的（忽略 gas 费用）。

我们相信闪贷费用最终会在诸如闪铸（flash minting）等高级功能的加持下，最终走向底部。整个生态从简单和免费的闪电贷中获得的系统性收益超过闪电贷费用带来的相对温和的收益。

风险调整借款能力 - Risk-adjusted Borrowing Capacity

Like other lending protocols, Euler requires users to ensure that the value of their collateral remains higher than the value of their liabilities (except during the intermediate period when liquidity checks have been deferred). Over-collateralisation is encouraged by limiting how much borrowers can take out as a loan in the first place.

Compound achieves this in a one-sided way by using collateral factors to adjust down the value of a borrower's collateral assets when deciding how much they can borrow. This gives rise to a 'risk-adjusted collateral value' that helps to create a buffer that can be drawn upon by liquidators in the event that the value of a borrower's assets and liabilities changes over time. One of the problems with this approach is that it only adjusts for the risks associated with a borrower's collateral assets decreasing in value. There may be an asymmetric risk, however, of the borrower's liabilities increasing in value. This risk is not factored into the collateral factors.

跟其他借贷协议一样，Euler 需要用户确保他们的抵押物的价值比他们的债务更高（除非在流动性检查延期的时候）。Euler 一开始就通过限制借款人的借款额来鼓励超量抵押。Compound 使用一种单向的方法：当决定一个用户能借多少钱的时候，平台会根据抵押参数(因子)来做低抵押物的价值。这样会使得“风险调整后抵押物价值”上升，从而形成一个缓冲带——清算人可以据此在借款人的资产或者负债变化的时候（主动管理）。但一个问题是这样总会向借款人抵押物资产贬值的方向调整。于是就可能存在一个非对称风险，借款人的负债在增加，但这没有作为抵押参数进行考虑。

On Euler, we therefore use a two-sided approach where we also adjust up the market value of a borrower's liabilities to arrive at a 'risk-adjusted liability value'. This approach improves capital efficiency on the protocol because it allows Euler to factor in the asset-specific risks of both downside and upside price movements. These risks are encapsulated in asset-specific collateral factors (as on Compound) and borrow factors (new to Euler). Ultimately, this approach means that the liquidation threshold of every borrower is tailored to the specific risk profiles associated with the assets they are borrowing and using as collateral.

To give an example, suppose a user has \$1000 worth of USDC, and wants to borrow UNI. How much can they borrow? If USDC has a collateral factor of 0.9, and UNI has a borrow factor of 0.7, then a user can borrow upto $\$1000 \cdot 0.9 \cdot 0.7 = \630 worth of UNI. At this level of borrowing, the risk-adjusted value of their collateral is $\$1000 \cdot 0.9 = \900 , and the risk-adjusted value of their liabilities is $\$630 / 0.7 = \900 . If UNI increases in price, then the risk-adjusted value of their liabilities will also increase to $>\$900$, and they will be eligible for liquidation. The buffer allowing for liquidation is $\$1000 - \$630 = \$370$.

在 Euler,我们使用一种双向的方法：我们也将借款人债务的市场价值上调，来产生“风险调整后抵押物价值”。这使得资本使用效率大大提升——因为它使得 Euler 将资产价值涨跌这两种情况带来的风险都考虑进来。这些风险有的被概括为特定的抵押物因子（类似 Compound），有的则作为借贷因子（Euler 的创新）。最终，这一套方法意味着每个借款人的清算条件都是根据他们自身借入和抵押品资产情况量身定做。举例来说，假设一个用户有价值 1000USDC 的抵押物，然后他想借入 UNI。那他可以借出多少钱？如果 USDC 有一个抵押参数 0.9，而 UNI 则有一个 0.7 的抵押参数，那么用户就可以借价值 $\$1000 \cdot 0.9 \cdot 0.7 = \630 的 UNI。在这个情况下，风险调整后价值就是 $\$1000 \cdot 0.9 = 900$ 美金，而风险调整负债就是 $\$630 / 0.7 = \900 。如果 UNI 价格上涨，那么风险调整负债也会增加，大于 $\$900$ ，并且会作为清算资产。这时清算的缓冲是 $\$1000 - \$630 = \$370$ 。

去中心化价格预言机 - Decentralised Price Oracles

To be able to calculate whether a loan is over-collateralised or not, Euler needs to monitor the value of

users' assets. On Compound, Maker, and Aave, various systems are used to get prices from off-chain sources and put them on-chain so that they can be accessed by the relevant smart contracts.

This approach is unsuitable for Euler's purposes because it requires centralised intervention whenever a new lending market needs to be created. Euler therefore relies on Uniswap v3's decentralised time-weighted average price (TWAP) oracles to assess the solvency of users (4). The reference asset used to normalise prices on Euler is Wrapped Ether (WETH), which is the most common base pair on Uniswap (5).

要能够计算一笔贷款是否超额抵押，Euler 需要监控用户资产价值。在 Compound, Maker 和 Aave 上，各类系统被应用，来从链下获得价格信息并且将其上传到链上，从而使得相关智能合约能够访问。这个方法对实现 Euler 的目的来说就不适合了，因为每当要创造一个新的借贷市场时，就会需要一个中心化的干预（审批）。Euler 因此使用 Uniswap v3 的去中心化时间加权平均价格（TWAP）预言机来评估用户的偿付能力 (4)。Euler 作为标准化定价单位的是 Wrapped Ether (WETH), 这是 Uniswap 最常见的基础交易对 (5)。

TWAP

Uniswap TWAP is calculated using the geometric mean price of an asset over some interval of time. TWAP in general is both a smoothed and lagging indicator of the trade price: a TWAP over a short interval is a less smooth function, but more up-to-date, whilst a TWAP over a long interval is a smoother function, but less up-to-date. TWAP is ideal for Euler's purposes for several reasons.

