History / Home

Revisions	Compare revisions
Updated Home (markdown) MazyGio committed on Mar 9	55623a7
Updated Home (markdown) CPSTL committed on Mar 9	4c72b66
Updated Home (markdown) CPSTL committed on Mar 8	6e59f50
Updated Home (markdown) CPSTL committed on Mar 8	40e73a1
Updated Home (markdown) CPSTL committed on Jan 11	80c8009
update home copy CPSTL committed on Jan 11	a9dad6a
Updated Home (markdown) CPSTL committed on Jan 10	09d8f3d
Initial Home page CPSTL committed on Jan 10	654ec96

Governance Proposal Framework Template (Council)

Jump to bottom

Alim edited this page on Mar 8 · 9 revisions

Guidelines to keep in mind when creating a Governance Proposal Framework

1. **Explicitness:** A proposal should clearly define and address a specific behavior or single responsibility. If it proposes multiple actions, it should be split into multiple explicit proposals.
2. **Completeness:** A proposal should be thorough. Relevant particulars should not be left undefined or unreferenced.
3. **Avoid Overlap:** Multiple independent proposals should not propose the implementation of the same type of behavior. Review active proposals to avoid overlap.
4. **Clarity:** A proposal should be as clear and as easy to understand as possible. Be succinct, use simple language, and consider having a peer in the community review prior to submission.

Create Venues For Participating in Governance & Discussion 💡

Some examples of common practices for governance venues:

Governance Forum: Forums are the best way to get yourself up to date with the current state of governance and join in on community discussions. The forum should be the go-to place for submitting proposals, discussing governance proposals, DAO discussions, voting on both off-chain and on-chain proposals, and much more. Some examples of forums used to date are:

- Commonwealth
- Discourse

Snapshot (Off-Chain Polling): Snapshot is an off-chain polling system that provides a simple voting interface that allows users to signal sentiment on proposals (directly). This is typically used as a way to gauge if a proposal should move onto an on-chain vote.

- Snapshot.org

Governance Dashboard (Council Reference UI): This is a part of the Council Kit tooling framework, and allows your governance system to have a dedicated UI for on-chain voting. This UI allows users to review information on proposals, delegate voting power, view other voter profiles and their voting activity, and participate in on-chain voting.

Discord #Governance Channel: If your project already uses Discord as a community management system, think about creating a dedicated governance channel in your server to allow governance participants to discuss proposal ideas, provide feedback on current proposals, share opinions on current polls and votes, and more.

- Discord

Example of a Governance Process for DAOs 🔁

Proposal Types

1) Protocol (Executable) Proposals

- This category of proposals should be used when implementation involves the execution of one or more smart contract operations by accounts controlled by the DAO.
- Examples: Technical upgrades to the governed Protocol, additional features, incentive programs, and more.
- Executable Proposals will pass by a simple majority if the quorum has been met. Note that quorum isn't a percentage but rather an absolute amount. There are specific types of votes that can have lower or higher quorums. For example, grants spending may entail various amounts of disbursements and having distinct quorums for each level of spending. Or, for governance actions such as freezing the protocol, one might choose a high quorum for further security.

2) DAO (Social) Proposals

- This category of proposals should be used when a community member makes a request of the DAO that cannot be executed or enforced on-chain.
- Examples: Adding, improving, or removing governance processes, such as updating the Governance process itself or migrating forums or communication channels to other venues.

An Example of the Proposal Process:

Detailed Summary

1. Anyone can post a proposal on the forum for discussion within the community. All proposals follow the same general process from idea to acceptance, with the exception of a review period for Protocol proposals after the Off-chain (Snapshot) Polling stage, for code review.
2. Using the governance proposal framework guidelines, a proposal is created and submitted to the DAO governance forum as an idea or skipping the idea and going to a Draft. In the forum, proposals are typically posted under the proposal type topics/categories.
    i. Example: DAO (Social) Proposals
3. Once it has been posted as a Draft, all questions and comments should be addressed and taken into consideration, to further improve the proposal before moving to the next stage. If they are ignored, it is less likely that the proposal will pass the subsequent off-chain poll on Snapshot.
4. Once the author of the proposal has incorporated any relevant feedback and made the necessary revisions, the forum admin will mark the proposal as officially "Proposed".

5. Once the proposal has been marked with the “Proposed” status on the forum or the author has stated it is officially ready to move forward, there will be a **1 week review period** for the community to review, discuss, and prepare for the Off-Chain Poll.
6. After the 1 week review period, the proposal will officially proceed to the off-chain poll step.
7. After the Snapshot poll has finished (the voting period is 5 days):
    i. For DAO Proposals, this is the final decision of the proposal being officially accepted or rejected by the community.
    ii. For Protocol Proposals, the sentiment of the community will be officially captured. If the poll passes, it will advance to the next stage of the lifecycle. If the poll is rejected, the author may go back and update it to better position it for future consideration.
8. After the successful passing of the off-chain poll, the Protocol Proposal must go through a long enough review period that allows sufficient time for an official code audit/review and be deemed safe to advance to an on-chain poll and possible execution.
9. Once the review period is complete (along with the audit/review), the proposal will move to an on-chain vote (the voting period is 8 days max). The outcome of the on-chain vote will dictate whether or not it is executable. If it passes, it is officially accepted and can move to execution into the governed protocol. If it is rejected, no execution will occur.
10. After a Protocol proposal has been executed, the changes are updated on-chain to the protocol, and the lifecycle of the proposal concludes.

Proposal Lifecycles

An Example of a DAO (Social) Proposal Lifecycle Breakdown & Criteria

Official Status Lifecycle for a DAO (Social) Proposal

[Idea] [Draft] [Proposed] [Off-chain Poll] [Accepted/Rejected]

Official Status Lifecycle for a DAO (Social) Proposal (including review periods)

[Idea] [Draft] [Proposed] [Review] [Off-chain Poll] [Accepted/Rejected]

Idea

- The **Idea** status is an informal stage of the lifecycle of a DAO governance proposal. This stage involves an idea being seeded for a proposal and made available for discussion in Discord, the forum, or Twitter (whatever is the preferred venue).

Draft

- The **Draft** status is a formal status on the governance forum, where a proposal has been created using the provided template in hopes of obtaining feedback and ultimately garnering the community's support. Based on community feedback, the author of the proposal should make updates and edits to the proposal. The goal of the `Draft` proposal status is to obtain sufficient feedback and support to achieve soft consensus in order to confidently proceed with the formal process without it being immediately shot down. The draft proposal status also helps avoid having “garbage proposals” make it through to Off-chain polls and helps eliminate voting fatigue that the community would face without this stage.
- Once posted on the governance forum as a **Draft**, all questions and comments should be addressed and taken into consideration, to further improve the proposal before moving to the next stage. If they are ignored, it is less likely that the proposal will pass the Off-chain Poll.

Proposed

- The **Proposed** status is a formal status on the governance forum, where a proposal has officially been submitted to proceed to the Off-Chain Poll stage after the dedicated review period. If the Draft proposal did not contain any official code and just a specification, this proposal would include the exact code that will be executed. This way anyone could verify the hash that is included on-chain.
- Once a proposal has been submitted under this status, it will officially be given an ID. IDs are given based on increments of submission. For example, the first proposal will be `[EGP-1]: Title of Proposal`. The second proposal would be `[EGP-2]: Title of Proposal` and so on.

Review Period (suggested)

- The **In-Review** status is a formal status on the governance forum, where a proposal sits in review for the community to discuss and prepare for the upcoming **Off-Chain Poll.** A proposal that has completed the **review period**, will proceed to a final Off-chain poll (Snapshot).
- `DAO (Social) Proposals Review Period`: 1 week

Off-Chain Poll (Snapshot)

- The **Off-Chain Poll** status is a formal status on the governance forum. Off-chain polls can be voted on directly on the Snapshot.org page and will ultimately decide if a DAO proposal is accepted or rejected by the community.

Accepted/Rejected (Final Status for DAO Proposals)

- The **Accepted/Rejected** status is the final state of a proposal after being voted on. If it passes, it is officially accepted by the community (with actions of the proposal being implemented) and is given the Accepted status. If not, the proposal is rejected and labeled as such.

An Example of a Protocol (Executable) Proposal Lifecycle Breakdown & Criteria**

Official Status Lifecycle for a Protocol (Executable) Proposal

[Idea] [Draft] [Proposed] [Off-chain Poll] [On-chain Vote] [Accepted/Rejected] [Execution]

Full Lifecycle For A Protocol (Executable) Proposal (including review periods)

[Idea] [Draft] [Proposed] [Review] [Off-chain Poll] [Review] [On-chain Vote] [Accepted/Rejected] [Execution]

Idea

- The **Idea** status is an informal stage of the lifecycle of a DAO governance proposal. This is when an idea is seeded for a proposal and has been posted on Discord, the forum, or even Twitter.

Draft

- The **Draft** status is a formal status on the governance forum, where a proposal has been created using the provided template in hopes of obtaining feedback and ultimately garnering the community's support. Based on community feedback, the author of the proposal should make updates and edits to the proposal. The goal of the `Draft` proposal status is to obtain sufficient feedback and support to achieve soft consensus in order to

confidently proceed with the formal process without it being immediately shot down. The draft proposal status also helps avoid having “garbage proposals” make it through to Off-chain polls and helps eliminate voting fatigue that the community would face without this stage.

- Once posted on the governance forum as a **Draft**, all questions and comments should be addressed and taken into consideration, to further improve the proposal before moving to the next stage. If they are ignored, it is less likely for the proposal to pass the Off-chain Poll.

Proposed

- The **Proposed** status is a formal status on the governance forum, where a proposal has officially been submitted to proceed to the Off-Chain Poll stage after the dedicated review period. If the Draft proposal did not contain any official code and just a specification, this proposal would include the exact code that will be executed. This way anyone could verify the hash that is included on-chain.
- Once a proposal has been submitted under this status, it will officially be given an ID. IDs are given based on increments of submission. For example, the first proposal will be `[GP-1]: Title of Proposal`. The second proposal would be `[GP-2]: Title of Proposal` and so on.

Review Period (not an official proposal status)

- The **In-Review** status is a formal status on the governance forum, where a proposal sits in review for the community to discuss and prepare for the upcoming **Off-Chain Poll.**
- `Protocol (Executable) Proposals Review Period`: 1 week

Off-Chain Poll (Snapshot)

- The **Off-Chain Poll** status is a formal status on the governance forum and can be voted on directly in the forum or directly on the Snapshot page (link). A proposal that has completed the **review period**, will proceed to an Off-chain poll (Snapshot). The main requirement of a Protocol proposal on Snapshot is to include Calldatas and Targets should be included to verify the on-chain proposalHash when it is created.
- The goal of the Off-Chain Poll is to gather community sentiment towards the proposal and gather feedback to direct the preferred path. This Poll allows the community to reach a rough consensus to maximize the chances of a positive outcome of the On-chain vote submission.

Review Period (not an official proposal status)

- This review period is for Protocol proposal code review. If the code is already audited it may be a short period of time but if the audit or review has not started, the proposal should not move to an on-chain vote until it has been complete. The review period is meant to start out as flexible but may become more strict as the governance process evolves. Transparency and communication are encouraged by the proposal author during this stage.

On-chain Vote

- The **On-chain Vote** status is a formal status on the governance forum where proposals can be voted on directly on the official governance dashboard/UI.

Accepted/Rejected

- The **Accepted/Rejected** status is the final stage of a proposal after being voted on. If it passes, it is officially accepted and is given the Accepted status. If not, the proposal is rejected and labeled as such.

Executable/Executed (For Protocol Proposals only)

- Once a proposal has reached the winning quorum (specific number), the proposal may be executed. This status states that the proposal is ready to be officially executed.

Governance Proposal (GP) Templates:

DAO (Social) Proposal Template

Protocol (Executable) Proposal Template

What Governance Operates (Using Council)

Governance Parameters

General Governance:

- **Quorum:** The required minimum number of voting power in support of a proposal for it to succeed (this value is fixed and not a percentage of total voting power).
- **Timelock Duration:** The timelock is a speed bump for calls. It requires calls to wait for a defined period before they can be executed. After a call is registered, the call can be removed during the waiting period, and an authorized address can extend the waiting period only once.
- **Voting Period:** The total voting period in which votes can be cast. [Minimum voting period][extra voting period] is the layout of the total time for voting.
- **Minimum Voting Period:** The minimum voting time before a vote can be executed.
- **Extra Voting Period:** The remaining time beyond the minimum voting period in which votes can be cast.
- **Proposal Threshold:** The amount of voting power needed to submit a governance proposal on-chain.
- **Vesting Vault Token Multiplier (Optional - depending on whether or not you use this voting vault):** The vesting vault contract is used to set up long-term grants for core contributors to the governed protocol's ecosystem. It is also a voting vault so that community members can vote with vested but unclaimed tokens and unvested tokens. The **multiplier** for voting power on the vesting vault allows locked / vesting positions to still have voting power in the governance system and does so by using a defined multiplier for the vested tokens over unvested.

