**Rialto- KoreConX Integration – Main functional concepts**

**Index**

1-Roles, 1

2-Trading, 2

3-Onbarding, 6

4-Settlement, 14

Introduction, 1

**Introduction**

The purpose of this document is to describe all the technical, functional and nonfunctional features involved in the ***Rialto-KoreConX*** integration. It is dedicated to non-technical users, and it expects to have in an easy and straight language a full description about everything that has been learned during the integration (and everything that´s yet to be learned)

This document does not pretend to be technical, but a few technical concepts will be described deep enough so that a non-technical users can understand the main concepts that have to be known, but without digging into too much detail that will be beyond the scope of this document.

It is worth to mention that throughout this document you might find some sentences in red color. This sentences are written in this color as a convention to indicate that at the time of writing this document, further research had to be taken in that given subject.

**1- Roles**

The role of every company in this project is oriented to integrate ***Rialto*** **Alternative Trading System** (ATS) with ***KoreConX*** financial records which will allow users to trade, settle and track private equity portfolios like had always been done in public regulated markets.

In this context, throughout the whole document, we pretend to describe all the functional and non-functional concepts that the users has to get familiar in order to understand how the interaction between KoreConX and Rialto works.

In this project Rialto ATS system pretends to give the user the usual experience it was used to have when trading stocks in the public markets. Usually in the public markets, users deal with what is called a CLOB: A Central Limit Order Book and they buy and sell securities issuing orders will become a trade as soon as they find an opposite counterparty in this CLOB.

Rialto ATS system pretends to give the user the same experience. **There will be some differences with the traditional CLOB exchanges** in public markets, but basically a given shareholder will have access to securities and it will be able to issue orders to buy and sell those securities.

On the other hand, every time a trade takes place, a given amount of fiat currency will be transferred from one bank account to another and a given number of shares will change ownership in a given transfer agent. It is in this context that a lot of communication has to take place between the following parties:

* The ATS System (Rialto)
* The Transfer Agent (KoreConX)
* The Bank accounts ( managed through Plaid platform)

And some of the following question will have to be answered throughout this document and the whole project

* What does the Transfer Agent need to know every time a new order is sent to an exchange?
* How will the securities onboarding be made? What will identify a given security in both systems? Will there be a security that could be traded in multiple transfer agents?
* How will the different shareholders onboarding be made? How will KYC procedures be conducted? What will identify a legal entity in both systems? And what about a physical person?
* How will the settlement be made? Do you have to analyze some sort of default protocol? How will ATS system handle data inconsistency situations?

**2- Trading**

Whenever trading, the responsibilities of an ATS lie mostly in avoiding the shareholders to take actions that might end up in some sort of non-compliance.

At the time of writing this document, this will be mostly related to validating that there is enough funds or shares for a given action.

**Funds validation**

It is worth to mention that the balance validation before sending buy orders to the exchange is done offline. This means that it will have to be the ATS the one that synchronizes with the banks through the ***Plaid*** platform. Then, every time an order is sent to the exchange, the ATS has to validate that the order being created and all the open orders have an equal or lower notional than all available funds.

So, how is this synchronization going to be made? In the following checkpoints

* Whenever the Plaid token is received from Solidus
* Whenever the user´s log in to the platform

So, how is this synchronization implemented? Basically through the following fields:

* **Firm Limit:** A Kore Con X shareholder will be a firm marked as individual investor. So it will have only one user. In consequence, the firm limit will be the shareholder limit.
* **Buying Power**: Because of what was mentioned in the previous paragraph, this field will have the same effect as the ***Firm Limit*** field.
* **Trade Limit**: Limit per individual order. It will be ***Buying Power*** minus ***Pledged Funds*** (open order’s notional)

So, basically, if the shareholder has *10,000* USD in his ***Piermont*** account, if he has no open orders, a new order will have a maximum notional value of *10,000* USD. If he has open orders for *1,000* USD, he will only be able to create new orders for *9,000* USD and so on.

**<to add; When the login synchronization is implementd,talk about the Balance synchronization service>**

**Holdings validation**

When buying something, the validation consists of assuring that there is enough funds to pay for the underlying shares in case the new order is filled. There risks are big because the validation is made at very specific points in time giving quite a lot of time for rogue traders, to empty their bank accounts and force defaults.

