Cordite Documentation

Release 1

Cordite Foundation

Nov 11, 2018

Contents

CHAPTER 1

Introduction

1.1 History

Cordite is as much a philosophy as it is an application, and to understand it's significance, you must also understand its journey.

The Distributed Ledger Technology space is fueled by disruption and innovation with which unavoidably attracts challenges and at times reservations. 2016 was all about performance and scalability, 2017 was all about security. These are technical challenges. The harder challenges are the social and political ones.

One of the key unsolved challenges is how to support a decentralised business model. We believe that there is no point replacing one centralised business model with another. 2018 will be the year that many in the blockchain domain wake up to the fact that there is little point in adopting DLT and distributing their technology if they are not going to distribute the business model as well.

See the case for decentralisation: https://medium.com/@rickcrook/the-case-for-de-centralisation-1ac14935a3fc

And it is this same philosophy, that lies at the core of Cordite. Cordite builds on the work of Richard Gendal Brown, James Carlyle, Ian Grigg and Mike Hearn in their Corda whitepaper in August 2016. The Corda whitepaper introduces us to a Corda Distributed Application ("CorDapp"). We expect a great number of CorDapps will be built to provide improved or new services and to make operational efficiencies. One of the core features of a CorDapp is that it is distributed with no centralised aspect of the technology layer. We believe that distribution should not be limited to the technology layer. Distribution needs to also permeate through the business process the application supports. We would propose that any centralisation in either the technology or business layers would make the use of a distribute ledger technology redundant. A traditional server-client based application would suffice if there is centralisation in either the technology or business layers.

See the Corda Whitepaper at: https://docs.corda.net/_static/corda-technical-whitepaper.pdf

The technology itself is travelling broadly in the right direction, with open-source protocols remaining dominant over proprietary schemes. The initial vendors in this space have largely pivoted their business models after clients were reluctant to engage in full stack solutions. The large technology vendors have attempted to gain ground in the space, and to date no single vendor has managed to dominate and create vendor lock in.

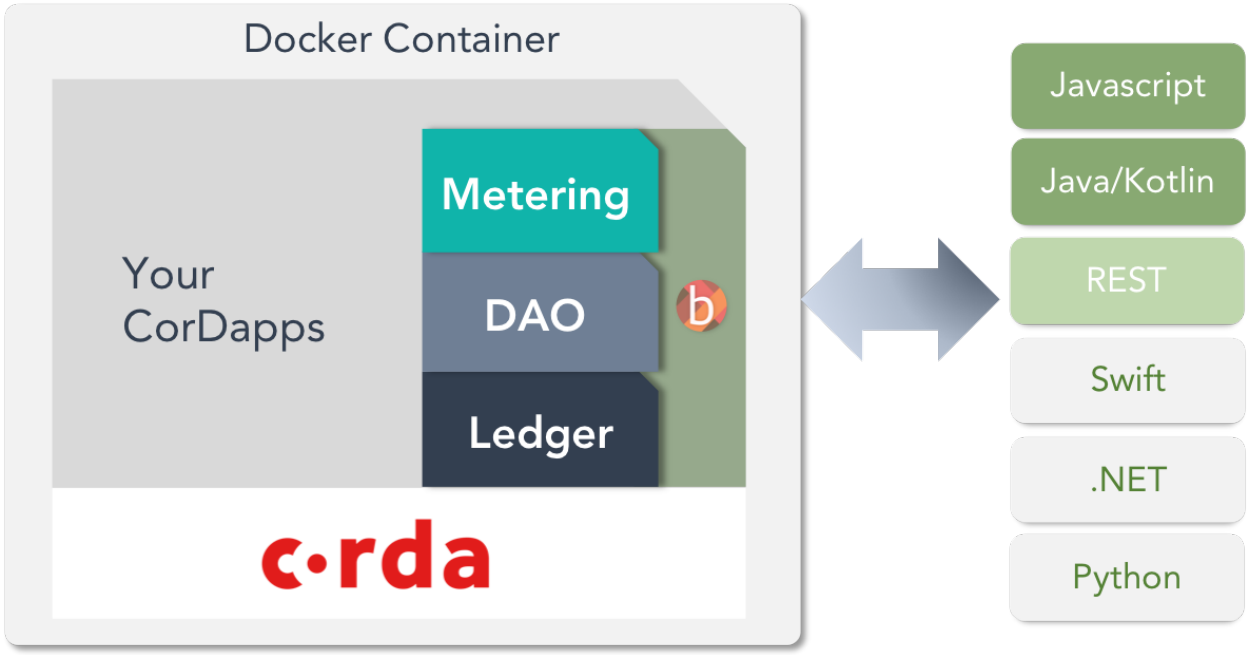
The activity is moving away from its exploration and research phase, and towards targeted delivery and commercialisation of distributed applications. This is largely being driven by consortiums, such as R3, who are working with

financial institutions and other leaders to launch propositions into the market.

Many organisations have learnt that there is only so much fun you can have on your own with a distributed ledger and collaborated to build CorDapps. Some have chosen to create joint ventures or centralised legal entities to fund and operate these CorDapps. The centralised business layer renders the use of a distributed ledger technology redundant. These CorDapps need a distributed funding and incentive for the build and operation of these respectively. For these CorDapps to be truly distributed their governance and economic model needs to be de-centralised too.

As a result, we have Cordite, an open source CorDapp that provides companies and corporations the distributed governance and economic services that CorDapps will need to be truly distributed.

1.2 Architecture



1.3 Roadmap

One of the key unsolved challenges is how to support a decentralised business model. When building decentralised applications, we believe that there is no point replacing one centralised business model with another. 2018 will be the year that many in the blockchain domain wake up to the fact that there is little point in adopting DLT and distributing their technology if they are not going to distribute the business model as well.

Cordite provides decentralised economic and governance services including:

- decentralised stores and transfers of value allowing new financial instruments to be created inside the existing regulatory framework. eg. tokens, crypto-coins, digital cash, virtual currency, distributed fees, micro-billing
- decentralised forms of governance allowing new digital autonomous organisations to be created using existing legal entities eg. digital mutual societies or digital joint stock companies
- decentralised consensus in order to remove the need for a central operator, owner or authority. Allowing Cordite to be more resilient, cheaper, agile and private than incumbent market infrastructure