Uniswap 的时间加权平均价格（TWAP）是由某项资产在某几个时间段的几何平均数计算出来的。一般来说，TWAP 是一个对交易价格既平滑也相对滞后的指标：一个短期的 TWAP 是一个不那么平滑的函数，但是却更具时效性；但长期的 TWAP 看起来更加平滑，时效性却相对差一些。TWAP 对 Euler 来说是非常理想的，原因如下：

First, TWAP is resistant to price manipulation attacks. It cannot be manipulated within a transaction or block (for example with flash loans or flash bots), because it is calculated using historic data. It is also expensive to manipulate using large market orders, because the manipulated price must be maintained for some period of time relative to the TWAP time interval. During this time, other users can take advantage of the manipulated price with arbitrage which will cause it to revert back to the broader market price. Arbitrage is especially practical on the blockchain because arbitrageurs have access to large amounts of capital (including from flash loans) and the atomic nature of transactions means that arbitrage transactions have a low execution risk. For these reasons, manipulating the price on a single decentralised exchange usually requires more widespread manipulation of all on-chain exchanges simultaneously, although even this can't prevent the (less practical but still possible) arbitrage between on and off-chain exchanges.

首先，TWAP 可以对抗价格操纵攻击。用（一笔）链上交易和区块是不能操纵市场的（比如用闪贷等手段），因为使用的是历史数据。如果要通过市场大单来操纵，那会非常昂贵，因为这需要内操纵的价格在 TWAP 的时间段内维持一定时间的稳定。在这段时间内，其他用户就可以利用这个情况来套利，接着价格就会回归。套利在区块链中是非常具有可操作性的，因为套利者一般资金充沛，而（区块链）的原子化交易又意味着套利交易在执行上的风险很低。因为上述原因，在一个去中心化交易所里操纵价格经常意味着要在更大的规模上同时操控所有的链上交易所（实际上更不可能但是理论上成立），但即使是这样还是无法阻止链上链下的套利行为。

Second, the smooth nature of TWAP helps to remove the impact of price shocks on borrowers. In the event

of a large trade, the current price on Uniswap can be moved significantly. Usually arbitrage bots will quickly converge this to the broader market value, so the maximum deviation of the TWAP will only be a fraction of the temporary price movement. This prevents some unnecessary liquidations and loans that may quickly become undercollateralised.

Third, instead of instantly jumping between two price levels, TWAPs change continuously, second-by-second. This property is used by Euler's liquidation process to implement Dutch auctions that reduce the value captured by miners and front-running bots.

其次，TWAP 平滑的天性可以帮助移除价格冲击对借款人的影响。假如有大单，Uniswap 的当前价格就会迅速变化。通常情况下套利机会会迅速使得（Uniswap）的价格和其他市场（更广泛市场）的价值趋同，所以这个最大的波动（偏离）在 TWAP 上就仅仅只是一个暂时价格变动的一部分。这样的机制就可以防止一些不必要的，会迅速变得抵押不足的清算或贷款。

时间间隔 - Time Interval

One of the challenges in using TWAP is determining the right interval over which it should be calculated for a given asset. The trade-offs involved with shorter (longer) intervals may sometimes need to be taken into consideration and altered for specific assets. Euler therefore allows the default time interval to be updated by governance if EUL holders deem it necessary.

清算 - Liquidations

A borrower is considered to be in violation on Euler when the value of their risk-adjusted liabilities exceeds the value of their risk-adjusted collateral. A borrower that has just become in violation still has enough collateral to repay their loan, but is adjudged to be at risk of defaulting on their loan. Consequently, they may be liquidated in order to limit the potential for them to default.

当 Euler 用户的风险调整后负债超过了风险调整后债务时，就会被认为“违约”了。一个借款人刚刚进入“违约”状态时依然有足额的抵押来偿付它的贷款，但是会有被调整到可能无法偿付贷款的风险。结果来说，为了防止他们违约，就可能会对它们进行清算。

MEV 抵抗 - MEV-resistance

On Compound and Aave, liquidations are incentivised by offering up a borrower's collateral to liquidators at a fixed percentage discount, which typically ranges between 5-10%. One of the issues with this strategy is that would-be liquidators often have no choice but to engage in priority gas auctions (PGA) for profitable liquidations, which exposes the liquidation bonus as so-called miner extractable value (MEV) (7). Another issue with this approach is that a fixed discount can be punitive for large liquidations, and therefore discourage large borrowers, whilst being insufficient to cover costs and incentivise smaller liquidations.

在 Compound 和 Aave 上，系统给出 5%-10%抵押物的折扣来奖励清算人。这个策略的问题之一就是清算人常常别无选择,只能为了进行有利润的清算而参与 Gas 优先拍卖（priority gas auctions,PGA），这样就引发了“矿工可提取价值”（miner extractable value, MEV）的问题(7)。另一个问题是这样提供一个固定折扣的方法对大型清算成本过高，也（变相）阻碍了借款人（继续借款），同时小型清算又变得费用不足，激励不够。

To remedy these issues, Euler uses a different approach. Rather than a fixed discount percentage, we allow

the discount to rise as a function of how under-water a position is. This turns a one-shot opportunity, where liquidators have no option but to engage in a PGA, into a type of Dutch auction. As the discount slowly increases, each would-be liquidator must decide whether or not to bid for a liquidation at the current discount on offer. Liquidator A might be profitable at 4%, but liquidator B might run a more efficient operation and be able to jump in sooner at 3.5%. The Dutch auction is aided by the TWAP oracles used on Euler, because a shock to the price does not bring with it a singular point at which every liquidator becomes profitable all at once. Instead the price moves more smoothly over time leading to a continuum of opportunities to liquidate, which further helps to limit PGAs. Overall, this process should help to drive the discount price towards the marginal operating cost of liquidating a borrower.

为了解决这些问题，Euler 采用了不同的方式。我们不采用固定折扣，而是采用一个公式来确定某个仓位到底“缩水”多少。这是“一锤子买卖”，清算人从别无选择参与 PGA 变成参与一种荷兰式拍卖。随着折扣慢慢增加，每个有意参与的清算者必须在当前的折扣水平下做出是否参与的决断。清算人 A 也许在 4% 折扣时就可以盈利，但是清算人 B 的行动效率可能更高，在 3.5% 折扣的时候就会果断出手。荷兰式拍卖会受到 Euler 上 TWAP 预言机的协助，因为一个价格冲击并不会使得价格立刻到达一个每个清算人都会盈利的奇点。随着时间变长，价格只会变得更加平滑，从而产生一系列可以清算的机会，而这还会限制 PGA。总的来说，这个过程会使得折扣价格和清算借款人的边际成本趋同。

However, by itself, this process does not prevent MEV, because miners and front-runners can still steal a liquidator's transaction. To limit this form of MEV, we allow liquidity providers on Euler to make themselves eligible for a "discount booster", which allows them to become profitable in the Dutch auction before miners and front-runners (who do not have the booster).