GSC Parameters (Optional - depending on whether or not you use a GSC in your model):

- **GSC Delegation Threshold:** A threshold of delegated voting power, giving delegates a seat on the GSC. This comes with additional governance powers within the system, and as a result, additional responsibilities.
- **GSC Quorum:** The required minimum number of GSC members needed to support a proposal for it to succeed.
- **GSC Proposal Threshold:** The amount of GSC members needed to propose a GSC vote on-chain.

Protocol Parameters

- **Protocol Upgrades:** Should the governed DAO community decide to upgrade the governed Protocol, it will go through governance. These types of changes will go through the process defined for Protocol Proposals.

Clone this wiki locally

https://github.com/delvtech/council-kit.wiki.git

June 17, 2023 – June 24, 2023

Period: 1 week

Overview

7 Active pull requests

0 Active issues

7 Merged pull requests	**0** Open pull requests
0 Closed issues	**0** New issues

Could not load contribution data

Please try again later

7 Pull requests merged by **2** people

Add changeVaultStatus to SDK

#397 merged 22 minutes ago

[SDK] Add set lock duration
#396 merged 7 hours ago

Add fallback UI on mobile when no proposals exist
#395 merged 11 hours ago

Update localhost.ts default config
#394 merged 12 hours ago

[CLI] Fix voting power errors on localhost due to stale block lag
#393 merged 2 days ago

Fix voting power stat in locking vault
#392 merged 2 days ago

[CLI] Add simple deploy command
#391 merged 3 days ago

2 Unresolved conversations

Sometimes conversations happen on old items that aren't yet closed. Here is a list of all the Issues and Pull Requests with unresolved conversations.

Added Opt-in/out button to start receiving notifications from Push
#342 commented on 3 days ago • 0 new comments

feat: add airdrop ui
#388 commented on last week • 0 new comments

Spender

Jump to bottom

CPSTL edited this page on Jan 10 · 2 revisions

- Contract Name: Spender.sol
- Type/Category: Treasury/Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/features/Spender.sol

1. Introduction (Summary)

The spender contract is designed to leverage the variable quorum requirements of the core voting contract to enable a very small-scale grant program.

2. Contract Details

Key Functionalities:

- Spend a number of tokens up to a high, medium, or low threshold via separate functions for high, medium, and low spends. Allow the owner to remove the tokens from this contract.

State:

- High, medium, and low spending thresholds
- Tracking the block by historical block spending ensures that a threshold is violated in no block.

3. Key Mechanisms & Concepts

The contract contains some tokens and then allows an owner address to spend tokens from this contract. The owner of the contract will be the core voting contract. In the core voting contract, each of the spending functions should be given its own quorum. In this setup, this contract enables small and correspondingly low-security grants to fund non-contentious community-driven projects which are low expenditure.

4. Gotchas (Potential source of user error)

The expenditure in each block is tracked in aggregate, so if a low spend is successful, a following high spend in the same block can only spend up to (high spend bound - low spend bound).

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

In the case of contract failure, the owner should remove funds from the contract.

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Timelock

Jump to bottom

CPSTL edited this page on Jan 10 · 2 revisions

- Contract Name: Timelock.sol
- Type/Category: Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/features/Timelock.sol

1. Introduction (Summary)

The timelock is a speed bump for calls. It requires calls to wait for a period before they can be executed. After a call is registered, the call can be removed during the waiting period, and an authorized address can extend the waiting period only once.

2. Contract Details

- The key functions in the time lock are the ability for an owner to register a call via a 32 byte hashed identifier, to remove the registered call, to allow anyone to run that call after a customizable period of time, and the ability for authorized addresses to add an optional one time delay in addition to the regular delay in the contracts.
- The state in the timelock is a mapping of registered calls to execution timestamps, authorized addresses, and an address that can register calls.

3. Key Mechanisms & Concepts

The timelock is a mechanism to ensure that any security-critical votes that execute a call or bundle of calls have a minimum review time to allow the community to check that the calls have no unintended side effects. After the core voting contract votes something that is high security-critical, a hash of the information of the calls to be executed will be registered with the timelock.

After waiting a minimum amount of time, anyone can call the timelock to execute the call or set of calls. Suppose the proposal is reviewed and an error is identified during the review period. In that case, the core voting contract must vote again to remove the call from the timelock. Because core voting has a minimum vote time, the effective timelock period is the timelock period minus the minimum voting time. To mitigate this, authorized addresses [such as the GSC] can do a one-time per call increase in lock time for a registered call. The time bump should be slightly more than the minimum voting time to enable removal votes.

The timelock can upgrade its stored parameters such as the delay time and the amount of bump time by making a call to itself which sets these pieces of information.

4. Gotchas (Potential source of user error)

- Calls registered with the contract must all succeed or the whole bundle will fail.
- Calls cannot re-access the timelock as a reentrancy lock protects it.
- Timelocks should not be used for emergency mitigation functionality and security response functionality, such as pause controls. Because of the slowdown, they may not be able to respond to disclosures of security bugs in time.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

Since the timelock controls its parameters, the timelock can upgrade itself to a non-functional state by, for instance, making the address that registers calls one which cannot send transactions that register calls or setting the delay to a very large number. Parameter setting should be done with a high amount of care.

▸ **Pages** 25

Council Wiki

- Home
- Council Protocol 101

Technical Guides

- Getting Started
- Council Deployment Guide
- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault

- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Treasury

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: Treasury.sol
- Type/Category: Treasury/Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/features/Treasury.sol

1. Introduction (Summary)

The Treasury contract is an isolated holder of Element governance funds that allows an authorized address to move them. This isolation improves the security of the system and prevents a bug from compounding to total fund loss.

2. Contract Details

The contract only contains the functionality to allow an authorized address to move the ERC20 and ETH tokens it contains; to set ERC20 allowances or to make generic calls. The stored data is the authorized address.

3. Key Mechanisms & Concepts

The contract receives a call from an address and then checks if it is authorized to move funds. If the address is authorized, it makes an ERC20 transaction that can move funds or set allowances.

4. Gotchas (Potential source of user error)

Sending the funds to the wrong address (person or contract) or approving the wrong contract can result in the loss of funds.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

It is important to note that, if the ownership of the contract is reset, it should be reset to an address that can make calls or the funds in the treasury will be frozen in place.

▸ Pages 25

Council Wiki

- Home
- Council Protocol 101

Technical Guides

- Getting Started
- Council Deployment Guide
- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Vesting Vault

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: VestingVault.sol
- Type/Category: Voting Vault/Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/vaults/VestingVault.sol

1. Introduction (Summary)

The vesting vault contract is used to set up long-term grants for core contributors to the Element ecosystem, but its flexible approach enables it to be used for many other types of grants as well. It is also a voting vault so that community members can vote with vested but unclaimed tokens and unvested tokens.

2. Contract Details

Key Functionalities:

- Ability to create grants which release over time with an optional cliff.
- Ability to remove grants for exited contributors.
- Grant holders should be able to vote and delegate votes with their locked tokens.
- Grant holders should be able to claim their tokens on schedule.

State:

- A mapping tracking of the historical voting power.
- A mapping tracking grants.
- Contract solvency information.
- Ownership information.

3. Key Mechanisms & Concepts

The contract allows a manager to create vesting grants using a balance of tokens that are stored in the contract. Each grant has a non-vesting period, and an optional cliff followed by a linear vesting period. The created grants cannot add up to a total amount more than what is in the contract. Each grant holder can delegate their votes to other addresses, and the contract state keeps a record of historical votes that are compatible with the core voting contract requirements. In addition to a historical voting power record, the contract also stores user grants in a mapping. When a grant is removed for a contributor, any unlocked tokens are transferred to them, and their grant is removed.

4. Gotchas (Potential source of user error)

- The grant administrator cannot create more outstanding grants than the token which have been deposited. They must use the deposit method for that deposit to be recognized.
- Any tokens or ETH directly transferred to the contract cannot be removed and are locked forever.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- The vesting vault contract should only be operated as an upgradeable proxy component because the storage library used is intended for that application.
- In the case of compromise of the grant funding address, all funds in this contract are at risk. And, in the case of compromise of the upgrade system, all funds in this contract are at risk.

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Voting Vaults Overview

Jump to bottom

CPSTL edited this page on Jan 11 · 2 revisions

Overview

Voting Vaults provide the ability to assign voting power to specific types of tokens/positions. The result is beautiful — governance users can maximize capital efficiency while maintaining the ability to delegate or vote when the time comes. Voting vaults are designed to be upgraded, allowing for voting vault logic to be updated by governance and to avoid the pains that come with governance upgrade migrations. The creation of a voting vault begins with defining a strategy for counting votes / providing voting power. Once the logic has been established, governance will vote to approve the method, and if successful, the integration is complete, and users can start voting.

What are voting vaults?

Voting vaults are simply smart contracts that allow any programmable logic to be used for allocating voting power to governance participants.

Some example metrics include:

- Reputation or merit-based systems
- User protocol usage metrics
- User governance participation data
- Token-holding
- Positions in DeFi protocols (staked assets, collateral, LP positions, etc.)
- Any other metric or combination of metrics

This gives the community complete modularity and flexibility over how to structure a DAO's governance framework.

- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Home

Jump to bottom

MazyGio edited this page on Mar 9 · 8 revisions

Introduction to Council

Council represents the next evolution of on-chain governance, allowing anyone to build adaptable governance systems that meet both the practical needs of day-to-day activities and the required flexibility of long-term governance.

The Council Protocol and Council Kit enable builders to use the security of on-chain governance while allowing for unprecedented modularity and flexibility.

Council is here to reinvigorate the standard model for DAO governance - keeping decentralization at the forefront and allowing DAOs to scale their decision-making.

Key Features of the Protocol

The Council Protocol's key value proposition is the introduction of new governance primitives, which will enable a new era of experimentation:

⚖️ **Core Voting:**

- This system begins with the core which is all that is needed at the base layer. All additional functionality can be added by selecting desired modules. The Core defines the voting process for those with governance power, tracks proposals, and retrieves user voting power from approved Voting Vaults.

Voting Vaults:

- Voting Vaults enable governance participation to be more inclusive and capital efficient. Vaults can be created by defining strategies for assigning voting power to different use cases.

🏛️ **GSC:**

- The Governance Steering Council (GSC) enables scalable decision-making for DAOs. The GSC is a group of representatives elected on a rolling-basis model of delegation and must maintain a minimum level of voting power to remain on the Council. Council members can create, vote, and execute proposals and can be assigned further duties through the governance process.

🔴 **Optimistic Rewards and Grants:**

- A new structure that assumes the best out of grantees & offers tools to empower the delivery of grant outlines
- A way to move rewards calculation off-chain and alleviate the limitations of traditional rewards programs.

Build with Council Kit

Council Kit goes beyond the smart contracts and positions Council as an all-in-one governance framework. The Kit comes with a suite of tools to create and bootstrap DAOs.

🚀 **Deployment Template:**

- Select, configure, and deploy the smart contracts in a guided walk-through with the deployment template.

🎛️ **Reference UI:**

- Make the creation of your governance portal simple with a fully customizable reference UI built with React, TypeScript, and NextJS.

🛠️ **SDK:**

- A TypeScript SDK which interfaces with the Council Protocol smart contracts and unlocks the ability to create custom scripts.

Support & Contribute

Building Council has been an absolute pleasure and we couldn't have done it without the feedback and support from you all. We're always looking to make Council better, so please let us know how we can continue to improve by filing Issues or ask how you can start making PRs and contribute! Check out the contributor guide.