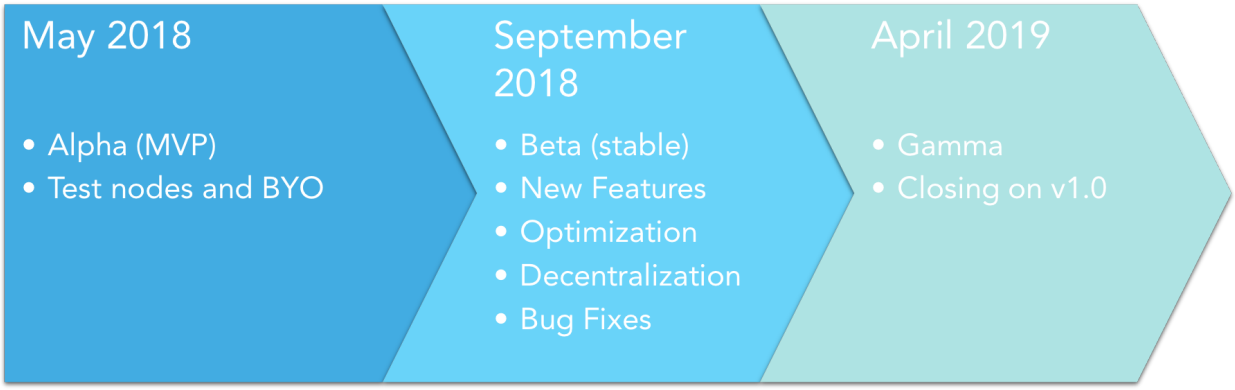


Fig. 1: Cordite Roadmap

1.4 Built on Corda

Distributed Ledger Technology (DLT) ('blockchain') is a disruptive technology that could enable institutions to reduce costs, improve product offerings and increase speed. There has been an explosion of activity - with institutions actively researching, testing and investing in this technology. Financial institutions have remained at the forefront of this development, and have established a strong insight into the underlying technology, the market landscape and the potential value applications for their businesses.

Cordite is built on Corda, a finance grade distributed ledger technology, meeting the highest standards of the banking industry, yet it is applicable to any commercial scenario. The outcome of over two years of intense research and development by over 80 of the world's largest financial institutions.

Cordite is open source, regulatory friendly, enterprise ready and finance grade.

1.5 Cordite Design Principles

1. Don't fork Corda. Stronger together
2. Minimal core system with flexibility to extend
3. Support widest functionality and be least prescriptive
4. Provide exemplar CorDapps using Cordite
5. Minimise CorDapp knowledge of Cordite
6. Code is not law. Cordite will re-use existing legal frameworks to make law is code

1.6 Cordite Cordapp Design Principles

1. Don't be opinionated. You lose people for every opinion
2. Don't implement anything more than the minimum required. Let others extend
3. Don't wait. WYSIWYG. Raise feature requests on Corda and work round
4. Be Open Source (Presently 24 contributors, multiple maintainers from disparate orgs)
5. No manual test. Aggressive use of automated testing from unit to full scale environments
6. Sustain code quality and test coverage with code reviews and commit gates
7. Maintain a bi-weekly cadence. Have the ability to release at any moment in time from a "green" build

CHAPTER 2

Concepts

2.1 DGL

The Distributed General Ledger provides a de-centralised way to manage accounts and issue tokens by: - Providing de-centralised stores and transfers of value allowing new financial instruments to be created inside the existing regulatory framework. eg. tokens, crypto-coins, digital cash, virtual currency, distributed fees, securitisation - Allowing participants to record, manage and execute financial agreements in perfect synchrony with their peers, creating a world of frictionless commerce. - Providing a financial toolkit that can include promissory notes, settlement issuances and account aliases.

2.1.1 Why are we interested in a DGL?

When a company provides goods or services to another, they tend to expect payment in return. For hundreds of years, companies have recorded such sales or purchases on their respective ledgers. These general ledgers represent what they own (their assets) and what they owe (their liabilities) and both companies will endeavor to ensure both ledgers match.

In order to do this, they rely on sharing purchase orders, invoices, payment instructions and receipts. These formal documents are followed up with a range of communication including meetings, emails, facsimile and phone calls. All of these resources and efforts are exhausted, simply to try and keep the two over lapping accounts (sales and purchase in this example) matched on their respective ledgers.

A company will likely have many of these overlapping accounts between themselves, their clients and suppliers. As you can imagine, keeping these all aligned becomes a tremendous drag on all parties involved.

2.1.2 Where do banks fits in?

Banks will also have a general ledger representing their assets and liabilities. One of the liabilities on this ledger will be the deposits their corporate customers have made into the bank. The bank calls these deposit 'accounts' on their ledger, but the corporate customer will see these as 'cash (or bank) accounts' on their ledger.

Just another case of overlapping accounts between two general ledgers which need to be aligned. Another case of expended effort and drag.

2.1.3 More madness

In order to make payments between corporate customers at different banks, banks also need to have relationships with each other. Be it between a corresponding bank, a central bank, a reserve bank or a clearing bank.

Each bank will hold deposits with these other banks to cement the relationship. Bank A will hold a nostro account to record the banks money held in a deposit at another bank B. Bank B will hold a vostro account to record the deposits they have received from bank A. All of these banks might have many of these types of vostro or nostro accounts on their respective ledgers and guess what, each bank needs to keep all these overlapping accounts aligned. More madness, more drag.

2.1.4 Distributed General Ledger

One way of keeping all the overlapping accounts aligned would be to share all the transactions on a single public ledger and have all the corporates validate each other's transactions. The obvious disadvantage of this solution is that everyone can see each other's transactions.

A second way would be to allow each corporate over time to connect their general ledger with the general ledger of their clients, suppliers and banks. Thus, creating a private peer to peer network of inter-connected ledgers. Enter the Distributed General Ledger.

The Distributed General Ledger would allow corporates to record, manage and execute financial agreements in perfect synchrony with their client, suppliers and banks, creating a world of frictionless commerce.

2.1.5 Cordite DGL / Tokens requirements summary

Some have asked for the requirements for Tokens in Cordite. This a precis of a large number of requirements captured as project tickets. Built for a NatWest project but with an eye on new use-cases being discovered.

1. Define a unit level instrument (Token type) that can represent fiat, crypto currencies, reward tokens, audio credits or units of basic tokenised assets - equities, bonds, mortgages
2. Should be able to support aliasing - it multiple 'instrument' codes - for example XBT, BTC
3. Aliasing should also support validity date ranges (for example ISINs)
4. Must be able to support micro units and hence exponent should be part of definitiion
5. Corda Parties should be able to create Token types with 'strong' issuance such that Tokens are unique by Issuer and Token Name and be universally unique
6. Tokens should be referenced by their unique identifier across nodes without the need for copying the whole token definition
7. Corda Parties must be able to create accounts to store tokens
8. Accounts must be able to support multiple token types
9. Accounts should be uniquely identifiable across all nodes - using node identity plus account name
10. Accounts must support aliases so that they can be identified by existing methods, such as IBAN, Sortcode and Account Number
11. Accounts must support multiple financial structures such as trading book hierarchies and accounting hierarchies so that aggregation of positions and financial position can be calculated

12. Accounts should support multiple owners (e.g. joint accounts, business accounts) and only allow transfers when a quorum of signatures is achieved
13. Accounts should be able to span multiple nodes for resilience and decentralisation
14. Accounts should be able to be owned by non node (i.e. user) parties
15. Tokens must only be issued to accounts
16. Tokens must be able to be transferred between accounts on the same node or between nodes using primary or aliased account identifiers
17. All Token transactions must produce a summary of debits and credits
18. Tokens and Accounts should be searchable by primary and secondary (alias) identifiers
19. We must be able to listen to account changes by specifying single or groups of primary identifiers or tags.
20. Tokens may be issued back-to-back from existing off-ledger assets, in which case it should be possible to immobilise the off-ledger asset thereby ensuring that the underlying assets cannot be 'spent' independantly from the issued token. The Token should be a specialised form a PromissoryNote with immediate or future maturity and partial redemption.