然而，这个过程自身并不能防止 MEV，因为矿工和抢跑者可能会偷掉一笔清算人的交易。要限制这种 MEV，Euler 会给流动性提供者一个“折扣推进器”，让他们在荷兰拍卖期间就可以盈利（矿工和抢跑者则不行，因为没有这个“推进器”）。

稳定池 - Stability Pools

On other lending protocols liquidations are usually processed using an external source of liquidity. That is, a liquidator will generally source the repayment amount of the borrowed assets from a third-party exchange, repay the loan, and receive the collateral and any bonus for themselves. One of the downsides of this approach is that the price feed used to determine the liquidation price of a borrower will not always accurately reflect the exchange rate on external markets, meaning that liquidators will not always be able to liquidate at that price. Reasons for this include slippage, swap fees, extreme volatility, the use of price-smoothing algorithms such as TWAP (as on Euler), and delays posting new prices.

在其他借贷协议中，清算中经常使用外部的流动性。也就是说，一个清算人一般会从第三方交易所借入资金，来偿付（被清算人的）债务，然后接管抵押物和奖励。这其中不好的一点，就是借款人的清算执行价格并不总是能准确反应外部市场的汇率，这意味着清算人并不能总是以这个价格执行清算。这种现象的原因有很多，比如滑点，交易费用，极端波动，TWAP 等算法的应用（Euler 也使用 TWAP），报价延迟等。

To alleviate this issue, Euler enables lenders to support liquidations by providing liquidity to a stability pool

associated with each lending market. Liquidity providers in the stability pool deposit eTokens and earn interest whilst they wait for liquidations to be processed. An unstaking period prevents them from moving assets in and out of the pool to try to game the system. When a liquidation is processed the liquidator uses liquidity from the stability pool to cancel a borrower's debts and they return discounted collateral to the stability pool in return (minus a fee, which they keep for themselves). Stability pool liquidity providers essentially end up swapping their eTokens for a discounted index of collateral assets.

要解决这些问题，Euler 允许出借人对清算人提供支持，方式就是出借人向每个借贷市场的稳定池注入流动性。稳定池的流动性提供者存入 eTokens 然后获得利息，同时等待清算发生。为了保持系统稳定，不被恶意操纵，存入 eTokens 后需要一定时间之后才能提出。当一笔清算发生时，清算人将使用稳定池的资金来偿还借款人的债务，然后抵押物也会（按比例）存入稳定池（扣除归属清算人其他费用后）。稳定池的流动性提供者最终可以用 eTokens 交换（池中）相应的抵押物资产。

This approach can be thought of as an extended multi-collateral form of the stability pool idea pioneered by Liquity protocol (8). The main advantage of using a stability pool is that liquidations can be processed immediately using an internal source of liquidity at the point at which a borrower is deemed by the protocol to be in violation, without a liquidator needing to source the assets themselves from a third-party exchange. See Table 1 for some of the benefits of performing liquidations using internal versus external liquidity.

这被认为是多渠道抵押稳定池形式的一种扩展，Liquity protocol (8) 是这方面的先驱。使用稳定池主要的好处是，在借款人因被认为“违约”而面临清算时，可以使用内部流动性来进行清算，清算人则不必向外部/第三方交易所寻求资源。具体的好处请查看表 1：

Table 1. Comparison of using an internal stability pool for liquidations rather than using an external source of liquidity.

Text	External	Internal
Liquidity source	Liquidator typically purchases from a DEX or has existing source of funds themselves	Liquidator uses internal liquidity in the stability pool
Transaction costs	Gas costs may be high for DEX trades and cross-contract calls	Gas costs often relatively cheap for internal token transfers
Explicit trade costs	Swap fees	No swap fees
Implicit trade costs	Slippage on illiquid markets	No slippage
Liquidation price	Liquidation expected to take place at price determined by the wider market	Liquidation expected to take place at price determined by the internal price feed
Liquidation timing	Liquidation expected to take place only after the dynamic discount exceeds operating costs and trade costs	Liquidation expected to take place soon after the dynamic discount exceeds the operating cost of liquidation

Table 表 1. 使用内部/外部资源进行清算的比较:

文字	外部	内部
清算资源	清算人大部分从其他 DEX 购入或者外部资金支持	清算人使用稳定池的内部流动性
转账费用	Gas 费用在不同的 DEX 或者智能合约间可能会很贵	内部转账的 gas 一般更加便宜
显性交易成本	闪兑 (swap) 费用	没有闪兑 (swap) 费用
隐形交易成本	流动性不佳产生的滑点	没有滑点
清算价格	清算预计以外部价格 (更广泛市场的价格) 展开	清算预计以内部价格展开
清算时机	只能在动态折扣超过执行成本和交易成本后执行	动态折扣超过执行成本和交易成本后就能迅速执行

软性清算 - Soft Liquidations

The fraction of a borrower's debt that can be paid off by liquidators in one go is referred to by Compound as the 'close factor.' On both Compound and Aave, the close factor is currently fixed at 0.5, meaning liquidators can pay off up to half a borrower's loan in one go regardless of how underwater their position is. This approach has a couple of potential drawbacks.

一个清算人能够一次性偿还借款人债务的比例在 Compound 上叫做“偿还因子”。在 Compound 和 Aave 上，偿还因子目前是固定的 0.5，这意味着清算人可以最多一次性偿还借款人 50% 的贷款，不管借款人的抵押物如何“缩水”。这种方式有一些潜在的不宜之处。

First, allowing liquidators to liquidate half a loan could be considered excessive if a smaller liquidation would have been sufficient to bring the borrower back to health. Larger borrowers are likely to be put off by such a process. Second, a large fixed discount can sometimes drive a borrower closer to insolvency and disincentivise them from repaying their loans (see (8)).

首先，如果一个小型清算就足以使得借款人的（仓位）回到健康状态，那让清算人清算一半的贷款就显得比较“过”了。规模大一些的借款人对这样的清算方式可能不感兴趣。其次，一个大额固定折扣会让借款人更接近资不抵债的状况，从而打消他们偿还贷款的积极性。

On Euler, we therefore use a dynamic close factor to try to 'soft liquidate' borrowers. Specifically, we allow liquidators to repay up to the amount needed to bring a violator back out of violation (plus an additional safety factor). This means that borrowers who are only slightly in violation will often have much less than half their debts repaid during a liquidation, whilst borrowers who are heavily in violation will often have much more than half their debts repaid during a liquidation (their whole position might be closed in some circumstances).