When selling shares things are not that easy, at least when interacting with ***Kore Con X***. Here we face multiple levels of synchronizations, validations and freezes, that make much more difficult the possibility to implement any kind of fraud.

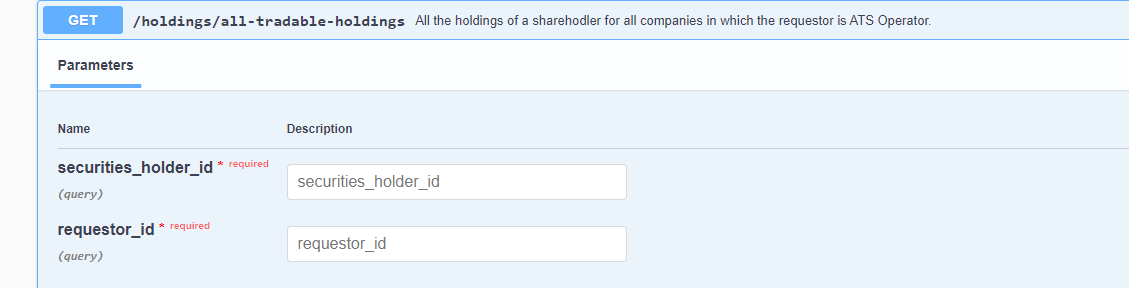
Basically the security when selling shares takes place in multiple layers.

Synchronization

A shareholder’s positions are synchronized with Kore Con X in the following checkpoints:

* Onboarding
* Logging in

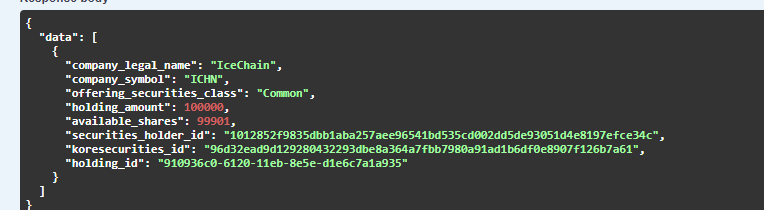
This synchronization is made calling to the following endpoint

****

Where

* **Securities\_holder\_id**: It is the Kore Shareholder Id of the shareholder we want to synchronize, that can be found in table **share\_holder**.
* **Requestor\_id:** It is the ATS kore chain id that can be found in table **transfer\_agent.**

The response will look as follows



The response from this service will be mapped to the positons table and it will be these values the ones used for validating sell orders.

Note that it will be the ***available\_shares*** the amount that will be considered for balance synchronization. The difference between ***available\_shares*** and ***holding\_amount*** is expected to be the securities put on hold and they will not be considered for synchronization.

Offline Validations

Once the portfolio is synchronized, there will be an offline synchronization whenever a trade is executed. This means, if by the time I log in, I am informed that I have 100 shares of a given security, if I sell 10 shares of that security, that holding will be updated to 90 shares. However, this is not safe enough, as holdings could be received or transferred outside of Rialto. So basically the holding validations will be done in two steps: one offline and one online.

So basically, you will not be able to send sell orders for a given quantity of shares that the ones that you have in your holdings screen.

Online Validations

The previous offline validations could be easily hacked if a holdings record was manually created. However, when selling a security that is managed by **Kore Con X**, there is a second validation that is implemented through the following service:

<http://3.210.205.44:3001/api/docs/#/KoreSecurities/get_holdings_available_shares>

Note: The previous IP address could have changed.

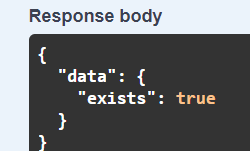
So basically, every time before sending a sell order of a ***Kore Con X*** security, the following parameters will be sent to the previous service:



Where

* securities\_holder\_id: Shareholder kore chain id (***kore shareholder id***)
* koresecurities\_id: Security kore chain id (***kore security id***)
* requestor\_id: ATS kore chain id.
* number\_of\_shares: number of shares to be sold

The following is how the response will look like.



Basically a very simple true or false, describing if the user has enough or not shares for the previous ***number\_of\_shares*** value.

Freezes

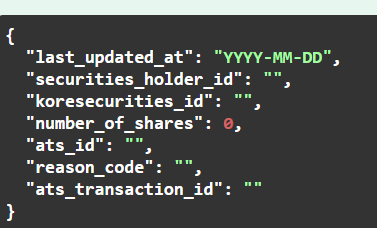
Having enough shares is not enough to be fully certain that unwanted situations will arise when clearing the trades. This means that the shareholder might transfer or sell the shares outside of Rialto ATS, just before the order has been sent. Given that he will receive the money before transferring the shares, it’s important to block/freeze the shares just after the order has been created.