2.1.6 DGL : Token issuance and transfer

Georgina's great-great-great-grandfather (**grandfather**) was shipwrecked on Kirrin Island. On the **Ship** was a 100 gold coins (**XAU**). **Georgina** lives with her **Uncle Quentin** at **Kirrin Cottage**. **Georgina** and her friends - **Julian**, **Dick** and **Anne** find the gold on Kirrin Island. **Georgina** puts the gold in her **pocket**. She returns to **Kirrin Cottage** and gives it to **Uncle Quentin**. **Uncle Quentin** splits the gold between the four children - **Georgina**, **Julian**, **Dick** and **Anne**. A simple test of the Cordite DGL functionality. Each actor can run a separate Cordite node creating a Distributed General Ledger (DGL).

2.1.7 Direct pay: FX transfers

Anne has moved to the US, and **Julian** wants to send some pocket money to **Anne**. However, **Julian** only has British pounds (GBP) while **Anne** needs US dollars (USD). Therefore, Julian sends GBP to a **FX provider** with instructions of converting the GBP to USD before sending to **Anne**. The **FX provider** carries out the currency conversion using the current FX rate, sends USD to **Anne**, and levies a fee that is deducted from the transaction. **Anne** then receives USD in her account.

2.1.8 Multi-currency wallet

Anne realises that she still needs GBP when she goes back the UK, so she created a **Multi-Currency Wallet** that can hold both USD and GBP. Next time, **Julian** can send GBP directly to **Anne**, who can receive the GBP and store it in her **Wallet**.

Apologies to Enid Blyton, the author of The Famous Five Series. The characters and stories are based on this series.

2.1.9 CLI Demonstration

Click here for the CLI Demonstration

2.2 DAO

The DAO provides de-centralised governance by: - Allowing new digital autonomous organisations to be created using existing legal entities. For example, digital mutual societies or digital joint stock companies. - Provides de-centralised consensus in order to remove the need for a central operator, owner or authority. This allows cordite to be more resilient, cheaper, agile and private than the incumbent financial infrastructure. - Facilitates the funding, building and organising of decentralised applications. - Encapsulates voting, raising issues, marshaling changes to the system and deciphering economic models.

2.2.1 Why are we interested in DAOs?

It is really hard to bring together a group of organisations to fund and build a decentralised application that they would all like to use.

We have repeatedly seen that if you try and build a "blockchain app" and sell it, no one is interested - and you can't have a blockchain app on your own! However if you talk to other institutions to investigate co-creation, then they are often really interested. However actually getting a disparate group of organisations together to fund the building of a decentralised app is really hard.

- getting budgeting cycles aligned is hard
- getting experts together is hard
- who would actually build the app?
- who will make changes?
- who chooses which changes to build next?
- who will get those changes deployed safely to many institutions' nodes?

However imagine if we spin up a decentralised autonomous organisation - a DAO. We could create governance, economic and membership models, in code, that make it easier to bring organisations together to:

- fund the building of a decentralised app
- propose, and vote for, changes to the app
- propose, and vote for, changes to the dao itself - e.g. changes to voting or membership rules

Some DAOs may:

- offer incentives for joining the dao early
- have rules for distributing proceeds for deploying the system to another geography

A **Decentralised Autonomous Orgnisation (DAO)** is an organisation that is governed by rules based on computer code or network protocols and its decisions are made through the voting of its members.

We think that DAOs represent a really elegant way to bring organisations together to fund the building, and running, of decentralised applications between institutions much easier, without having to resort to a centralised company sitting in the middle monetising this.

Open source foundations represent existing example that is not that far away from what we are talking about here.

For an example of DAO, see Dash

DAO : Membership

Julian starts a gang with his brother **Dick** called the **Famous Five**. **Dick** proposes their sister **Anne** joins and **Julian** accepts. **Anne** proposes their friend **Georgina**. Both **Dick** and **Julian** accept. **Georgina** proposes her dog **Timmy** joins but none of the others accept her proposal. A simple test of the Cordite DAO functionality. Each actor can run a separate Cordite node where **Cordite** is the collection of nodes.

DAO: Proposals

Julian proposes to his gang **FamousFive** that they head to **Treasure Island**. **Anne** proposes an alternative to **Go Off in a Caravan**. **Dick** and **Georgina** vote for the **Treasure Island** proposal. **Anne** finds her proposal has been rejected, while **Julian**'s has been accepted. The **FamousFive** head to **Treasure Island**. A simple test of the Cordite DAO functionality. Each actor can run a separate Cordite node where **FamousFive** is a distributed autonomous organisation (DAO) running on Cordite.

Apologies to Enid Blyton, the author of The Famous Five Series. The characters and stories are based on this series.

2.2.2 CLI Demonstration

Click here for the CLI Demonstration

2.3 Metering

Metering incentivises people to fairly participate in de-centralised organisations by: • Incentivising parties to run metering notaries that can receive payments for notarisation. • Accommodating a variety of economic models, such as PayGo and other customised structures. • Handling invoice and dispute resolution.

Incentives—We all need them and we all use them. From how you are remunerated for your time to the ice cream you give your kids to reward them for good behaviour; distributed ledgers are no exception. Since there is no central party governing the network of a distributed ledger, we rely upon the participants to provide services to keep the show on the road, and no one does this for free.

Different distributed ledgers will need different incentives to reward participants providing services to the network. More than one economic model is expected to emerge in order to server the different applications of a distributed ledger technology.

2.3.1 Lovely Rita, meter maid

Corda introduced us to notaries to verify the transactions. Building on this, Cordite introduces the concept of Metering Notaries. Metering notaries verify the transaction and sends an invoice to the party originating the transaction including the notary's fees, much like a gas or electric utility company does. The transacting party can accept or dispute these invoices. They can pay the invoice using *Cordite DGL tokens*. Metering notaries should be able to refuse to verify transactions for parties that do not pay their invoices.

Metering fees are collected in a *Cordite DAO* (aka Digital Mutual) and distributed back to the metering notaries. This provides the ability for pools of metering notaries to operate where they share the fees and the DAO provides governance over the metering notary pool in order to allow the metering pool to evolve over time. The DAO itself may also take a share of the fees in order to provide services

Metering Notaries have their own transactions verified by a guardian notary. The guardian notaries can also be run as Metering Notaries to create two inter-locked sets of metering notaries checking each other's transactions in a de-centralised model. Depending on preference or application, you can adopt a variety of pricing models for your Dapp. It allows organisations with differing economic needs and applications to adapt respectably.

The key point here, is that Metering Notaries represent an elegant and versatile method to provide the necessary incentives while maintaining the benefits of de-centralisation.

Below is a story showcasing a simple token transaction and how metering works.