在 Euler，我们选择采用动态偿还因子来对借款人进行软性清算。具体来说，我们允许清算人偿还债务到违约

的临界点（再附加一些其他的安全因子修正）。这就意味着那些只是轻微违约的人在清算时往往偿还的比例要显著小于 50%，当然同时严重违约的借款人就要在清算时偿还显著高于 50% 的比例（某些情况下仓位可能被关闭）

储备 - Reserves

In rare circumstances the value of a borrower's collateral might become less than the value of their liabilities. In this situation the borrower is said to be 'insolvent.' Insolvent borrowers will typically be liquidated repeatedly until they have little to no collateral left. Any leftover liabilities after liquidations have stopped can be considered 'bad debt' that we can assume will never be repaid. If bad debt accumulates on the protocol, it increases the chance that lenders might all rush at once to withdraw their funds (to avoid becoming the bearer of the bad debt). This phenomenon is known as 'run on the bank.'

在一些极少的情况下，借款人的抵押物（价值）会低于他们的债务。在这种情形下，借款人就“资不抵债”。资不抵债的借款人一般会被反复清算直到他们只有极少的抵押物留存。任何清算停止后还存留的债务就成为“坏账”，我们基本可以认为“坏账”无法清偿。如果池中的坏账持续累积，就会增加出借人挤兑的风险（为了避免成为坏账的最终承担者而在同一时间大量提出资产）。

To reduce this risk, Euler follows Compound by allowing a portion of the interest paid by borrowers in each market to accumulate into a reserve. The idea behind this is to allow the reserves to act as a lender of last resort in the event of a run on the bank. Providing that reserves accumulate at a faster pace than bad debt, lenders do not need to worry about being able to withdraw their funds. Euler reserves operate similar to those on Compound, except that Euler reserves are tracked in eToken units, rather than underlying units, which means that Euler reserves earn interest automatically whereas Compound reserves do not.

为了减少这个风险，Euler 向 Compound 取经，将借款人在市场中支付利息的一部分积累起来，成为了储备金。这么做背后的想法是当挤兑发生时让储备金成为最后的出借人。只要储备金的增速高于坏账的增速，那么出借人就没有必要担心是否能取出自己的资产。Euler 的储备金与 Compound 的类似，但 Euler 储备金以 eTokens 为单位来追踪（价格），而不是标的资产，这意味着 Euler 的储备金会自动获取利息，而 Compound 储备金则不会。

The proportion of interest paid into the reserves is called the 'reserve factor' and it is a parameter specific to each lending market. There are trade-offs to consider when setting the reserve factor. A reserve factor of zero would mean no reserves accrue, which could stifle lending because of the bad debt issue. Nevertheless, a high reserve factor would mean a large portion of interest is diverted away from lenders, which could also stifle lending as lenders seek a better rate elsewhere. Thus EUL holders may wish to use governance to select a reserve factor that balances these trade-offs for each type of asset.

利息存入储备金的比例就叫做“储备因子”，而且每个特定的借贷市场都不相同。设置储备因子有一些问题需要取舍。如果储备因子为 0，则意味着不归集储备金，这可能会因为坏账和伤害出借方（积极性）。然而，储备因子如果太高，那就意味着大部分的利息从出借方流失了，从而也会损伤出借方（积极性）——他们可能就去别的地方寻求更好的回报率了。所以，EUL（Euler 治理代币）的持币人可能就需要在治理时选择能够平衡上述情况的储备因子。

清算的额外费用 - Liquidation Surcharge

During a liquidation, the liquidator is required to provide a slightly larger amount of the borrowed asset than

is being repayed on behalf of the violator. This extra amount is contributed to the reserves for the borrowed asset as a fee. The base liquidation discount starts at the level of this fee, so it is ultimately paid by the violator.

As a result, more volatile assets, which generally trigger more liquidations, will tend to accrue reserves at a faster pace than less volatile assets helping to protect lenders of those assets. Additionally, this fee ensures that 'self-liquidating' is always net-negative, which adds a profitability threshold that some undesirable manipulation strategies are unlikely to meet.

在清算的时候，清算者需要代表违约者来证明已经借出的资产比要偿还的要稍微多一些。这些多余的就会当做已借资产的保证金费用。基础清算折扣就从这个费用的水平开始（计算），最后由违约者偿付。最终，越来越多的违约资产会引发更多的清算，（也会）导致归集保证金的速度比更少违约资产保护出借人的速度快。此外，这个费用还保证了“自清算”永远为净负值，从而为我们不愿见到的某些市场操纵行为添加了利润上的门槛。

利率 - Interest Rates

Both Compound and Aave use static linear (or piecewise linear) interest rate models to guide the cost of borrowing on their protocols. Broadly speaking, as demand for borrowing from the pool increases or supply decreases, interest rates go up, and when supply increases or the demand for borrowing decreases, interest rates go down.

Static models work well if they are appropriately parameterised ahead of time, but can be problematic when parametrised incorrectly. For example, if the slope of the static linear function is too shallow, it can lead to the cost of borrowing being underpriced, with lenders unable to withdraw their assets because a pool has become over-utilised. On the other hand, if the slope of the static linear function is too steep, it can lead to the cost of borrowing being too expensive, which can stifle borrowing and lead to low capital efficiency.

Compound 和 Aave 都使用静态线性（或分段线性）利率模型来指导协议中的借款成本。宽泛地说，随着池中借款需求的升高或者供给的降低，利率就会下降。如果提前设置好了恰当的参数，静态模型就会较为有效，但如果参数设置不好，就会有问题。举例来说，如果静态线性函数的斜率太平缓，就会导致借款成本被低估，出借人就会因池中（资产）被超额使用而无法提现自己的资产。相反，如果斜率太陡，就会导致借款成本过高，妨害借款人借款，也影响出借人资金利用效率。

回应式汇率 - Reactive Interest Rates

To avoid the problem of having to choose the right parameters for every lending market, Euler uses control theory to help autonomously guide the cost of borrowing towards a level that maximises capital efficiency on the protocol. Specifically, we use a PID controller to amplify (dampen) the rate of change in interest rates when utilisation is above (below) a target level of utilisation. This gives rise to reactive interest rates that adapt to market conditions for the underlying asset in real-time without the need for ongoing governance intervention. A similar approach has also recently been described by the Delphi Labs team (9).

