Vocdoni Knowledge Base

Introduction

> Vocdoni is a project born with the aim of making digital participation easy and secure by building an anonymous, universally verifiable and scalable voting system.
>
> This infrastructure is accessible via Vocdoni Platform, Vocdoni Bridge and Vocdoni API.

Welcome to Vocdoni Knowledge base

Getting started

If you want to know how to start using **Vocdoni App** start here:

→ Vocdoni App /vocdoni-platform-1/vocdoni-app

If you need help in the use of **Vocdoni Platform (App and manager)** check our FAQs:

→ Vocdoni FAQ /faq

If you want to start using Vocdoni API check the following page:

→ Vocdoni API /other-components/vocdoni-api

To know more about the Vocdoni Open Stack check the docs:

Vocdoni Open Stack

What is Vocdoni?

Vocdoni is a universally verifiable, censorship-resistant and anonymous governance system, designed to be scalable and easy to use on modest devices. The goal is to provide the foundation to run national elections, general shareholders meetings, assemblies, etc.

Our main aim is a trustless voting system, where people can speak their voice and everything can be audited. We are engineering the building blocks for a permissionless, private and censorship-resistant democracy.

The algorithms, systems and software that we build are intended to be a contribution toward making *violence in these cryptonetworks impossible by protecting users privacy with cryptography*. In particular, our aim is to provide the tools for the political will of network participants to translate outwardly into real political capital, without sacrificing privacy.

Vocdoni chronology

2021

January - Vocdoni Platform available in Open Access.

2020

July - First relevant voting process hold with a 280.000 user base with a client (Òmnium Cultural).

September - Early Access program starts with +20 organizations using Vocdoni Platform on regular basis.

October - Vocdoni Bridge, a new project for token-based governance on top of Vocdoni Open Stack starts.

December - Vocdoni Bridge MVP released.

2019

July - Seed funding.

December - First version of Vocdoni Open Stack Infrastructure working.

February - Vocdoni Platform MVP released.

2018

April - First donations are received. Project starts.

June - First testing with ZCash.

October - First design of Vocdoni Open Stack.

November - Team consolidation. Development starts.

2017

October - Censorship during the 2017 Catalan referendum catalyzes the birth of the project

Vocdoni FAQ

What is Vocdoni?

Vocdoni is a governance platform that allows you to participate in organizations online with the maximum guarantees of security and privacy. Vocdoni’s digital voting system is designed to be universally verifiable, secure, and resistant to attack and censorship through the use of blockchain technology, other decentralized technologies, and advances in cryptography, such as zero-knowledge proofs.

Is Vocdoni anonymous?

Currently, Vocdoni has an in-built anonymity system through the use of information containers and identity obfuscation. A careful design of the frontend prevents third parties from knowing the meaning of the vote of each member and the possibility of crossing data.

In Q12020, we will release a new version of our voting protocol that makes use of zk-SNARKs technology to get full anonymity into the infrastructure layer.

Is Vocdoni secure?

Vocdoni is a secure voting system that takes into account not only the vote privacy, but also users personal data. Therefore, the system we have designed prevents any sensitive data from being exposed to the Internet. In addition, thanks to the use of decentralized technologies, the system is resilient to all sorts of attacks that normally affect centralized voting systems.

I'm an organization, what can I do with Vocdoni?

As an organization, you can:

- Import your verified user base (email)
- Create verifiable and immutable communications.
- Register new users through QR codes and manage their participation rights.
- Create multiple censuses by segmenting by attributes. For example, an NGO may decide to hold a vote in which only women over the age of 18 living in Barcelona can participate.
- Create participation processes of different types: public consultations, single-choice / multiple-choice voting.

What kind of technologies does Vocdoni use?

Vocdoni platform integrates different distributed technologies:

- **Ethereum**: public and immutable blockchain where the structure and components of voting processes are indexed through smart-contracts, including the public key tree.
- **Vochain**: Allowed blockchain based on Tendermint that records votes transparently and immutably.
- **IPFS**: distributed file system (https://ipfs.io/)
- **LibP2P**: distributed messaging protocol.
- **zk-SNARKs**: to perform zero-knowledge tests that allow anonymous voting and the ability to change the direction of the vote before a vote ends, thus avoiding vote-buying and coercion.