2.3.2 Metering: Token Transaction

Georgina sends a token to **Dick**, and the token transaction is notarised by a **Metering Notary** who charges a fee and issues a Metering Invoice to **Georgina**. The Metering Invoice transaction is then notarised by a **Guardian Notary** before the Metering Invoice is paid by **Georgina** with tokens to the **DAO**. The **DAO** then distributes the funds according to the **DAO** rules.

Apologies to Enid Blyton, the author of The Famous Five Series. The characters and stories are based on this series.

2.3.3 CLI Demonstration

Click here for the CLI Demonstration

2.4 Foundation

The Cordite Foundation is, currently, a necessary thin legal wrapper around Cordite, which is a DAO in Cordite.

The Foundation will be bootstrapped with a set of simple rules around:

- membership - ie rules around joining
- governance - who can make proposals, vote for proposals, what happens to accepted proposals
- economics - how proceeds are used to run and evolve Cordite

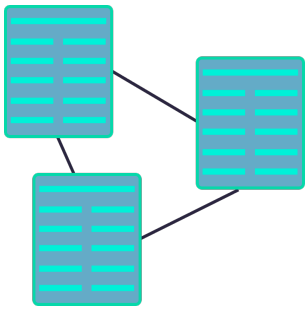
Members of the Cordite Foundation are expected to:

- have proof of stake

- provide core infrastructure
- make proposals for evolving all aspects of cordite

The aim of the foundation is to create a body of people who can be trusted to over see the funding, running and evolution of cordite over time. Much like an open source foundation.

DGL



ledger with accounts *DAO*

governance *Metering*

economics

CHAPTER 3

Cordite Test

3.1 DGL CLI Demonstration

Current node addresses

- https://amer-test.cordite.foundation:8080
- https://apac-test.cordite.foundation:8080
- https://emea-test.cordite.foundation:8080

In this CLI demonstration we will: - Create two accounts on seperate nodes - Create a token type - Issue tokens of the created type to an account - Look at the balance of that account to check the issuance was succesful - transfer those tokens to the other account on the other node - Look the balances of the accounts to check the transaction was succesful

3.1.1 Commands used

Node	Command
amer	`notaries`
amer	`ledger.createAccount("myBankAMER4", "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB")`
apac	`ledger.createAccount("myBankAPAC4", "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB")`
amer	`ledger.createTokenType("TOK4", 2, "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB")`
amer	`ledger.issueToken("myBankAMER4", 100, "TOK4", "my first token", "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB")`
amer	`ledger.balanceForAccount("myBankAMER4")`
amer	`ledger.transferToken(10, "TOK4:2:OU=Cordite Foundation, O=Cordite AMER, L=New York City, C=US", "myBankAMER4", "myBankAPAC4@OU=Cordite Foundation, O=Cordite APAC, L=Singapore, C=SG", "my first transfer", "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB")`
apac	`ledger.balanceForAccount("myBankAPAC4")`
amer	`ledger.balanceForAccount("myBankAMER4")`

3.2 DAO CLI Demonstration

Current node addresses

- https://amer-test.cordite.foundation:8080
- https://apac-test.cordite.foundation:8080
- https://emea-test.cordite.foundation:8080

In this CLI demonstration we will: - Create a dao - Add two members in addition to the dao creator - Create two proposals - Vote on the proposals - Accept the proposal which has the vote of all members

3.2.1 Commands used

Node	Command
amer	`notaries`
amer	`dao.createDao("Famous-Five", "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB")`
amer	`dao.daoInfo("Famous-Five")[0].members`
emea	`julian = network.getNodeByLegalName("OU=Cordite Foundation, O=Cordite AMER, L=New York City, C=US").legalIdentities[0]`
emea	`proposalState = dao.createNewMemberProposal("Famous-Five", julian)`
emea	`dao.acceptNewMemberProposal(proposalState.proposal.proposalKey, julian)`
emea	`dao.daoInfo("Famous-Five")[0].members`
apac	`dick = network.getNodeByLegalName("OU=Cordite Foundation, O=Cordite EMEA, L=London, C=GB").legalIdentities[0]`
apac	`anneProp = dao.createNewMemberProposal("Famous-Five", dick)`
amer	`ffkey = dao.daoInfo("Famous-Five")[0].daoKey`
amer	`apkey = dao.newMemberProposalsFor(ffkey)[0].proposal.proposalKey`
amer	`dao.voteForMemberProposal(apkey)` `
apac	`apkey = anneProp.proposal.proposalKey`
apac	`dao.acceptNewMemberProposal(apkey, dick)`
apac	`dao.daoInfo("Famous-Five")[0].members`
amer	`treasureState = dao.createProposal("Treasure Island", "head to treasure island", ffkey)`
apac	`ffkey = dao.daoInfo("Famous-Five")[0].daoKey`
apac	`caravanState = dao.createProposal("Caravan", "go off in a caravan", ffkey)`
emea	`ffkey = dao.daoInfo("Famous-Five")[0].daoKey`
emea	`treasureKey = dao.normalProposalsFor(ffkey)[0].proposal.proposalKey`
emea	`dao.voteForProposal(treasureKey)`
amer	`treasureAccept = dao.acceptProposal(treasureState.proposal.proposalKey)`
amer	`treasureAccept.lifecycleState`

3.3 Metering CLI Demonstration

Current node addresses

- https://amer-test.cordite.foundation:8080
- https://apac-test.cordite.foundation:8080
- https://emea-test.cordite.foundation:8080

In this CLI demonstration we will: - Transfer some tokens between accounts on different nodes - Look at outstanding invoices on the metering notary - Pay the metering invoice for the transaction - Look at the balances of the metering notary, guardian notary and committee node to check that the funds from the invoice payment have been dispersed correctly

3.3.1 Commands used

Node	Command
amer	`ledger.transferToken("50.00", Token1.descriptor.uri, "Georgina", Dick, "transfer", notaries.corditeMeteringNotary.name)`
emea	`ledger.balanceForAccount("Dick")`
amer	`meterer.listInvoices("ISSUED")`
amer	`transactionId = meterer.listInvoices("ISSUED"[0]. meteringInvoiceProperties.me teredTransactionId`
amer	`meterer.payInvoice({meteredTransactionId:transactionId,from Account:"Georgina"})`
metering notary	`ledger.balanceForAccount("metering-notary-account1")`
guardian notary	`ledger.balanceForAccount("guardian-notary-account1")`
commit-tee	`ledger.balanceForAccount(dao-foundation-account")`

3.4 Braid JS Client Example

3.4.1 Aim

The aim of this tutorial is to create a small cordite client in node js using braid. We will connect this to the braid endpoint of the emea test node in the cordite test network. We will then proceed to make a few doc type queries and also to create your very own Dao!

At the end of this tutorial you should be comfortable creating javascript cordite clients to call your serverside braid endpoints.

The completed code for this tutorial can be found here.