为了避免强行为每个借贷市场都单独设定“正确”的参数（因为这样做耗费精力和各项成本都很巨大），Euler 采用了控制论来帮助指导借款成本达到一个最大化资本利用率的水平。具体来说，当利用率高于某个目标水平时，我们使用一个 PID 控制器来调整利率变化的幅度。无需持续干涉，就提升了回应式利率实时响应标的资产市场情况的能力(9)。

复利 - Compound Interest

Compound interest is accrued on Euler on a per-second basis. This differs from other lending protocols, where interest is typically accrued on a per-block basis. A per-second basis is generally expected to perform more predictably in the long-run even if upgrades to Ethereum lead to changes in the average time between blocks.

Euler 上的复利是按秒归集的。这就和其他以区块归集复利的借贷协议不一样。以秒计算从长期来看更加有可预测性，即使因以太坊更新导致按区块间平均（出块）时间来计算。

最优化 - Gas Optimisations

Euler's smart contracts minimise the amount of storage used, implement a module system to reduce the amount of cross-contract calls, and have had a number of other gas usage optimisations applied. This makes the protocol cheaper on most operations than other lending protocols.

Euler 的智能合约将存储用量最小化，并且实施了一个模块系统来减少智能合约间的（无效）交互，（除了这些）还应用了一系列其他优化 gas 的措施。这使得整个协议在大部分运营成本上都比其他协议更廉价。

交易建设者 - Transaction Builder

The user interface includes a convenient tool to help users batch up multiple transactions and reduce their gas costs, which we call a transaction builder. Advanced users can use this feature in conjunction with a defer liquidity option provided on the protocol to rebalance loans or perform flash loans.

用户界面包含了一套方便的，被称作交易建设者的工具来帮助用户管理多笔交易和减少 gas 费用。高级用户可以利用这个特性，来连接协议提供的延期清算选项，从而实现贷款的再平衡或实行闪贷。

子账号 - Sub-accounts

Asset tiers help to isolate risks on Euler, but they open up a new user-experience problem. Specifically, it would quickly become cumbersome for borrowers to use Euler if they had to send collateral to a new Ethereum account for each new isolation-tier loan they wanted to take out.

Euler therefore enables every Ethereum account using the protocol to access up to 256 sub-accounts (including the primary account), which can be used to cost-effectively manage multiple positions at the same time. A user only needs to approve Euler's access to a token once and can then deposit into any sub-account. Additionally, no approvals are required to transfer assets and liabilities between sub-accounts, which allows users to isolate and segregate their collateral and debts as desired.

资产分层帮助隔离 Euler 上的风险，但是产生了一个用户体验的新问题。具体来说，如果用户每次都要把抵押物转账到新的以太坊账号上，那确实太繁琐了。Euler 于是采用了一个可以同时有效管理多账号的新办法：允许每个使用（Euler）协议的以太坊账号使用最多 256 个账号（含主账号）。除此之外，子账号之间转移资产和债务是不需要批准的，这也使得用户能按照自己的意愿隔离和配置他们的资产和债务。

治理 - Governance

Euler will broadly follow the governance model pioneered by Compound (10). The protocol will be managed

by holders of a protocol native governance token called Euler Governance Token (EUL). EUL tokens will represent voting shares. Holders with enough EUL tokens will be able to make a formal proposal for change on the protocol. Token holders will then be able to vote on the proposal themselves or delegate their vote shares to a third party. Examples of the kinds of decisions token holders might vote on include proposals to alter include:

- The tier of an asset
- Collateral and borrow factors
- Price oracle parameters
- Reactive interest rate model parameters
- Reserve factors
- Governance mechanisms themselves

Euler 在治理方面会较多采用 Compound (10) 提出和发展的一些方式。Euler 会被持有原生治理代币 Euler Governance Token (EUL)的人管理。EUL 代币将代表投票的比例。有足够数量 EUL 代币的人能够提出改变协议（治理）的正式提案。代币持有人也能够将投票权让渡给第三方。以下是持币人可能投票的内容：

- 资产分层
- 抵押物和借款因子
- 价格预言机参数
- 回应式利率模型的参数
- 储备因子
- 治理机制本身

鸣谢 - Acknowledgements

With special thanks to [Shaishav Todi](#), [Luke Youngblood](#), [Charlie Noyes](#), [samczsun](#), [Hasu](#), [Dave White](#), [Rick Pardoe](#), [Ayana Aspembitova](#) and the [Delphi Labs](#) team, [Mariano Conti](#), [Lev Livnev](#), and [Chainguys](#).

特别感谢以下诸位 [Shaishav Todi](#), [Luke Youngblood](#), [Charlie Noyes](#), [samczsun](#), [Hasu](#), [Dave White](#), [Rick Pardoe](#), [Ayana Aspembitova](#) and the [Delphi Labs](#) team, [Mariano Conti](#), [Lev Livnev](#), and [Chainguys](#).

引用文献 - References

1. <https://docs.compound.finance/built-on-ethereum/open-market/what-is-open-market/>
2. <https://compound.finance/documents/Compound.Whitepaper.pdf>
3. https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf
4. <https://uniswap.org/whitepaper-v3.pdf>
5. <https://weth.io/>
6. <https://www.theblockcrypto.com/post/82721/makerdao-issues-warning-after-a-flash-loan-is-used-to-pass-a-governance-vote>
7. <https://research.paradigm.xyz/MEV>
8. <https://docsend.com/view/bwiczmy>
9. <https://members.delphidigital.io/reports/dynamic-interest-rate-model-based-on-control-theory>
10. <https://medium.com/compound-finance/compound-governance-5531f524cf68>

Community Translations

Read the Euler white paper in different languages.

Members from the Euler community have translated the white paper into a number of other languages below. These are not hosted by official channels nor fully verified for word-for-word accuracy, but they may be used as reference to better understand the white paper.

Community translators' efforts are greatly appreciated! Feel free to submit other translations to the Euler community [Discord](#).

Korean: <https://m.blog.naver.com/PostView.naver?blogId=ahrmina&logNo=222627632140&proxyReferer=>

Japanese: <https://www.notion.so/Euler-Whitepaper-Japanese-044d74772b3541c58ba9b644c27e6b7c>

Russian: <https://gitbook-guru.gitbook.io/euler/white-paper>

Ukrainian: https://medium.com/@nina_b33/white-paper-ed9cfb390c2

Legal

Terms and Conditions

Terms of Use

Please read these terms of use carefully before you start to use our Website, as these will apply to your use of our Website. We recommend that you print a copy of this for future reference. By using our Site, you confirm that you accept these terms of use and that you agree to comply with them. If you do not agree to these terms of use, you must not use our Site.