How is the transparency and integrity of the voting ensured?

Unlike other electronic voting solutions, any person or entity can verify the integrity of the process and its outcome. We built a tool that specifically serves this purpose: Vochain Explorer.

All our code is open and can be audited by the entire community.

Why does Vocdoni help users have more control over their data?

We use a self-sovereign identity scheme. The keys that identify the user are created and stored in the user's terminal, along with the rest of their details. An exchange protocol allows you to share the necessary credentials with the organization's servers so that they can verify users.

A built-in consent mechanism generates a signature to indicate how the data should be used. This allows complete control of the current and historical use of the data by the user, while allowing organizations to manage their data with full GDPR compliance.

Vocdoni Platform

Vocdoni App

Getting started

What is Vocdoni Platform?

...

Install Vocdoni app

Vocdoni app is available to download from **Google Play** or **Apple App Store**.

→ Creating my identity
/vocdoni-platform-1/vocdoni-app/getting-started/creating-my-identity

→ First steps with the App
/vocdoni-platform-1/vocdoni-app/getting-started/first-steps-with-the-app

Creating my identity

Can I become who I want to be?

That's a tough question but thankfully, our team is on it. Please bear with us while we're investigating.

Have you had a chance to answer the previous question?

Yes, after a few months we finally found the answer. Sadly, Mike is on vacations right now so I'm afraid we are not able to provide the answer at this point.

1. Create an unlock pattern

Backup your identity

...

First steps with the App

- [ ] Interface
- [ ] Follow an organization
- [ ]

General Troubleshooting

Can I become who I want to be?

That's a tough question but thankfully, our team is on it. Please bear with us while we're investigating.

Have you had a chance to answer the previous question?

Yes, after a few months we finally found the answer. Sadly, Mike is on vacations right now so I'm afraid we are not able to provide the answer at this point.

My unlock pattern doesn't work

I lost my identity

Vocdoni Manager

Other components

Vochain explorer

Vochain explorer is a tool to draw blockchain data and make it human-readable and searchable.

> Vochain is a *Proof-of-Authorithy* blockchain network that uses Tendermint as its consensus algorithm. The main purpose of Vochain is to register polls, votes, and backing processes in a decentralized and verifiable way.
> More info here: https://docs.vocdoni.io/#/architecture/services/vochain

For most users, a blockchain explorer will allow you to search and explore data about recently mined blocks or recently carried out transactions on a blockchain. Ideally, they allow you to view a live feed of blocks as they are mined, as well as the data related to the blocks.

You can use our Vochain explorer available here explorer.vocdoni.net or deploy your own instance in your server following the guide.

→ Guide - Deploy Vochain Explorer /other-components/vochain-explorer/guide-deploy-vochain-explorer

Guide - Deploy Vochain Explorer

Dependences

Golang 1.14+

Running

Docker

Navigate to `vocexplorer/docker/vocexplorer`

```
docker-compose build
docker-compose up
```

That's it!

Manual

1. Frontend

On the command line navigate into the directory `./frontend` and run

```
env GOARCH=wasm GOOS=js go build -ldflags "-s -w" -trimpath -o ../static/mair
```

to compile the frontend code into wasm.

2. Static files

Make a copy of the `wasm_exec.js` file from `$GOROOT/misc/wasm/` directory and put it in the `./static` directory. This must be from the same golang version that you used to build `main.wasm`.

Get back to the root path and run `yarn` to install and compile the required style assets.

If you want to renew the styles, run `yarn gulp`, or in case you want to watch for changes, `yarn gulp sass:watch`.

There's also a gulp task for watching `.go` files changes in `./frontend` files and regenerating the `main.wasm` file: `yarn gulp go:watch`.

You can watch both `.go` and `.scss` file changes by just using

```
yarn gulp watch
```

4. Backend

After steps 1, and 2, navigate back into `vocexplorer` and run

```
go run main.go
```

Then in your favourite web browser navigate to localhost at the specified port.

