corda

CORDA CHEAT SHEET

Useful links:

Documentation:	docs.corda.net
Slack:	slack.corda.net
Forum:	discourse.corda.net

INSTALLING CORDA

a. Download latest JDK

http://www.oracle.com/technetwork/java/javase/downloads

b. Clone template app

```
git clone https://github.com/corda/cordapp-template
```

c. Check out latest milestone (e.g. M10)

```
cd cordapp-template && git checkout release-M10
```

RUNNING CORDA

a. Deploy the nodes

```
./gradlew clean deployNodes
```

b. Run the nodes

```
sh kotlin-source/build/nodes/runnodes
```

WRITING CORDAPPS

a. Subclass `CordaPluginRegistry`

```
class MyPlugin : CordaPluginRegistry() {…}
```

b. Register the fully qualified class name of the plugin

…under src/main/resources/META-INF.services

STATES

`ContractState`

The base class for on-ledger states

- **`.contract`** — The `Contract` governing this state's evolution
- **`.participants`** — The parties able to consume this state

`LinearState` (extends `ContractState`)

State representing a 'shared fact' evolving over time

- **`.linearId`** — An ID shared by all evolutions of the 'shared fact'
- **`.isRelevant(Set<PublicKey>)`** — Should our vault track this state?

`OwnableState` (extends `ContractState`)

State representing fungible assets (cash, oil…)

- **`.owner`** — The state's current owner

CONTRACTS

`Contract`

Establishes whether a transaction is valid

- **`.verify(TransactionForContract)`** — Throws an exception if the transaction is invalid
- **`.legalContractReference`** — A hash of the contract's legal prose

TRANSACTIONS

`TransactionType.General.TransactionBuilder`

A mutable container for building a general transaction

- **`.withItems(vararg Any)`** — Adds items (states, commands…) to the builder
- **`.signWith(KeyPair)`** — Adds a digital signature to the builder
- **`.toWireTransaction()/.toSignedTransaction(Boolean)`** — Converts the builder to a wire/signed transaction

`WireTransaction`

An immutable transaction

- **`.toLedgerTransaction(ServicesForResolution)`** — Converts the transaction to a ledger transaction

`SignedTransaction`

A wire transaction, plus associated digital signatures

- **`.signWithECDSA(KeyPair)`** — Generate a digital signature over the transaction
- **`.withAdditionalSignature(DigitalSignature.WithKey)`** — Add a digital signature to the transaction
- **`.verifySignatures(vararg CompositeKey)`** — Verify the transaction's signatures

`LedgerTransaction`

A transaction that is checkable for contract validity

- **`.verify()`** — Checks transaction validity based on contracts

FLOWS

`FlowLogic`

The actions executed by one side of a flow

- **`.call()`** — Defines the flow-logic's actions
- **`.send(Party, Any)/receive(Party)/sendAndReceive(Party, Any)`** — Sends data to/receives data from the specified counterparty
- **`.subFlow(FlowLogic<R>, Boolean)`** — Invokes a sub-flow that may return a result
- **`.serviceHub`** — Provides access to the node's services

SERVICE HUB

- **`.networkMapCache`** — Provides info on other nodes on the network (e.g. notaries…)
- **`.vaultService`** — Stores the node's current and historic states
- **`.storageService`** — Stores additional info such as transactions and attachments
- **`.keyManagementService`** — Manages the node's digital signing keys
- **`.myInfo`** — Other information about the node
- **`.clock`** — Provides access to the node's internal time and date
- **`.schedulerService`**
- **`.transactionVerifierService`**