3.4.2 Pre-requisites

- NodeJS installed on your machine

3.4.3 Steps

1. In a terminal, create a directory in which to put your code and change to that directory

```
mkdir braidJsClient
cd braidJsClient
```

2. Next, initialise a node project:

```
> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (wibble) cordite-braid-js-client
version: (1.0.0)
description:
entry point: (index.js) client.js
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /private/tmp/wibble/package.json:

{
"name": "cordite-braid-js-client",
"version": "1.0.0",
"description": "",
"main": "client.js",
"scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
},
"author": "",
"license": "ISC"
}

Is this ok? (yes) y
```

3. Install the braid javascript client

```
npm install --save braid-client
```

4. Connect braid to the cordite emea test test node
    - create the client.js file we specified above: `touch client.js`.
    - edit this file to import the dependency and connect to emea test

```
const Proxy = require('braid-client').Proxy;

const emea = new Proxy({url: 'https://emea-test.cordite.foundation:8080/api/'},␣
↪onOpen, onClose, onError, {strictSSL: false})
```

(continues on next page)

(continued from previous page)

```
function onOpen() {
    console.log("connected")
}

function onClose() {
    console.log("closed")
}

function onError(err) {
    console.error(err)
}
```

If we run this using `node client.js` we should get the output: `connected!`.

5. Next we want to see what is in the braid endpoint. So change the `onOpen()` function to:

```
function onOpen() {
    console.log("connected to the emea test cordite node")
    console.log(JSON.stringify(emea))
    console.log("\ndao functions:")
    Object.getOwnPropertyNames(emea.dao).forEach(f => console.log(f))
    console.log("")
    emea.dao.daoInfo.docs()
}
```

This should give the output:

```
node client.js
connected to the emea test cordite node
{"network":{},"flows":{},"ledger":{},"dao":{}}

daoFunctions:
getServiceHub
voteForProposal
daoInfo
createDao
newMemberProposalsFor
normalProposalsFor
modelDataProposalsFor
createProposal
createNewMemberProposal
createRemoveMemberProposal
acceptNewMemberProposal
acceptRemoveMemberProposal
acceptProposal
voteForMemberProposal
requestProposalConsistencyCheckFor
createModelDataProposal
voteForModelDataProposal
acceptModelDataProposal

API documentation
```

(continues on next page)

(continued from previous page)

```
-----------------
* daoInfo(daoName) => array

@param daoName - string
```

Which we know means we have four `ServiceProxy` objects, one each for network, flows, ledger and dao. The second line prints out all the available methods and the last line prints out the docs for the daoInfo method.

6. Finally lets create a dao:

```
let saltedDaoName = 'testDao-'+new Date().getTime()

function onOpen() {
    console.log("connected to the emea test cordite node")

    emea.dao.daoInfo(saltedDaoName).then(daos => {
        console.log("there were", daos.length, "existing daos with name",␣
↪saltedDaoName )

        return emea.dao.createDao(saltedDaoName, "O=Cordite Metering Notary,␣
↪OU=Cordite Foundation, L=London,C=GB")
    }).then(dao => {
        console.log(saltedDaoName,"created with key",JSON.stringify(dao.daoKey))
    }).catch(error => {
        console.error(error)
    })
}
```

Running this gives us:

```
connected to the emea test cordite node
there were 0 existing doas with name testDao-1524060634372
testDao-1524060634372 created with key {"name":"testDao-1524060634372","uuid":
↪"f99c32c4-7e9c-4c3a-af99-9765d8e6e5b4","uniqueIdentifier":{"externalId":
↪"testDao-1524060634372","id":"f99c32c4-7e9c-4c3a-af99-9765d8e6e5b4"}}
```

The full code is now:

```
const Proxy = require('braid-client').Proxy;

const emea = new Proxy({url: 'https://emea-test.cordite.foundation:8080/api/'},␣
↪onOpen, onClose, onError, {strictSSL: false})

let saltedDaoName = 'testDao-'+new Date().getTime()

function onOpen() {
    console.log("connected to the emea test cordite node")
```

(continues on next page)

(continued from previous page)

```
    emea.dao.daoInfo(saltedDaoName).then(daos => {
        console.log("there were", daos.length, "existing daos with name",␣
↪saltedDaoName )

        return emea.dao.createDao(saltedDaoName, "O=Cordite Metering Notary,␣
↪OU=Cordite Foundation, L=London,C=GB")
    }).then(dao => {
        console.log(saltedDaoName,"created with key",JSON.stringify(dao.daoKey))
    }).catch(error => {
        console.error(error)
    })
}

function onClose() {
    console.log("closed")
}

function onError(err) {
    console.error(err)
}
```

3.5 Token JS Example

Cordite has tokens, just like Corda. Cordite however is capable of issuing tokens to specific accounts on a node. Token issuance must be notarised. You can create token types and accounts to your heart's content!

By this stage you know how to connect using the CLI or the *Braid JS Client*, so follow along with your chosen client.

This page explains how to:

- create a token type
- create an account
- issue tokens to an account
- display the balance of an account

Connect to a cordite node Firstly, connect to a cordite node - emea test in this case. Create variables for the desired notary, token type name, and account name.

```
const Proxy = require('braid-client').Proxy;
const emea = new Proxy({url: 'https://emea-test.cordite.foundation:8080/api/'},␣
↪onOpen, onClose, onError, {strictSSL: false})

let saltedTokenName = 'TOK-'+new Date().getTime()
let saltedAccountName = 'Acc-'+new Date().getTime()
let notary = "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB"

function onOpen() {
    console.log("connected to the emea test cordite node")
}

function onClose() {
    console.log("closed")
```

(continues on next page)

(continued from previous page)

```
}

function onError(err) {
    console.error(err)
}
```

Create token type and account Now create the token type and the Account. For the `createTokenType()` function you must specify a name or symbol such as USD, an exponent, and a notary. When creating an account with `createAccount()`, just specify the desired name/ID and a notary.

```
function onOpen() {
    console.log("connected to the emea test cordite node")

    emea.ledger.createTokenType(saltedTokenName, 2, notary).then( a => {
        console.log("Token with name " + saltedTokenName + " created")
        return emea.ledger.createAccount(saltedAccountName, notary)
    }).then( b => {
        console.log("Account with name " + saltedAccountName + " created")
    }).catch(error => {
        console.error(error)
    })
}
```

Issue tokens and check balance Finally, issue tokens of the type just created to your new account with `issueToken()`. Specify the account name, quantity, token type, a message, and a notary. We then query the balance of the account with `balanceForAccount()` to check that the tokens were issued. For this function we need only specify the account name.

```
function onOpen() {
    console.log("connected to the emea test cordite node")

    emea.ledger.createTokenType(saltedTokenName, 2, notary).then( a => {
        console.log("Token with name " + saltedTokenName + " created")
        return emea.ledger.createAccount(saltedAccountName, notary)
    }).then( b => {
        console.log("Account with name " + saltedAccountName + " created")
        return emea.ledger.issueToken(saltedAccountName, 100, saltedTokenName, "First␣
↪issuance", notary)
    }).then( c => {
        console.log("Tokens of type " + saltedTokenName + " issued to " +␣
↪saltedAccountName)
        return emea.ledger.balanceForAccount(saltedAccountName)
    }).then( d => {
        bal = (d[0].quantity * d[0].displayTokenSize) + " " + d[0].token.symbol
        console.log("Balance for " + saltedAccountName + ": " + bal)
    }).catch(error => {
        console.error(error)
    })
}
```