Options for `main.go`:

- `--chain` `(string)` vochain network to connect to (eg. main, dev (default "main")
- `--dataDir` `(string)` directory where data is stored (default "/Users/natewilliams/.vocexplorer")
- `--disableGzip` use to disable gzip compression on web server
- `--hostURL` `(string)` url to host block explorer (default "http://localhost:8081")
- `--logLevel` `(string)` log level <debug, info, warn, error> (default "error")
- `--refreshTime` `(int)` Number of seconds between each content refresh (default 10)
- `--vochainCreateGenesis` create own/testing genesis file on vochain
- `--vochainGenesis` `(string)` use alternative genesis file for the voting chain
- `--vochainLogLevel` `(string)` voting chain node log level (default "error")

- `--vochainNodeKey` `(string)` user alternative vochain private key (hexstring[64])
- `--vochainP2PListen` `(string)` p2p host and port to listent for the voting chain (default "0.0.0.0:26656")
- `--vochainPeers` `(stringArray)` comma separated list of p2p peers
- `--vochainSeeds` `(stringArray)` comma separated list of p2p seed nodes

Note: when updating to a new version of this program, you may need to refresh your data store:

```
rm -rf DBPATH
```

Where DBPATH by default is `~/.vocexplorer/CHAIN_ID` and CHAIN_ID is the ID of the blockchain you're exploring, eg. `vocdoni-release-06`

Vocdoni API

Start using Vocdoni API

Entity creation

Make sure you have Node.js 12 installed on your computer.

In an empty folder, create a `package.json` file by running

```
$ npm init -y
Wrote to /home/user/dev/governance-as-a-service/package.json:

{
  "name": "governance-as-a-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\\"Error: no test specified\\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Next, install the DVoteJS dependency and two dev dependencies to support TypeScript:

```
$ npm i dvote-js
$ npm i -D typescript ts-node
```

Edit the `scripts` section of `package.json` and leave it like this:

```
  "scripts": {
    "main": "ts-node index.ts"
  },
```

Then, let's create a file named `index.ts` and create a function to generate the wallet of our new entity:

```
import { Wallet } from "ethers"

function makeEntityWallet() {
    console.log("Creating entity wallet")
    const entityWallet = Wallet.createRandom()

    console.log("Entity address:", entityWallet.address)
    console.log("Entity private key:", entityWallet.privateKey)
}

makeEntityWallet()
```

To check that it works, let's run `npm run main`:

```
$ npm run main
Creating entity wallet
Entity address: 0x1048d8cB20fE806389802ba48E2647dc85aB779a
Entity private key: 0x6d88d78a4a76e42144b6867fdff89477f0a3b805d85b97cd46387
```

So here's our wallet, what's next?

In this example, we will be using an Ethereum testnet called Sokol. Write down your private key and and use the address to request some test coins from the faucet. You will need them to send some transactions.

Sokol faucet

Now, instead of creating a random wallet, we should use the one that received the test ether. In the lines above, replace this:

```
const entityWallet = Wallet.createRandom()
```

Into this:

```
// Use your private key here
const entityWallet = new Wallet("0x6d88d78a4a76e42144b6867fdff89477f0a3b805
```

Obviously, make sure to store any real private key nowhere within the source code or Git in general.

Now we need to get a pool of gateways and connect to one of them:

```
import { GatewayPool } from "dvote-js/dist/net/gateway-pool"

let pool: GatewayPool

async function connect() {
    // Get a pool of gateways to connect to the network
    pool = await GatewayPool.discover({ networkId: NETWORK_ID, bootnodesCon
    await pool.connect()
}

const disconnect = () => pool.disconnect()
```

Then, let's define the metadata of our Entity and make it available on IPFS and Ethereum. Add the following code to `index.ts` :

```
import { EntityMetadata } from "dvote-js"
import { updateEntity } from "dvote-js/dist/api/entity"

async function registerEntity() {
    // Make a copy of the metadata template and customize it
    const entityMetadata: EntityMetadata = Object.assign({}, Models.Entity.

    entityMetadata.name.default = "Vilafourier"
    entityMetadata.description.default = "Official communication and partic
    entityMetadata.media = {
        avatar: 'https://my-organization.org/logo.png',
        header: 'https://my-organization.org/header.jpeg'
    }
    entityMetadata.actions = []

    const contentUri = await updateEntity(entityWallet.address, entityMetad

    // Show stored values
    console.log("The entity has been defined")
    console.log(contentUri)
}
```

And then:

```
$ npm run main
Setting the entity metadata
WARNING: Multiple definitions for addr
WARNING: Multiple definitions for setAddr
The entity has been defined
ipfs://QmdK5TnHDXPt4xozkuboyKP94RTrUxFr1z9Pkv5qhfahFG
```

Done!

This is what we just did:

- We created JSON metadata containing the custom details of our entity
- We pinned the JSON content on IPFS using a gateway from the pool
- The hash of our metadata is `QmdK5TnHDXPt4xozkuboyKP94RTrUxFr1z9Pkv5qhfahFG` and should be available from any IPFS peer
- An Ethereum transaction was sent to the entities contract, defining the pointer to our new metadata.

The value on the smart contract can only be updated by our wallet, the blockchain ensures the integrity of our data and IPFS ensures its global availability.

Visualizer

To check that our entity is properly declared, we can check it on the visualizer: `https://app.dev.vocdoni.net/entities/#/<entity-id>`

This is the link in our case.

> Note: Keep in mind that we're using a testnet and some of the data might be eventually disposed.

Census generation

Our entity is ready, now it can handle voting processes. But before we can create one, we need a census of the people who can vote.

As we said, digital identities arise from a cryptographic keypair. The ideal scenario is when the identity is generated on the user's device and only the public key is shared, but for the sake of simplicity, we will create a few identities ourselves.

```
let votersPublicKeys: string[] = []

function populateCensusPublicKeys() {
    for (let i = 0; i < 10; i++) {
        const wallet = Wallet.createRandom()
        votersPublicKeys.push(wallet["signingKey"].publicKey)
    }
    console.log("Voter's public keys", votersPublicKeys)
}
```

Which results in

```
Voter's public keys [
 '0x046f5ae433845ddc5d93a44b4aa3b9f654861372ada5f074b88bd18623dcbac5e4c3495
 '0x0460aaecd59d68dc63aa173f77d89476280d77b67c5793c5f166ff951eb4d761df8d57e
  ...
]
```

Then we can upload the census claims to a gateway and publish it:

```
import { addCensus, addClaimBulk, getRoot, publishCensus } from "dvote-js/d

let merkleTreeOrigin: string
let merkleRoot: string

async function publishVoteCensus() {
    // Prepare the census parameters
    const censusName = "Vilafourier all members #" + Math.random().toString
    const adminPublicKeys = [await entityWallet["signingKey"].publicKey]
    const publicKeyClaims = votersPublicKeys.map(k => digestHexClaim(k)) //

    // As the census does not exist yet, we create it (optional when it exi
    let { censusId } = await addCensus(censusName, adminPublicKeys, pool, e
    console.log(`Census added: "${censusName}" with ID ${censusId}`)

    // Add claims to the new census
    let result = await addClaimBulk(censusId, publicKeyClaims, true, pool,
    console.log("Added", votersPublicKeys.length, "claims to", censusId)
    if (result.invalidClaims.length > 0) console.error("Invalid claims", re

    merkleRoot = await getRoot(censusId, pool)
    console.log("Census Merkle Root", merkleRoot)

    // Make it available publicly
    merkleTreeOrigin = await publishCensus(censusId, pool, entityWallet)
    console.log("Census published on", merkleTreeOrigin)
}
```

Which results in:

```
Census added: "Vilafourier all members #550262" with ID 0x1048d8cB20fE80638
Added 10 claims to 0x1048d8cB20fE806389802ba48E2647dc85aB779a/0x8e0c00cda17
Census Merkle Root 0xbf8572159929b4e37c40e2ce18fd46156e750a4aac1f06dbda3237
Census published on ipfs://QmcTuUJbN1tEWA3nqHFU8N8iuUN2ymJoqW4UcKBzHuYrPw
```

Cool! Our entity is ready, and our first census is now public.

Process creation

Now we are ready to create our fist process. To do so, we will use the values we generated previously and define the topic, the questions, the vote options, etc.

