Advanced Corda

Corda Token SDK

Course Agenda

Tokens

The Token SDK

A Tokenization Platform

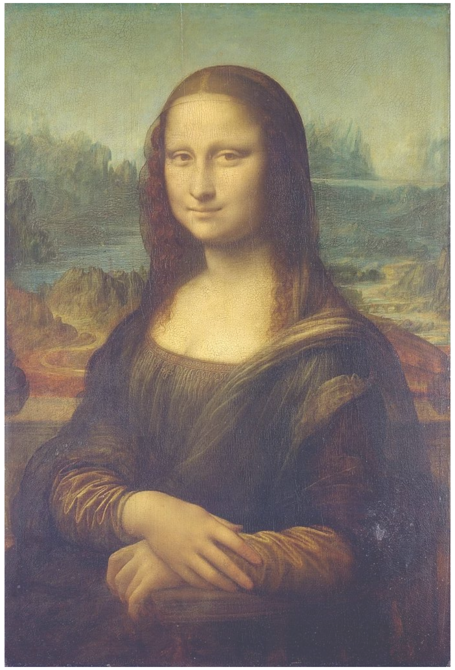
Design

Implementation in CorDapps

Tokens

- Can create new markets for previously illiquid assets
- Reduces risk and cost in post-trade systems
- Enabling end-to-end solutions that combine trading, settlement, and custody services.
- Tokenization allows for fractional ownership of assets
- Opens liquid and illiquid assets to a wider investor base
- Enables the creation of new financial products through securitization of asset-backed cash flows

Example: Fine Art



- Artwork can be difficult to sell
    - Have to sell at auction
    - Hard to gauge price

- Buy tokenized percentages of a piece of artwork
    - stakes in the piece are traded on an exchange like any other asset
    - Able to gauge real-time prices by looking at the tokenized art market

The Token SDK is a standardized developer toolkit that establishes a consistent developer experience for end-to-end management of digital assets.

The Token SDK

- Standard Library
- “Issue-List-Exchange-Settle” Workflow
- Replaces the built-in “finance” CorDapp for using tokens on Corda.

- Corda natively supports the identity, privacy, scalability and finality requirements of the digital assets markets
- Secure
- Scalable
- All-in-one solution

CorDapps

Contract
s

Workflo
ws

Money

The SDK builds on the functionality from the finance package by providing a common way to

Issue tokens

Define types of tokens

Use and trade tokens

Corda as a Tokenization Platform

Secure Cryptographic Assets

- Privacy and confidentiality built-in
- Regulatory friendliness
- Integration with existing financial systems
- 51% attacks don’t exist on Corda
- No global chain that can be split

Speed and Scalability

- DTCC and Accenture study showed that Corda can handle the US Equity Markets of **6,300 trades per second.**
- **Optimally sharded**, i.e. nodes only process data related to them.
- Network level increases **linearly** as you add nodes.
- **UTXO** model

All-in-one Solution for Digital Assets

- Corda combines *trading*, *settlement*, and *custody* services into one platform.
- Settlement of tokens into fiat currency through legacy payment rails is possible with **Corda Settler**
- **Digital asset exchanges** are being created using the Token SDK to represent debt, equity and cash instruments on Corda

Using Token SDK

Start with the training template

“**tokens-template**” branch of **corda-template-kotlin** repository

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

cd cordapp-template-kotlin

git checkout token-template

./gradlew clean deployNodes

./build/nodes/runnodes
```

Add Token SDK to an existing CorDapp

Add to build.gradle:

```
buildscript {
   ext {
      tokens_release_version = '1.0'
      tokens_release_group =
'com.r3.corda.lib.tokens'
   }
}
repositories {
   maven { url
'https://ci-artifactory.corda.r3cev.com/artifactory
/corda-lib' }
   maven { url
'https://ci-artifactory.corda.r3cev.com/artifactory
/corda-lib-dev' }
}
dependencies {
       …
       cordaCompile
"$tokens_release_group:tokens-contracts:$toke
ns_release_version“
       cordaCompile
"$tokens_release_group:tokens-money:$tokens_
release_version"
}
```

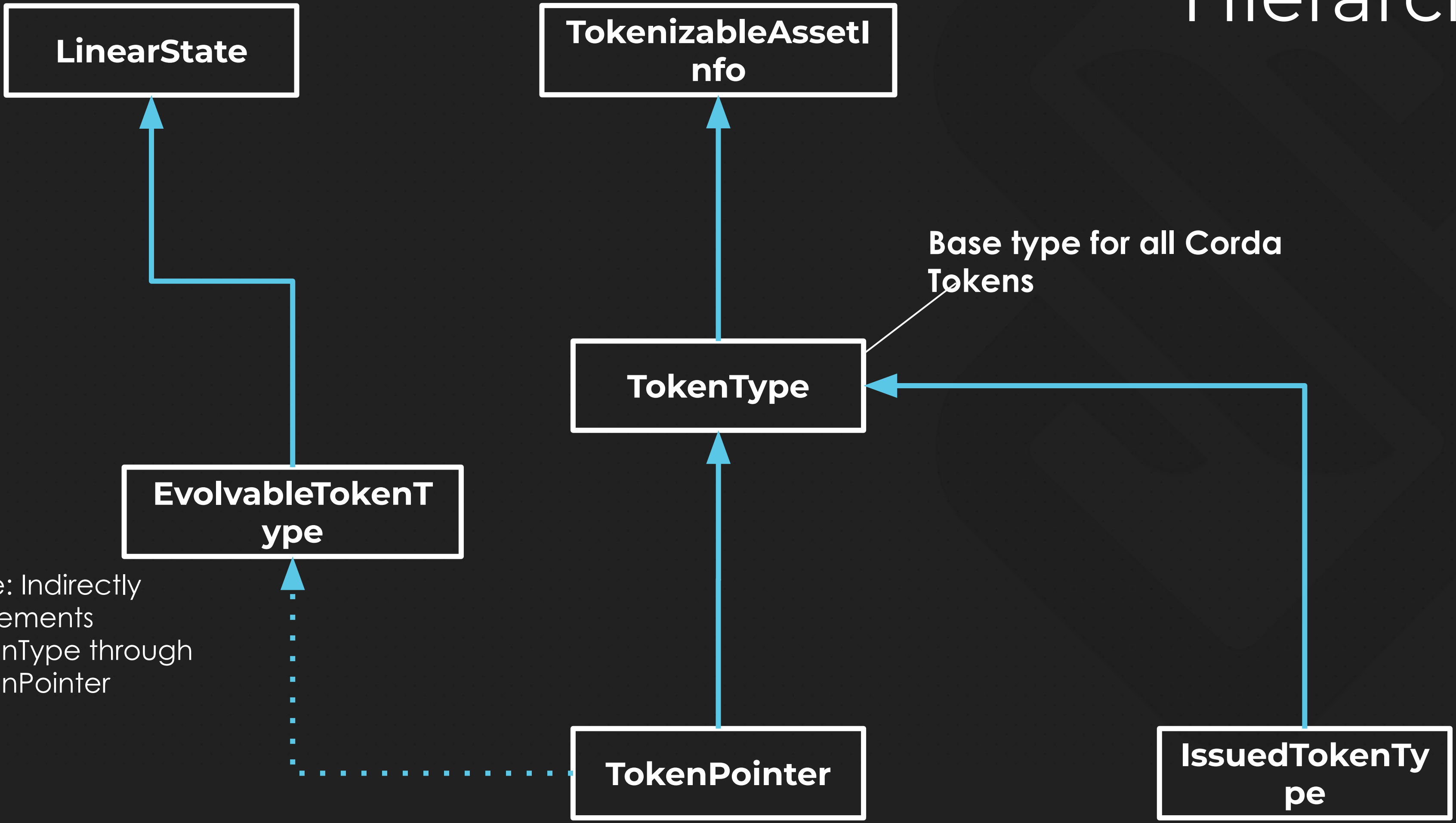
See:
https://github.com/corda/token-sdk for detailed instructions.

Types of Tokens

There are two types of tokens in the Token SDK.

- Fixed tokens - represented by TokenType
  - Do not change over time
  - Example: USD, GBP
- Evolvable tokens - represented by EvolvableTokenType
  - Expected to change over time
  - Extension of the LinearState interface
  - A TokenPointer is used to point the token state to the LinearState containing the token information.

Token Class Hierarchy



LinearState

TokenizableAssetInfo

TokenType

Base type for all Corda Tokens

EvolvableTokenType

TokenPointer

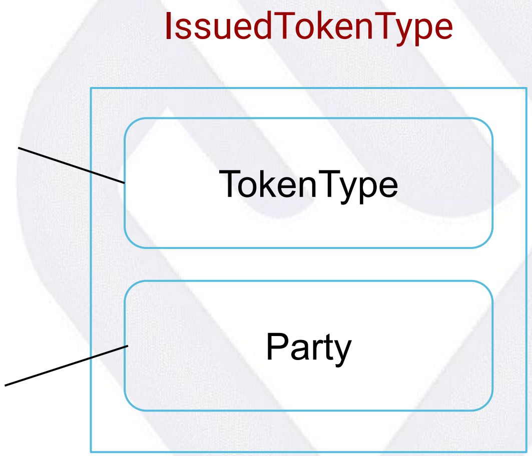
IssuedTokenType

Note: Indirectly implements TokenType through TokenPointer

Note: Uses a *LinearPointer* which is resolved into data during verify()

IssuedTokenType

- An IssuedTokenType is a wrapper class containing a:
  - TokenType
  - reference to an issuing Party



This could also be a **TokenPointer**

Party object referencing node on network.

Fixed Token Types

- All fixed token types must implement the TokenType interface.

- Two pieces of information are required:
  - Token identifier
  - Fractional amount allowed for token