So just after sending invoking the ***available-shares*** service, the ATS will call the following service:

<http://3.210.205.44:3001/api/docs/#/KoreSecurities/post_holdings_hold_shares>

Note: The previous IP address could have changed.

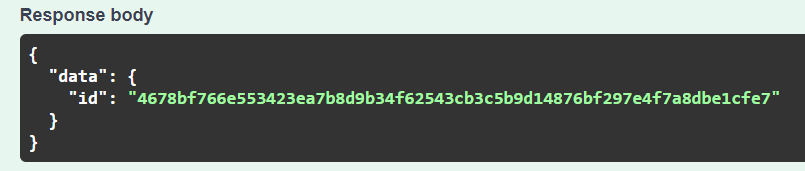
We will have to provide the following structure:



Where:

* *securities\_holder\_id*: Shareholder kore chain id (***kore shareholder id***)
* *koresecurities\_id*: Security kore chain id (***kore security id***)
* *ats\_id*: ATS kore chain id.
* *number\_of\_shares*: number of shares to be sold/put on hold
* *reason\_code*: any internal ATS id
* *ats\_transaction\_id*: any random transaction id identification. This will not be needed in this service and this might be removed in the future. However, it will be important when calling the release shares service, because we will have to provide the *transaction\_id* received when putting the shares on hold.

The answer will look like the following



This is will have to be provided when releasing the shares.

Release:

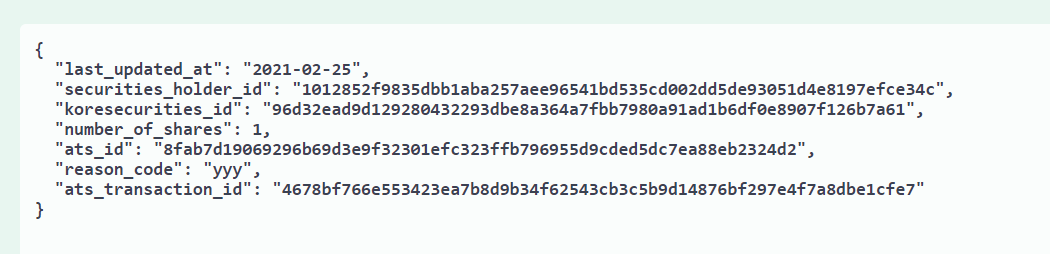
The frozen shares will be released when an order is cancelled or rejected. This means that if there is actually a trade, the shares will stay on hold until the settlement takes place.

So whenever rejecting or cancelling a new sell order, the ATS will invoke the following ***Kore Con X*** service:

<http://3.210.205.44:3001/api/docs/#/KoreSecurities/post_holdings_release_shares>

Note: The previous IP address could have changed.

We will have to provide the following structure, which is basically the same as the one provided when putting shares on hold, but the only difference that we will be providing the ***ats\_transaction\_id*** received when putting the shares on hold.



The response will look like the following:



We should also save the previous ***transaction\_id*** but there is not usage for it at the time of writing this document.

**3-Onboarding**

There will be basically two things that will be shared between ***Rialto*** and ***KoreConX***:

* Securities
* Shareholders

Entity identification - Securities

Securities can have different asset classes and be related to different companies and even the same company can have different securities of the same asset class. In this context, the worldwide identification number for securities is the CUSIP. A CUSIP is a nine-digit numeric (e.g. 037833100 for Apple) or nine-character alphanumeric (e.g. 38259P508 for Google) code that identifies a North American financial security for the purposes of facilitating clearing and settlement of trades. The CUSIP was adopted as an American National Standard under Accredited Standards X9.6.

In Rialto world it will be the CUSIP the id that will identify a security. This way, the ATS will be 100% aligned with any other system that has a given security registered, and a trading user will be able to use the security symbol, to create his orders.