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

All workflows

Showing runs from all workflows

Filter workflow runs

2,488 workflow runs

Event Status Branch Actor

Add changeVaultStatus to SDK (#397)

Deploy Council UI to GH Pages #339: Commit 24ad1eb pushed by ryangoree

21 minutes ago 3m 31s `main`

Add changeVaultStatus to SDK

Lint #662: Pull request #397 opened by ryangoree

25 minutes ago 3m 17s `ryan-change-vault-status`

Add changeVaultStatus to SDK

Format Check #173: Pull request #397 opened by ryangoree

25 minutes ago 2m 9s `ryan-change-vault-status`

Add changeVaultStatus to SDK

Build #173: Pull request #397 opened by ryangoree

25 minutes ago 3m 23s `ryan-change-vault-status`

Add changeVaultStatus to SDK

Dependabot auto-approve #660: Pull request #397 opened by ryangoree

25 minutes ago 2s `ryan-change-vault-status`

[SDK] Add set lock duration (#396)

Deploy Council UI to GH Pages #338: Commit 272887d pushed by ryangoree

7 hours ago 3m 24s `main`

[SDK] Add set lock duration

Dependabot auto-approve #659: Pull request #396 opened by ryangoree

7 hours ago 3s `ryan-lock-duration`

[SDK] Add set lock duration

Build #172: Pull request #396 opened by ryangoree

7 hours ago 4m 2s `ryan-lock-duration`

[SDK] Add set lock duration

Lint #661: Pull request #396 opened by ryangoree

7 hours ago 2m 48s `ryan-lock-duration`

[SDK] Add set lock duration

Format Check #172: Pull request #396 opened by ryangoree

7 hours ago 2m 16s `ryan-lock-duration`

Add fallback UI on mobile when no proposals exist (#395)

Deploy Council UI to GH Pages #337: Commit 398ac29 pushed by DannyDelott

11 hours ago 3m 41s `main`

Update localhost.ts default config (#394)

Deploy Council UI to GH Pages #336: Commit b336b52 pushed by ryangoree

12 hours ago 3m 18s `main`

Add fallback UI on mobile when no proposals exist

Format Check #171: Pull request #395 opened by DannyDelott

12 hours ago 2m 10s `danny-fallback-proposals-…`

Add fallback UI on mobile when no proposals exist

Dependabot auto-approve #658: Pull request #395 opened by DannyDelott

12 hours ago 2s `danny-fallback-proposals-…`

Add fallback UI on mobile when no proposals exist

Build #171: Pull request #395 opened by DannyDelott

12 hours ago 3m 35s `danny-fallback-proposals-…`

Add fallback UI on mobile when no proposals exist

Lint #660: Pull request #395 opened by DannyDelott

12 hours ago 2m 40s `danny-fallback-proposals-…`

Update localhost.ts default config

Format Check #170: Pull request #394 opened by ryangoree

12 hours ago 2m 0s `ryangoree-patch-1`

Update localhost.ts default config

Lint #659: Pull request #394 opened by ryangoree

12 hours ago 3m 7s `ryangoree-patch-1`

Update localhost.ts default config

Dependabot auto-approve #657: Pull request #394 opened by ryangoree

12 hours ago 2s `ryangoree-patch-1`

Update localhost.ts default config

Build #170: Pull request #394 opened by ryangoree

12 hours ago 3m 23s `ryangoree-patch-1`

[CLI] Fix voting power errors on localhost due to stale block lag (#393)

Deploy Council UI to GH Pages #335: Commit e21230b pushed by ryangoree

2 days ago 1h 8m 25s `main`

[CLI] Fix voting power errors on localhost due to stale block lag

Build #169: Pull request #393 synchronize by ryangoree

2 days ago 3m 29s `ryan-testnet-accounts`

[CLI] Fix voting power errors on localhost due to stale block lag

Dependabot auto-approve #656: Pull request #393 synchronize by ryangoree

2 days ago 2s `ryan-testnet-accounts`

[CLI] Fix voting power errors on localhost due to stale block lag

Format Check #169: Pull request #393 synchronize by ryangoree

2 days ago 2m 32s `ryan-testnet-accounts`

[CLI] Fix voting power errors on localhost due to stale block lag

Lint #658: Pull request #393 synchronize by ryangoree

2 days ago 2m 50s `ryan-testnet-accounts`

Core Voting Contract

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: CoreVoting.sol
- Type/Category: Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/CoreVoting.sol

1. Introduction (Summary)

The core voting contract allows a voting process for those who have been allocated governance power. It allows them to select which calls are executed from the core voting contract address and does not make assumptions about how the voting power is allocated. Instead, it calls designated contracts that calculate how much voting power a user has for the voting contract. Calls can be executed at a variety of different security thresholds for various different external functionalities. The contracts which calculate votes for the core voting contract are called voting vaults, and the sum of what they indicate for a user at a timestamp is that user's voting power.

2. Contract Details

The core voting allows the proposal of a package of calls to be made from the core voting contract. Each of these calls corresponds to some action on the Ethereum blockchain. Each proposal must be proposed by either an authorized user [Governance steering council] or by someone with sufficient voting power. Then, each of the users who have voting power can cast a vote for, against, or neutral on each proposal. If the proposal gains more votes than a security threshold, the proposal can then be executed after some minimum time, and before an expiration time. An authorized address can change all parameters of the voting contract. In a normal operation, that address is the timelock contract. The core voting contract stores a mapping containing previous proposals organized according to an id associated with each proposal, timelock address, minimum and maximum voting times. Also, the minimum power needed to propose a vote, a mapping that records custom quorums, a default quorum, and a listing of approved voting vaults.

3. Key Mechanisms & Concepts

The voting contract accepts in calldata a list of vaults in which the user may have voting power to calculate the voting power for a user, the core voting contract then checks that each vault in the list is approved, and calls them with the sender's address, timestamp, and optional extra data. The voting vault responds with a number of votes allocated to the user. To create a proposal the creator's voting power is checked and then the proposal is created by assigning the last unused proposal id to the new proposal, and storing its data into the proposals mapping. After the proposal is created, a user can vote on a proposal by indicating the proposal id they would like to vote on. Then, their voting power is calculated, and added to a running tally of the votes for yes, no, or abstain. Votes cannot be made on a proposal after a max time from proposal creation. If a proposal has more votes cast than the quorum threshold for the action/s being taken, and more yes votes than no votes, then the proposal can be executed. Proposal execution will fail if the proposal is past an expiration timestamp.

4. Gotchas (Potential source of user error)

- Votes can be changed or removed by calling the vote function again, with a new ballot to change them, or with an empty set of voting vaults to remove them.
- All actions in a proposal must succeed or the execution of that proposal will revert. Therefore, any proposal must be created without any calls that have a high probability to revert.
- Anyone can execute the transactions approved by the governance system, therefore, any bundle of transactions made should be vetted to not have exploitable downstream effects if sandwiched by an adversary.
- The quorum of a bundle of onchain actions is set as the maximum in that bundle, meaning that it should be vetted if the quorum is lowered. For an action it should be secure even if called multiple times in the same transactions or with other low quorum actions.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- Since the voting contract can change its own configuration it can put itself into a position where it is no longer functional by making improper changes to its own variables. There is no recovery mechanism for this format of bricking so the governance contract variable upgrades must be held to the highest security standards.
- The modular nature of the governance systems is designed to enable upgradeability. In the case of a component like a voting vault fails, it should be voted on to be removed from the governance contract. In a case where the core voting contract is out of date, it can change the core voting contract in the time lock and other permissioned contracts to replace itself.

▸ **Pages** 25

Council Wiki

- Home
- Council Protocol 101

Technical Guides

- Getting Started
- Council Deployment Guide
- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Council Deployment Guide

Jump to bottom

William Villanueva edited this page on Mar 28 · 6 revisions

If you're interested in learning how to deploy your own version of the Council governance contracts on the blockchain, you've come to the right place.

Through this guide you will learn how to do the following:

1. Set up the Council deployment scripts
2. Adjust Council to suit your DAO's needs
3. Deploy the Council contracts to the blockchain

The Council Kit includes apps and packages that will help you with various tasks around the deployment and usage of the Council governance framework. This guide will focus on how to use the `council-deploy` package to deploy the Council contracts onto the blockchain.

Requirements

- Git
- Node 14+
- Yarn package manager
- Etherscan API Key
- Node provider URI and API Key (e.g. Alchemy)
- Private Key for an Ethereum address to deploy the contracts

Clone the council-kit repository

The first step is just getting the code. We'll do that by using `git` to clone the repository from GitHub, so open up your preferred command line terminal and run the following command:

```
git clone git@github.com:element-fi/council-kit.git
```

> **NOTE** If you want to make some changes to the codebase and/or eventually deploy your own flavor of the Council Reference UI, you'll probably want to fork the repo on GitHub and clone your fork instead

If you take a look at the directory structure, you'll find that this guide will focus on the `packages/council-deploy` package.

Fetch Contract Code and Install Dependencies

Run the following command on the root directory council-kit to install the required dependencies:

```
yarn install
```

Since we're deploying contracts onto the blockchain, and using the Etherscan API to verify them, we also need to fetch the contract code with the following command:

```
yarn workspace @council/deploy run load-contracts
```

Now we can build the project's packages:

```
yarn build
```

Finally, don't forget to create a `packages/council-deploy/.env` file and add your environment variables, namely the Alchemy and Etherscan API keys, and the deployer address private key. You can just use the `packages/council-deploy/.env.sample` file as a template.

Customizing Council

If you've followed the guide up to this point, you can already deploy a default version of Council by using the `packages/council-deploy/src/deployCouncil.ts` script (just jump to the Deploying Council Contracts section)

However, chances are that you'll need to make some configuration adjustments to the script for this Council deployment to fit your DAO's structure and needs, such as:

- Will the DAO use a Governance Token? If so, does it already have one on the blockchain?
- Will the DAO use a Governance Steering Council?
- Does the DAO have an existing Treasury contract?
- Which Voting Vaults will the DAO use?

The Council Kit provides a default implementation for all of these situations. This section will explore some of these configuration options.

Customizing: Governance Token

The first few lines of the `deployCouncil.ts` script execute the deployment of a standard ERC20 token contract with the given token name and symbol, to be used for Governance through Council:

```
// The voting token is used to determine voting power in the Locking Vault and
// Vesting Vault. It has no dependencies on any of the council contracts.
const votingToken = await deployVotingToken({
    tokenName: "Council Voting Token",
    tokenSymbol: "CVT",
    signer,
});
```

Scenario 1: The DAO doesn't have a Governance Token, but wants to use one

In this case, just change the `tokenName` and `tokenSymbol` parameters into the DAO's preferred values for their Governance Token. Council's deployment script will handle the token contract's deployment and verification.

Scenario 2: The DAO already has a Governance Token, or wants to use a custom one

If the DAO already has its own Governance token, you can skip this token deployment by removing (or commenting out) the aforementioned lines of code.

You'll also have to adjust the code that depends on the votingToken address, namely the Locking vault and Vesting vault deployments. To do that, change the lines:

```
votingTokenAddress: votingToken.address
```

into your Governance token address. For example:

```
votingTokenAddress: "0x5c6D51ecBA4D8E4F20373e3ce96a62342B125D6d"
```

In this example, we've set the votingTokenAddress to Element's ELFI contract, whose address is `0x5c6D51ecBA4D8E4F20373e3ce96a62342B125D6d`

Customizing: The Governance Steering Council (GSC)

The next few lines of code in the `deployCouncil.ts` script handle the deployment of the GSC Core Voting contract, which gives the Governance Steering Council its privileged permissions:

```
// The GSC Core Voting is a privileged voting contract
...
const gscCoreVoting = await deployGSCCoreVoting({
  signer,
  ...
    votingVaultAddresses: [],
  ...
    ownerAddress: signer.address, // temporary assignment
  ...
  baseQuorum: "1",
  lockDuration: 10,
```

```
  extraVotingTime: 15,
});
```

It also uses a gscVault to keep track of voting power:

```
const gscVault = await deployGSCVault({
  signer,
  ownerAddress: timelock.address,
  // GSC vault depends on core voting contract to prove that members meet the
  // voting power minimum to be on the GSC.
  coreVotingAddress: coreVoting.address,
  // any test account can get onto GSC with this much vote power
  votingPowerBound: "100",
  // members are idle for 60 seconds after they join the GSC
  idleDuration: 60,
});
```

Scenario 1: The DAO wants to have a GSC

In this case, you'll just want to adjust the parameters that control the GSC's privileges in the respective gscCoreVoting and gscVault contracts:

gscCoreVoting:

- `baseQuorum`: Controls how many GSC members are needed to pass a proposal. Default value is 1.
- `lockDuration`: Controls how many blocks to wait from creation time before a proposal can be executed (provided it has reached quorum and is passing). For reference, 1 week is around 300,000 blocks.
- `extraVotingTime`: Controls how long to wait from unlocking time (see `lockDuration`) before a proposal is closed for voting.

gscVault:

- `votingPowerBound`: How much voting power a user needs to be able to join the GSC. This value should be set according to the DAO's token supply plan.
- `idleDuration`: This is the amount of time a new member of the GSC must wait before they can do anything as a GSC member.

Scenario 2: The DAO doesn't need a GSC

In this case, you can skip deployment of the gscCoreVoting contract and the gscVault contract:

Adjust references to gscCoreVoting by setting them to the `0x00` address:

```
const timelock = await deployTimelock({
...
  gscCoreVotingAddress: "0x0000000000000000000000000000000000000000"
...
```

```
const coreVoting = await deployCoreVoting({
...
  gscCoreVotingAddress: "0x0000000000000000000000000000000000000000"
...
```

Remove the following code, which updates some parameters on the gscCoreVoting contract:

```
await gscCoreVoting.contract.changeVaultStatus(gscVault.address, true);
console.log("Approved GSCVault on GSCCoreVoting");

// Now we transfer ownership to the Timelock, any future upgrades to
// GSCCoreVoting must go through the normal proposal flow.
await gscCoreVoting.contract.setOwner(timelock.address);
console.log("Set owner of GSCCoreVoting to Timelock");
```

With that, your deployment will not include any functionality related to the GSC.