```
import { getEntityId } from "dvote-js/dist/api/entity"
import { estimateBlockAtDateTime, createVotingProcess, getVoteMetadata } fr

async function createVotingProcess() {
    const myEntityAddress = await entityWallet.getAddress()
    const myEntityId = getEntityId(myEntityAddress)

    const startBlock = await estimateBlockAtDateTime(new Date(Date.now() +
    const numberOfBlocks = 6 * 60 * 24 // 1 day (10s block time)

    const processMetadata: ProcessMetadata = {
        "version": "1.0",
        "type": "poll-vote",
        "startBlock": startBlock,
        "numberOfBlocks": numberOfBlocks,
        "census": {
            "merkleRoot": merkleRoot,
            "merkleTree": merkleTreeOrigin
        },
        "details": {
            "entityId": myEntityId,
            "title": { "default": "Vilafourier public poll" },
            "description": {
                "default": "This is our test poll using a decentralized blo
            },
            "headerImage": "https://images.unsplash.com/photo-1600190184658
            "streamUrl": "",
            "questions": [
                {
                    "type": "single-choice",
                    "question": { "default": "CEO" },
                    "description": { "default": "Chief Executive Officer" }
                    "voteOptions": [
                        { "title": { "default": "Yellow candidate" }, "valu
                        { "title": { "default": "Pink candidate" }, "value"
                        { "title": { "default": "Abstention" }, "value": 2
                        { "title": { "default": "White vote" }, "value": 3
                    ]
                },
                {
                    "type": "single-choice",
                    "question": { "default": "CFO" },
                    "description": { "default": "Chief Financial Officer" }
                    "voteOptions": [
                        { "title": { "default": "Yellow candidate" }, "valu
                        { "title": { "default": "Pink candidate" }, "value"
                        { "title": { "default": "Abstention" }, "value": 2
                        { "title": { "default": "White vote" }, "value": 3
                    ]
                },
            ]
        }
    }
    const processId = await createVotingProcess(processMetadata, entityWall
    console.log("Process created:", processId)

    // Reading the process metadata back
    const metadata = await getVoteMetadata(processId, pool)
    console.log("The metadata is", metadata)
}
```

Which looks like this:

```
Process created: 0x82f42dc48bfbfa0bb1018bcb0f3a31f77d12ced1fda9566990d64d07
The metadata is {
  version: '1.0',
  type: 'poll-vote',
  startBlock: 2203,
  numberOfBlocks: 8640,
  census: {
    merkleRoot: '0x2044a23e328d75276a7e9e6bc1774eb67707597beba1cfdfde5e7fa1
    merkleTree: 'ipfs://QmTFGgoDnMWhMU3BXxm4zTxeFZLPnRV1jiQgBqukzJPmo9'
  },
  details: {
    entityId: '0xdeea56f124dd3e73bcbc210fc382154a11f3bab227af55927139c887e1
    title: { default: 'Vilafourier public poll' },
    description: {
      default: 'This is our test poll using a decentralized blockchain to r
    },
    headerImage: 'https://images.unsplash.com/photo-1600190184658-4c4b088ec
    streamUrl: '',
    questions: [ [Object], [Object] ]
  }
}
```

So, we just...

- Defined the metadata of our process
- Defined the block at which we would like the process to start
- Declare the metadata of the process

As it happened before, the JSON metadata is pinned on IPFS and pointed to from the process smart contract. In addition, some metadata fields are also stored on the smart contract so they can be accessed on-chain.

> The contract flags define **how** the process will behave, whereas the metadata is used to store the **human readable** content.

In about 2-3 minutes, the Ethereum transaction will be relayed to the voting blockchain as well. When the block number reaches `startBlock`, the process will become open to those who are part of the census. The `startBlock` value should be **at least 25 blocks ahead** of the current value.

Visualizer

To check the new process, there are two web sites we can use:

- `https://app.dev.vocdoni.net/processes/#/<entity-id>/<process-id>`
- `https://explorer.dev.vocdoni.net/process/<process-id>`

In our case:

- `https://app.dev.vocdoni.net/processes/#/0xdeea56f124dd3e73bcbc210fc382154a11f3bab227af55927139c887e15573e4/0x82f42dc48bfbfa0bb1018bcb0f3a31f77d12ced1fda9566990d64d07be9a48a6`
- `https://explorer.dev.vocdoni.net/process/0x82f42dc48bfbfa0bb1018bcb0f3a31f77d12ced1fda9566990d64d07be9a48a6`

Again, this is a testnet and some of this data may have been cleaned up at some point.

Voting

The previous code typically runs on the backend of your organization -you will simply choose to generate a census based on your specific organization's parameters.

This next portion, however, is meant to be executed on the voter's device. The specific method used to create and manage user keys is an important decision which presents a trade off between usability and security.

- Users create the keys on their end
    - This allows for a self sovereign identity and provides higher privacy
    - However, users have to register the public key on your backend and you need to validate it
- The organization creates one-time keys
    - The keys are not in the user's control
    - They don't need to register
    - They can simply receive a link and use it to vote, and then the key expires

Whatever your case is, we will illustrate a web browser environment where the voter fetches the process metadata, signs his/her choices and submits them to a gateway.