But things get a little bit more complicated when interacting with ***KoreConX***, as its financial records are stored in a technology called ***Hyperledger***. This technology is an open

source ***blockchain*** (distributed ledger) where, as any other ledger, we will have owners, assets and exchanges. But the identification of the different entities in this technologies is not as friendly as it is in ***Rialto***, and every entity will have its own identification which will be called ***KoreChainId.***

So basically:

* **KoreChainId**, is the name that a given entity id will receive in ***KoreConX***´s Hyperledger. Weather it is an ATS Operator, a security , a shareholder (physical or legal) or a company, every entity will have an Id, and they are usually called ***KoreChainIds***

These Ids look all the same and it´s not possible to identify the entity that is being referenced by a given Id. It is basically a 64 bytes (characters) string that looks like the following and it will have to be the invoking app the one that knows which one is the entity that is being referenced behind that string:

91e6149d2fab8bf5ef1d48970456a342460c58b38a3de01e388fcd19858f12ff

Remember that in different parts of the documentation this id can be referred as ***Kore Security Id,*** to make clear that we are referring to a ***Kore Chain Id*** associated with a security.

Also remember that a given firm can issue different kinds of securities. For example they could issue shares (equity), bonds, etc. so in ***Kore Con X*** world, a given security under a given CUSIP will have basically 2 ids.

* A ***Kore Chain Id*** related to the security (also known as ***Kore Security Id***)
* A ***Kore Chain Id*** related to the firm (also known as ***Kore Company Id***)

In Rialto database structure this can be seen in table **security\_chain** where you will have field **chain\_id** and **company\_id** respectively.

Entity identification – Shareholders

As was mentioned in section “*Entity identification – Securities*” the shareholders will have their own ***Kore Chain Id*** which will look exactly the same as the one that refers to a security. Remember that in different parts of the documentation this id can be referred as ***Kore Shareholder Id,*** to make clear that we are referring to a ***Kore Chain Id*** associated with a shareholder.

Kore Con X - Automatically trigger the onboarding of a shareholder

As was mentioned before, shareholders will be onboarded automatically through the ***Kore Con X*** platform. This means that the initial point of contact between an investor and Rialto will be ***Kore Con X*** as its starting point.

In summary, ***Kore Con*** X will suggest one of his users (shareholders) the possibility to be part of Rialto platform. After accepting the suggestion, the user will be forwarded to new screens that will request the shareholder to load missing information that will be needed in the onboarding process.

After filling all the missing fields, ***Kore Con X*** will invoke a ***Rialto Onboarding Service*** that will be responsible of the decryptions, personal data requests, running all the proper validations and sending all these data to ***Solidus***.

So basically the onboarding steps are:

Onboarding request – input data – Encryption mechanisms

The encryption topic is always complex and interesting enough to be a topic of its own. Basically an encryption is the process of obfuscating certain information in a way that only a specific recipient can de obfuscate it and access its content.

In the encryption algorithm world there are 2 kinds of algorithms: symmetrical and asymmetrical. **Symmetric encryption** uses a single key that needs to be shared among the people who need to receive the message while **asymmetric encryption** uses a pair of public key **and** a private key to **encrypt and** decrypt messages when communicating.

Asymmetric encryption was introduced to complement the inherent problem of the need to share the key in symmetric encryption model, eliminating the need to share the key by using a pair of public-private keys.

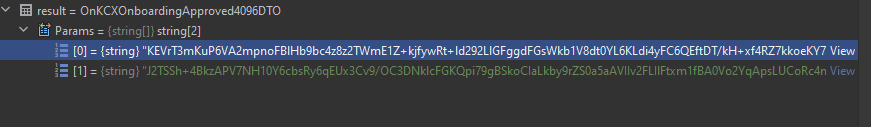
In the context of the interaction between ***Rialto*** and ***Kore Con X*** we will use a combination between both models. The simplest option would be using a ***symmetric algorithm,*** but there would be a *public key* that would have to be shared between both counterparties. Then that key could travel as part of the message, but we would have to send it encrypted and then we will need an ***asymmetrical algorithm*** to decrypt this message.

