



CORDA CHEAT SHEET

Useful links:

Website: corda.net
GitHub org.: github.com/corda
Documentation: docs.corda.net
Slack: slack.corda.net
Stack Overflow: stackoverflow.com/questions/tagged/corda

RUNNING CORDA

a. Set up your dev environment

<https://docs.corda.net/getting-set-up.html>

b. Clone the template app in Kotlin or Java

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

c. Check out the latest version (e.g. V4)

```
cd cordapp-template-kotlin && git checkout release-V2
```

d. Deploy the nodes

```
./gradlew clean deployNodes
```

e. Run the nodes

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

Windows: `call kotlin-source/build/nodes/runnodes.bat`

STATES

ContractState

The base class for on-ledger states

.participants

The parties for which this state is relevant

LinearState (extends ContractState)

State representing a 'shared fact' evolving over time

.linearId

An ID shared by all evolutions of the 'shared fact'

OwnableState (extends ContractState)

State representing fungible assets (cash, oil...)

.owner

The state's current owner

.withNewOwner(AbstractParty)

Creates a copy of the state with a new owner

CONTRACTS

Contract

Establishes which transactions are valid for a given state

.verify(LedgerTransaction)

Throws an exception if the transaction is invalid

TRANSACTIONS

TransactionBuilder

A mutable container for building a general transaction

.withItems(vararg Any)

Adds items (states, commands...) to the builder

ServiceHub.signInitialTransaction(TransactionBuilder)

Converts the builder to a signed transaction

TRANSACTIONS (CONT.)

SignedTransaction

An immutable transaction plus its associated digital signatures

.verifyRequiredSignatures()

Verify all the transaction's required signatures

.verifySignaturesExcept(vararg List<PublicKey>)

Verify all the transaction's required signatures except those listed

.verify(ServiceHub, Boolean)

Verify the transaction

.toLedgerTransaction(ServiceHub, Boolean)

Resolve transaction into a LedgerTransaction for extra verification

ServiceHub.addSignature(SignedTransaction)

Add a digital signature to the transaction

FLOWS

FlowLogic

The actions executed by one side of a flow

.initiateFlow(Party)

Initiates communication between two flows

FlowSession.send(Party, Any)/FlowSession.receive(Party)

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

FLOW ANNOTATIONS

@InitiatingFlow

A flow that is started directly

@InitiatedBy(KClass)

A flow that is only started by a message from an InitiatingFlow

@StartableByRPC

Allows the flow to be started via RPC by the node's owner

SERVICE HUB

.networkMapCache

Provides info on other nodes on the network (e.g. notaries...)

.vaultService

Stores the node's current and historic states

.validatedTransactions

Stores all the transactions seen by the node

.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

PROVIDING AN API

a. Subclass WebServerPluginRegistry

```
class MyWebPlugin : WebServerPluginRegistry() {...}
```

b. Override webApis

```
override val webApis = listOf(Function(::MyApi))
```

c. Register the fully qualified class name of the plugin

...under `src/main/resources/META-INF/services/WebPluginRegistry`