Your console output should now show the balance of your account. The token type and quantity are displayed. For example:

```
connected to the emea test cordite node
Token with name TOK-1532429563901 created
Account with name Acc-1532429563901 created
```

(continues on next page)

(continued from previous page)

```
Tokens of type TOK-1532429563901 issued to Acc-1532429563901
Balance for Acc-1532429563901: 100 TOK-1532429563901
```

The full code is now:

```
const Proxy = require('braid-client').Proxy;
const emea = new Proxy({url: 'https://emea-test.cordite.foundation:8080/api/'}, onOpen, onClose, onError, {strictSSL: false})

let saltedTokenName = 'TOK-'+new Date().getTime()
let saltedAccountName = 'Acc-'+new Date().getTime()
let notary = "OU=Cordite Foundation, O=Cordite Guardian Notary, L=London, C=GB"

function onOpen() {
    console.log("connected to the emea test cordite node")

    emea.ledger.createTokenType(saltedTokenName, 2, notary).then( a => {
        console.log("Token with name " + saltedTokenName + " created")
        return emea.ledger.createAccount(saltedAccountName, notary)
    }).then( b => {
        console.log("Account with name " + saltedAccountName + " created")
        return emea.ledger.issueToken(saltedAccountName, 100, saltedTokenName, "First issuance", notary)
    }).then( c => {
        console.log("Tokens of type " + saltedTokenName + " issued to " + saltedAccountName)
        return emea.ledger.balanceForAccount(saltedAccountName)
    }).then( d => {
        bal = (d[0].quantity * d[0].displayTokenSize) + " " + d[0].token.symbol
        console.log("Balance for " + saltedAccountName + ": " + bal)
    }).catch(error => {
        console.error(error)
    })
}

function onClose() {
    console.log("closed")
}

function onError(err) {
    console.error(err)
}
```

3.6 DAO JS Example

Cordite has the concept of *DAOs*. There is a Cordite Foundation DAO, called the committee. You can also create your own DAOs, either by using the `DaoService` or by extending this code for your own purposes (in which case please *contribute* your awesome changes back to Cordite).

By this stage you know how to connect using the CLI or the *Braid JS Client* so follow along with your chosen client.

This page explains how to:

- create your own DAO
- other people can request membership

- create proposals
- vote for proposals
- accept proposals

The full code can be found here

3.6.1 Connecting to two cordite nodes

DAOs don't make sense unless you have a few Parties that want to be part of the DAO. In this case we are going to work with two parties; the EMEA and AMER nodes on the Test network. This introduces some slight complexity over and above the *Braid JS Client* tutorial because we need to wait for two braid proxies to connect before we can start. There are many ways of solving this, one of which is shown below:

```
const Proxy = require('braid-client').Proxy;

// set up using test network
let emeaAddress = "https://emea-test.cordite.foundation:8080/api/"
let amerAddress = "https://amer-test.cordite.foundation:8080/api/"

const emea = new Proxy({url: emeaAddress}, onOpenEmea, onClose, onError, {strictSSL:␣
↪false})
var amer

function onOpenEmea() {
    console.log("connected to emea.  connecting to amer...")
    amer = new Proxy({url: amerAddress}, onBothReady, onClose, onError, {strictSSL:␣
↪false})
}

function onBothReady() {
    console.log("also connected to amer...starting test")
}

function onClose() {
    console.log("closed")
}

function onError(err) {
    console.error(err)
}
```

With this we will be adding most of our code to the `onBothReady` method.

3.6.2 Create your own DAO

First, let's create a new DAO - you need to give this a unique name (atleast for the node you're running on). For the purposes of this walkthrough we will call our DAO "My Dapp Dao". Its purpose is to gather together some interested parties to fund, build and then later run a decentralised app that several people want to build together.

Note this assumes you have two js apps connected to the emea and amer nodes in the *cordite test network*.

```
let saltedDaoName = 'testDao-'+new Date().getTime()
var daoKey
let meteringNotaryName = "O=Cordite Metering Notary, OU=Cordite Foundation, L=London,
↪C=GB"
```

(continues on next page)

(continued from previous page)

```
function onBothReady() {
    console.log("also connected to amer...starting test")

    emea.dao.createDao(saltedDaoName, meteringNotaryName).then(daoState => {
        daoKey = daoState.daoKey
        console.log("emea created dao with name",saltedDaoName,"and key",daoKey)

    }).catch(error => {
        console.error(error)
    })
}
```

3.6.3 Amer joins the DAO

Next, the amer party would like to join the DAO. So it needs to get the sponsoring node (in this case the only existing DAO member is emea, so we use this) and the daoName.

In this case, the proposer implicitly supports the new member so there are already enough supporters to propose acceptance of the proposal so we will do that here. If there were more DAO members we would have garner more support before proposing acceptance.

```
var emeaParty
let emeaNodeName = "OU=Cordite Foundation, O=Cordite EMEA, L=London, C=GB"

function onBothReady() {
    console.log("also connected to amer...starting test")

    emea.dao.createDao(saltedDaoName, meteringNotaryName).then(daoState => {
        daoKey = daoState.daoKey
        console.log("emea created dao with name",saltedDaoName,"and key",daoKey)
        return amer.network.getNodeByLegalName(emeaNodeName)
    }).then(emeaNode => {
        emeaParty = emeaNode.legalIdentities[0]
        console.log("amer asking to join dao")
        return amer.dao.createNewMemberProposal(saltedDaoName, emeaParty)
    }).then(proposalState => {
        console.log("proposalKey:",proposalState.proposal.proposalKey)
        console.log("both members already support so we can just propose acceptance")
        return amer.dao.acceptNewMemberProposal(proposalState.proposal.proposalKey,␣
↪emeaParty)
    }).then(proposal => {
        console.log("proposal state now:", proposal.lifecycleState)
        console.log("dao members now", proposal.members.map(x => x.name).join())
    }).catch(error => {
        console.error(error)
    })
}
```

3.6.4 Amer proposes change to voting rules

Finally, the amer node would like to propose making a small change to the voting rules. So it must:

- create a new proposal

- talk the emea node into voting for it
- propose acceptance of the proposal

The full code is now:

```
const Proxy = require('braid-client').Proxy;

// set up using test network
let emeaAddress = "https://emea-test.cordite.foundation:8080/api/"
let amerAddress = "https://amer-test.cordite.foundation:8080/api/"

const emea = new Proxy({url: emeaAddress}, onOpenEmea, onClose, onError, {strictSSL:␣
↪false})
var amer

let saltedDaoName = 'testDao-'+new Date().getTime()
let meteringNotaryName = "O=Cordite Metering Notary, OU=Cordite Foundation, L=London,
↪C=GB"
let emeaNodeName = "OU=Cordite Foundation, O=Cordite EMEA, L=London, C=GB"

var daoKey
var emeaParty
var normalProposalKey

function onOpenEmea() {
    console.log("connected to emea.  conneting to amer...")
    amer = new Proxy({url: amerAddress}, onBothReady, onClose, onError, {strictSSL:␣
↪false})
}

function onBothReady() {
    console.log("also connected to amer...starting test")

    emea.dao.createDao(saltedDaoName, meteringNotaryName).then(daoState => {
        daoKey = daoState.daoKey
        console.log("emea created dao with name",saltedDaoName,"and key",daoKey)
        return amer.network.getNodeByLegalName(emeaNodeName)
    }).then(emeaNode => {
        emeaParty = emeaNode.legalIdentities[0]
        console.log("amer asking to join dao")
        return amer.dao.createNewMemberProposal(saltedDaoName, emeaParty)
    }).then(proposalState => {
        console.log("proposalKey:",proposalState.proposal.proposalKey)
        console.log("both members already support so we can just propose acceptance")
        return amer.dao.acceptNewMemberProposal(proposalState.proposal.proposalKey,␣
↪emeaParty)
    }).then(proposal => {
        console.log("proposal state now:", proposal.lifecycleState)
        console.log("dao members now", proposal.members.map(x => x.name).join())
        console.log("now amer proposes to change the membership rules")
        return amer.dao.createProposal("change voting percentage","change the voting␣
↪percentage of people needed to accept", daoKey)
    }).then(normProposalState => {
        normalProposalKey = normProposalState.proposal.proposalKey
        console.log("new proposal created with key",normalProposalKey)
        console.log("emea decides to support proposal")
        return emea.dao.voteForProposal(normalProposalKey)
    }).then(votedProposalState => {
```

(continues on next page)

(continued from previous page)

```
        console.log("should be two supporters now", votedProposalState.supporters.
↪length)
        console.log("amer proposes to accept")
        return amer.dao.acceptProposal(normalProposalKey)
    }).then(acceptedProposal => {
        console.log("should be accepted",acceptedProposal.lifecycleState)
        console.log("and we are done :-)")
}).catch(error => {
        console.error(error)
    })
}

function onClose() {
    console.log("closed")
}

function onError(err) {
    console.error(err)
}
```

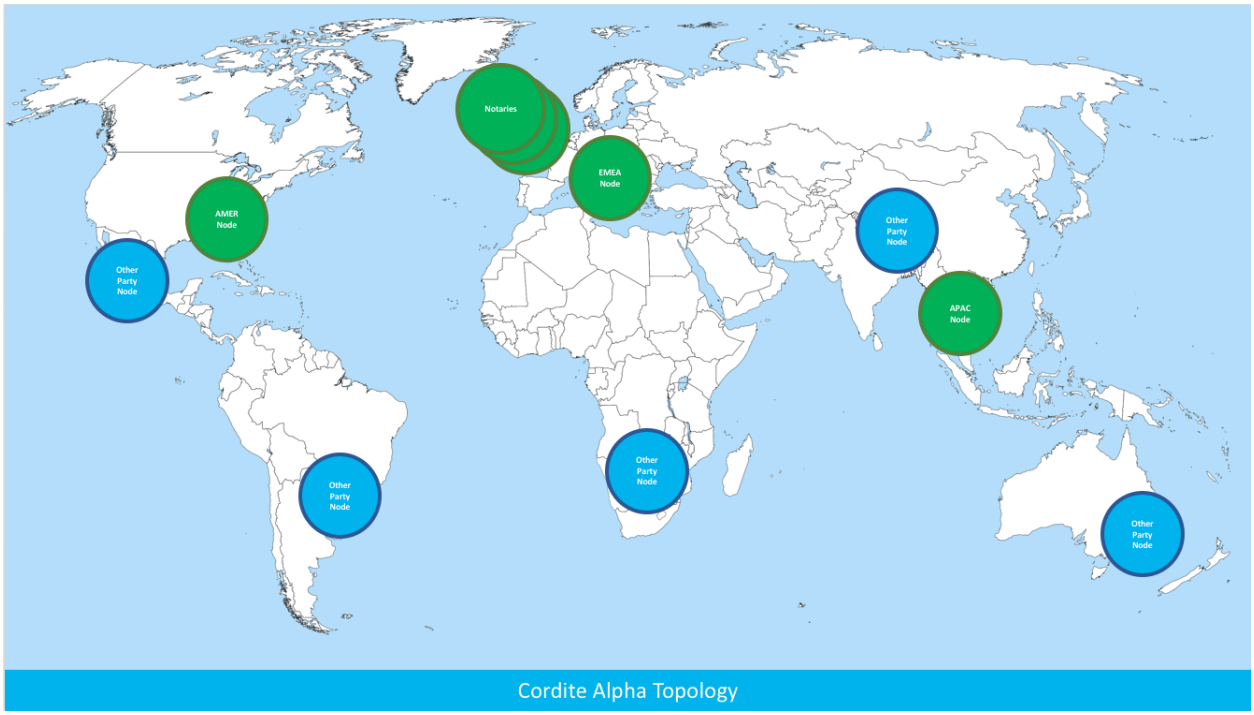


Fig. 1: Cordite World Picture

There is also https://network-map-test.cordite.foundation which provides a UI to view the network.

Edge network has the most bleeding edge version of Cordite deployed. This is the latest successful build of master in the Cordite repo. It should be seen as an unstable environment. Data is persisted but can be removed at any time without warning.

Test network is the latest release of Cordite deployed. This is the latest version to be pushed to the public registries e.g. DockerHub. It should be seen as a stable environment and a pre-cursor to the main network which will follow. Data is persisted and should remain. No guarantees are made.

The aim of the test network is to allow people to: - *Connect* to cordite - Build decentralised applications using cordite - *Build DAOs* - Add your own node

3.7 Connection details

Node name (endpoint hyperlink)	Node location	Party name
amer	eastus	O=Cordite AMER, OU=Cordite Foundation, L=New York City, C=US
apac	southeast asia	O=Cordite APAC, OU=Cordite Foundation, L=Singapore, C=SG
emea	westeurope	O=Cordite EMEA, OU=Cordite Foundation, L=London, C=GB

3.8 Cordite Entities

Node name	Node location	Party name
Cordite Metering Notary	westeurope	O=Cordite Metering Notary, OU=Cordite Foundation, L=London,C=GB
Cordite Committee	westeurope	O=Cordite Committee, OU=Cordite Foundation, L=London,C=GB

CHAPTER 4

Contributing

4.1 Code of Conduct

4.1.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

4.1.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.1.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

4.1.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

4.1.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at community@cordite.foundation. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

4.1.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

4.2 Issue Management

4.2.1 Labels

The issues backlog is actively managed. All issues are labeled for `importance` using MSCW method and `urgency` using AQL method. Please use the MUST, SHOULD, COULD, WONT labels to define important and CRITICAL, MAJOR, MINOR for urgency on all issues raised using the label defintions.

4.2.2 Milestones

The next two milestones are open. Issues that are being actively worked on and expected to be delivered in time for the milestone are added to the milestone. Adding issues to milestones does not make them actively worked on. Only issues that are actively worked tend to make it into milestones!