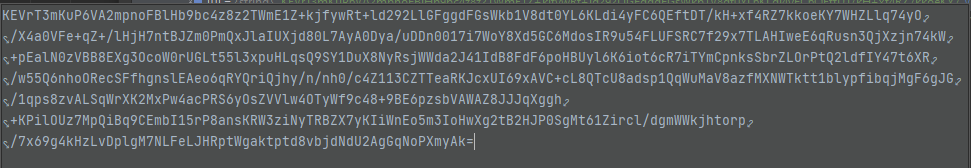
To solve the circularity, the following procedure was followed:

**RSA algorithm**

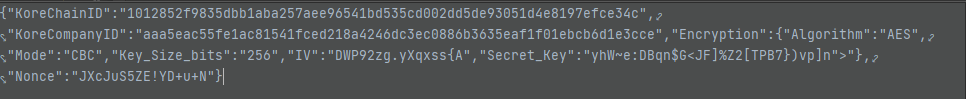
* The onboard request will consist of a Json message encrypted using the ***RSA algorithm*** (RSA/ECB/PKCS1Padding 4096 bits asymmetric).
* This message will be encrypted using the Rialto private key and decrypted using Rialto private key. **<TODO: include more info about the location of these public and private key>**

Given that it was chosen to use 4096 bit keys, the incoming message will arrive in an array that looks like follows:





Once decrypted every position in the ***Params*** array, they should be concatenated, and then we will be able to visualize onboarding Json:



In the previous Json we will have 4 parameters that are

* **KoreChainId**: The kore chain id of the shareholder to onboard
* **KoreCompanyId:** The kore chain id of the company that the shareholder belongs to. The shareholder could actually belong to multiple companies so this value is not going to be used except as a parameter to the service described in section “Onboarding – Personal Data download”
* **Key and IV:** This is the public symmetric key of the ***AES algorithm*** described as follows that will be used when decrypting the shareholder personal data.

**AES algorithm**

Having the ***key*** and ***IV*** parameters that were retrieved from the previous step (RSA) decryption, we are ready to download all the new shareholders personal information and decrypt it using the AES algorithm, using the previously mentioned **IV** and **Key** parameters.

The only thing worth to mention about this AES algorithm is that is a symmetric key, and that we will be using:

* Mode CBC
* Key: 256 bits
* IV: 128 bits
* Encoding: Base 64

These **Key** and ***IV*** are what we could call the algorithm public keys. Usually they can be stored in weather an xml file or what is known as a ***pem file***. Using the parameters in the ***.dot net*** core *System.Security.Cryptography* library was good enough. We don’t have to store or persist these keys in any kind of repository as they will be sent with every new shareholder to be onboarded.

Onboarding – Personal Data download

Once we have decrypted all the input parameters we have to download the shareholder personal data that will be stored in Kore Con X Kore Node. We will basically access this environment by a REST service using an URL like the following.

**http://<kore\_node\_ip\_port>/person/show/**

The Swagger that fully describes this service can be found in :

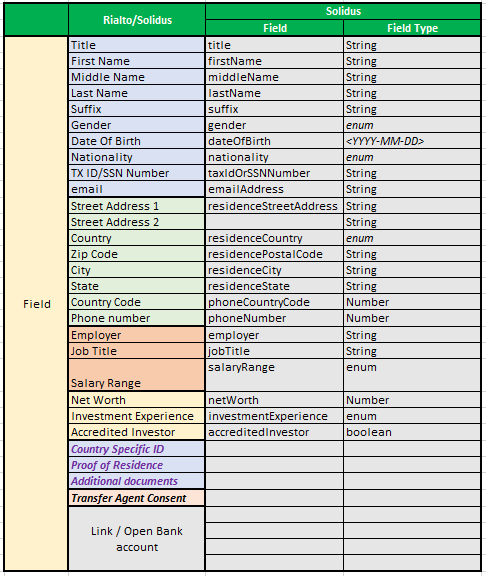
[**http://3.210.205.44:3001/api/docs/#/Person/get\_person\_show\_\_id\_**](http://3.210.205.44:3001/api/docs/#/Person/get_person_show__id_)

**Note: Beware that the previous IP address belongs to the new Rialto Kore Node and may change in future deployments.**

In this context, it is important to know:

* The REST service URL will have to be retrieved from the ***kcx\_connection\_setting*** table.
* When invoking the Kore Con X personal data service, we will have to provide:
  + The ***Kore Shareholder Id*** received (**id** field)
  + The ***Kore Company Id*** received (***company\_id*** field)
  + The ***Rialto Kore ATS Id*** , which can be found in table **transfer\_agents,** column ***transfer\_agent\_chain\_id*** (**requestor\_id** field)
    - **Beware that this table should be refactored in the near future to support more transfer agents!**

The following is a quick snapshot of all the fields that are downloaded from ***Kore Con X***. This field list was inspired in the all the information that ***Solidus*** established that will be needed to validate that a new shareholder can be onboarded in the ATS platform (***KYC validations***). This will change over time so it is a good practice to manually validate for new fields/removed fields. But this is basically also the field list that will be provided to every new transfer agent and their respective onboarding process will have to, somehow, provide all these fields.