Customization: The CoreVoting Contract

- `baseQuorum`: Controls how much voting power is needed for a proposal to pass. This value should be set according to the DAO's token supply plan.
- `minProposalPower`: Controls how much voting power is needed to create a proposal. This value should be set according to the DAO's token supply plan.
- `lockDuration`: Controls how many blocks to wait from creation time before a proposal can be executed (provided it has reached quorum and is passing). For reference, 1 week is around 300,000 blocks.
- `extraVotingTime`: Controls how long to wait from unlocking time (see `lockDuration`) before a proposal is closed for voting.

Customization: The Timelock Contract

- `waitTimeInBlocks`: This is the amount of blocks that you have to wait before a call can be executed by the Timelock.
- `gscCoreVotingAddress`: The GSC has a special permission to increase the time that a proposal must wait before it can be executed. This is a security feature. If your deployment does not include a GSC, you should set this to the 0x00 address.

The Timelock contract is where proposals get executed. It is owned by the CoreVoting Contract. This way, proposals can register calls on the timelock.

At the same time, the Timelock contract owns the CoreVoting contract. As such, the only way to upgrade the CoreVoting contract is through a proposal which gets executed through the timelock.

Customization: The Treasury Contract

If the DAO does not have a treasury contract they want to use with Council, the script will deploy a treasury contract.

However, if the DAO does have its own treasury contract, it doesn't need to deploy a new one, so you can just remove the following code:

```
const treasury = await deployTreasury({
  signer,
  ownerAddress: timelock.address,
});
```

It is recommended that Council's timelock contract should be the owner of the Treasury contract, so that any usage of the treasury must go through the proposal process.

Customization: Voting Vaults

The Council Kit includes two default vault implementations: the Locking vault and the Vesting vault. They both require the use of a Governance Token or Voting Token. If the DAO already has its own Governance Token that they'd like to use with Council, make sure to update both vault deployments with the token's contract address by changing:

```
votingTokenAddress: votingToken.address
```

into your Governance token address. For example:

```
votingTokenAddress: "0x5c6D51ecBA4D8E4F20373e3ce96a62342B125D6d"
```

In this example, we've set the votingTokenAddress to Element's ELFI contract, whose address is `0x5c6D51ecBA4D8E4F20373e3ce96a62342B125D6d`

The Vesting Vault gives voting power to tokens that are still being vested, but applies a multiplier to the unvested tokens, which allows the DAO to reduce their voting power. This multiplier can be set in the `packages/council-deploy/src/vaults/deployVestingVault.ts` file, by adding a line to set this value before the timelock address is set:

```
await vestingVault.initialize(signer.address, signer.address);

// Set the initial unvestedMultiplier value
await vestingVault.changeUnvestedMultiplier(10)

// Only the Timelock can update things like the unvestedMultiplier.
await vestingVault.setTimelock(timelockAddress);
```

In this example, the line `await vestingVault.changeUnvestedMultiplier(10)` sets the multiplier to 10%, so vested tokens will have 0.10 as much voting power as unvested tokens.

Deploying the Council Contracts

Contracts are deployed using the private key of the wallet specified in the .env file, see: `GOERLI_DEPLOYER_PRIVATE_KEY`.

Once all contracts are deployed, each one will be verified on Etherscan automatically. (Don't forget to add your `ETHERSCAN_API_KEY` to the `.env` file.)

The entire process takes around 10 minutes. This is because some contracts need to interact with other contracts. It also takes a little extra time for Etherscan to index new contracts before they can be verified.

Goerli: Run the following command:

```
yarn workspace @council/deploy run goerli:deploy-contracts
```

Mainnet:

```
yarn workspace @council/deploy run mainnet:deploy-contracts
```

This will run the `deployCouncil.ts` script to deploy all the included contracts:

```
council-kit % yarn workspace @council/deploy run goerli:deploy-contracts
yarn workspace v1.22.18
yarn run v1.22.18
$ cross-env TS_NODE_TRANSPILE_ONLY=1 npx hardhat run src/goerli.ts --network goerli --no-compile
prompt: Name this deployment:  (Goerli Deployment #2) Mazy's Council
Signer: 0xcF3b7bcBbcEFF836F81f6EAd914706E699267bca
Deployed VotingToken
Deployed GSCCoreVoting
Set GSCCoreVoting lockDuration and extraVoteTime
Deployed Timelock
Deployed Treasury
Deployed LockingVault
Deployed LockingVault proxy
Deployed VestingVault
Set Timelock permissions on VestingVault
Deployed VestingVault proxy
Deployed CoreVoting
Set CoreVoting lockDuration and extraVoteTime
Set owner of CoreVoting to Timelock
Deployed GSCVault
Approved GSCVault on GSCCoreVoting
Set owner of GSCCoreVoting to Timelock
Set owner of Timelock to CoreVoting
All contracts deployed!
Updated goerli.deployments.json!
```

And with that, you're done. All the required contracts are now deployed on the blockchain, so once you've distributed some tokens, you can start doing some Governance.

You'll find that running the script generated a `packages/council-deploy/src/deployments/goerli.deployments.json` file which contains data about all the contracts that were deployed, should you need it for future reference.

The next step would be to create a web UI to interact with the Council contracts. Luckily, the Council Kit includes a Reference UI, so you can follow the Reference UI Deployment Guide to get started!

Clone this wiki locally

https://github.com/delvtech/council-kit.wiki.git

Council Glossary

Jump to bottom

Alim edited this page on Mar 14 · 13 revisions

Voting

Voting Power:

The basic unit of governance power used for voting on proposals, which determines whether proposals pass or fail. Governance participants hold Voting Power, which they can then delegate for it to be usable in governance.

Delegation:

Governance participants can delegate their voting power/voting rights to an address. Delegation can be given to one address at a time, including the holder’s own address. Note that delegation does not lock tokens; it simply adds votes to the chosen delegate address. This allows token holders' votes to be used in governance through their delegates, which reduces both the financial and mental cost of participating in governance for individual holders.

Proposal:

Formal request to make a change on how any part of the protocol works. This can include, but is not limited to: adding, deleting or updating smart contract code, changing the protocol's stated mission or vision, defining how to spend treasury assets, executing any kind of on-chain transaction, changing governance processes and parameters, among others. Users will decide on whether a proposal passes or fails by using their Voting Power. Proposal types may include Protocol (Executable) Proposals and DAO (Social) Proposals, but the specific proposal types depend on the governance framework defined by the DAO.

Voting on Protocol Proposals:

Users can vote for or against single proposals once they have voting rights delegated to their wallet address. Votes can be cast while a proposal is in the “Active” state; the proposal may be queued in the Timelock.

Governance Steering Council (GSC):

Group of delegates who have surpassed a pre-established threshold of delegated Voting Power (by the DAO) and have claimed this governance role. The GSC holds additional governance powers, as well as the additional responsibilities that come with them. The extent of the GSC's functions and permissions is to be decided by the DAO through a governance proposal. If a GSC member’s delegated voting power falls below the DAO-defined member threshold, any delegate can “kick” them from the GSC at which point they will lose their additional powers.

Quorum:

Minimum amount of votes required for a proposal's voting results to be considered valid. The purpose of this quorum is to ensure that the only measures that pass have adequate voter participation. More specifically, for a proposal to succeed, a minimum amount of voting power (defined by the DAO) must participate in the vote. The quorum value may change as governance begins to understand participation rates but all changes to quorum requirements must be ratified by the governance community.

Voting Period (Off-chain polling):

The total voting period in which votes can be cast on an off-chain polling platform (e.g. Snapshot), if the DAO has chosen to include an off-chain polling step in their governance framework before moving proposals to an on-chain vote.

Voting Period (On-chain):

The total voting period in which votes can be cast. [Minimum voting period][extra voting period] is the layout of the total time for voting. On-chain proposals have a voting period that is to be defined by the DAO.

Timelock:

The timelock is a speed bump for calls (i.e., the smart contract code to be executed if the proposal passes). It requires calls to wait for a period before they can be executed. After a call is registered, the call can be removed during the waiting period, and an authorized address can extend the waiting period only once.

Voting Vaults

Voting Vaults:

Set of smart contracts that determine how voting power is counted in the system. Each voting vault can be programmed to measure any metric to distribute voting power. For example, one voting vault could assign 1 vote per token deposited, another one could assign 0.05 votes per token vested, while another could assign 1000 votes per successful governance proposal submitted by a user. This gives the system enough flexibility to evolve alongside the needs of the DAO.

Voting Vaults provide the ability to assign voting power to specific types of tokens/positions. The result is beautiful — governance users can maximize capital efficiency while maintaining the ability to delegate or vote when the time comes. Voting vaults are designed to be upgraded, allowing for voting vault logic to be updated by governance and to avoid the pains that come with governance upgrade migrations.

The creation of a voting vault begins with defining a strategy for counting votes / providing voting power. Once the logic has been established, governance will vote to approve the method, and if successful, the integration can be completed, following which users can start voting.

Locking Vault:

The locking vault allows users to deposit their tokens into a contract in exchange for voting power, which can also be delegated to a different user. The vault tracks the historical voting power of each address and, when asked for voting power, searches the historical record of that address' voting power at the time when the vote was proposed.

Vesting Vault:

The vesting vault has the same functionality that the locking vault has, but for vesting. This vault allows locked / vesting positions to still have voting power in the governance system and does so by using a defined multiplier for the vested tokens over unvested. For example, vested tokens could count for 0.05 votes each while unvested tokens count for 1 vote each.

Governance Steering Council (GSC) Vault:

The GSC vault enables a mode of governance that is distinct from other voting vaults. It gives one vote to each member of the Council. When attached to a core voting contract, it enables the council members to vote on proposals and execute them only with agreement from the Council. The Council is selected by a continuous election of governance power in another core voting system.

- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

README.md

Council-Kit

Council kit is the complete toolkit for scaffolding your own DAO using the council protocol. This repository is designed to be forkable for your own council protocol variant.

Get started with the Docs.

Requires `Node 14+`.

What's inside?

This monorepo uses Yarn as a package manager. It includes the following packages/apps:

Apps

Name	Description
council-ui	A reference user interface built with NextJS and using the SDK.
council-sdk-starter	A boilerplate TypeScript project that uses the SDK.

Packages

Name	Description
council-sdk	A TypeScript SDK for interfacing with the Council protocol.
council-deploy	A template for deploying council contracts on Goerli.
council-typechain	Type classes generated from the council protocol contracts using Typechain.

Utility Packages

Name	Description
eslint-config	Package for static type checking.
prettier-config	Package for code formatting.
tsconfig	Package for TypeScript configuation.

Each package/app is 100% TypeScript.

Monorepo commands

Build

To build all apps and packages, run the following command:

```
yarn build

# build only packages (useful for app development)
yarn build:packages
```

Develop

To run the development server for all apps, run the following command:

```
yarn dev
```

Linting

To run linting on all apps and packages, run the following command:

```
yarn lint
```

Formatting

To run formatting on all apps and packages, run the following command:

```
yarn format:check

# write formatting changes to files
yarn format
```

Development

1. Clone the repo: `git clone git@github.com:delvtech/council-kit.git`
2. Run `yarn` at the top-level to install all packages across every workspace

Installing new packages

Here are a few examples of how to install packages in this repo:

```
# Install prettier for the top-level package.json, useful for tooling that
# runs against the entire monorepo
yarn add prettier

# Install lodash for the packages/council-sdk workspace.
# Note: specify the workspace by the name in its `package.json`, ie: `@council/sdk` not `c
yarn workspace @council/sdk add lodash
```

Installing a workspace package

To install a project from the packages/ directory as a dependency, copy it directly into your app package.json like this, then run `yarn`.

```
{
  "dependencies": {
```

```
    "@council/sdk": "*",
    "@council/typechain": "*"
  }
}
```

Running workspace scripts

To run scripts in workspaces, use the following command:

```
yarn workspace <workspace-name> <package.json script>
```

Example

```
yarn workspace council-ui start
```

Turborepo

This package uses the turborepo monorepo framework. Turborepo provides additional features compared to a plain monorepo such as local/remote command caching and parallel npm script execution.

Remote Caching

Turborepo can use a technique known as Remote Caching to share cache artifacts across machines, enabling you to share build caches with your team and CI/CD pipelines.

By default, Turborepo will cache locally. To enable Remote Caching you will need an account with Vercel. If you don't have an account you can create one, then enter the following commands:

```
cd my-turborepo
npx turbo login
```

This will authenticate the Turborepo CLI with your Vercel account.