```
import { Wallet } from "ethers"
import {
    getVoteMetadata,
    estimateDateAtBlock,
    getBlockHeight,
    getEnvelopeHeight,
    packagePollEnvelope,
    submitEnvelope
} from "dvote-js/dist/api/vote"
import { getCensusSize, digestHexClaim, generateProof } from "dvote-js/dist
import { waitUntilVochainBlock } from "dvote-js/dist/util/waiters"

async function submitSingleVote() {
    // Get the user private key from the appropriate place
    const wallet = new Wallet("voter-privkey")  // TODO: Adapt to your case

    // Fetch the metadata
    const processMeta = await getVoteMetadata(processId, pool)

    console.log("- Starting:", await estimateDateAtBlock(processMeta.startB
    console.log("- Ending:", await estimateDateAtBlock(processMeta.startBlo
    console.log("- Census size:", await getCensusSize(processMeta.census.me
    console.log("- Current block:", await getBlockHeight(pool))
    console.log("- Current votes:", await getEnvelopeHeight(processId, pool

    await waitUntilVochainBlock(processMeta.startBlock, pool, { verbose: tr

    console.log("Submitting vote envelopes")

    // Hash the voter's public key
    const publicKeyHash = digestHexClaim(wallet["signingKey"].publicKey)

    // Generate the census proof
    const merkleProof = await generateProof(processMeta.census.merkleRoot,

    // Sign the vote envelope with our choices
    const choices = [1, 2]
    const voteEnvelope = await packagePollEnvelope({ votes: choices, merkle

    // If the process had encrypted votes:
    // const voteEnvelope = await packagePollEnvelope({ votes, merkleProof,

    await submitEnvelope(voteEnvelope, pool)
    console.log("Envelope submitted")
}
```

That's quite a lot so far!

To check for the status of a vote, you can append these extra calls at the end:

```
import { getPollNullifier, getEnvelopeStatus } from "dvote-js/dist/api/vote

{
    // ...

    // wait 10+ seconds
    await new Promise(resolve => setTimeout(resolve, 1000 * 10))

    // Compute our deterministic nullifier to check the status of our vote
    const nullifier = await getPollNullifier(wallet.address, processId)
    const status = await getEnvelopeStatus(processId, nullifier, pool)

    console.log("- Registered: ", status.registered)
    console.log("- Block: ", status.block)
    console.log("- Date: ", status.date)
}
```

The output:

```
- Starting: 2020-09-17T15:50:44.000Z
- Ending: 2020-09-18T15:50:44.000Z
- Census size: 10
- Current block: 5
- Current votes: 0
Waiting for Vochain block 35
Now at Vochain block 6
Now at Vochain block 7
[...]
Now at Vochain block 35`

Submitting vote envelopes
Envelope submitted

- Registered:  true
- Block:  36
- Date:  2020-09-17T15:51:06.000Z
```

This is what we just did:

- Initialize the wallet with the voter's private key
- Fetch the vote metadata
- Wait until `startBlock`
- Request a census merkle proof to a random Gateway, proving that the voter's public key matches the census Merkle root
- Package our choices into an envelope signed with the voter private key (and optionally encrypt it)
- Submit the envelope using a random Gateway
- Retrieve the status of the vote

Process results

If we look at the visualizer or the block explorer we should see votes already submitted.

If the process type was set to `encrypted-poll` then, results would remain unavailable until the process ends. As our process is an open poll, however, we can ask for the results in real time:

```
import { getResultsDigest } from "dvote-js/dist/api/vote"

async function fetchResults() {
    const { questions } = await getResultsDigest(processId, pool)

    console.log("Process results", questions)
}
```

After all users have voted, the visualizer shows:

Process results

However, if you can't wait for an encrypted process to end later or you simply want to stop vote submissions, there's a trick you can use. This will change in the near future, but for now you can `cancel` the rest of the lifespan of a process, if needed. The scrutiny will begin a few seconds later.