```
val fixedToken = new

ExampleFixedToken("CUSTOMTOKEN", 2);
```

- Here, with fractional amount 2 we can create tokens like: 100.51

Issue Fungible Tokens

Create fungible fixed token

- Creating and issuing fungible tokens is very similar to creating non-fungible tokens.
- The primary difference is the inclusion of an “amount” property and exclusion of the “linearId” on the FungibleToken to allow splitting and merging.

```
val token =
    ExampleFixedToken("CUSTOMTOKEN",
2);

val issuedTokenType =
    IssuedTokenType(issuer, token);

val fungibleToken =
    FungibleToken(
        Amount(10000,
    issuedTokenType),
        recipient,
        /* Jar attachment Secure Hash
    */
    );

subFlow(IssueTokens(fungibleToken));
```

Use Case

Delivery Versus Payment

- Applying what we learned, we'll look at a simple delivery versus payment example.

- Let's represent the swapping of two different types of tokens, one fungible and one non-fungible:
  - "House" token
    - Non-fungible digital asset
  - GBP money tokens
    - Fungible currency asset

Fixed Token Types

Creating our own fixed token type:

```
data class
ExampleFixedToken(
      override val
tokenIdentifier: String,
      override val
fractionDigits: Int = 0
) : TokenType(tokenIdentifier,
fractionDigits)
```

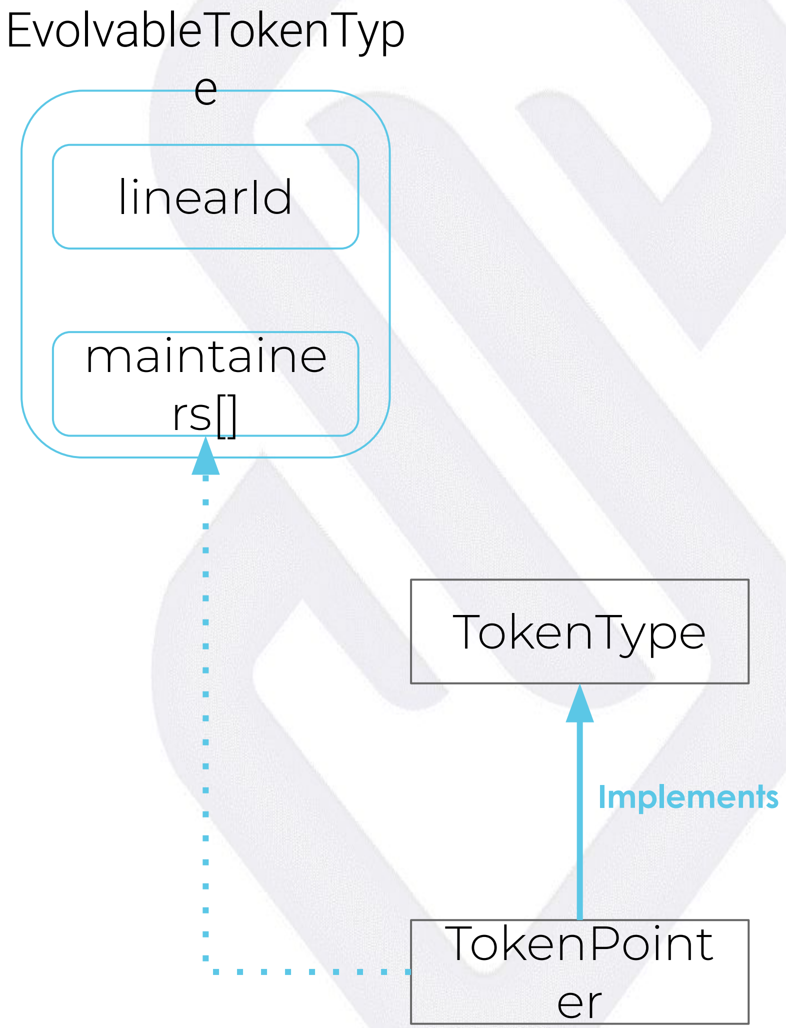
Evolvable Token Types

- A class implementing an evolvable token must extend the **EvolvableTokenType** class.

- EvolvableTokenType extends **LinearState** hence we have a *linearId* to keep track of the changes to the state over time.

- We also have a set of *maintainers* who would be informed on any state update.

- CreateEvolveableTokenFlow



Evolvable Token Types

Creating our own evolvable token type:

```
data class
ExampleEvolvableToken(
    override val maintainers:
List<Party>,
    override val fractionDigits: Int,
    val exampleDataProperty:
String,
            override val linearId:
        UniqueIdentifier =
        UniqueIdentifier()
) : EvolvableTokenType()
```

Issuing Tokens

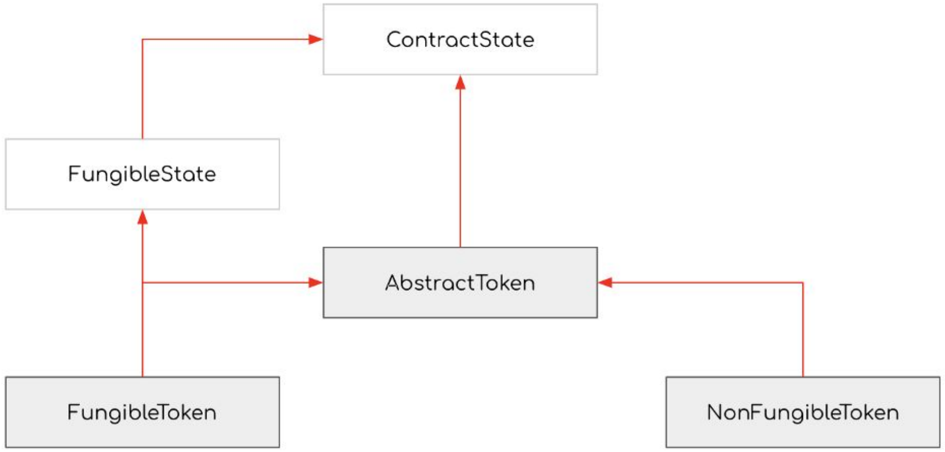
- **TokenType** objects are just that -- *objects*. In order to apply them we need to **ISSUE** them onto the Corda ledger as **ContractStates**.

- There are two high level types of States for representing tokens:
  - Fungible - can be split and merged
    - ex. USD, GBP, stocks, bonds
  - Non-fungible - cannot be split or merged
    - ex. loans, deeds

- **A fungible token is not unique,** a different amount of it can be owned by multiple holders.

Fungible and Nonfungible Tokens

- Assets are represented on the ledger as either FungibleTokens or NonFungibleTokens.
- Both interfaces extend the top-level ContractState interface.
- FungibleTokens extend FungibleState.
- NonfungibleTokens extend AbstractToken.



Fungibility versus Evolvability

- In review, Tokens can be:
  - Fungible or Non-fungible
  - *Fixed* or *Evolvable*

	Fixed	**Evolvable**
Fungible	Money	Stock, security
Non-fungible	Digital asset	Deeds and titles

Issue Non-Fungible Tokens

- Issuing tokens works as you would expect:
  - First we create the token objects.
    - Create a IssuedTokenType object
    - Create a NonfungibleToken object
  - Then we “Issue” them onto the ledger by using a transaction and collecting signatures.
    - IssueTokensFlow

Issue Non-Fungible Tokens

Create non-fungible fixed token

- To create a non-fungible token we first need to create an IssuedTokenType object with an issuing party reference.

- Create IssuedTokenType for Fixed token:

```
val token = ExampleFixedToken("CUSTOMTOKEN", 2);
val issuedFixedToken =
        IssuedTokenType(issuer, token);
```

Reference to **Party** object - node that is serving at the issuance authority on this token type.

Reference to fixed token object.

Issue Non-Fungible Tokens

Create non-fungible evolvable token

- To create a non-fungible token we first need to create an IssuedTokenType object with an issuing party reference.

- Create IssuedTokenType for Evolvable token:

```
val token = ExampleEvolvableToken(...);
val linearPointer = LinearPointer(
        token.linearId, ExampleEvolvableToken::class.java
);
val tokenPointer = TokenPointer(linearPointer,
token.fractionDigits);
val issuedToken = IssuedTokenType(issuer, tokenPointer)
```

Issue Non-Fungible Tokens

Issue non-fungible fixed token

- Then, we can Issue the tokens on the leger using the IssueTokensFlow.

- Issue the IssuedTokeType for our Fixed non-fungible token:

```
val nonFungibleToken = NonFungibleToken(
    exampleFixedToken, recipient, /* omitting .. */);
subFlow(IssueTokens(listOf(nonFungibleToken)));
```

- Issue the IssuedTokeType for our Evolving non-fungible token:

```
val nonFungibleToken = NonFungibleToken(
    exampleEvolvableToken, recipient, /* omitting ..
*/);
subFlow(IssueTokens(listOf(nonFungibleToken)))
```

NonFungibleToken data type

NonFungibleToken takes 4 parameters:

- the IssuedTokenType object
- the recipient Party
- linearId of our EvolvableTokenType
- SecureHash of the jar which implements the TokenType

```
NonFungibleToken(
        issuedTokenType,
        recipient,
        UniqueIdentifier.Companion.fromString(UUID.randomUUID().toString()),
        tokenPointer.getAttachmentIdForGenericParam()
);
```

Use Case

Delivery Versus Payment - Issue tokens onto ledger

- Issue GBP token onto ledger - Fixed Fungible Token

```
subFlow(IssueTokens(100.GBP issuedBy issuerParty heldBy
ownerParty));
```

From “Money” module in Token SDK

Helper methods from TokenSDK Utilities

- Issue House token onto ledger - Evolvable Non-Fungible Tokens

```
val house: House = House(“100 Maple Lane”, …)
val housePtr = house.toPointer<House>()
subFlow(IssueTokens(housePtr issuedBy issuerParty heldBy
ownerParty));
```

Use Case

Delivery Versus Payment - Create flow

Create flow to “Move” the GBP and House tokens.

- ○ use subflows or utility helper methods

```
@StartableByRPC
@InitiatingFlow
class SellHouseFlow(val house: House, val
newHolder: Party) : F
lowLogic<SignedTransaction>() {
@Suspendable
override fun call(): SignedTransaction {
    txBuilder = TransactionBuilder(notary
=
ServiceHub.networkMapCache.notaryIdentitie
s(0))

    addMoveTokens(txBuilder,
house.toPointer<House>(), newHolder)
    addMoveFungibleTokens(
        txBuilder, serviceHub, 100.GBP,
getOutIdentity(), newHolder, /* optional
query criteria */
    )
    // .. Collect signature and finalize
```

Use Case

Delivery Versus Payment - Finalization

For Evolvable states, we need to notify all parties on the distribution list

```
// Update distribution list.
subFlow(UpdateDistributionListFlow(stx))
```

Finalize transaction with optional observers

```
return subFlow(ObserverAwareFinalityFlow(
    stx, listOf(observerSessions))
)
```

Subflows Versus Utility Methods

- We can Issue, Move, and Redeem tokens
- For each operation there are built-in flows:
    - IssueTokensFlow
    - MoveTokensFlow
    - RedeemTokensFlow
- And utility methods:
    - addIssueTokens(..)
    - addMoveTokens(..)
    - addRedeemTokens(..)
- Built-in flows include finalization, while utility methods are used when we want to do multiple operations in an atomic transaction

Token SDK Recap

- Tokens create new markets for previously illiquid assets and reduce risks for trading among other benefits.
- Corda is a ideal platform for tokenization due to its scalability and privacy concerns.
- Tokens can be Fungible or Non-fungible
- Tokens can be Fixed or Evolvable
- We can use built-in flows or helper methods to Issue, Move, and Redeem tokens.