Next, you can link your Turborepo to your Remote Cache by running the following command from the root of your turborepo:

```
npx turbo link
```

Resources

Learn more about the power of Turborepo:

- Pipelines
- Caching

- Remote Caching
- Scoped Tasks
- Configuration Options
- CLI Usage

Releases

No releases published

Packages

No packages published

Contributors 9

Languages

TypeScript 93.2% **Solidity** 6.3% **Other** 0.5%

Council Protocol 101

Jump to bottom

Alim edited this page on Mar 8 · 8 revisions

The current state of most on-chain governance is often a monolithic contract with little room for adaptability, customization, or scalability.

Most on-chain contracts require large transaction and audit costs. Monolithic on-chain contracts also create significant friction in conducting routine operations. Each transaction, update, or change to the core protocols requires the consensus of the entire DAO, leading to governance fatigue, slow-moving operations, and overall inefficiencies.

Consequently, DAO governance has skewed heavily towards hybrid governance, which foregoes on-chain execution in favor of off-chain signaling tools such as Snapshot. DAOs employing this model typically verify the chain's status to endow voting power. Once consensus is achieved, they often depend upon trusted community members to manually execute those decisions using multisigs, introducing a single point of failure for operations and a potential security risk.

The Council Protocol introduces a new governance system that allows for improvements, flexibility, and experimentation while successfully maintaining the security and robustness of the Protocol. It entails a set of foundational contracts designed with modularity in mind. Modular governance frameworks allow for customization, enabling DAOs to add and remove modules based on their evolutionary needs. It empowers decentralized governance by leveraging:

- Voting power delegation (i.e. a refined form of representative democracy)
- Capital-efficient voting via a novel governance primitive called Voting Vaults
- A community entrusted Governance Steering Council (GSC)

Core voting and delegation of voting power provide basic functionality for the governance system while voting vaults allow the governance ecosystem to be more inclusive and give governance participants a capital efficient way to be active in protocol governance without sacrificing yield opportunities across DeFi. Delegation, Voting Vaults and the GSC will be explained in more detail later in this guide.

Core Voting

Central to the Council Protocol is the Core Voting Contract. The Core Voting Contract defines voting power, tracks proposals, and measures voting power prior to the execution of any proposal. It also allows various contract parameters to be configured:

- **Customizable Voting Strategies**: When a user votes, the Core Voting Contract will reference a pre-approved list of voting strategies. These strategies define the calculation of voting power and validate a user's voting power for that particular strategy, enabling multiple methods to exist simultaneously.
- **Customizable Quorum**: A customizable quorum threshold can be set which checks that the minimum amount of votes required for a proposal's voting results to be considered valid before a contract call to be executed. The quorum value may change as governance begins to understand participation rates but all changes to quorum requirements must be ratified by the governance community. Ultimately, this means that governance can vary the security threshold on any on-chain action with very high granularity.
- **Optional Timelock**: A timelock is typically only used for security-critical votes. This means increased efficiency for more minor proposals and measured protections for highly sensitive proposals.

An overview of governance roles, voting vaults, and delegation will provide the reader with the requisite knowledge to understand how the baseline framework works and what properties can be configured to suit the particular needs of a DAO.

Governance Roles

Delegates

You can assign all of your Voting Power in the governance system to yourself or someone else, and they can vote on your behalf. This is called Delegation. It's important that you select a delegate who is aligned with your vision for how the DAO should evolve, as your votes would be counted towards their selection.

Governance Steering Council (GSC)

The GSC is a group of delegates, each of whom has reached a pre-established threshold of delegated voting power, giving them additional governance powers within the system, and as a result, additional responsibilities.

The opportunity to be on the GSC is open to anyone and can be earned by garnering enough delegated voting power. GSC members must stay aligned with their delegators and relate to the general sentiments of the governance community to maintain their positions on the Council. Otherwise, they risk losing their delegated votes to another GSC member or new governance community member aspiring to better represent the interests of the community. Removal from the GSC happens as a result of failing to maintain delegated votes and could indicate the outgoing GSC member’s lack of impact or disconnect to the general community of voters.

It is possible for the DAO to change the function of the GSC, including the possibility of adding special functions (propose votes directly on-chain, spend a portion of treasury funds at their discretion, etc.), and various responsibilities (DAO2DAO relationships, collaborations, treasury management, community engagement, etc.), and might (depending upon a vote) be compensated for the time and effort that they dedicate to improving the governed protocol. All of these functions and responsibilities must be defined and ratified through the governance process.

Note: being a delegate and a GSC member is not necessarily mutually exclusive. Learn more in this blog post.

Voting Vaults and Delegation

What are Voting Vaults?

Voting vaults are smart contracts that allow any programmable logic to be used for allocating voting power to governance participants. Some example metrics include:

- Reputation or merit-based systems
- User protocol usage metrics
- User governance participation data
- Token-holding
- Positions in DeFi protocols (staked assets, collateral, LP positions, etc.)
- Any other metric or combination of metrics

Voting Vaults provide the ability to assign voting power to specific types of tokens/positions. The result is beautiful — governance users can maximize capital efficiency while maintaining the ability to delegate or vote when the time comes. Voting vaults are designed to be upgraded, allowing for voting vault logic to be updated by governance and to avoid the pains that come with governance upgrade migrations.

The creation of a voting vault begins with defining a strategy for counting votes or providing voting power. Once the logic has been established, governance will vote to approve the method, and if successful, once the integration is complete, users can start voting.

Examples of commonly used Voting Vaults include:

- **Locking Vault** – allows users to deposit their tokens into a contract in exchange for voting power, which can also be delegated to a different user.
- **Vesting Vault** – allows locked / vesting positions to have voting power in the governance system and does so by using a defined multiplier for the vested tokens over unvested tokens.
- **GSC Vault** – enables a mode of governance that is distinct from other voting vaults; it provides one vote to each member of the GSC. Members can create, vote, and execute proposals if the GSC quorum is met, with quorum set by the DAO.

Learn about some of the possible voting vaults that can be created in this blog post.

How does delegated voting work?

Anyone with Voting Power can assign all of their Voting Power in the protocol to someone else (i.e., a delegate), who may vote on their behalf. This is called Delegation. It is important that the assignor selects a delegate who is aligned with their vision for how the protocol should evolve, as the assignor’s votes would be counted towards their voting selection.

This alleviates the issues of having to keep up with the multitude of discussions that happen surrounding the protocol, and having to interact with on-chain contracts for each decision that needs to be made. Reducing the governance load on each participant ultimately helps to reduce some of the friction and encourage higher levels of voter participation in governance decisions.

Parameters that Governance Operates

General Governance

- **Quorum**: The required minimum number of voting power in support of a proposal for it to succeed (this value is fixed and not a percentage of total voting power).
- **Timelock Duration**: The timelock is a speed bump for calls. It requires calls to wait for a defined period before they can be executed. After a call is registered, the call can be removed during the waiting period, and an authorized address can extend the waiting period only once.
- **Voting Period**: The total voting period in which votes can be cast. [Minimum voting period][extra voting period] is the layout of the total time for voting.
- **Minimum Voting Period**: The minimum voting time before a vote can be executed.
- **Extra Voting Period**: The remaining time beyond the minimum voting period in which votes can be cast.
- **Proposal Threshold**: The amount of voting power needed to submit a governance proposal on-chain.
- **Vesting Vault Token Multiplier**: The vesting vault contract is used to set up long-term grants for core contributors to the DAOs ecosystem. It is also a voting vault so that community members can vote with vested but unclaimed tokens and unvested tokens. The multiplier for voting power on the vesting vault allows locked / vesting positions to still have voting power in the governance system and does so by using a defined multiplier for the vested tokens over unvested.

GSC Parameters

- **GSC Delegation Threshold**: A threshold of delegated voting power, giving delegates a seat on the GSC. This comes with additional governance powers within the system, and as a result, additional responsibilities.
- **GSC Quorum**: The required minimum number of GSC members needed to support a proposal for it to succeed.
- **GSC Proposal Threshold**: The amount of GSC members needed to propose a GSC vote on-chain.

Protocol Parameters

- **Protocol Upgrades**: Should the governed DAO community decide to upgrade the governed Protocol, it will go through governance. These types of changes will go through the process defined for Protocol Proposals.

▸ **Pages** 25

Council Wiki

- Home
- Council Protocol 101

Technical Guides

- Getting Started
- Council Deployment Guide
- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Council Protocol Overview

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

Welcome to the council-monorepo wiki!

- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Council Protocol Smart Contracts

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

Introduction

The Council Protocol contains several architectural choices which make it a distinct new primitive in the decentralized governance space:

- Council does not have a single security threshold to make a call, instead, various actions can be given different security threshold requirements.
- Council abstracts the vote allocation process for assigning voting power away from the actual voting process meaning that multiple complex vote allocation systems can run in parallel in the contracts.
- By default, Council ships with a Governance Steering Council (GSC) enabled which can be assigned different powers than the core voting system. Together, these features allow a wide range of voting processes and security procedures can be seamlessly integrated into one governance system.

Architecture Overview

Resources

- Introduction to Element's Governance System blog post.
- Github Repository
- To learn more about the technical architecture of Council, read this blog post.

- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

DAO (Social) Proposal Template

Jump to bottom

CPSTL edited this page on Mar 7 · 1 revision

GP-X: [Title]

```
GP#: <# to be assigned>
Title: <GP title>
Author(s): <list of authors' names and/or Twitter/GitHub handles>
Type: DAO Proposal
Status: <Assigned by Forum moderators >
Date Proposed: <yyyy-mm-dd>
Date Ratified: <yyyy-mm-dd>
```

References

- A list of supporting materials referenced by this GP

Sentence Summary

- A description of what the Governance Proposal (GP) is focused on. Suggest 30 words max.

Paragraph Summary

- A description of what the Governance Proposal (GP) is focused on. 100 words max is recommended.

Motivation

- A short description of the motivation behind the GP.

Specification / Proposal Details

- Proposed process standard details - describes the new process or feature and the problem it is solving.

Next Steps (Voting Outcome Summary)

- Clearly outline the next steps for this proposal. This includes what voting “yes” and “no” entails and what the outcome and next steps would be if the proposal were to pass the Off-Chain Poll and On-Chain Vote.
- **For example:**
  - Discuss this proposal on the forum and discord.
  - Implement a poll on this proposal on Snapshot.
    - If passed: This proposal will move to an On-Chain Vote
  - Implemented a vote on this proposal on the Governance Dashboard
    - If passed: This proposal will be executed and the following will be deemed as a new DAO process, etc.

- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Getting Started

Jump to bottom

Ryan Goree edited this page on Mar 9 · 11 revisions

Deploying your Council Smart Contracts (CLI)

- The process for properly deploying the smart contracts is detailed in the Contracts deployment guide.
- @council/deploy: This is a command-line tool that simplifies the process of deploying smart contracts to the blockchain. It also serves as a helpful reference for developers who want to understand how the deployment process works in more detail.
- Council Smart Contracts repo mock deploy script: This is another deployment script that can be used as a reference to understand how to deploy contracts manually, without using the CLI tool.

Setting up the UI config object

- The process for setting up and deploying the Reference UI to GitHub Pages is detailed in the UI deployment guide.
- Before deploying the reference UI, you'll need to adjust the UI config object to point to the correct addresses for your deployed smart contracts.
- If you've deployed your contracts to the Goerli test network, start by editing the Goerli config in the `apps/council-ui/src/config/goerli.ts` file. If you used @council/deploy, you can use the helper code to grab addresses from the latest deploy. They should all be gathered in `packages/council-deploy/src/deployments/goerli.deployments.json`
- For the Ethereum mainnet, edit the Mainnet config instead in the `apps/council-ui/src/config/mainnet.ts` file.
- If you've deployed to a network other than Goerli or the mainnet, you'll need to create a new config in the `apps/council-ui/src/config/` directory based on one of the existing configs. Then, add the new config to the `council.config.ts` file.

Deploying the static UI

- Now that the UI config is done, you can deploy the Council Kit Reference UI. This is also covered in the UI deployment guide.
- The Reference UI uses Alchemy to fetch data, so you'll need to acquire Alchemy API keys for each configured chain.

- If you added new chains, you'll have to add them to the node provider object in the `apps/council-ui/src/provider.ts` file.
- There are two deployment options outlined in this section of the guide: GitHub Pages and Vercel.
- To use GitHub Pages, fork the repo and configure the Pages settings to use "GitHub Actions" as the build and deployment source. You'll also need to add some repository secrets to enable the deployment workflow. You can follow this UI deployment guide for more details on this process.
- To use Vercel, follow their documentation on deploying with Git.

Updating a new proposal when it gets created

- When a proposal is submitted on-chain, it should show up in the list of proposals in the user interface right away. However, the config can be updated with additional metadata that isn't available on-chain, as well as the targets and calldatas for the proposal.
- The process for adding new proposal data is outlined in the Adding New Proposal Data section of the README.

- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Governance Steering Council (GSC) Vault

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: GSCVault.sol
- Type/Category: Voting Vault/Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/vaults/GSCVault.sol

1. Introduction (Summary)

The GSC vault enables a different mode of governance than the rest of the voting vaults. It gives one vote to each member of a council. When attached to a core voting contract, it enables the council members to vote on proposals and execute them only with agreement from the council. The council is selected by a continuous election of governance power in another core voting system.

2. Contract Details

Key Functionalities:

- Allow anyone to join the governance steering council if they prove they meet a minimum voting power in the DAO.
- Allow anyone to kick a member if their voting power falls below a threshold.
- Give each member one vote after they have been on the council for a warmup period.

Storage layout:

- A mapping tracks who is a member and when they joined.
- An upgradable bound on how many votes are needed to join or remain on the council.
- An upgradable warmup period length.

3. Key Mechanisms & Concepts

The Governance Steering Council (GSC) is designed to be a group of people who take on extra responsibility in the DAO and gain extra powers. To join, the members must maintain some minimum amount of voting power in the DAO. The members can vote on issues through a copy of the core voting contract [which is not the same voting contract as in the primary DAO]. To maintain the list of the GSC members, we use a model where the prospective member should prove membership by proving they have enough voting power in the DAO. To remove a member, they are challenged. The voting vaults which initially contained their voting power are queried and, if they have not maintained enough power, they are removed.

4. Gotchas (Potential source of user error)

- If there is a vault migration where the source of member voting power changes, they have to reprove their membership. This reproof does not cause another idle period before they can vote.
- Since the GSC vaults stores the data of the voting vaults, the member votes and queries them again on kick. The voting vaults of the DAO must enforce that a voting power query that previously didn't revert does not start reverting. If that happens, members may become un-kickable.
- Some vaults require extra data, for example, to provide a Merkle proof or signature, in order to kick a member who joins using those vaults the kicker must provide valid data as well.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- In the case of corruption or inaction by the GSC, the DAO can vote to vote on the GSC core voting contract and be given a high number of votes. Meaning that the decisions of the GSC can be overridden by the DAO via vote.
    - This also is a remediation path to many bugs in the GSC vault.

▸ **Pages** 25

Council Wiki

- Home
- Council Protocol 101

Technical Guides

- Getting Started
- Council Deployment Guide
- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview

- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

How to Create Your Own Voting Vault with Council

Jump to bottom

CPSTL edited this page on Mar 7 · 2 revisions

Introduction

This tutorial is at an intermediate level and is aimed at developers who have some experience with writing smart contracts in Solidity. In this tutorial, we explore the requirements of writing a voting vault and give an example of how to create an NFT voting vault.

The boundaries on what is allowed as a voting vault are endless and are only limited by the interface that the Core Voting contract must be able to use to load data from the Vault. The Core Voting contract does not make any assumptions on the allocation of votes between voting vaults, the source of the votes, or the length of time the user has had them.

WARNING: if the total number of votes in the system exceeds size u128, overflows are possible. The interface that the vault must match looks like:

```
interface IVotingVault {
        /// @notice Attempts to load the voting power of a user
        /// @param user The address we want to load the voting power of
        /// @param blockNumber the block number we want the user's voting power at
        /// @param extraData Abi encoded optional extra data used by some vaults, such as merkle
proofs
        /// @return the number of votes
        function queryVotePower(
        address user,
        uint256 blockNumber,
        bytes calldata extraData
        ) external returns (uint256);
}
```

The voting vault must be able to retrieve the voting power for a user at the timestamp that the vote was created. Not following this temporal ordering can result in attacks on the governance system such as flashloans, or in the worst cases votes being falsely created. The vault then returns this power to the core voting contract when asked to do so. The voting vault can use the extra data provided to load or manipulate votes, or it can be used for custom instructions.

Example: https://github.com/element-fi/council/blob/main/contracts/vaults/OptimisticRewards.sol uses merkle proofs in the extra data to enable voting with unclaimed but distributed tokens.

For our first example of a voting vault, consider a vault that allows anyone who calls it to have any number of votes per request.

```
contract FriendlyVault is IVotingVault {

        // extraData encodes a uint128 of votes to give to the user via abi.encode
        function queryVotePower(
        address user,
         uint256 blockNumber,
         bytes calldata extraData
        ) external override returns (uint256) {
        uint128 votes = abi.decode(extraData, (uint128));
        return uint256(votes);
        }
}
```

The voting vault contracts that have been deployed for the Element DAO all use upgradable proxies to ensure that when users deposit governance tokens (or other voting power systems such as NFTs, other types of tokens, reputation or identity, etc) and a change is made to the voting algorithms, they do not need to move the governance tokens to retain their votes/voting power. This upgradability design is not mandatory. However, the tooling for historical lookups developed for the Council Protocol uses hash based data locations for storage, and as a result, may have incompatibilities with other systems.

WARNING: Council Protocol Proxies must use the hash based storage addressing for ALL variables as the proxy itself uses slots 0 and 1. Using any standard storage will corrupt the proxy system.

The History.sol library is an assembly implementation of a binary search over an array of timestamps and value pairs, with optional garbage collection. It can be used to load and store timestamped user data for your implementation.

In this tutorial, we will use it to build an implementation of an NFT vesting vault without any delegation. First, we will create a deposit function which transfers a user's NFT into the contract. Note: users can maintain control of the NFT if the NFT contract contains a historical ownership lookup system such as the Compound (COMP) token.

```
// SPDX-License-Identifier: Apache-2.0
pragma solidity ^0.8.3;

import "../libraries/History.sol";
import "../libraries/Storage.sol";
import "./IERC721.sol";

contract NFTVault {
        // Bring our libraries into scope
        using History for *;
```

```
        using Storage for *;

        // Only immutables can be declared without using the hash locations
        IERC721 immutable token;

        constructor(IERC721 _token) {
        token = _token;
        }

        /// @notice Returns the historical voting power tracker
        /// @return A struct which can push to and find items in block indexed storage
        function _votingPower()
        internal
        pure
        returns (History.HistoricalBalances memory)
        {
        // This call returns a storage mapping with a unique non overwrite-able storage location
        // which can be persisted through upgrades, even if they change storage layout
        return (History.load("votingPower"));
        }

        /// @notice Transfers one NFT of our collection to this contract and then adds one vote
to the user's voting power
        /// @param tokenId The token Id, not the NFT to transfer.
        function deposit(uint256 tokenId) external {
        // Get the token from the user
        token.transferFrom(msg.sender, address(this), tokenId);
        // Get the hash pointer to the history mapping
        History.HistoricalBalances memory votingPower = _votingPower();
        // Load the user votes
        uint256 currentVotes = votingPower.loadTop(msg.sender);
        // Push their new voting power
        votingPower.push(msg.sender, currentVotes + 1);
        }

        /// @notice Attempts to load the voting power of a user
        /// @param user The address we want to load the voting power of
        /// @param blockNumber the block number at which we want the user's voting power
        /// @return the number of votes
        function queryVotePower(
        address user,
        uint256 blockNumber,
        bytes calldata
        ) external override returns (uint256) {
        // Get our reference to historical data
        History.HistoricalBalances memory votingPower = _votingPower();
        // Find the historical data in our mapping
        return
                votingPower.find(
                user,
                blockNumber
                );
        }
}
```

This NFT voting vault contact is defined with an immutable NFT token and then for each token added the user who transferred it gets one vote. When the user adds a new token, their sum is incremented in the history mapping, and when the vault is queried the sum of the most recent timestamp before the query is returned. It does not support withdrawals or delegation but these actions flow from this same pattern.

Some simple but possibly instructive ways to tweak this NFT voting vault to get different results: Change the addition of 1 to any other constant to make NFTs worth more votes. Replace the addition of 1 with a lookup of the traits and then map traits by rarity and add to sum to get votes based on rarity. Replace the 'queryVotePower' function with one which calculates percent of deposited NFTs the user holds and multiplies it by a global constant for capped votes distributed proportionally to users.

By tweaking either the storage or the vote counting functionality you can arrive at any number of possibilities from only small changes.

- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

delvtech / **council-kit** Public

Code | Issues 14 | Pull requests 2 | Actions | Projects 1 | Wiki | Security | Ins

Labels | Milestones | New issue

is:issue is:open

14 Open ✓ 13 Closed

Author | Label | Assignee | Sort

Only run Vercel preview deployment for PRs after the build check has passed
enhancement
#383 opened 3 weeks ago by ryangoree

Add proposal creation UI
enhancement ui
#373 opened on May 22 by ryangoree

Council Kit v0.1.0
1
#349 opened on May 12 by ryangoree

Update UI README regarding Push settings
documentation ui
#347 opened on May 12 by ryangoree

Add ability to disable Push integration
1
enhancement ui
#346 opened on May 12 by ryangoree

Update wiki regarding Push settings
documentation ui
#345 opened on May 12 by ryangoree

Add missing contracts to the SDK
1
enhancement sdk
#343 opened on May 9 by ryangoree

Add guide for migrating from another gov system / token
1
documentation
#338 opened on May 3 by akhamisa

Publish packages to NPM
1
#337 opened on May 3 by akhamisa

Add % of VP that is delegated

enhancement ui

#336 opened on May 3 by akhamisa

Add voting power update UI for vesting vault

enhancement ui

#335 opened on May 3 by akhamisa

Add proposal execution UI

enhancement ui

#334 opened on May 3 by akhamisa

Add airdrop claim UI

1 14

enhancement ui

#333 opened on May 3 by akhamisa

Karma integration of data points into Council UI

3

enhancement ui

#332 opened on May 3 by akhamisa

ProTip! Mix and match filters to narrow down what you're looking for.

Locking Vault

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: LockingVault.sol
- Type/Category: Voting Vault/Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/vaults/LockingVault.sol

1. Introduction (Summary)

The Locking vault is a contract that allows the DAO to allocate votes to the token holders. It works by having the holders of the token deposit them into the Locking Vault. After votes have been allocated to users, they can then delegate votes to other addresses to vote on their behalf. To use their tokens in other contexts, the users must withdraw them from the locking vault.

2. Contract Details

Key Functionalities:

- Users can deposit into the locking vault and then either be allocated votes or have their votes be delegated to another address.
- Users can change which address their votes are delegated to.
- Users can withdraw their tokens to move them back into their wallets.
- The core voting contract can query the LockingVault to check the historical voting power of users in it.

Storage Layout:

- A system of state which tracks the historical power of addresses using the logic of the History.sol library.
- A mapping that tracks a user's deposited amount and who they have delegated to.
- Configuration parameters such as the token used and how much voting power history to retain.

3. Key Mechanisms & Concepts

The Council DAO framework can allocate votes to multiple sources but cannot employ the Compound governance contracts technique that tracks voting power inside the ERC20 itself. Therefore, for a user to have their voting power recognized, they must deposit into the voting contracts. This contract enables a user to deposit their tokens and receive voting power. When they deposit, they are allowed to select an address that may vote with their voting power. Each address has a historical tracker of voting power indexed by block number, and when they deposit or are delegated to the historical tracker records this. The core voting contract then calls the locking vault to ask how many votes an address has when a user votes on a proposal.

4. Gotchas (Potential source of user error)

- A user can fund another user's account and, on deposit, select the first delegation address. The 'firstDelegation' field in the call after the first call is ignored.
- If a malicious actor deposits to another user's address before that user deposits, then that malicious actor can select the delegation for the user. If the user deposits, they should confirm they have not been front run and, if they have, they must change delegation.
- Vote power queries will revert for users who have never been deposited or delegated to.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- The Locking Vault contract is intended to be used as the logic contract for an upgradable proxy, and it is recommended that it be only used in this context.
- Since the Locking Vault is upgradeable, its funds have the same risk profile as the governance system as a whole.
- In the case of a bug, the governance system should upgrade the locking proxy to protect user funds. However, because the locking vault is a primary source of funds, certain critical bugs may compromise its ability to do so.

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Optimistic Grants

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: OptimisticGrants.sol
- Type/Category: Grants/Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/features/OptimisticGrants.sol

1. Introduction (Summary)

This contract allows for grants to be made to contributors of the Element protocol with less overhead than other grant programs. When the grant is allocated, it contains an expiration date, and if not removed before then, it is assumed the grantee did what was expected, and they are allowed to withdraw.

2. Contract Details

Key Functionalities:

- Allow the owner to create and remove grants.
- Allows grantees to claim grants after they have completed their work.

Storage:

- A mapping tracking grants.
- An address authorized to create and remove grants.
- A tracker of the number of tokens the contract holds.

3. Key Mechanisms & Concepts

The optimistic grants program can be run directly by a DAO-wide governance vote, but it is more likely that it will handle grants allocated by a permissioned group like the GSC. Unlike other grants programs, the optimistic grants smart contract sets a schedule so that, after the grant period is up, the grantee can claim the grant. The default assumption of someone receiving an optimistic grant is that they will perform as the grant would indicate. Only in the case of non-performance, the grant is removed by the funding body.