1. Your Relationship With Us

1.1 Welcome to euler.finance and app.euler.finance (the “Website”), provided by The Euler Foundation (“Company”, “we” or “us”). The Company is registered in the Cayman Islands and our registered office is at 4th Floor, Harbour Place, 103 South Church Street, P.O. Box 10240, Grand Cayman KY1-1002, George Town, Cayman Islands.

1.2 This page (the “Terms”) forms an agreement between you and us and sets forth the terms and conditions by which you may access and use our protocols, applications and content (including but not limited to the Website) (collectively, the “Interface”). For purposes of these Terms, “you” and “your” means you as the user of the Interface.

1.3 **The Terms form a legally binding agreement between you and us.** Please read them carefully and we recommend that you seek legal advice on these Terms to understand your use of the Interface. If you do not agree to these Terms, you must not access or use the Interface.

2. Interface

2.1 The Interface provides functionality for you to interact with the Euler Protocol (a permissionless non-custodial protocol for the lending and borrowing of Ethereum-based crypto assets). The Interface also provides the following non-custodial functions, including: depositing, withdrawal, minting, burning, transferring, swapping, and shorting of crypto assets.

2.2 For the avoidance of doubt, the Company does not control or operate the Euler Protocol and that protocol is managed, controlled and operated by the holders of the protocol-native governance tokens called the Euler Governance Token (“EUL”). For further information on the Euler Protocol, please visit: docs.euler.finance. Such information is provided for informational purposes only and we do not make any representation or warranty in any form whatsoever in relation to such information.

2.3 Holders of EUL are not granted any legally enforceable rights or entitlements. Rather, the Euler Protocol may enable them to make certain proposals with respect to the Euler Protocol for all EUL holders to perform several functions.

3. Fees

3.1 Neither the Company, nor any of its parents, subsidiaries, and affiliates, and each of their respective officers, directors, employees, agents and advisors, charges or receives any form of fee from any user of the Euler Protocol for operating, maintaining or developing the Interface.

4. Accepting the Terms

4.1 **By accessing or using the Interface, you confirm that you can form a binding contract with us, that you accept these Terms and that you agree to comply with them.** Your access to and use of the Interface is also subject to our Privacy Policy, the terms of which can be found directly on the Website, and is incorporated herein by reference. The Interface may directly or indirectly collect and temporarily store personally identifiable information for operational purposes, including for the purpose of identifying blockchain addresses or IP addresses that may indicate use of the Interface from prohibited jurisdictions or by sanctioned persons or other Prohibited Uses. Except as required by applicable law, there will be no obligation of confidentiality with respect to any information collected.

4.2 If you are accessing or using the Interface on behalf of a business or entity, then (a) “you” and “your” includes you and that business or entity, (b) you represent and warrant that you are an authorized representative of the business or entity with the authority to bind the entity to these Terms, and that you agree to these Terms on the entity’s behalf, and (c) your business or entity is legally and financially responsible for your access or use of the Interface.

4.3 You can accept the Terms by accessing or using the Interface. You understand and agree that we will treat your access or use of the Interface as acceptance of the Terms from that point onwards.

4.4 You should print or save a local copy of the Terms for your records.

5. Changes to These Terms and to the Interface

5.1 We may amend these Terms from time to time, for instance when we update the functionality of the Interface or when there are regulatory changes. We may use commercially reasonable efforts to generally notify all users of any material changes to these Terms, such as through a notice on the Interface, however, you should look at the Terms regularly to review the most up-to-date version and to check for such changes. We will also update the “Last Updated” date at the top of these Terms, which reflect the effective date of such Terms. Your continued access or use of the Interface after the date of the new Terms constitutes your acceptance of the new Terms. If you do not agree to the new Terms, you must stop accessing or using the Interface.

6. Accessing the Interface and Prohibited Activities

6.1 Your access to and use of the Interface is subject to these Terms and all applicable laws and regulations. You may not, either directly or through a third party:

- (a) Access or use the Interface if you are not fully able and legally competent to agree to these Terms;
- (b) Modify, adapt, translate, reverse engineer, disassemble, decompile or create any derivative works based on the Interface, including any files, tables or documentation (or any portion thereof);
- (c) Distribute, license, transfer, reproduce, duplicate, copy, sell or resell, in whole or in part, any of the Interface or any derivative works thereof except as authorized by these Terms;
- (d) Market, rent or lease the Interface for a fee or charge, or use the Interface to advertise or perform any commercial solicitation;
- (e) Interfere with or attempt to interfere with the proper working of the Interface, disrupt the Interface or any networks connected to the Interface, or bypass any measures we may use to prevent or restrict access to the Interface;
- (f) Incorporate the Interface or any portion thereof into any other program or product. In such case, we reserve the right to refuse service, terminate accounts or limit access to the Interface in our sole discretion;
- (g) Use automated scripts to collect information from or otherwise interact with the Interface; or
- (h) Impersonate any person or entity, or falsely state or otherwise misrepresent you or your affiliation with any person or entity.

6.2 We reserve the right, at any time and without prior notice, to remove or disable access to content at our discretion for any reason or no reason.

6.3 You further agree that you will not use the Interface to perform any type of illegal activity of any sort or take any action that negatively affects the performance of the Interface. You may not engage in any of the following activities, either directly or through a third party:

- (a) attempt to gain unauthorized access to the Interface or another user's account; or
- (b) Engage in any activity that is abusive or interferes with or disrupts the Interface. Use of the Interface in connection with any activity involving illegal products or services is prohibited.

6.4 We may suspend your access to the Interface in the event of any breach of these Terms.

6.5 By using the Interface you represent and warrant that you:

- (a) Do not reside;
- (b) Are not located;
- (c) Do not have a place of business; and
- (d) Are not conducting any business, (any of which makes you a "Resident") in any jurisdiction in which your use of the Interface is prohibited by any applicable statutes, laws (including common law), ordinances, rules, regulations, codes, orders (including any temporary, preliminary or permanent order, judgment, injunction, decree, ruling or other similar event or action), or government or regulatory agency orders or guidance (collectively, "Laws") or where under such Laws the operator of the Interface would be required to be registered or licensed, to seek any consent or approval, or to make any filing with respect to your use of the Interface.