****

Once all these fields have been downloaded we will have to prepare them to be sent to Solidus. At the time of writing this document, the Solidus test environment was not ready, so all that is going to be mentioned here will be based in known documentation. In the previous table, all the fields that are described as *Strings , Numbers or Dates* will be sent as they are received (beware the date format in the previous table). The only special fields are the enumerators, as they will have to be translated to the different options expected by Solidus.

These enumerators are

* Gender
  + MALE | FEMALE | NON\_BINARY | PREFER\_NOT\_TO\_IDENTIFY
* Nationality
  + [Alpha-2 country code from ISO-1366](https://www.iso.org/iso-3166-country-codes.html).
* Residence Country
  + [Alpha-2 country code from ISO-1366](https://www.iso.org/iso-3166-country-codes.html).
* Salary Range
  + 0\_TO\_50000 | 51000\_TO\_100000 |101000\_TO\_200000 | 200000\_AND\_UP
* Investment Experience
  + EXPERT | EXPERIENCED | BEGINNER

By the time of writing this document, the following fields were missing in ***Kore Con X*** response:

* phone number (mobile), net worth, salary , investment experience and accredited investor

Other missing fields that are not mandatory are

* Title, Suffix, phone country code

**<TODO: By the time Solidus test environment is ready, update the previous table and enums>**

Onboarding – Images download

**<TODO: Complete with all the mechanisms to download images provided by Kore Con X>**

Onboarding – Validations

After all the onboarding process has taken place we will have to run different comparisons to know who is this new shareholder.

Basically we will have 2 main shareholder ids, and 4 different onboarding scenarios.

* The 2 ids
  + The ***Kore Chain Id*** (***Kore Shareholder Id***)
  + The user ***national id***: Which will be the shareholder´s tax Id.

So in summary we will have an id which is only known to ***Kore Con X*** (the ***Kore Chain Id***) and another Ids which is a shareholder personal identification in the whole country (the ***tax id***). A shareholder could already be onboarded, but we might just not know his ***Kore Chain Id*** or we could even have a situation where the shareholder might be onboarded with a different tax id or kore chain id!

So, in summary:

* **Scenario #1:** Found shareholder by Kore Chain Id and Tax Id
  + Are these shareholders the same person?
    - If not an error will be thrown. This is a critical error and this can be considered some sort of inconsistency.
    - If they are the same person , all the shareholders fields are updated, the shareholders is put in an onboarding situation and sent to Solidus.
* **Scenario #2:** We find a shareholder by ***Kore Chain Id,*** but its **Tax id** does not match with the one that could be downloaded from ***Kore Con X***.
  + This is one of the possible inconsistent scenarios. Basically we have found “someone” already onboarded for the provided ***Kore Chain*** Id, but with a different ***Tax Id***. Is it the same user but its ***Tax Id*** has changed? Is it a different user with the same ***Kore Chain Id***? Anyway, this requires a case by case analysis. An error will be thrown and the new shareholder will be notified of this inconsistency in the ***Kore Con X*** platform.
* **Scenario #3:** We find a shareholder by the ***Tax Id*** received from ***Kore Con X***, but it does not have a ***Kore Chain Id*** loaded. This is a much more consistent scenario. Maybe this shareholder was just previously onboarded by some other transfer agent, but we can be sure that it’s the same person, as the ***Tax id*** is a much stronger key. Basically there should not be 2 people with the same ***Tax id***. In this scenario we will
  + Update the existing shareholder ***Kore Chain Id***
  + Mark the found shareholder to be in an ***onboarding status***.
  + All his information is sent to ***Solidus***.
* **Scenario #4:** This is the most traditional scenario. There is not a shareholder, weather we look by ***Kore Chain Id*** or ***Tax Id.*** In this case , the procedure is quite linear:
  + We create the shareholder and its user , using all the information received from ***Kore Con X***
  + Mark the new shareholder to be in an ***onboarding status***.
  + All his information is sent to ***Solidus***.

After any of the 4 previous scenarios have taken place, the user will be given a confirmation message or the onboarding will be rejected from scratch. If the application goes on and the shareholder personal data is sent to Solidus, the workflow will be under Solidus control.