4. Gotchas (Potential source of user error)

- The authorized grant address must fund the contract via the deposit method
- The grant contract tracks solvency, so it must be funded before grants are created.
- Funds directly transferred to this contract may not be recoverable.

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Optimistic Rewards

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: OptimisticRewards.sol
- Type/Category: Rewards/Governance
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/vaults/OptimisticRewards.sol

1. Introduction (Summary)

The Optimistic Rewards contract is a system for allocating rewards over time without writing new rewards smart contracts. It allows a proposer to propose a new set of rewards and then, after a review period, it is made official. During the review period, governance can reject the proposed rewards. The name optimistic comes from the fact that the mechanism is a play on an optimistic rollup, where the fraud proofs are governance votes but, instead of L2 transactions, the updates are new rewards balances.

2. Contract Details

Key Functionalities:

- The ability for addresses who have confirmed rewards to claim those rewards, or to claim them and then move them into the DAO to continue voting.
- The ability for addresses who have confirmed rewards but have not claimed them to vote using them as votes.
- A proposer can propose a new Merkle root encoding rewards.
- Governance can reject a new Merkle root encoding rewards.

Storage layout:

- A Merkle root containing confirmed rewards.
- A mapping of how many rewards each address has claimed.
- A proposed root and when it was proposed.
- A challenge period length.

3. Key Mechanisms & Concepts

Rewards systems voted in DAOs are often complex mathematical formulas on how much incentive to give to taking actions in the protocol. Each of these mechanisms must often be encoded into smart contracts. Given the security risk of smart contracts and implementation complexity, this naturally limits what can be done with rewards, how complex they are, and how fast they can change or be implemented. The optimistic rewards paradigm is a way to move rewards calculation off-chain and alleviate those limitations. To accomplish this, a proposer address is selected by governance. The proposer runs a deterministic program that calculates rewards, and then submits this on-chain for review. Community members also run the rewards program and, if fraud is detected, they can submit a DAO vote to remove the proposer and block the rewards. Given an active community, there is no incentive for a malicious proposer to submit rewards.

4. Gotchas (Potential source of user error)

- Rewards Merkle roots must contain unique accounts, they must be append-only, and the accounts must only increase in rewards allocated. The root tracks the total all-time rewards granted.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- An engaged community of verifiers of rewards is required for this system to work. The DAO should consider allocating rewards to chain watchers to encourage this.
- The total tokens in this contract should be limited so that, if a bug or hack occurs, the risk is mitigated.

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Possible Voting Vault Examples

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

Ideas for Voting Vaults

In terms of future voting vaults to experiment with, once the process for establishing logic for counting votes / providing voting power is complete, governance will vote to approve the method, and if successful, the integration is complete, and users can start voting!

Some of the possible future voting vaults consist of:

- **Compound/Aave Vault**— use tokens as collateral and/or earn interest while keeping voting power.
- **LP Vault**— get voting power while providing liquidity.
- **Identity Verified Vault**— verify your digital identity as a unique person to get voting power.
- **L2-L1 Synthesis Vault**-L2 posts balance tree hashes and L1 votes using the Merkel balance proof of L2.
- **NFT Vault**-allow all owners of a specific NFT to have voting power.

▸ Pages 25

Council Wiki

- Home
- Council Protocol 101

Technical Guides

- Getting Started
- Council Deployment Guide
- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

is:open

1 Open 0 Closed Sort

Council

#7 updated 3 weeks ago

The Council framework comprises both the Council Protocol smart contracts and Council Kit. Council Kit is the SDK for launching, building and maintaining DAOs and their governance systems. It includes a reference UI, deployment template, CLI, and Typescript SDK.

Protocol (Executable) Proposal Template

Jump to bottom

CPSTL edited this page on Mar 7 · 3 revisions

GP-X: [Title]

```
GP#: <# to be assigned>
Title: <GP title>
Author(s): <list of authors' names and/or Twitter/GitHub handles>
Type: Protocol Proposal
Status: <Assigned by Forum moderators >
Date Proposed: <yyyy-mm-dd>
Date Ratified: <yyyy-mm-dd>
```

References

- A list of supporting materials referenced by this GP.

Sentence Summary

- A description of what the Governance Proposal (GP) is focused on in 30 words maximum.

Paragraph Summary

- A description of what the Governance Proposal (GP) is focused on.

Motivation

- A short description of the motivation behind the proposed technical feature, solution, or upgrade.

Specification

Describe the details of the proposed technical solution. The specification should be detailed enough to allow an implementation to begin as well as testing. The specification for technical GPs must include the following components:

Proposed Code

- The final code that could be used directly in the on-chain vote to execute the GP.

Test Cases

- For the implementation or testing of the proposed code.

Security Considerations

- This section is to include or (proactively document) any security-relevant design information, potential failure modes, implementation details, and important discussions related to the proposed change.

Technical Review Plan or Audit Information (If already available)

- This section is to include the process for reviewing and auditing the code. If there are no audits available yet, describe the plan for it. A Protocol proposal should NOT move to a final vote without being reviewed/audited. Therefore, it is recommended to be as detailed as possible in this section.

Next Steps (Voting Outcome Summary)

- Clearly outline the next steps for this proposal. This includes a recommended review period for the technical solution, what voting “yes” and “no” entails, and what the outcome and next steps would be if the proposal were to pass the Off-Chain Poll and On-Chain Vote.
- **For example:**
  - Discuss this proposal on the forum and discord.
  - Implement a poll on this proposal on Snapshot.
    - If passed: This proposal will move to an audit review period.
  - A code review process will take X and should not be moved to an On-Chain vote before it has been deemed safe.
  - Implemented a vote on this proposal on the Governance Dashboard
    - If passed: This proposal will be executed and the following will be implemented to the governed Protocol.

- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview
- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Labels Milestones New

is:pr is:open

2 Open ✓ 368 Closed

Author Label Assignee Sort

feat: add airdrop ui ✕ 1 24

#388 opened 2 weeks ago by Chakravarthy7102

Added Opt-in/out button to start receiving notifications from Push ✕ 1 9

#342 opened on May 9 by Siddesh7

ProTip! Filter pull requests by the default branch with base:main.

Reference UI Deployment Guide (GitHub Pages)

Jump to bottom

MazyGio edited this page on Mar 9 · 5 revisions

If you've deployed the Council contracts on the blockchain, and are looking for a UI to interact with them, the Council Kit provides a Reference UI that you can deploy and plug into your contracts. You can also follow the video tutorial.

This guide will show you how to deploy the Reference UI on GitHub Pages, but you can use a similar process to deploy it onto your preferred platform instead

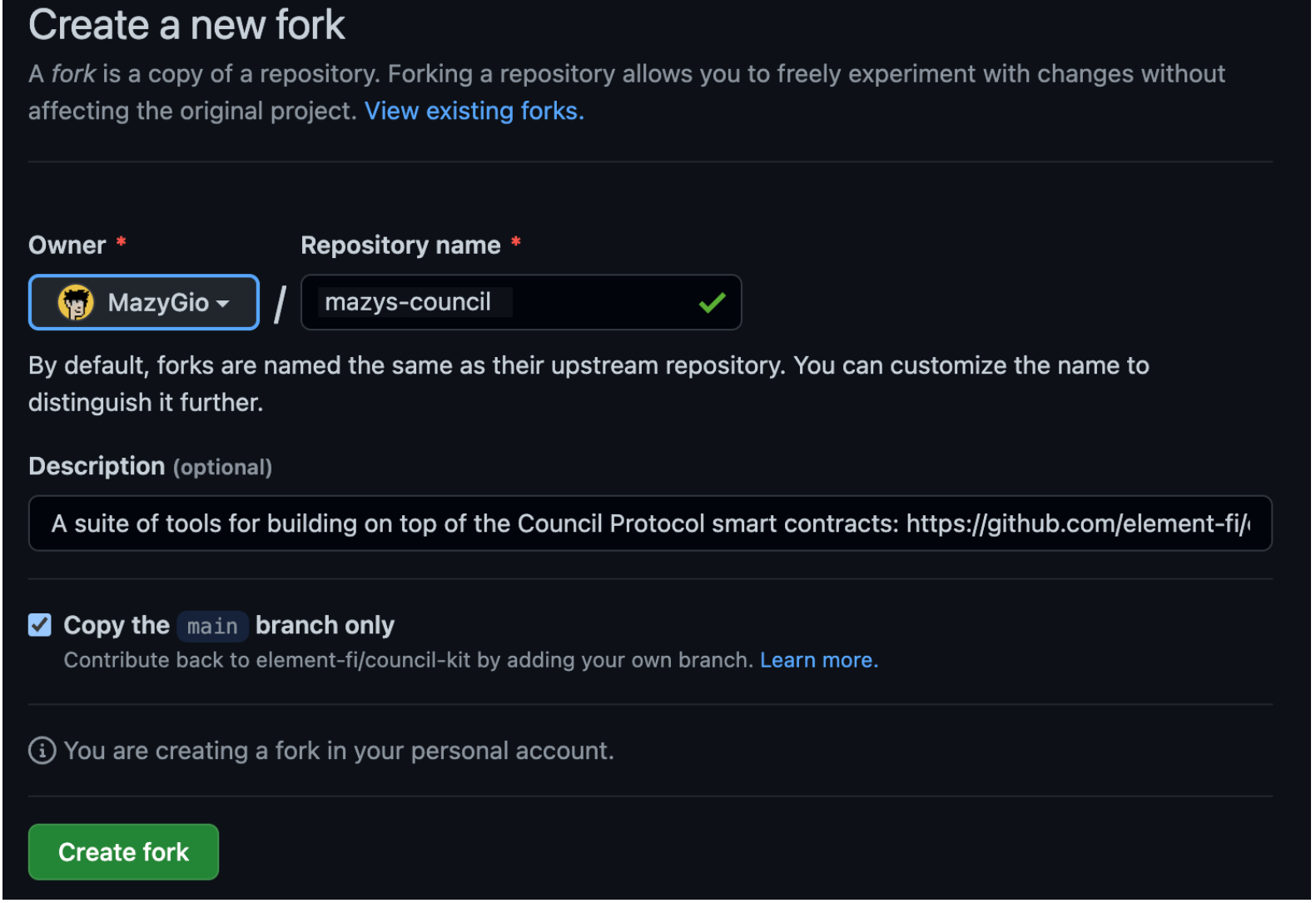
Requirements

- A previous deployment of the Council Contracts on the blockchain (follow the deployment guide if you need help)
- GitHub account
- Node provider (e.g. Alchemy)
- More?

Fork the repository

If you already made a fork of the repository to deploy the contracts, you can use that fork. Otherwise, visit the Council Kit repository page on GitHub and use the buttons on the top-right section to make your own fork.

You'll be prompted for a repository name. Keep in mind that this name will become the base path on the URL for this deployment.



Configure GitHub for UI deployment

The Council Kit codebase includes some GitHub Workflows to facilitate deployment in GitHub Pages. You can find these in the path `.github/workflows`, although you won't need to modify them for this guide.

One of these workflows is used to Deploy the web app to GitHub Pages every time there's a new push to the `main` branch. To use it, you'll need to follow a few steps on GitHub.

Set GitHub Pages to build from Actions

Go to your repository's Settings tab. Once there, click on the Pages section, and under Build and Deployment, set the Source to GitHub Actions (beta)

Add your environment variables

GitHub Pages will automatically set your page's path to be the name of your repository, so we need to inform the code what that name is. We'll do that by adding a Secret to the repository, which is just a variable. We'll need two:

`COUNCIL_UI_BASE_PATH`: The base path that GitHub Pages assigns to your deployment. This is our way of letting the app know about it. `GOERLI_ALCHEMY_KEY`: The code uses Alchemy as its default node provider. We'll go over how to use a different one later on.

Once there, set the new Secret's name to `COUNCIL_UI_BASE_PATH`, and its value to `/<your-repo-name>`. Don't forget the forward slash (`/`).

Use the same process to add your Alchemy API Key with the name GOERLI_ALCHEMY_KEY.

Deploy the council-ui app to GitHub Pages

Now that all our variables and configurations are set, we're ready to do the actual deployment. On the repository page, go to the **Actions** tab. It will ask for your consent to run Workflows. Once you accept, Look on the left side menu for the **Deploy Council UI to GH Pages** option.

This workflow is set to run after any changes are pushed to the `main` branch, but for this first time, we'll run it manually. Look on the right side for the **Run workflow** dropdown, and click the **Run workflow** button.

After a few seconds, the screen will update and show that the workflow has started running. It will take a few minutes. What's happening in the background is that the project is getting built with Next.js and uploaded as a static site onto GitHub Pages. You can check more details and even modify the workflow in the `.github/workflows/gh-pages-council.yml` file.

Once it's finished running, you can click into the workflow task, where you'll find a link to your newly-deployed page under the **deploy** subtask.