6.6 By using the Interface you represent and warrant that you are not a Resident of any state or country:

- (a) That requires entities engaged in token sales or token offerings to be registered or licensed; or
- (b) Where the sale or purchase of the tokens pursuant to the Terms would be unlawful.

6.7 By using the Interface you represent and warrant that you are not a Resident of the United States or a “U.S. person” within the meaning of Rule 902(k) under the United States Securities Act of 1933 (the “Securities Act”).

7. Intellectual Property Rights

7.1 The Interface, including its “look and feel” (e.g., text, graphics, images, logos, page headers, button icons, and scripts), proprietary content, information and other materials, and all content and other materials contained therein, including, without limitation, all designs, text, graphics, pictures, data, software, sound files, other files, and the selection and arrangement thereof are the proprietary property of the Company or our affiliates or licensors.

7.2 The Company’s name, logo, trademarks, and any Company product or service names, designs, logos, and slogans are the intellectual property of the Company or our affiliates or licensors and may not be copied, imitated or used, in whole or in part, without our prior written permission in each instance. You may not use any metatags or other “hidden text” utilizing the Company’s name or any other name, trademark or product or service name of the Company or our affiliates or licensors without our prior written permission.

7.3 We respect intellectual property rights and ask you to do the same. As a condition of your access to and use of the Interface, you agree not to use the Interface to infringe on any intellectual property rights. We reserve the right, with or without notice, at any time and in our sole discretion to block access to any user who infringes or is alleged to infringe any copyrights or other intellectual property rights.

8. Indemnity

8.1 You agree to defend, indemnify, and hold harmless the Company, its parents, subsidiaries, and affiliates, and each of their respective officers, directors, employees, agents and advisors from any and all claims, liabilities, costs, and expenses, including, but not limited to, attorneys’ fees and expenses, arising out of a breach by you of these Terms or arising out of a breach of your obligations, representations and warranties under these Terms.

9. Exclusion of Warranties

9.1 Nothing in these Terms shall affect any statutory rights that you cannot contractually agree to, alter or waive and are legally always entitled to as a user.

9.2 The Interface is provided “as-is” and we make no warranty or representation to you with respect to them. In particular we do not represent or warrant to you that:

- (a) your use of the Interface will meet your requirements;
- (b) your use of the Interface will be uninterrupted, timely, secure or free from error;
- (c) any information obtained by you as a result of your use of the Interface will be accurate or reliable;
and
- (d) defects in the operation or functionality of any software provided to you as part of the Interface will be corrected

9.3 No conditions, warranties or other terms (including any implied terms as to satisfactory quality, fitness for purpose or conformance with description) apply to the Interface except to the extent that they are expressly set out in the Terms. We may change, suspend, withdraw or restrict the availability of all or any part of our platform for business and operational reasons at any time without notice.

10. Limitation of Our Liability

10.1 Nothing in these Terms shall exclude or limit our liability for losses which may not be lawfully excluded or limited by applicable law. This includes liability for death or personal injury caused by our negligence or the negligence of our employees, agents or subcontractors and for fraud or fraudulent misrepresentation.

10.2 Subject to paragraph 10.1, we shall not be liable to you for any:

- (a) loss of:
 - (i) profit;
 - (ii) goodwill;
 - (iii) opportunity; or
 - (iv) data suffered by you, (in each case whether direct or indirect);
- (b) indirect or consequential losses which may be incurred by you; or
- (c) loss or damage which may be incurred by you as a result of:
 - (i) any reliance placed by you on the completeness, accuracy or existence of any advertising;
 - (ii) any changes which we may make to the Interface, or for any permanent or temporary cessation in the provision of the Interface (or any features within the Interface);
 - (iii) the deletion of, corruption of, or failure to store, any content and other communications data maintained or transmitted by or through your use of the Interface; or
 - (iv) your failure to provide us with accurate account information.

10.3 Subject to paragraphs 10.1 and 10.2, our total aggregate liability to you, whether based on an action or claim in contract, tort (including negligence), breach of statutory duty or otherwise arising out of, or in relation to, these Terms, the Interface or service, will be limited to USD 50.00.

10.4 Unless you notify us that you intend to make a claim in respect of an event within the notice period, we shall have no liability for that event. The notice period for an event shall start on the day on which you became, or ought reasonably to have become, aware of your grounds to make a claim in respect of the event and shall expire six (6) months from that date. The notice must be in writing and must identify the event and the grounds for the claim in reasonable detail.

11. Anti-Money Laundering, Economic Sanctions, Anti-Bribery and Anti-Boycott Representations

11.1 You represent and warrant that neither you, nor any of your directors, officers, or to the best of your knowledge and belief, your employees, affiliates or associates or anyone acting on your behalf (as applicable) is:

- (a) the subject or target of any economic or financial sanctions, trade embargoes or export controls administered, enacted or enforced from time to time by the United States of America ("U.S.") (including those administered by the U.S. Treasury Department's Office of Foreign Assets Control or the U.S. Department of State), the United Nations Security Council, the European Union ("EU"), any EU member state, the United Kingdom or any jurisdiction in which you operate (collectively "Sanctions");
- (b) organised, operating from, incorporated or resident in a country or territory which is the subject or target of comprehensive export, import, financial or investment embargoes under any Sanctions (which, as of the date of these Terms are Cuba, Iran, North Korea, the Crimea region of Ukraine, Syria, the so-called Donetsk People's Republic or the so-called Luhansk People's Republic); or
- (c) is a senior political figure or any immediate family member or close associate of a senior political figure.

11.2 For the purposes of this paragraph 11.1:

- (a) a "senior political figure" is a senior official in the executive, legislative, administrative, military or judicial branches of a government (whether elected or not), a senior official of a major political party, or a senior executive of a government-owned corporation. In addition, a "senior political figure" includes any corporation, business or other entity that has been formed by, or for the benefit of, a senior political figure;
- (b) an "immediate family member" of a senior political figure are such person's parents, siblings, spouse, civil partner, children, step-children and in-laws; and
- (c) a "close associate" of a senior political figure is a person who is widely and publicly known to maintain an unusually close relationship with the senior political figure, and includes a person who is in a position to conduct substantial financial transactions on behalf of the senior political figure.