**<To be completed: Is there some sort of confirmation email to be sent from Rialto or Solidus?>**

From this point on, the new shareholder will be interacting with Solidus site and all the relevant notifications regarding his KYC process will be managed by Solidus.

At the moment of writing this document, there is a high level definition about the user experience interacting with Solidus UI, but basically Solidus will have to interact with Rialto services , sending three different messages

* New shareholder user and password message
* Plaid onboarding message
* Application Approval message

Solidus – User and password update

**<To be completed – We still need to know more things like the encryption mechanism >**

Solidus - Plaid Onboarding

One of the most important actions that the new shareholder will take in ***Solidus*** website will be the possibility to incorporate his baking account into the Rialto platform. At the time of writing this document, the user will be requested to open a ***Piermont*** account and it will be this account the one that will be used for sending and receiving fiat money. It is beyond the scope of this document to specify how this whole signup process will take place as it is being designed and implemented by ***Solidus***, but basically after completing all this process, ***Solidus*** will inform ***Rialto*** the ***Plaid token assigned*** to this user.

A full description of the ***Plaid*** role as a banking hub and the use of the *access token* as an authentication/authorization mechanism can be found in the following links:

https://plaid.com/

<https://plaid.com/docs/api/tokens/#token-exchange-flow>

But the final result, and this is what matters most to ***Rialto***, is that a ***Solidus*** will invoke a Rialto service called ***Plaid/OnPlaidCredentialsLoad*** which will have the user’s email address and the ***Plaid Access Token*** described in the previous link.

After receiving this token, the ***PlaidCredentialsApp*** service will:

* Encrypt the ***access token*** using the same ***RSA algorithm*** that was described in the “Onboarding request – input data – Encryption mechanisms” section and persist this information in the ***plaid\_credentials*** table.
* The system will retrieve the **Rialto** ***client id and secret*** from the ***plaid\_settings*** table (populated through a UI in the administrator panel).
* The system will download the shareholder’s ***Piermont*** balance using the received access token plus the retrieved ***Rialto’s*** ***client id and secret***.
* The service will update the shareholders ***order lim***it (table ***firm***, column ***firm\_limit***), his ***buying power*** (table ***user\_info***, column ***buying\_power***), **cash on hand** (table ***user\_info***, column ***cash\_on\_hand***) and his ***trade limit*** (table ***user\_info***, column ***trade\_limit***)

Note: For more details about the validations implemented, refer to section “***2-Trading***”.

It’s important to remark that this step can take place in any given order, before or after the arrival of the ***Solidus Application Approval*** message, but the ATS will need all the three Solidus messages to login and start trading.

The other important message that will be received from ***Solidus*** is the ***application approval message***. At the time of writing this document this message has not been defined, but it will basically consist of the shareholder’s email (in ***Solidus*** world, this will be the main Id of a given shareholder).

The only effect that will be triggered after receiving this message will be to nullify the ***onboarding\_status*** column in the ***firm*** table, which will allow the new shareholder to login (assuming that his password has been properly updated as was described in the “*Solidus – User and password updat*e”)

In the repository the app is ***ApplicationApprovalService*** and the app will be deployed under the following URL:

**http://<ip\_address>/Management/ApproveApplication**

Sensitive data – Logging

As was mentioned before and described throughout the document the Rialto ATS will be interacting with several applications, own or third party, and this means that a lot of action will be taken under curtains which makes so important having a mechanism to track and troubleshoot all these interactions.

In this context, at the time of writing these lines, there were not a UI dedicated to tracking some sort of logging mechanism but the tables and endpoints existed.

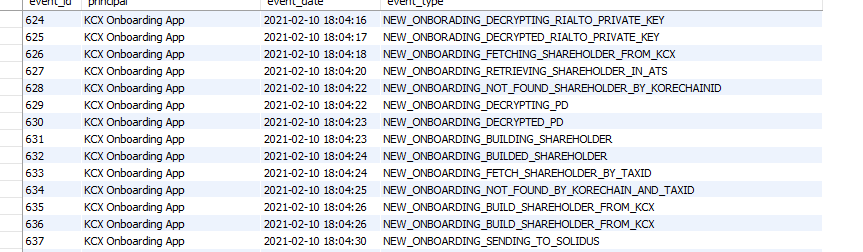
These tables are:

* + jhi\_persistent\_audit\_event
  + jhi\_persistent\_audit\_evt\_data

The mechanics of both tables is pretty simple, and reading ***jhi\_persistent\_audit\_event*** rows allows us to know different actions that took place throughout the execution of a given functionality.