> **NOTE** The contracts that the deployed UI will plug into are gathered in the `packages/council-deploy/src/deployments/goerli.deployments.json` `apps/council-ui/src/`

Customization: How to use a different node provider

This guide used Alchemy as the node provider for on-chain data, but you can use any other node provider by making a few adjustments:

- Add the required environment variables on GitHub

If your chosen node provider requires an API key or other environment data, you must add it to GitHub Pages in the same way we added the base path and the Alchemy API key: go to Settings > Secrets and variables > Add Secret and input the Secret's name and value.

- Adjust the Deploy workflow so it reads the newly added variable

You must let the deploy workflow know about the new variables, which you can do by adding them to the file `.github/workflows/gh-pages-council.yml`.

Look for the following lines:

```
- name: Build with Next.js
  env:
    NEXT_PUBLIC_MAINNET_ALCHEMY_KEY: ${{ secrets.MAINNET_ALCHEMY_KEY }}
    NEXT_PUBLIC_GOERLI_ALCHEMY_KEY: ${{ secrets.GOERLI_ALCHEMY_KEY }}
    NEXT_PUBLIC_COUNCIL_UI_BASE_PATH: ${{ secrets.COUNCIL_UI_BASE_PATH }}
```

That's where and how you would add the new Secret you've created on GitHub Pages.

- Add the node provider to the configuration in `apps/council-ui/src/providers.ts`

You can follow the same template on the `apps/council-ui/src/providers.ts` file to replace Alchemy for your preferred node provider.

- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

SDK Guide: Using Typescript to interact with Council Contracts

Jump to bottom

Alim edited this page on Mar 14 · 3 revisions

The Council Kit includes an SDK to help you build on top of it. You can read on to learn more, or follow the video version of this guide.

First step: get the code

Use git to clone the council-kit repository:

```
git clone git@github.com:element-fi/council-kit.git
```

> **NOTE** This is a monorepo (turborepo). This means that there's a `package.json` configuration file at the top level (or root) directory, and another one on each app or package which lists their own configuration options and subdependencies. This allows apps and packages in your project to share dependencies without having to publish them in a registry (such as npm).

To begin, run yarn on the root directory to install all dependencies:

```
yarn
```

Once installation finishes, use yarn to build the packages too:

```
yarn build:packages
```

This command runs the `build` script on all the packages that have one under the `packages/` directory

SDK app config

package.json npm scripts

The SDK starter app includes three example scripts you can use as reference to help you get started. You'll find them in the `apps/council-sdk-starter/src/scripts` directory. You'll also find an entry for each of these scripts in the `package.json` file:

```
"scripts": {
  ...
  "createProposal": "ts-node -r dotenv/config -r tsconfig-paths/register
src/scripts/createProposal.ts",
  "getGSCMembers": "ts-node -r dotenv/config -r tsconfig-paths/register
src/scripts/getGSCMembers.ts",
  "getProposalResults": "ts-node -r dotenv/config -r tsconfig-paths/register
src/scripts/getProposalResults.ts"
},
```

The commands called for each entry are pretty similar:

- `ts-node` is the command used to run the typescript file
- `-r dotenv/config` tells it to use the config variables from your `.env` file
- `-r tsconfig-paths/register` tells it to also load the paths from the `tsconfig.json` file when running the script

It's an easy template to follow if you want to add any new scripts:

1. Create the typescript file under the `apps/council-sdk-starter/src/scripts` directory
2. Add the script entry in `package.json`
3. Run the script using `yarn workspace council-sdk-starter your-script-name-here`

Environment variables

The SDK will read environment variables from your `apps/council-sdk-starter/src/.env` file. If you don't have a `.env` file, use the `.env.example` file as a template to create your own.

Blockchain node provider

The SDK includes the file `apps/council-sdk-starter/src/provider.ts` to handle which node provider to use for making on-chain calls. It reads the `PROVIDER_URI` variable from your `.env` file.

If you don't set the `PROVIDER_URI` variable, the SDK defaults to using a Goerli provider. Any adjustments you'd like to make to the logic around node providers can be made in the `apps/council-sdk-starter/src/provider.ts` file.

Wallet address

If your scripts need to make on-chain transactions, you'll have to add a wallet private key to the `WALLET_PRIVATE_KEY` variable in the `.env` file to be able to execute them. Make sure the wallet is funded properly so that it is able to make transactions.

Write an example script: Get Voter Activity

Let's create an example script in Typescript to get the voting activity of one voter in the DAO, using the SDK. Here's the full code that we'll go through, for reference:

```
import { CouncilContext, VoteResults, VotingContract, Ballot } from "@council/sdk";
import { getElementAddress } from "src/addresses/elementAddresses";
import { provider } from "src/provider";

// wrap the script in an async function so we can await promises
export async function getVoterActivity(): Promise<void> {
  const addresses = await getElementAddress();

  // create a CouncilContext instance
  const context = new CouncilContext(provider);

  // Create a VotingContract instance.
  // The vaults array can be left empty since we won't be fetching any voting
  // power data.
  const coreVoting = new VotingContract(addresses.coreVoting, [], context);

  // get all votes made by an address
  const votes = await coreVoting.getVotes("0x7AE8b0D6353F0931EB9FaC0A3562fA9e4C6Ff933");

  // Create a data structure to print out to the console
  const votesByProposalId: Record<number, { ballot: Ballot, votingPower: string }> = {}
  for (const vote of votes) {
    votesByProposalId[vote.proposal.id] = {
      ballot: vote.ballot,
      votingPower: vote.power
    };
  }

  console.table(votesByProposalId);

  process.exit();
}

getVoterActivity();
```

Import Council types and functions

You can easily copy one of the existing example scripts to get a baseline for Council types and functions that you'll want to use for your development:

```
import { CouncilContext, VotingContract, Ballot } from "@council/sdk";
import { getElementAddress } from "src/addresses/elementAddresses";
import { provider } from "src/provider";
```

The first line imports several object types that map to the Council architecture:

- `CouncilContext`: Helper class that keeps track of data sources and cache values.
- `VotingContract`: Typescript class that maps to a specified CoreVoting contract.
- `Ballot`: Interface which stores the voter's choice on a specific vote (yes | no | maybe)

The second and third lines import helper functions or objects for Council data and for blockchain interaction:

- `getElementAddress`: Helper function that fetches all contract addresses from a specified Council deployment. You can see the logic in the `apps/council-sdk-starter/src/addresses/elementAddresses.ts` file.
- `provider`: Node provider object which facilitates blockchain interaction from the Typescript script.

You can see all the SDK's exports in its docs.

If you copied one of the existing scripts, you'll see the template we've followed wraps the script's logic in an `async` function:

```
export async function getVoterActivity(): Promise<void> {

...

}

getVoterActivity();
```

This is done so we can `await` certain values within the script.

Get Council objects to interact with

Within the actual logic of the script, the first few lines fetch some useful data:

```
  const addresses = await getElementAddress();

  // create a CouncilContext instance
  const context = new CouncilContext(provider);

  // Create a VotingContract instance.
  // The vaults array can be left empty since we won't be fetching
  // any voting power data.
  const coreVoting = new VotingContract(addresses.coreVoting, [], context);
```

We've explained how to obtain the addresses and what the context is for. The CouncilContext takes a `provider` object, which you would have previously set up in your `.env` file and `provider.ts` file.

We're also instantiating a `VotingContract` and passing it the coreVoting contract's address. You can optionally tell it the addresses of which Voting Vault contracts you want to consider for this instance. The `addresses` object would have all of those addresses handy for easy access.

Get vote data from the VotingContract

At this point you have a `coreVoting` object which is a `VotingContract`. We can call its `getVotes()` method to fetch all votes that have occured in it, and we can filter it out by a specific voter address:

```
  // get all votes made by a specified address
  const votes = await coreVoting.getVotes("0x7AE8b0D6353F0931EB9FaC0A3562fA9e4C6Ff933");
```

This line actually fetches data from the blockchain by using the `provider` that you've set up earlier through your `PROVIDER_URI` environment variable.

We'll now make a data structure for this data that is easier to output to the console in a readable format:

```
  // Create a data structure to print out to the console
  const votesByProposalId: Record<number, { ballot: Ballot, votingPower: string }> = {}
  for (const vote of votes) {
    votesByProposalId[vote.proposal.id] = {
      ballot: vote.ballot,
      votingPower: vote.power
    };
  }

  console.table(votesByProposalId);
  process.exit();
}

getVoterActivity();
```

This part of the code just iterates through the `VoteData` that was returned by the `coreVoting.getVotes()` method, and creates a `Record` which we can then print out as a table on the console.

Each entry in the `Record` holds an index, which stores the proposal ID, a `Ballot` object which stores the voter's choice, and the `votingPower` used on this vote.

Finally, we exit the process to close the program, and call the `getVoterActivity` async function that we've just created.

Register the script and execute

To execute this from the command line, we can add a script to the `apps/council-sdk-starter/src/package.json` file, just like the other example scripts:

```
...
  "scripts": {
    ...
    "getVoterActivity": "ts-node -r dotenv/config -r tsconfig-paths/register
src/scripts/getVoterActivity.ts"
  },
...
```

Once you've registered the `getVoterActivity` script, we can call it from the command line and see the results. Just run the following command:

```
yarn workspace council-sdk-starter getVoterActivity
```

```
council-kit % yarn workspace council-sdk-starter getVoterActivity
yarn workspace v1.22.18
yarn run v1.22.18
$ ts-node -r dotenv/config -r tsconfig-paths/register src/scripts/getVoterActivity.ts
```

(index)	ballot	votingPower
0	'yes'	'123191.101513823336749'
1	'yes'	'123600.367291019047649'
2	'yes'	'125473.452857948808081'
3	'yes'	'125473.452857948808081'

```
✨  Done in 6.31s.
✨  Done in 6.61s.
```

With this, you've successfully instantiated a Council VotingContract and queried the votes that a specific voter has made on it, and interacted with a few parts of the SDK along the way. We're excited to see what you'll build from here on out! Don't hesitate to contact us through our Discord!

▸ **Pages** 25

Council Wiki

- Home
- Council Protocol 101

Technical Guides

- Getting Started
- Council Deployment Guide
- Reference UI Deployment Guide (GitHub Pages)
- How to Create Your Own Voting Vault with Council
- SDK Guide: Using Typescript to interact with Council Contracts
- Contributors Guide

Video Tutorials

- Council UI Overview
- Deploying the Council UI to Github Pages
- Get Started with the Council SDK
- Extending the SDK

Governance Proposal Framework

- Governance Proposal Framework Template
- DAO (Social) Proposal Template
- Protocol (Executable) Proposal Template

Smart Contract Documentation

- Overview

- Core Voting
- Voting Vaults Overview
- Voting Vault Example: Locking Vault
- Voting Vault Example: Vesting Vault
- Voting Vault Example: Governance Steering Council (GSC)
- Possible Voting Vaults Examples
- Timelock
- Treasury
- Spender
- Optimistic Grants
- Optimistic Rewards
- Simple Proxy

Glossaries

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```

Security

No security policy detected

This project has not set up a `SECURITY.md` file yet.

There aren’t any published security advisories

Simple Proxy

Jump to bottom

CPSTL edited this page on Jan 10 · 1 revision

- Contract Name: SimpleProxy.sol
- Type/Category: Proxy
- Contract Source: https://github.com/element-fi/council/blob/main/contracts/simpleProxy.sol

1. Introduction (Summary)

The simple proxy is used to make a contract upgradeable by splitting the state from the logic run by the smart contract. It is implemented in gas-efficient assembly and calls a changeable logic contract that runs the code in that contract using the storage of the proxy.

2. Contract Details

The key functionalities of the smart contract are the ability to accept calls from users and run the code of another contract as if they had called it, and the ability to change which contract's logic is used. The simple proxy stores the address of the implementation and the address allowed to upgrade that implementation.

3. Key Mechanisms & Concepts

The simple proxy's fallback function will take the data provided by the caller and forward that data as a delegate call to the implementation address, then propagate any return or error back to the caller. The fallback uses optimized assembly to achieve this efficiently. Since the proxy uses delegate call, the address of the proxy will have any state changes made in it, instead of in the address of the implementation. Therefore, if the implementation is changed, the new implementation will have access to the same storage. The only other function in the proxy allows a governance address to change the implementation address, which is called when the callback function executes.

4. Gotchas (Potential source of user error)

- The governance and implementation address is normally stored in the contract so any logic contract must use the storage library to manage storage or will almost certainly corrupt storage.
- Function names 'upgradeProxy(address)' 'resetProxyOwner(address)' and their corresponding selectors cannot be used in logic contracts and are reserved for the proxy.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- The governance address can make any change to the storage or usage of the proxy; therefore, upgradable contracts are only as secure as their governance mechanism. If governance is corrupted the whole contract is also corrupted.

- Council Glossary

Audits

- Audit Reports

Clone this wiki locally

```
https://github.com/delvtech/council-kit.wiki.git
```