12. General

12.1 Security

- (a) We do not guarantee that the Interface will be secure or free from bugs or viruses. You are responsible for configuring your information technology, computer programmes and platform in order to access the Interface. You should use your own virus protection software.
- (b) You must not misuse the Interface by knowingly introducing viruses, Trojans, worms, logic bombs or other material which is malicious or technologically harmful. You must not attempt to gain unauthorised access to the Interface, the server on which the Interface is stored or any server, computer or database connected to the Interface. You must not attack the Interface via a denial-of-service attack or a distributed denial-of-service attack. By breaching this provision, you acknowledge that you commit a criminal offence under the Computer Misuse Act 1990 (as amended, extended or re-enacted from time to time). We will report any such breach to the relevant law enforcement authorities and we will co-operate with those authorities by disclosing your identity to them. In the event of such a breach, your right to use the Interface will cease immediately.

12.2 Linking To Our Website

- (a) You may link to our home page, provided you do so in a way that is fair and legal and does not damage our reputation or take advantage of it. You must not establish a link in such a way as to suggest any form of association, approval or endorsement on our part where none exists.
- (b) You must not establish a link to our Website in any website that is not owned by you. Our Website must not be framed on any other website, nor may you create a link to any part of our Website other than the home page.
- (c) We reserve the right to withdraw linking permission without notice. If you wish to make any use of content on our Website other than that set out above, please contact: info@euler.foundation.

12.3 Third Party Links

- (a) Where the Interface contains links to other websites and resources provided by third parties, these links are provided for your information only. We have no control over the contents of those websites or resources.

12.4 Applicable Law and Jurisdiction

- (a) These Terms, its subject matter and its formation (and any non-contractual disputes or claims arising out of or in connection with the Terms) are governed by laws of the British Virgin Islands.
- (b) Any dispute, controversy, difference or claim arising out of or relating to these Terms, including the existence, validity, interpretation, performance, breach or termination thereof or any dispute regarding non-contractual obligations arising out of or relating to it shall be finally resolved by arbitration under the Arbitration Rules of the LCIA (the “Rules”), which are deemed to be incorporated by reference into this paragraph (save that any requirement in the Rules to take account of the nationality of a person considered for appointment as an arbitrator shall be disappplied and a person may be nominated or appointed as an arbitrator (including as chairman) regardless of nationality). There shall be three arbitrators, two of whom shall be nominated by the Company and you in accordance with the Rules and the third, who shall be the Chairman of the tribunal, shall be nominated by the two-nominated arbitrators within fourteen (14) days of the last of their appointments. The seat, or legal place, of arbitration shall be London, England. The language to be used in the arbitral proceedings shall be English. Judgment on any award may be entered in any court having jurisdiction thereover.

12.5 Entire Agreement

- (a) These Terms constitute the whole legal agreement between you and the Company and govern your use of the Interface and completely replace any prior agreements between you and the Company in relation to the Interface.

12.6 No Waiver

- (a) Our failure to insist upon or enforce any provision of these Terms shall not be construed as a waiver of any provision or right.

12.7 Severability

- (a) If any court of law, having jurisdiction to decide on this matter, rules that any provision of these Terms is invalid, then that provision will be removed from the Terms without affecting the rest of the Terms, and the remaining provisions of the Terms will continue to be valid and enforceable.

Contact Us

To contact us, please email contact@euler.foundation.

Last updated on: 06-June-2022

Effective date: 06-June-2022

Privacy Policy

Privacy Policy

This Privacy Policy describes the policies of The Euler Foundation, 4th Floor, Harbour Place, 103 South Church Street, P.O. Box 10240, Grand Cayman KY1-1002, George Town, Cayman Islands on the collection, use and disclosure of your information that we collect when you use our website (euler.finance or app.euler.finance) (the "Service"). By accessing or using the Service, you are consenting to the collection, use and disclosure of your information in accordance with this Privacy Policy. If you do not consent to the same, please do not access or use the Service.

We may modify this Privacy Policy at any time without any prior notice to you and will post the revised Privacy Policy on the Service. The revised Policy will be effective 180 days from when the revised Policy is posted in the Service and your continued access or use of the Service after such time will constitute your acceptance of the revised Privacy Policy. We therefore recommend that you periodically review this page.

How We Use Your Information

We will use the information that we collect about you for the following purposes:

- Marketing/ Promotional
- Customer feedback collection
- Support

If we want to use your information for any other purpose, we will ask you for consent and will use your information only on receiving your consent and then, only for the purpose(s) for which grant consent unless we are required to do otherwise by law.

How We Share Your Information

We will not transfer your personal information to any third party without seeking your consent, except in limited circumstances as described below:

- Analytics
- Data collection & process

We require such third party's to use the personal information we transfer to them only for the purpose for which it was transferred and not to retain it for longer than is required for fulfilling the said purpose.

We may also disclose your personal information for the following: (1) to comply with applicable law, regulation, court order or other legal process; (2) to enforce your agreements with us, including this Privacy Policy; or (3) to respond to claims that your use of the Service violates any third-party rights. If the Service or our company is merged or acquired with another company, your information will be one of the assets that is transferred to the new owner.

Your Rights

Depending on the law that applies, you may have a right to access and rectify or erase your personal data or receive a copy of your personal data, restrict or object to the active processing of your data, ask us to share (port) your personal information to another entity, withdraw any consent you provided to us to process your data, a right to lodge a complaint with a statutory authority and such other rights as may be relevant under applicable laws. To exercise these rights, you can write to us at contact@euler.foundation. We will respond to your request in accordance with applicable law.

You may opt-out of direct marketing communications or the profiling we carry out for marketing purposes by writing to us at [contact@euler.foundation].

Do note that if you do not allow us to collect or process the required personal information or withdraw the consent to process the same for the required purposes, you may not be able to access or use the services for which your information was sought.

Cookies

We do not use cookies.

Security

The security of your information is important to us and we will use reasonable security measures to prevent the loss, misuse or unauthorized alteration of your information under our control. However, given the inherent risks, we cannot guarantee absolute security and consequently, we cannot ensure or warrant the security of any information you transmit to us and you do so at your own risk.

Grievance

If you have any queries or concerns about the processing of your information that is available with us, you may email us at contact@euler.foundation and we will address your concerns in accordance with applicable law.

Last updated on: 06-June-2022

Effective date: 06-June-2022