4.2.3 Tagging / Release

Please do not use tags. Tagging is reserved for releases.

4.2.4 Continuous Integration / Continuous Deployment

We follow trunk based development. All successfull builds of master are released to Maven Central (as a snapshot), DockerHub (tagged EDGE) and the Cordite EDGE network is upgraded.

4.2.5 Releases

Master is released every two weeks as a new tagged release to the public binary stores (maven central, dockerhub, gitlab artefacts) and the Cordite Test upgraded. Maintainers follow Semantic Versioning when tagging the code and creating new releases.

4.3 Legal

All contributions to this project must be certified that:

1. The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or
2. The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or
3. The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.
4. I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

4.4 Licence

4.5 Getting in touch

- News is announced on @We_are_Cordite
- More information can be found on Cordite website
- We use #cordite channel on Corda slack
- We informally meet at the Corda London meetup
- email community@cordite.foundation

4.6 Troubleshooting

If you encounter any issues whilst using Cordite, and your issue is not documented, please raise an issue.

4.7 Contributing

We welcome contributions both technical and non-technical with open arms! There's a lot of work to do here, and we're especially concerned with ensuring the longevity and reliability of the project.

Please take a look at our issue backlog if you are unsure where to start with getting involved!

4.8 Repository Access

While we are in private alpha, please go to the slack channel (#cordite channel on Corda slack) to request access to the repository

4.9 Releases

The first release, v0.1.0, was tagged through gitlab, on a version of the code that we saw pass the pipeline. Unless mentioned otherwise the process will continue the same and we will be releasing every two weeks. Currently we will not really follow SEMVAR specification util the api settles down. Once it is, we will release v1.0.0 and follow SEMVAR from there onwards.

CHAPTER 5

Frequently Asked Questions

5.1 Can I run multiple nodes on a single machine?

Yes you can. You need to be aware of the ports you are using for each node on the same host to ensure none clash. For all new ports assigned to each node, you will need to update the firewall accordingly.

5.2 How do I build from source?

When building from source we recommend the following setup: * Unix OS (ie OS X) * Docker - minimum supported version 18.03.0 * NPM - minimum supported version 5.6.0 * Oracle JDK 8 JVM - minimum supported version 8u131

For those wishing to build Cordite from source run `./build.sh`. (NOTE: this script is not to designed to be run on Windows.) Cordite node is laid out in `./node` and gradle builds in `./cordapps`. To start node after the build run `(cd node && start.sh)`.

5.3 How do I connect to a Cordite node?

We have created a few test nodes: + https://amer-test.cordite.foundation:8080 + https://apac-test.cordite.foundation:8080 + https://emea-test.cordite.foundation:8080

These are available as public nodes hosted in East US, SE Asia and W Europe locations respectively. You can use the REST endpoint `/api` or use the following clients: + example node client + Interactive console

5.4 How do I deploy my own Cordite node?

We use Docker to deploy Cordite nodes. For instructions on starting up a Cordite node using Docker, please visit our Docker Hub Repo for more information on configuration including environment variables:

```
$ docker run -p 8080:8080 -p 10002:10002 cordite/cordite
```

Once the node is running, you will be able to see the REST API for accessing Cordite at https://localhost:8080/api

If you do not wish to use Docker then alternatively you can download the latest Cordite node and run without docker by running `./start.sh`. You will need Oracle JRE 8 JVM - minimum supported version 8u131.

5.4.1 Environment Variables

Cordite uses environment variables to configure Cordite and create amoungst other things the Corda node.conf file.

Env Name	Description	Default
CORDITE_LEGAL_NAME	The name of the node	`O=Cordite-XXX, OU=Cordite, L=London, C=GB`
CORDITE_P2P_ADDRESS	The address other nodes will use to speak to your node	`localhost:10002`
CORDITE_COMPATIBILITY_ZONE_URL	The address of the Network Map Service.	‘‘https://network-map-test.cordite.foundation‘‘
CORDITE_KEY_STORE_PASSWORD	Keystore password	`cordacadevpass`
CORDITE_TRUST_STORE_PASSWORD	Truststore password	`trustpass`
CORDITE_DB_USER	Username for db	`sa`
CORDITE_DB_PASS	Password for db	`dnpass`
CORDITE_DB_DIR	Path to db directory - only used for H2	`/opt/cordite/db/`
CORDITE_DB_URL	database JDBC URI	`<default cord a url>`
CORDITE_DB_DRIVER	driver class name for database access - this image comes preconfigured with postgres 42.2.5 - you can use `org.postgresql.ds.PGSimpleData Source` to enable it	‘‘org.h2.jdbcx. JdbcDataSource‘ ‘
CORDITE_BRAID_PORT	Braid port	`8080`
CORDITE_DEV_MODE	Start up node in dev mode	`true`
CORDITE_DETECT_IP	Allow node to auto detect external visible IP	`false`
CORDITE_TLS_CERT_PATH	Path to TLS certificate	`null`
CORDITE_TLS_KEY_PATH	Path to TLS Key	`null`
CORDITE_NOTARY	Set to true to be a validating notary, false for non-validating or do not set to be a notary	`null`
CORDITE_METERING_CONFIG	JSON to set metering notary configuration	`null`
CORDITE_FEE_DISPERSAL_CONFIG	JSON to set metering fee dispersal config	`null`

5.4.2 Volume Mounts

File/Folder	Description
/opt/cordite/node.conf	Configuration file detailing specifics of your node - will be created using env variables if a node.conf is not mounted
/opt/cordite/db	Location of the database for that node - for persistence, use volume mount
/opt/cordite/certificates	Location of the nodes' corda certificates - will be created if no certificates are mounted to node and devmode=true
/opt/cordite/tls-certificates	Location of TLS certificates - will use self-signed certificates if none are mounted to node

5.5 How do I use Cordite in my own project

The core of Cordite is a collection of CorDapps which are java libraries. These are releasd to Maven Central and can be used in your project

Cordite provides de-centralised governance and economic services to mutually distrusting corporations wishing to solve distributed challenges in their industries.

Cordite is different things to different people. You have to start with the aims of Cordite to understand why:

- Wide spread adoption of https://corda.net
- Build awareness and understanding of the benefits of Corda
- Encourage use of Corda by demonstrating how it will help improve your business
- Support thought leaders in using Corda

Firstly it is a community of contributors who believe in these aims and hang out on the #cordite channel on Corda slack - https://cordaledger.slack.com/messages/cordite/

Secondly it is an established public Corda network which you can join - https://network-map-test.cordite.foundation/

Thirdly it is packaged into a simple Docker image so you can run a Corda node easily with the Cordite CordApps pre-packaged - https://hub.docker.com/r/cordite/cordite/

Fourth it is a set of CordApps which you can use with other CordApps to provide tokens, digital mutuals and metering in your own eco-systems - https://search.maven.org/search?q=g:io.cordite

We. Are. Cordite. https://cordite.foundation/ @we_are_cordite Built on Corda, Cordite is open source, regulatory friendly, enterprise ready and finance grade.

You are warmly welcome to contribute by improving these docs or anyway you can - <content/contributing>