```
import { isCanceled, cancelProcess } from "dvote-js/dist/api/vote"

async function forceEndingProcess() {
    // Already canceled?
    const canceled = await isCanceled(processId, pool)
    if (canceled) return console.log("Process already canceled")

    // Already ended?
    const processMeta = await getVoteMetadata(processId, pool)
    const currentBlock = await getBlockHeight(pool)
    if (currentBlock >= (processMeta.startBlock + processMeta.numberOfBlock

    console.log("Canceling process", processId)
    await cancelProcess(processId, entityWallet, pool)
    console.log("Done")
}
```

Congratulations! You just conducted an election on your own

A recap of the examples featured In the article are available on this repository:

https://github.com/vocdoni/tutorials/tree/master/governance-as-a-service

Billing

Pricing

How much does Vocdoni cost?

At Vocdoni we have a mission: to democratize access to secure digital voting.

It is for this reason that we have implemented a fair trade policy that will allow us to offer universal and free access to our platform, while offering payment plans with advanced features for those organizations with wider needs.

End-users will never pay for the use of the platform. Vocdoni is currently in the restricted access phase. It is expected to move into the open access phase during Q1 2021. If you are an organization, and you are interested in using Vocdoni to make secure digital voting, you can contact us through **this form**.

Misc

Case studies

Organization name	Type	Members	Type of voting	Channel
Òmnium Cultural	Association	185000	AGM Board elections	Web
Centre Excursionista de Catalunya	Association	4500	AGM	Web

Centre Excursionista de Catalunya

> The Centre Excursionista de Catalunya (CEC) is a sports and cultural entity founded in 1876 that has about 4,500 members and a total of 7 shelters.
>
> The CEC is organized by committees that conduct a wide range of activities and sporting and cultural outlets.

Challenge

Due to the restrictions in mobility and gatherings because of COVID-19, the Centre Excursionista de Catalunya decided to hold its AGM 2020 in a way that guarantees the voting rights of its 4500 members through remote participation and in accordance with the Catalan associations' law.

They had certain requirements:

- Members should be able to vote from their homes using their own devices.
- To facilitate accessibility and avoid the digital gap, members could also go to the entity's headquarters and vote from devices provided by the entity.

Solution

To hold this

Vocdoni proposed the use of his voting system accessible via internet browsers.

The system generates an ephemeral private key for each user by using two attributes in a census.

Then users access to vote using their attributes

Outcome

Testimonial

Toni Barnils, CEO of Centre Excursionista de Catalunya

> *The organization and development of our first telematic Annual General Meeting has been a success. Vocdoni has provided us with tailor-made solutions and has diligently and professionally resolved all the incidences. Our partners' satisfaction with monitoring and voting has been very high.*

> *One of the decisive facts that led us to choose Vocdoni was that its platform guaranteed the participation and vote of all our members, with security, reliability and transparency in the whole process, including access and participation in the Annual General Meeting. We also valued the fact that the project was born in Catalonia.*

Successful case: Annual General Meeting 2020

It was organized with hybrid-voting: presential, presential digitally and remote

COVID

The organization and development of our first telematic Annual General Meeting has been a success. Vocdoni has provided us with tailor-made solutions and has diligently and professionally resolved all the incidences. Our partners' satisfaction with monitoring and voting has been very high.

Òmnium Cultural

> The commitment to Vocdoni has been clear, as from Òmnium Cultural we opted for a secure and verifiable voting system that would allow us to hold our statutory meetings with full guarantees to the more than 180,000 members who make up our organization.
> Anna Giralt Communication Head at Omnium Cultural

Other resources