It´s worth to mention that any logging in an application is very dynamic and everything that might be written one day will soon become outdated. So we will give a brief description about these tables through some example. But keep in mind that the main concepts described in this example, should apply for any other functionality that uses these tables.

The first example belongs, precisely to the onboarding service. Taking a look to the **jhi*\_persistent\_audit\_event*** tablewe can see that for an onboarding process, several rows, will be added with a title that can pretty much give an idea about what was the action referred in that row.



But then we can pick one specific row, and go into further detail using table ***jhi\_persistent\_audit\_evt\_data.*** If we take for example event 628 we can see that every event will be describe through 3 basics fields



* **id\_name** and **id\_value**: Every logging action will be related to something. This can be a security, a service, a shareholder, etc. Sometimes this shareholder could be identified by its firm id, his email or tax id. Other times, like the situation when someone is being onboarding and not much is known about him, it will be identified by exotic things like his *kore chain id*. Either way, in every recording, we will try to identify the entity affected by such action. So for every recording, the ***id\_name*** will describe the id that it´s used to identify the referred entity and the ***id\_value*** field, will precisely have such id.
* **Message:** Having identified the entity, we must have something to tell about it and a specific action that was taken in that pint in time. This will be recorded under the ***message*** key. In the previous example we can see that in an onboarding process, the onboarding service is informing that the onboarded shareholder, could not be found by the provided *kore chain id*. So at least, based in the received ***kore chain id,***  it´s a new shareholder.

At the time of writing this lines, there was not a UI that could create reports that would allow to have access to all of these records, but this will be one of the most important features to implement in the near future.

<**TODO: Design and Implement logging UI: tracking actions and Ids**>

**4-Settlement**

The settlement process is one of the most important in the operations of a stock exchange. Basically it will be one of the most complex processes of all, where a proper instrumentation (logging), a smooth, performant and intuitive interface have to be implemented to have solid and robust settlement.

**At the time of writing this document, most of this workflow is under design**. The document will be updated, but many changes can arise and this document will have to be updated accordingly. The special complexity lies in the fact that it is in this specific functionality that the ATS will be communicating through different steps with different actors (banking institutions, transfer agents, etc.) and it will be the ATS the one responsible of granting the consistency of all these interactions. The failure of a specific actor, should not propagate to an inconsistent state or to a different actor.

Matching trades

**To Complete the new CLOB (Central Limit Order Book) behavior.**

**If there is a match, an email will be sent to both participants and a new record will be created in table clob\_trade.**

Worfklow Commands

When clearing the trades, you will see the following commands that will allow to implement all the needed actions to finally have the money and the shares transferred.



Fiat Settlement



This button has to be pressed to create the NACHA file that will be sent to Piermont to transfer the fiat from buyer to seller.

After being pressed the trade will be marked as ***Bank Info Exported***.

Fiat Transfer Confirmation



This button has to be pressed to confirm that the fiat has been received by the seller. At the time of writing this lines it is being evaluated if this can be done automatically.

After being pressed the trade will be marked as ***Fiat Transferred***.

Shares Transfer Request



This button has to be pressed to initiate the shares transfer with the transfer agent. Some transfer agents have a synchronous transfer, other have an asynchronous one.

The important thing to mention is that the transfer request will be started with this button. Then we can go to the next step to confirm if the trades could be actually transferred.

After being pressed the trade will be marked as ***Shares Transferred Requested***, unless we have a synchronous confirmation, in which case , the trade will be marked as **Trade Cleared.**

Shares Transfer Confirmation



This utility of this button will depend in how is exactly the transfer agent confirming that the shares transfer is successful

* If the shares transfer is asynchronous, when pressing this button, depending on the transfer agent
  + The shares will be marked as transferred
  + The proper service will be called to confirm that the shares have been transferred
* If the transfer agent is synchronous, this button will be automatically pressed, after pressing the “Shares Transfer Request” button

At the time of writing this document, in Kore Con X, we are waiting for a “confirm shares transferred” service to be called.

After being pressed the trade will be marked as ***Trade Cleared***.

Default Scenarios

**<To be researched: Default Scenarios>**

**<To be completed/researched: Order´s expiration and cancellation>**